# Review of existing tools for working with schemas, metadata, and thesauri

John Gilbert[1], Mark H. Butler
Digital Media Systems Laboratory
HP Laboratories Bristol
HPL-2003-218
November 6th , 2003*

E-mail: gilberj@tcd.ie, mark-h butler@hp.com

SIMILE, RDF, semantic web, schemas, ontologies, thesauri

SIMILE, a joint project between HP, MIT Libraries and the W3C, is investigating applying Semantic Web tools, such as RDF (Resource Definition Framework) and associated schema languages, to the problem of dealing with heterogeneous metadata. Currently creating schemas involves writing them by hand using a text editor. As one of the aims of SIMILE is to simplify the deployment of heterogeneous data, it is desirable to investigate the applicability of schema authoring tools that reduce the need for expert knowledge and the risk of schema errors. This report reviews existing applications that support the creation of metadata schemas and graphical user interfaces for entering instance data based on those definitions. It also compares and contrasts these applications with existing applications used for the creation of thesauri, which have many similarities with schemas and ontologies.

* Internal Accession Date Only
Approved for External Publication
[1] Trinity College Dublin, Dublin, Eire.

# Review of existing tools for working with schemas, metadata, and thesauri

**John Gilbert (gilberj@tcd.ie)**[1]
**Mark H. Butler (mark-h_butler@hp.com)**
Hewlett Packard Laboratories, Bristol UK
01 October 2003

## Abstract

SIMILE, a joint project between HP, MIT Libraries and the W3C, is investigating applying Semantic Web tools, such as RDF (Resource Definition Framework) and associated schema languages, to the problem of dealing with heterogeneous metadata. Currently creating schemas involves writing them by hand using a text editor. As one of the aims of SIMILE is to simplify the deployment of heterogeneous data, it is desirable to investigate the applicability of schema authoring tools that reduce the need for expert knowledge and the risk of schema errors. This report reviews existing applications that support the creation of metadata schemas and graphical user interfaces for entering instance data based on those definitions. It also compares and contrasts these applications with existing applications used for the creation of thesauri, which have many similarities with schemas and ontologies.

## Keywords

SIMILE, RDF, Semantic Web, Schemas, Ontologies, Thesauri

# 1 Introduction

SIMILE[1] is a research project investigating how to extend DSpace[2], a digital asset management system developed by Hewlett-Packard Laboratories and MIT Libraries. It aims to provide an architecture for information search and retrieval across heterogeneous metadata that describes multiple collections of resources from disparate domains. Typically these collections use different metadata schema to define the structure and organization of metadata descriptions for a given domain.

### *What is the difference between ontologies, schemas and vocabularies?*

Currently terms like schema and ontology are used in multiple, conflicting ways. Therefore this report proposes some specific terms in order to distinguish between these conflicting definitions:

**Instance data**: Instance data is metadata that is specific to resources.

**Schemas**: Schemas are generic descriptions of collections of metadata. This report proposes that it is possible to distinguish between four different types of schemas:

[1] Summer intern at HP Labs Bristol from Trinity College Dublin, Eire.

vocabulary, ontological and constraint schemas and controlled vocabularies. In the Semantic Web specific schema information, possibly composed of all four types of schema, is identified using a namespace.

**Vocabulary schemas**: A vocabulary schema defines a set of classes and properties that can be used to describe a particular domain. Note a vocabulary schema is quite different to a controlled vocabulary.

**Ontological schemas**: Ontological schemas describe relationships between classes and properties, both within a namespace and between multiple namespaces. The Semantic Web community refers to ontological schemas as ontologies, but here we use the term schema to emphasise the similarity with the other schema language types.

**Constraint schemas**: A constraint schema describes constraints that are applied to classes and properties. Constraint schemas may be used to validate metadata associated with a particular schema, but they may also be used for inference.

**Controlled vocabularies**: A controlled vocabulary defines a set of terms that may be used as property values. The controlled vocabulary may optionally define relations between the terms such as synonyms.

**Namespace:** A namespace is a grouping of related information that uses a common name. A namespace may provide no information at all, but ideally it will provide a vocabulary schema, optionally supported by an ontological schema, a constraint schema and optionally one or more controlled vocabularies.

**Application profile:** An application profile is a vocabulary schema derived from subsets of several other vocabulary schemas and associated with a particular application.

SIMILE is investigating applying Semantic Web tools to the problem of dealing with heterogeneous metadata, so instance data will be represented using the Resource Definition Framework (RDF)[3], and associated schemas will be written in languages such as RDF Schema[4], DAML+OIL[5] or OWL[6] and XML Schema. In gross simplification, RDF Schema is primarily used for vocabulary schemas, OWL for ontological schemas and XML Schema for constraint schemas although both RDFS and OWL have vocabulary, ontological and constraint elements whereas XML Schema is primarily concerned with constraint elements.

## *Why are schemas necessary?*

RDF has much in common with the OEM semi structured database model[7] [8] [9] where property arcs are explicitly labelled so in principle schemas are unnecessary. So why do we need schema information in order to process RDF? Here we propose that there are several advantages for providing schemas in the SIMILE context:

- Providing vocabulary schemas means users only need to perform the operation of deciding what metadata is required to describe a collection of resources once, rather than for each individual resource.

- Schemas can provide labels and textual descriptions for classes and properties, that are more human readable than the URI notation used in the RDF model.
- Once a vocabulary schema has been defined, it is possible to reuse part of or the entire schema for describing a different collection of resources. This reduces the effort required to specify the metadata required by a collection, and may also simplify the task of searching multiple collections.
- Vocabulary and constraint schemas may be used to validate instance metadata[10].
- Vocabulary schemas can be used to automatically create user interfaces for entering metadata and to automatically create user interfaces for templated queries.
- Ontological schemas can be used to map one vocabulary on to another vocabulary, enabling searching across heterogeneous collections of resources.
- Ontological schemas can also be used to map between instance data that uses the same vocabulary, but where different approaches have been used for encoding data for different instances, for example the use of synoymns.

However, we note that there are limits to what can be described by ontological schemas based on hierarchical representations and hence the type of mappings that can be performed[11].

## *Why do we need tools for schema creation?*

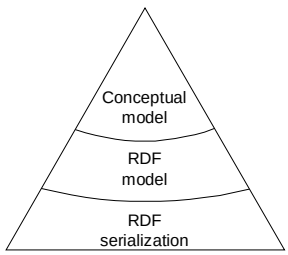
Regardless of the type of schema being created, creating schemas currently involves writing them by hand using a text editor, which is difficult, potentially error-prone, and requires expert knowledge of schema languages. As one of the aims of SIMILE is to simplify the deployment of heterogeneous data, it is desirable to investigate the applicability of schema authoring tools that provide graphical schema authoring environments, reducing the need for expert knowledge and the risk of schema errors. This report reviews existing applications that support the creation of metadata schemas for RDF based languages.

## *Why do we need tools for metadata creation and visualisation?*

Just as it is desirable to have tools to help create schemas, we also require tools that help users create metadata. However as metadata varies with schema, it is undesirable to require custom tools for each schema to be supported. Instead we need tools that make use of schema information, and optional additional information to automatically create an authoring environment for metadata that uses particular schemas.

## *How does this relate to existing work?*

Although the use of RDF based languages is relatively recent, the library community already uses similar applications for the creation of thesauri. Therefore this report also considers these applications, and compares and contrasts them with the applications designed for RDF based languages.




In the discussion that follows one important distinction between the applications is the level of abstraction from the underlying data representation. RDF can be serialised in

many forms, but the current serialisation recommended by the W3C is RDF/XML, an XML, text-based serialisation that can be created using any plain text editor (vim, emacs and others). Above this level, one can work with RDF at the data model level, where the data model is a graph composed of individual triples. Going up a further level, we reach the highest level of abstraction the underlying RDF data model is also hidden, and instead the tools focus on the conceptual model of a schema associated with a particular namespace or application profile. Here details such as anonymous nodes in the underlying RDF data model are hidden from users.

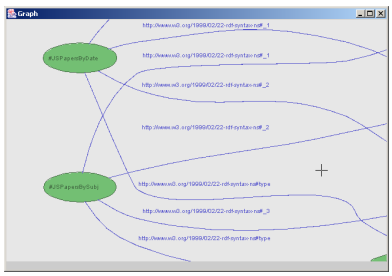
Note that although tools at the various levels use a graph as part of their interface, this graph represents different things: RDF editors use the graph to display the actual RDF graph model of the triples that underpin RDF, while the schema creation tools use the graph as a metaphor to represent the actual structure of the schema being developed.

# 2 RDF Editors

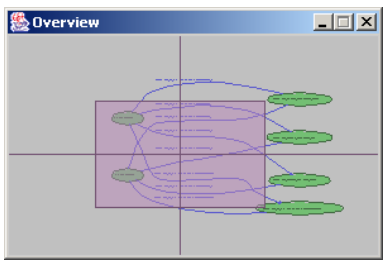
In the following section on RDF editors, the two tools evaluated display the RDF document as a two dimensional graph. The ultimate goal of both tools is to simplify editing RDF documents, without requiring knowledge of the underlying serialisation e.g. RDF/XML. As such, both tools described are visual with a drag and drop interface, where users 'draw' the nodes (either resource or literal) on screen, and click between them to add properties.

As RDF Schema, DAML+OIL, and OWL schema languages are all expressed in RDF these tools can be used for creating schemas in these languages as well as editing instance data. This is the lowest possible level of tool for working with RDF, short of using a text editor to create documents by hand.

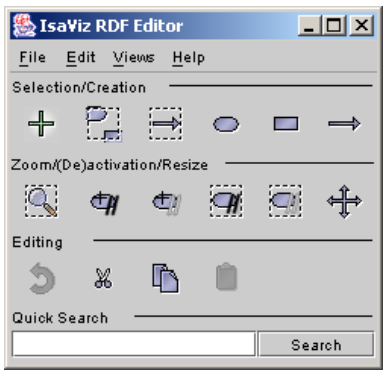
## *IsaViz*




The IsaViz[12] interface consists of a number of windows. The main window displays the RDF graph, which is easy to navigate by zooming or panning the display: functions readily available by dragging with the right mouse button pressed.



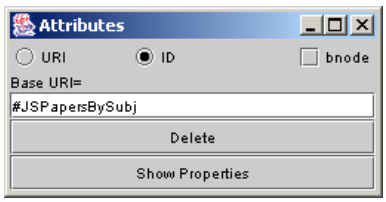

An overview window provides information about where in the graph the current section displayed in the main window has come from, and gives the ability of navigating at a high level.



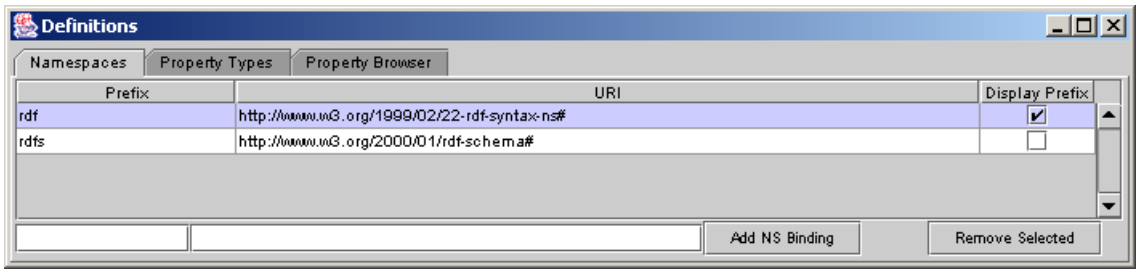

A toolkit window provides a toolbar with the main tools used in creating and modifying a graph, with options for selecting nodes and properties, or creating nodes (either resource or literal). It is possible to 'deactivate' a section of the graph, effectively commenting it out in the serialisation. This could be used to 'remove' nodes

and properties from a description, without losing the structure and information contained within them. It is also possible to resize and modify the layout of sections of the graph. This information is then stored in a custom IsaViz file that stores both the RDF and the layout information. When you wish to publish the model so that others can use it, you must export the RDF in an appropriate serialisation.

An attribute window displays the attributes of whatever node or property is currently selected in the graph, allowing for modification of the items property values.

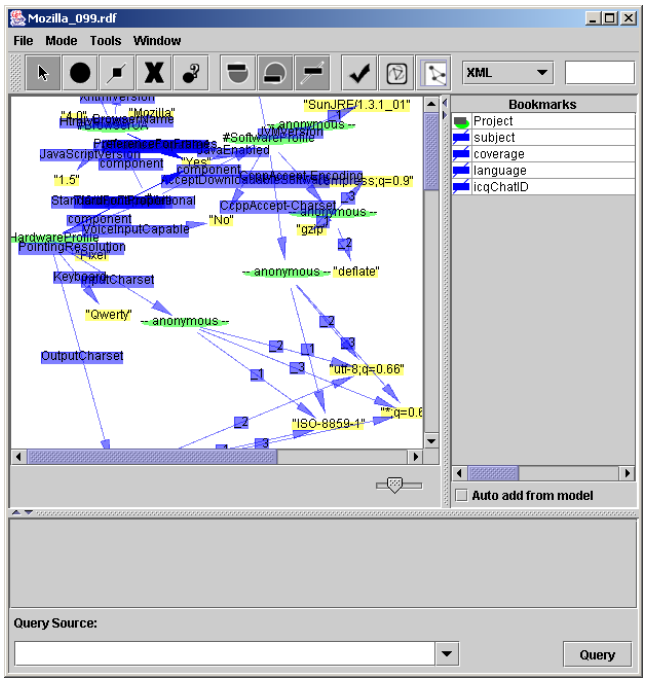



Finally a definitions window provides access to manage multiple namespaces (by associating prefixes with full URIs), property types used in the graph and the ability to browse properties of a selected node in the main window.




Schema support is lacking in the tool: building new metadata instances is done using an existing instance document rather than the actual schema. IsaViz extracts the properties from the existing instance document to make them available in the 'Property Types' tab of the Definitions window. Classes however must still be typed by hand.

One limitation with IsaViz is that it uses the AT&T GraphViz DOT layout algorithm to initially render the display. This graph toolkit is platform specific so the binary version is not portable between platforms, although IsaViz itself is written in Java and made available under a GPL-compatible license.

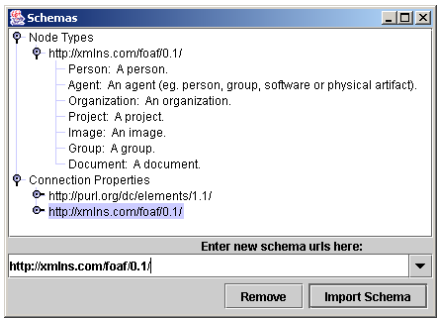



## *RDFAuthor*

RDFAuthor[13] has been developed for Mac OS X, though a Java Swing port is available for cross platform usage. It is distributed for free under the GNU GPL.

Although the graph layout is not as clear as the one provided by IsaViz, the tool provides better RDF Schema support. It is possible to load an RDF Schema, which makes the Classes and Properties defined in that schema available in a separate 'Schemas'

window, for drag-and-drop usage in creating an instance document. When classes and properties are added to the instance document, the application automatically types nodes and properties.




Navigation within the tool involves zooming in or out, and sliding the view using scrollbars. The user can configure the size of the area within which the graph is displayed. Depending on the size of the graph, the view can become cramped, unlike that of IsaViz where the space for rendering is considered infinite.

### *Summary*

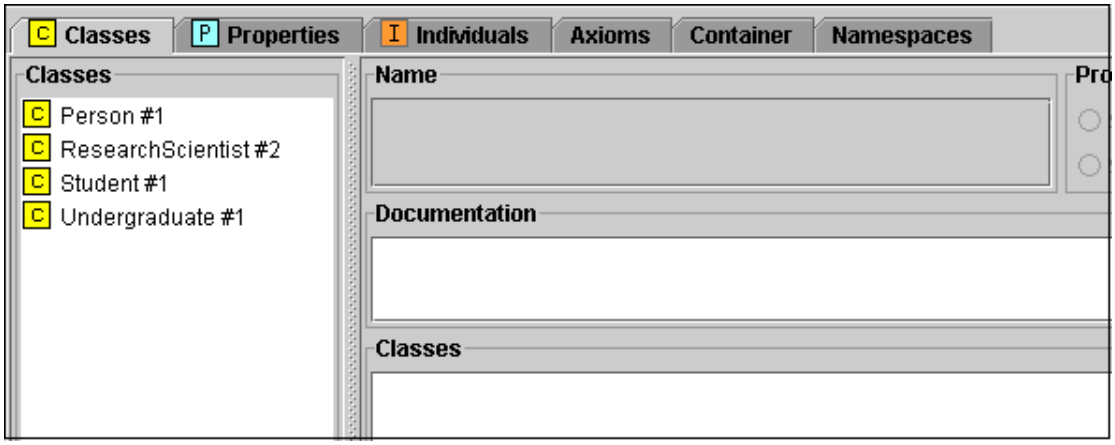
Both IsaViz and RDF Author are suitable for use by people familiar with the RDF data model. The tools are a step in the right direction as far as support for editing RDF/XML documents is concerned, but they are not necessarily suitable for users who do not require knowledge of the underlying data model. Furthermore using these tools to create schemas and ontologies requires specialist knowledge of the associated RDF vocabularies e.g. RDF schema. The tools avoid some syntactic errors in the creation of schema but they do not facilitate easily capturing the relationships between properties and classes as they operate at the base RDF level so the user manipulates nodes and arcs, not at the level of the conceptual model, where the user would manipulate classes, properties, relations and constraints.

## 3 Schema Editors

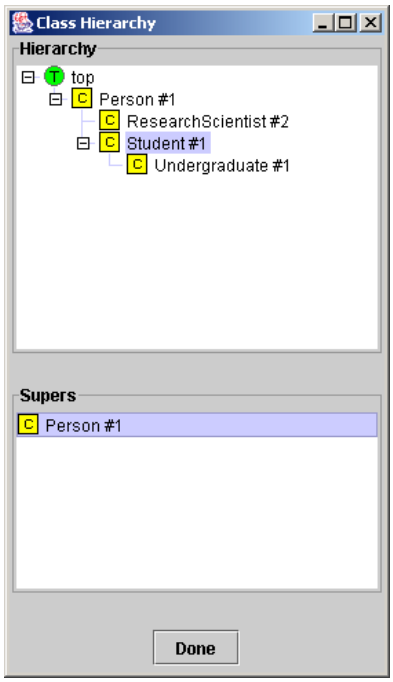
Here we consider tools specifically designed for the creation of schemas. Most of these tools also facilitate knowledge acquisition and any relevant features are noted. These tools all provide a conceptual model of the schema. For other reviews of tools for creating schemas and ontologies, see [14] [15] [16]. For an excellent introduction to how to create ontologies and schemas see [17].

### *OilEd*

OilEd[18] is a tool created specifically for editing DAML+OIL documents. DAML+OIL is a precursor to the W3Cs OWL. OWL was developed from the DAML knowledge interchange language commissioned by DARPA, and a language created by the description logic community called OIL. The main OilEd application interface consists of a number of tabs, each providing a different editing option. Within each tab the left of the screen is occupied by a pane listing the relevant items for that pane, for example the pane contains a list of all classes in the classes tab, while it contains all properties while the properties tab is selected:




- Classes: The classes tab allows the user to define classes, document those classes, define super / sub class relations and attach properties with specified restrictions.
- Properties: The properties tab allows the user to create and edit DAML+OIL properties. In addition to creating properties, DAML+OIL provides mechanisms for defining inverse, hierarchical and transitive relationships between properties, as well as domain and range restrictions amongst others.
- Individuals: This tab allows for the creation of instance data using the defined ontology, for example specifying what classes this instance is an instance of, applying properties to the instance etc.
- Axioms: This section allows for the definition of logical axioms that apply to the ontology, such as specifying that two classes are disjoint i.e. Class Male and Class Female may not have any instances in common.
- Two 'housekeeping' tabs: Container, which provides general information about the ontology and Namespaces, which allows for the control of multiple namespaces within the ontology.




The classification of classes and instances in OilEd is performed using a reasoner external to the system. Based on class definitions, it creates an appropriate class hierarchy (using both the specified sub/super class relations, and also using those inferred by the system). Rather than typing instance data explicitly, OilEd is designed so that the reasoner can infer which classes a particular instance belongs to and classify the instance accordingly. The class hierarchy output is shown in the adjacent window.

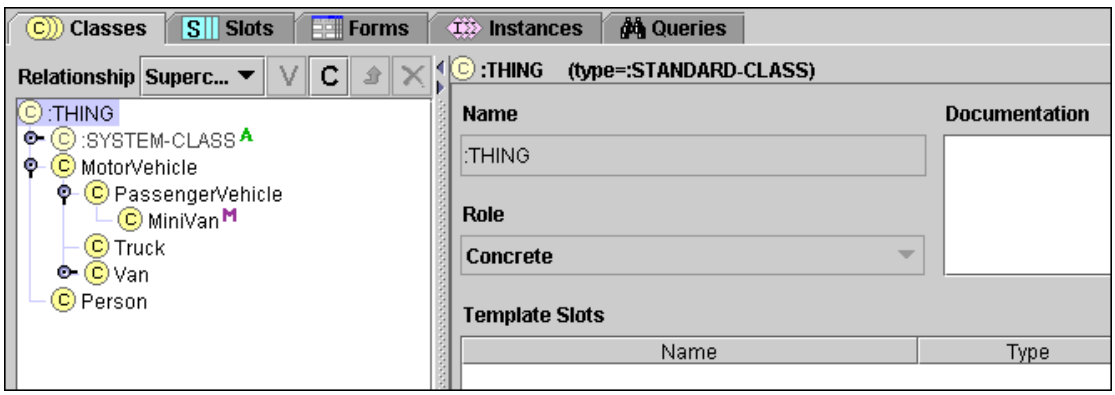
The main list of classes and properties displayed on the left of the interface does not reflect the sub/super class relationship hierarchy, but is simply ordered alphabetically. After invoking the reasoner, double clicking on any class in the class list will display a hierarchical classification of all classes. This view is not available for properties or instances however.

OilEd supports importing other DAML+OIL projects, allowing for mixing different schemas when creating metadata descriptions. It also supports the use of multiple namespaces using a prefix #(INT) on class and property names to indicate which user-configured namespace they are taken from. It is available open source under the GPL.

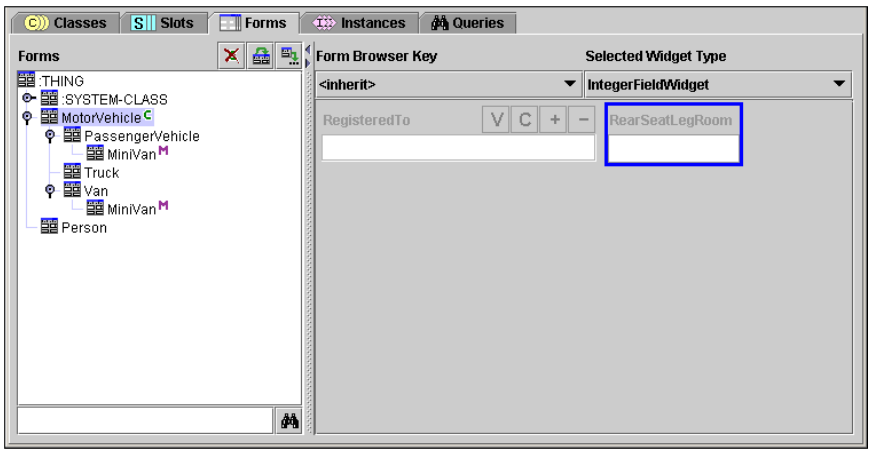
## *Protege-2000*

OilEd is strongly based on Protégé-2000[19], so there are a number of similarities in their user interfaces. Tabs in Protégé are available for:

- Classes: This tab is used to define classes, including sub/super class relationships and restrictions upon those properties attached to the class.
- Slots: Slots are another name for properties.
- Forms: This tab allows the user to create a form for entering individual class instances.
- Instances: This tab is used to create class instances.
- Queries: For searching and creating stored queries of the instance data in the knowledge base.




The class/slot lists are displayed as hierarchical trees modelling the defined sub/super class/property relationships. Elements within the hierarchy with multiple super class/slot relationships are replicated in the tree structure. Additional tabs are available depending on installed third party plug-ins.

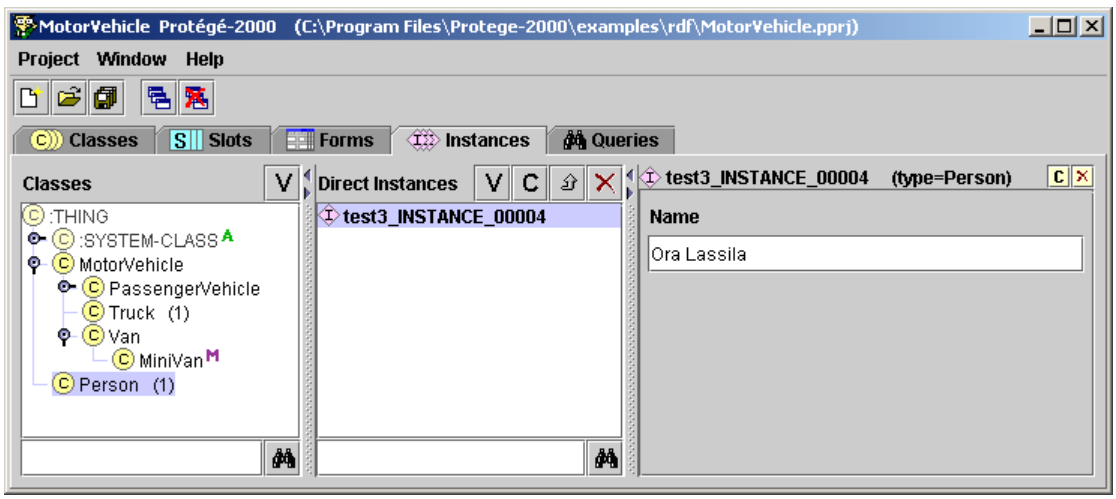


Amongst the applications considered here, the 'Forms' feature is unique to Protégé but is found in simple database applications such as Microsoft Access. Although it is possible to automatically create user interfaces for data entry from vocabulary schemas, sometimes such interfaces will be inappropriate for a number of reasons:

- It may be desirable to give a slot a different name to that used in the schema.
- It may be desirable to place slots in a particular order, for example to reflect work flow.
- It may be natural to associate some slots with different UI widgets apart from simple text boxes, for example a drop-down box in the case of a controlled vocabulary.
- The user may never provide values for certain slots, so they may be omitted from the user interface, for example when a slot value may be calculated from other slot values.
- The user may need to decide on object type first as this determines what slots will be available.

Protégé generates an initial knowledge acquisition interface for each class defined which can be tailored by hiding fields, and restructuring them on the screen.

Unfortunately when creating a metadata instance, it can only be created for one class: although multiple inheritance is supported in the class hierarchy, multiple typing is not supported for instance data. In this sense then, unlike OilEd, Protégé currently adheres strictly to the notion of an objects properties being dependant upon class type.

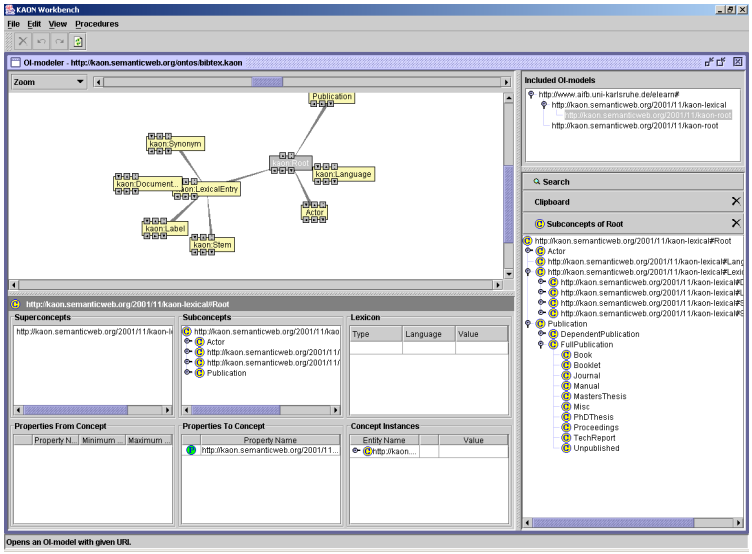



Navigation of instance data is done using a mirror of the class hierarchy, which displays a list of instances in a separate pane when a class is selected. This is in contrast to the OilEd navigation, which, like its classes, simply lists all instances alphabetically. The Protégé approach to metadata-browsing is more structured than that of OilEd, and it also offers better query support than the simple keyword search provided by OilEd.

An additional problem with the current implementation of Protégé is its lack of support for multiple namespaces. Although it supports 'including' external projects, to reuse their terms, the single-namespace problem means that terms which are repeated will conflict. Protégé is available open source under the Mozilla public licence.

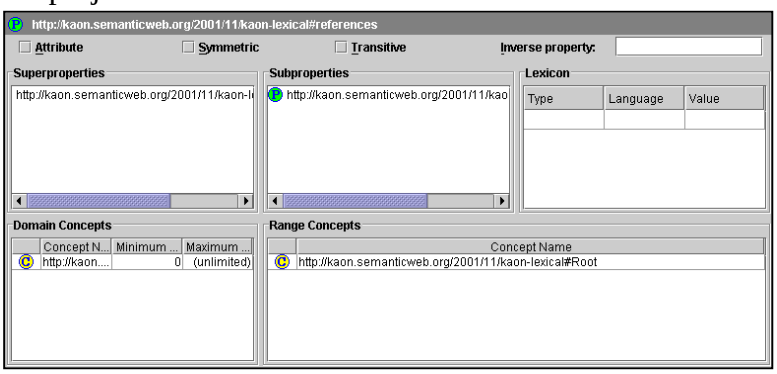
## *KAON OI-Modeller*

KAON[20] is an open-source ontology management infrastructure targeted for business applications and is made available under the Lesser GPL. It includes a comprehensive tool suite including OI-Modeller. The graph layout algorithms are based on the TouchGraph library[21]. The tool is still under development, and currently supports editing of the vocabulary schema only i.e. the class and property hierarchy.

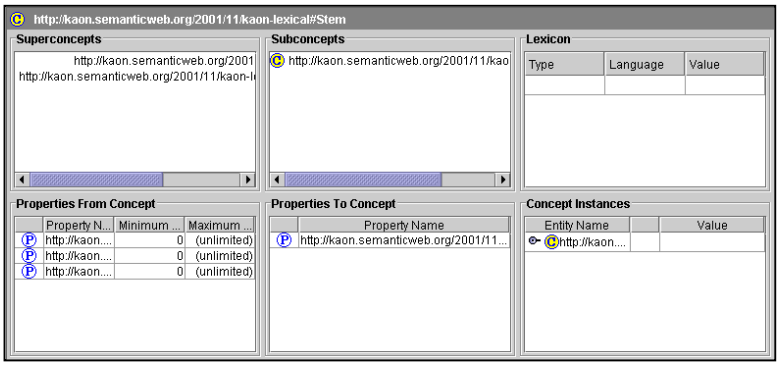



The user interface appears to be a cross between the simple RDF editors that use a 2D graph-tree visualisation, and the 'options' panes available in both OilEd and Protégé for modifying class and properties respective properties. The graph displayed can be expanded or contracted as required to display sub and super concepts. It is possible to view a concept hierarchy on the right of the screen, as well as inspect any other ontologies included within this project.

The available options at the bottom of the screen are context-based on the item(s) selected within the graph, and without the 'tabs' style of Protégé and OilEd, it is initially unclear what exactly you are working with.



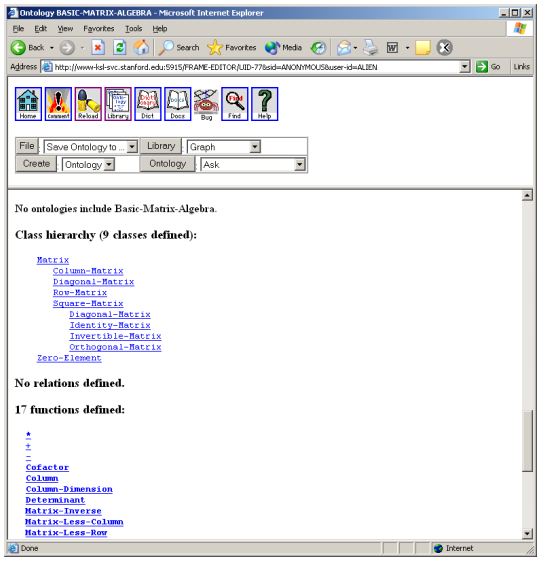

At the time of writing this report, this tool is at an early stage of development; in particular its graph display was found to be unreliable and the



software lacks documentation.

## *Ontolingua*

Ontolingua[22] is a browser based ontology editor, created by Stanford University. After logging in it is possible to either browse existing ontologies from the library, or create a new ontology. Creating a new ontology begins with a basic definition of what the ontology is about, and any ontologies from the library that should be included.




The screen is split into two frames, a top frame with options for searching, adding classes, axioms etc. This top frame controls the contents of the bottom frame.

The interface for editing concept properties is based on standard HTML forms, and hyperlinks are used throughout for both navigation and selecting editing options. Rich in functionality, the tool is slightly overwhelming as a web based application.

Simple features, taken for granted in the other tools evaluated, are lacking in this tool. For example, when creating a class, it must be a subclass of at least one existing class. This super class name must be entered by hand into a text box during the new classes creation. In a tool such as OilEd this is done by right clicking on the class to be sub-classed.

## *Summary*

With a wide user base, Protégé sets the de-facto standard against which most ontology creation software is compared. Although OilEd is strongly influenced by Protégé, the two tools diverge on many topics: multiple typing for instance data, how classes fit into the overall design and structure of instance data i.e. are they used to define templates for the data as in Protégé, or used to classify arbitrary descriptions as done using the external reasoner in OilEd. Possibly a side effect of the underlying ontology description, these differences highlight that there are several possible approaches to schema modelling.

The KAON OI-Modeller, although still in development provides an alternative approach to visualising the ontology under consideration, with its graph based conceptual model. Like Protégé and OilEd, OI-Modeller provides standard features for working with an ontology, modifying concepts and sub concepts, applying restrictions and so on. Currently it does not support more than simply editing the concept hierarchy, and its contractible-graph is unstable and difficult to navigate, making the tool awkward to work with.

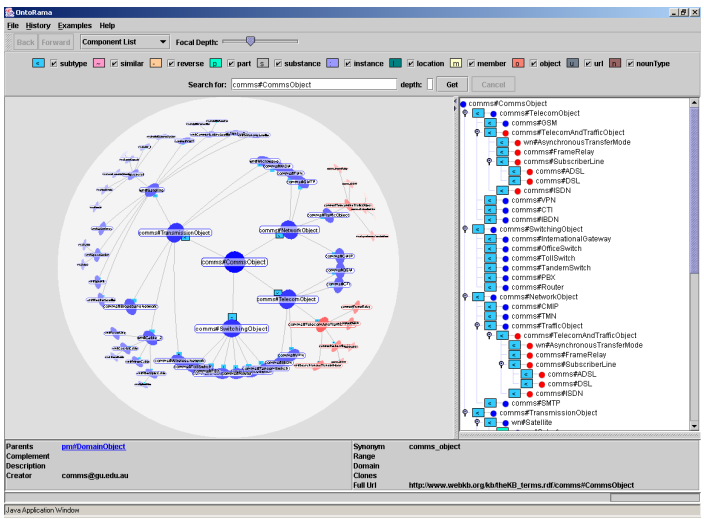
Protégé's support of knowledge acquisition is far superior to that of any other tool tested, giving excellent configurability via its forms designer. However the data navigation tools provided in both Protégé and OilEd are poor, OilEd offering none past a simple alphabetical list, and Protégé offering a 'browse-by-class' and query search. Such interfaces are unlikely to be sufficient for large collections of instance data that will be a feature of SIMILE. Perhaps a form of faceted search would be a flexible, scalable and user-friendly approach to the problem.

# 4 Ontology Visualisation software

A number of 'ontology' browsers exist, which place a different visualisation on the underlying structure of the ontology to make it more usable. These tools were included as none of the visualisations were used in the construction of the other tools evaluated. All provide a conceptual model of the ontologies being viewed.

## *OntoRama*

OntoRama[23] is a prototype ontology browser that uses a fisheye visualisation. On the right hand side of the screen a hierarchical tree is displayed, while on the left hand side (the main section of the applications interface) a graph view is provided, using a hyperbolic view, so that those classes near the centre of the window are displayed reasonably large while those at the extremities are relatively small.




Navigation can be performed using either the tree on the right, or dragging the graph displayed on the left. Class properties are displayed at the bottom of the screen.

Though multiple super classes are supported, the sub-hierarchies are 'cloned' to avoid the confusion of using a real graph i.e. the graph is converted into a tree. Cloned sections of the graph in both the hyperbolic graph view and the tree view are displayed in red, making it easy to recognise them. The tool can read in RDF/XML.

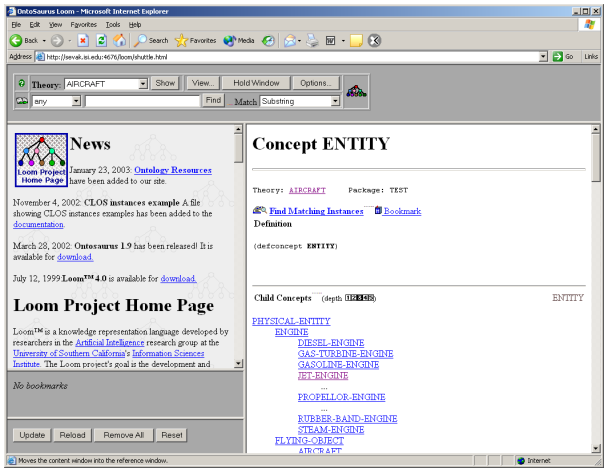

## *VIUM*

VIUM[24] (shown on the right) is another ontology visualisation tool that uses another type of fisheye view rather than a standard tree-based hierarchy or graph based view. Instead this visualisation allows something of a cross between the two: All classes are displayed, but once one is

selected, the view re-organizes itself, making the selected class and associated classes appear in a larger font, and pushes those very distantly related into the background, with others varying in between. Colour coding and indentation of terms can also be used to control the visualisation.

### *OntoSaurus*

OntoSaurus[25] is a web-based browser for the 'loom' language, with an interface much like that of Ontolingua. The top section of the screen is used to select 'theories' to work with or search for items in the knowledge base, which then replaces the right hand side of the screen with the chosen theory. This screen then displays top-level concepts, child theories etc. Clicking on a concept displays that concepts structure, axioms, sub/super-concept hierarchy, and instances of the concept. One section of the screen allows for 'bookmarking' a section of the ontology for quick-access. RDFAuthor also has a similar feature. The Interface uses the webs ease of navigation via hyperlinks to make navigating the ontology being viewed a simple task.



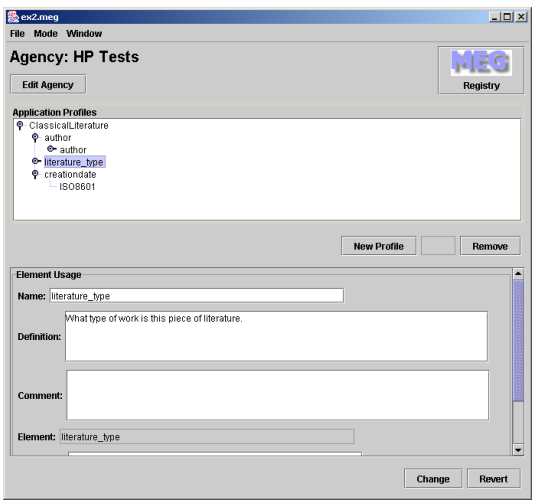

### *Summary*

Each of the three visualisation tools/ontology browsers evaluated provide a different view than that used by the other tools discussed in this report: Ontorama with its hyperbolic graph view which replicates nodes pointed to by multiple properties in order to provide a tree view, VIUM with its hybrid hyperbolic-graph/structured tree view, and finally the basic OntoSaurus web-based hierarchical viewer which is comparable to the initial thesauri view in TCS8 discussed in a later section.

# 5 Application profile editors

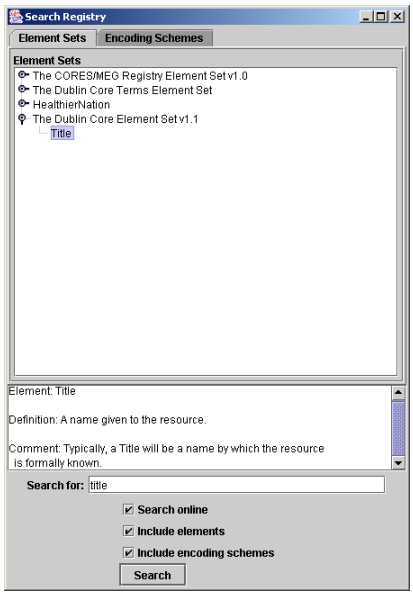
### *SCART: The MEG Registry Client*

The MEG Registry project[26] at UKOLN[27] has developed a schema creation tool specifically designed to encourage re-use of existing element sets and encoding schemes aimed specifically at librarians. An element set is a vocabulary schema i.e. a group of related properties used to specify property-value pairs in metadata descriptions, such as the 15 elements that form the well known Dublin Core[28]. Encoding schemes are used to control the form that values for an element may take. The MEG client supports two forms of

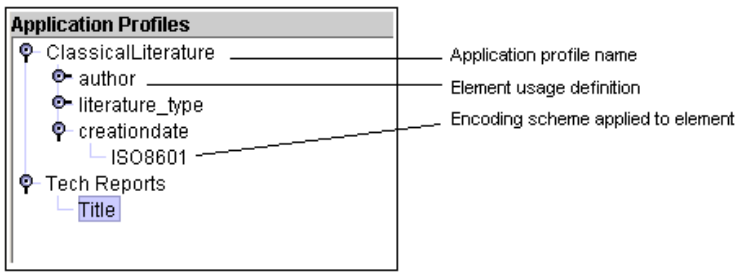


encoding scheme: data types that provide simple descriptions of the encoding scheme, including pointers to external specifications (e.g. a date format), and descriptions based on an enumeration of values (i.e. a controlled vocabulary). Due to the user interface provided, this enumeration is only suitable for small sets of terms with less than 15 members.

The tool supports the creation and editing of application profiles that reuse existing vocabulary schemas, as well as the creation of new vocabulary schemas and associated encoding schemes hidden away under 'advanced functionality'.

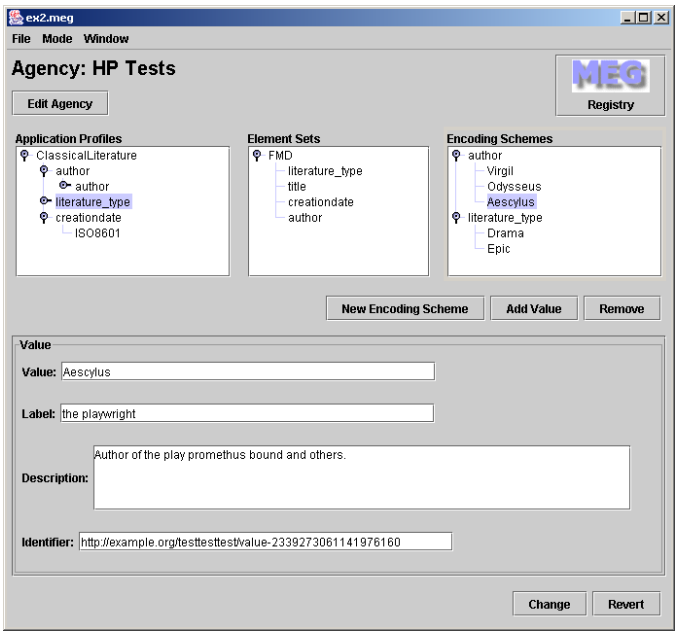



The main window allows the creation of application profiles, displaying all profiles as a tree. The lower section of the window provides a context sensitive property editor for whatever is selected from the application profile (or later, the element sets and encoding schemes).




A second window provides the ability to search a registry for existing element sets and encoding schemes. This is based on a keyword search, which returns a list of all related items in the registry. To use either an encoding scheme or element from the registry, it is a simple matter of dragging the item from the 'search results' window onto your application profile, and modifying its description. One problem here is the lack of any visual indication of the source of elements and encoding schemes within the application profile tree i.e. did the element come from the registry or from a locally defined schema?

By enabling advanced mode in the main window, two additional columns appear: one for element sets, and one for encoding schemes. As with the application profile column, selecting either of these columns will display an appropriate property editor in the lower half of the screen.




Note that the element sets and encoding schemes editor is very simple, and only creates

flat element sets with no hierarchical relationships.

Also the UI design is deliberate to try to encourage the re-use of existing elements and encoding schemes rather than the creation of new elements.

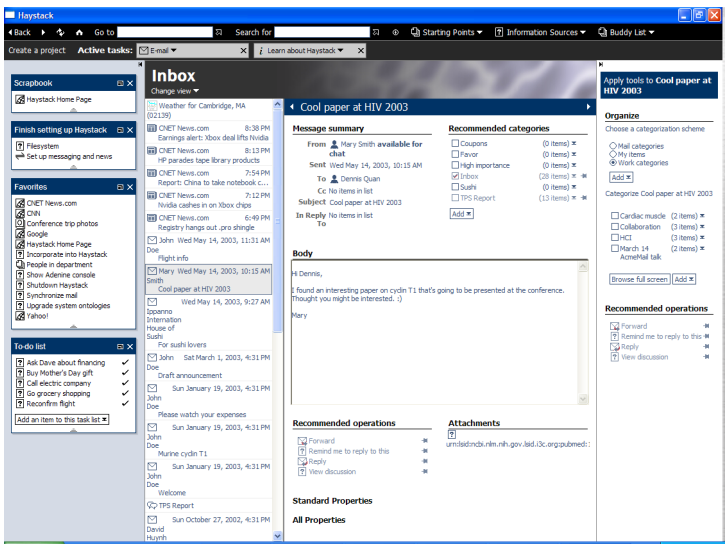
### *Summary*

The MEG registry client provides a basic tool for working with application profiles. The software performs its job admirably, however there are many possible improvements: for example provide support for dealing with larger controlled vocabularies, better tools for searching existing schemas, various possible user interface improvements and indicate the provenance of elements and encoding schemes in application profiles i.e. whether they have come from locally defined schemas or from an online registry.

# 6 Meta-data instance editors

### *Haystack*

Haystack[29] is a tool that has been developed to allow users to manage all of their information in a way that makes sense to them. The project aims to remove the barriers imposed by applications that are designed to work with specific information types. Haystack is built using RDF as its underlying data-model, and work has been done within the project on providing RDF authoring environments for end users[30].




Work within the project has identified three main types of user interface component, known as a view, which can be used to represent one or more resources on the screen. They are:

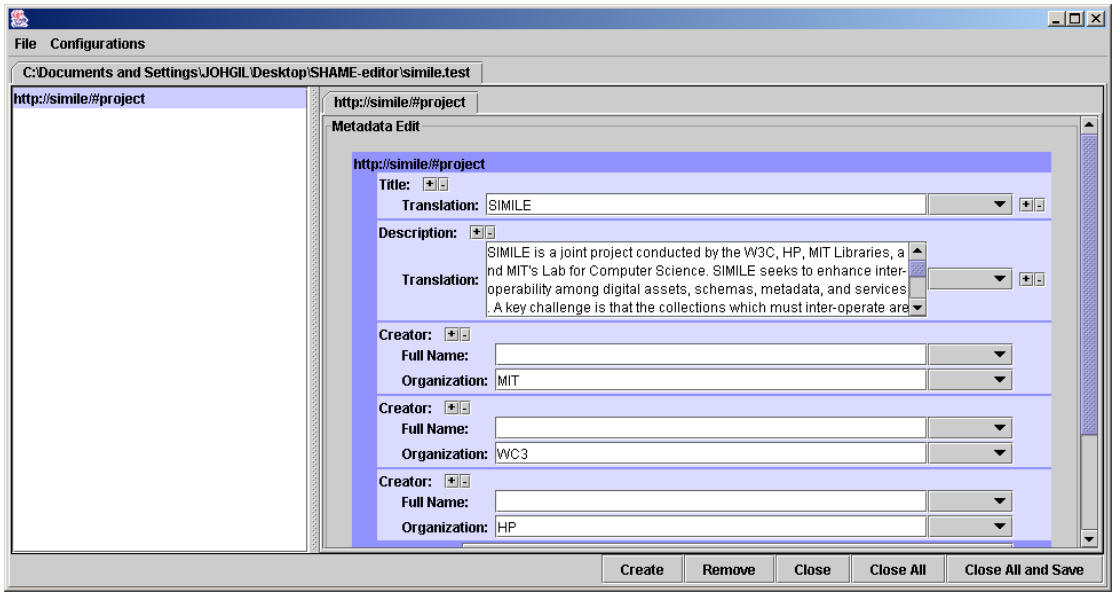
1. **Property editing** allowing users edit property-value pairs.
2. **Managing lists and taxonomies**, important for classification of data and enables users to quickly locate data again.
3. **Specifying and viewing relationships between objects**, while hiding the complexity of the overall data model (i.e. only those relationships of immediate interest are displayed).

There are various implementations of each view, and views can be nested within views to display an object in the most suitable manner for a given context, thus providing flexibility within the system. For example property editing views may be nested in a managed list view. By providing various views on a single underlying

resource, the problem of allowing users manipulate the properties of a resource can now be cast as a problem of providing appropriate views for a resource for the user(s) of the system.

## *SHAME*

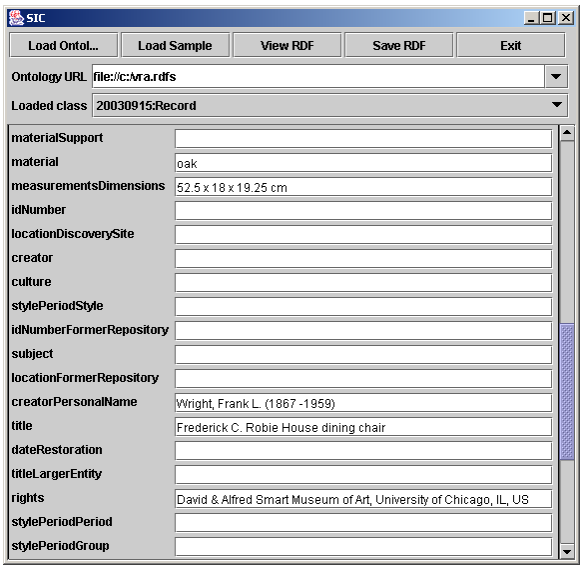
The Standardized Hyper Adaptable Metadata Editor (SHAME)[31] is a framework for building RDF based metadata editors. This framework aims to help provide appropriate abstract views on metadata for different users of the system, which may then be presented differently by different UI generators. Core to SHAME is its configurability through a data modelling part (SHAME Query Model), and a form specific part (SHAME Form Model). Configuration requires writing configuration files in RDF using reified statements to specify a Query Model and Form Model. A Query Model is used to identify data to bind to a view, while a Form Model is used to specify display information.




A demonstration application, Configurable Editor[32] (shown above), is available with sample configuration files for a LOM editor, a Dublin Core editor, and a Form-Form editor (used for editing SHAME form models).

### *SIC*

Simple Instance Creator[33] is a tool that allows users create DAML+OIL instance data using a form-based interface. The interface is generated based on the provision of an Ontology to the software. Forms are created based on daml:Class definitions, and can be nested hierarchically to form a single interface for a network of Classes which have been linked together using properties. daml:DatatypeProperty properties are displayed as the actual fields which users input data into.




### *Summary*

Haystack provides users much flexibility in both organizing their information, and displaying it on screen in a meaningful way. Views are described in a programming language written in N3 called adenine, rather than being derived from schemas, and must be customised for different schemas. SHAME on the other hand uses configuration files written in RDF that make extensive use of reification to specify paths to bind instance data in order to populate form views. Unlike Haystack, the framework does not specify any particular approach to presenting instance metadata via the UI, so it is left to the developer or a UI generator to define the interaction between user and data that may take place. SIC is the simplest of the tools, and although it provides no possibility of configuring the generated interface, it allows users to select a DAML+OIL ontology and then automatically generates a form from the ontology, allowing the user to enter instances of classes. Since forms are generated directly from the ontology, a limitation of the tool is it is not possible to specify when properties are required and when they are optional, or use conditional logic that only makes certain properties appear in certain contexts, which can assist users entering metadata into the form.

# 7 XForms

XForms [34] [35] is a technology being developed by the W3C for combining XML and Forms. It is a declarative way of creating user interfaces for creating, editing and viewing instance data represented in XML. It uses three other XML technologies, XML Schema and XPath, and a new technology created by the XForms and HTML Working Groups designed to reduce the need for scripting called XML Events. It provides for support collecting data, but can also perform additional processing such as calculations, validation, and can determine which form controls are relevant, read-only or required.

Unfortunately even though RDF can be serialised in XML, it is not always amenable to processing with XML tools, for two reasons: first because there are multiple XML serialisations of the same RDF model, and second because XML documents are modelled as trees whereas RDF models are graphs. Recently there has been growing interest in creating workarounds that overcome these problems to allow XML tools to be used with RDF/XML - for example RDF Twig[36]. This report will not discuss the suitability of XForms for use with RDF/XML, and hence its use in SIMILE, but mentions this technology because it is relevant and we would like to propose that any future work on declarative descriptions of UI's for manipulating RDF should consider the existing XForms work.

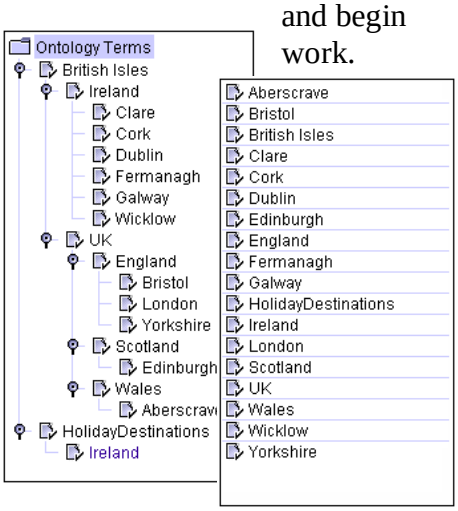
# 8 Thesaurus construction software

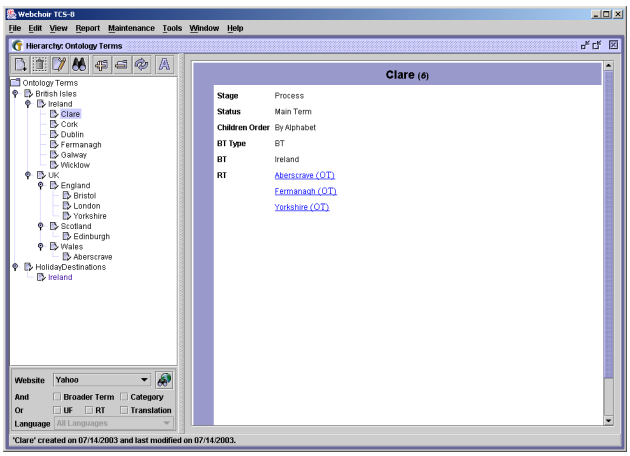
Thesauri are a particular type of controlled vocabulary that have additional relations between terms, for example indicating narrower and broader terms (abbreviated NT and BT respectively). Therefore the terms represent classes and the relations indicate subclass relationships, so thesauri are related to vocabulary schemas and ontological schemas. In addition the software used for the creation of both is reasonably similar regarding user interface requirements. When creating instance meta-data from schemas, we use specified controlled vocabularies and thesauri to source the values for some attributes.

*The following tools are all commercial, and have been evaluated using 'trial' versions. A variety of other commercial tools exist, but did not provide trail or demonstration versions, and hence are not included here.*

## *Webchoir TCS-8*

TCS-8[37] is a multi-user system. A manager uses the Thesaurus manager to control user accounts, and Thesauri/Permissions. The second tool provided as part of the package is the thesaurus editor. After initially logging in, one can open a thesaurus and begin work.

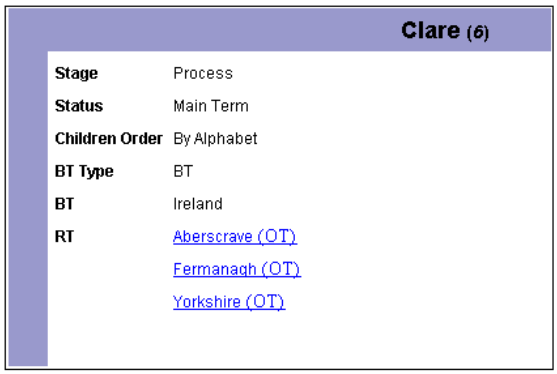



Like the Protégé UI, the left hand section of the screen is used to display the preferred terms used in the thesaurus – by default as a standard hierarchy, though it is possible to turn it into an alphabetically ordered list as in OilEd.

In WebChoir, a thesaurus that allows multiple inheritance is known as a polyhierarchy. When

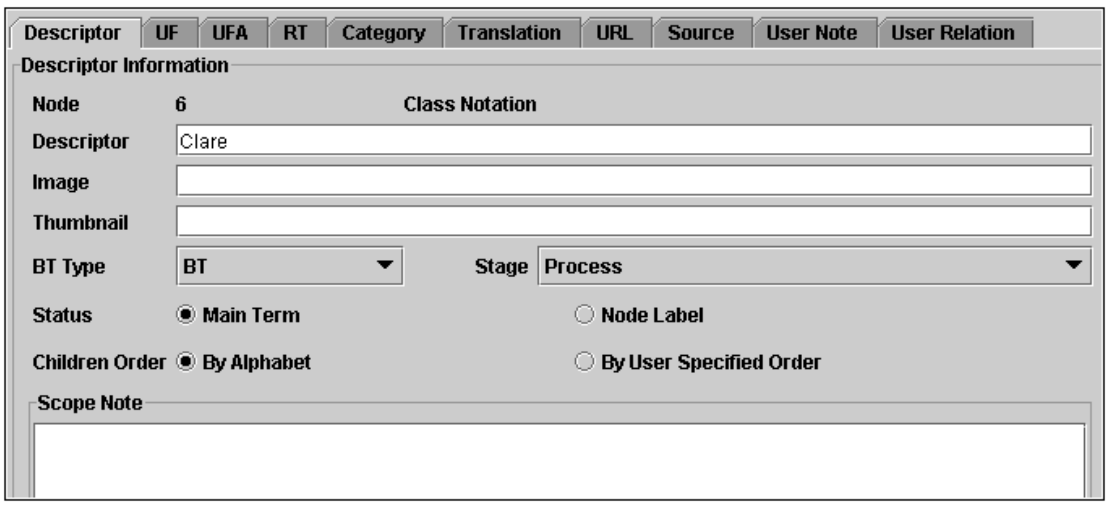
displaying polyhierarchies, WebChoir will replicate the term to which the multiple BTs have been applied to under the additional terms. Note that narrower terms (NT) will not be replicated in the tree view however.




Once terms have been created, it is possible to browse terms using a web-like interface based on hyperlinks as shown in the adjacent screen shot. Here a description of the term includes information such as Status, Broader Terms (BT) and Related terms (RT). Narrower Terms are not listed however. Any linkable terms are hyperlinked for browsing.

When editing a term descriptor, the right hand side of the main window takes a tabbed-appearance like Protégé, providing tabs for working with the terms properties: descriptor (name, image, stage, status, scope note), used for (UF) and used for and (UFA) relationships to denote synonyms and to indicate preferred terms, related terms, category, source etc.

As well as creating terms using the descriptor tab, it is also possible to create terms via relationships such as related terms. They are then added to a list view of all terms available in the thesaurus.

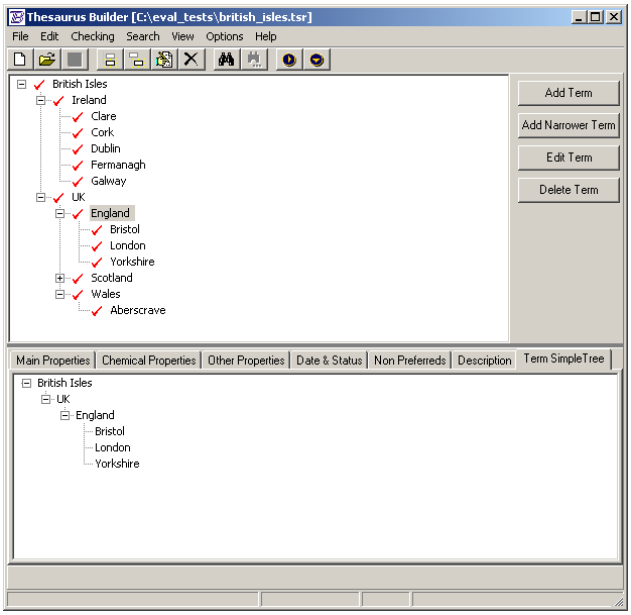



The tool supports adding translations for other languages, using the ‘translation’ tab although no translation options are available in the demo version.

It is possible to import and export different formats including ASCII, XML and MARC, but some of these features are disabled in the evaluation version. It is also possible to generate a hierarchical website of the terms. Finally the option is available to generate a number of different reports on the thesaurus: alphabetical, hierarchical, partial hierarchies, descriptor by category and so on.

## *Thesaurus Builder*

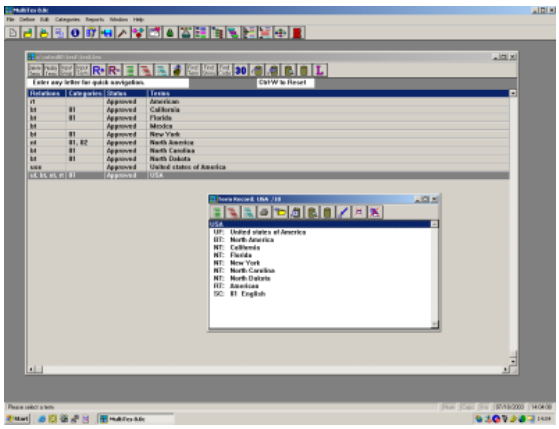
Thesaurus Builder[38] is the simplest of the thesaurus software evaluated. The main application window consists of a simple tree-view of the thesaurus hierarchy. Polyhierarchies are not supported. The lower section of the screen provides a tabbed-view for editing and displaying properties, such as translations, a hierarchy showing the path from the root to this term, non-preferred terms etc. It is possible to export these thesauri as XML, an MS Access database, or as a variety of RTF formatted documents.




One notable feature of this tool is the simple way that multilingual thesauri construction is encouraged, with the entering of the preferred term in other languages on the main properties tab – and an option to translate the current thesaurus into one of the alternative languages.

## *MultiTes*

The MultiTes[39] main window displays the preferred terms of the thesaurus alphabetically, in a tabular form with relations that apply to the term, categories, the terms status etc.

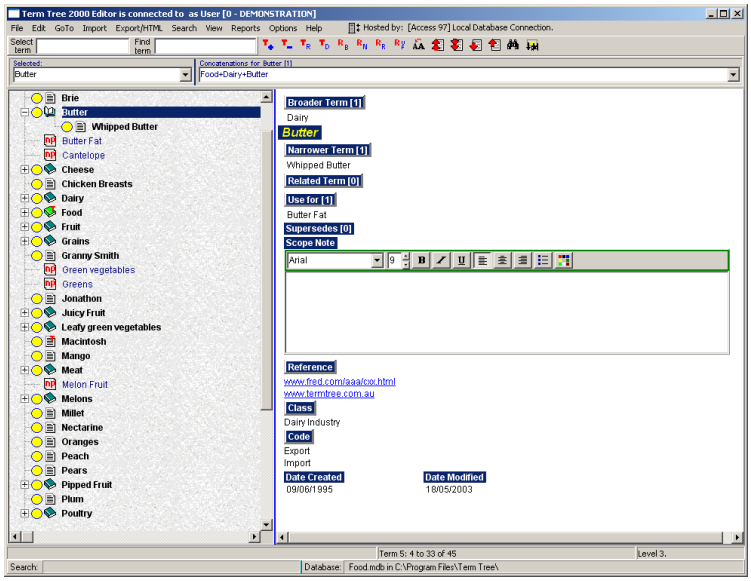


A toolbar with large buttons is used to add new terms to the thesaurus, add relationships to existing terms, and generate various views on the selected term from the list such as a hierarchy to show the path to this term, or a full term report displaying all broader terms, narrower terms, related terms and other relationships, which pops up in a small window within the application.

The toolbar also provides the ability to generate a number of reports (hierarchical, alphabetical, rotated etc.). The tool supports multilingual thesauri. In addition, it is possible to define custom relationships based on the existing built in ones: for example abbreviation can be defined in terms of being an equivalence relation.

## *Term Tree*

The Term Tree[40] display is easy to work with, despite using a rather old style of user interface. On the left hand side of the application a combination of a tree and list view is used to navigate terms. All terms are displayed, and anytime a broader term is specified, the term is replicated to become a child of the broader term. This extends to hierarchies, and polyhierarchies are supported.




The right hand side of the interface allows easy addition of broader terms, narrower terms, use for terms etc. by either adding a new term to the thesaurus, or simply adding structure to an existing set of terms by selecting terms from the existing list to add the relationship between.

The tool supports importing of thesauri either in Term Tree or Microsoft Access format, and exporting as XML, tab delimited, comma separated values, Term Tree, HTML, or metabrowser format. Like the other thesaurus construction software, it supports creating reports in a number of formats.

## *Summary*

The functionality and interface of the four thesauri creation tools is more standardised than that of the ontology/metadata instance tools evaluated. Preferred terms were generally shown hierarchically (with MultiTes being the exception), with TermTree displaying all preferred terms at the root level, with hierarchical relationships replicating terms under their specified broader terms. This replication expanded to terms that had sub-hierarchies. Tabbed views for editing properties are used in TCS8 and Thesaurus builder, while Term Tree displays all properties in a single pane on the right of the tools interface. MultiTes, slightly older than the other tools, used a multiple-window based view for the various views of terms and those terms property editors. As already noted thesauri can be regarded as a simple type of ontology, so these tools are simpler than the more general ontology tools previously described: for example there is no requirement for specifying complex property restrictions or logical axioms that were found in OilEd and Protégé.

# 9 Conclusions

First, SIMILE is aimed at users who are not Semantic Web experts so the UI of any tools provided by the project must reflect this. RDF editors are too low level for such users, and tools more akin to ontology and thesaurus editors must be provided. It may also help the target audience if the user interface adopts terms that are familiar to the user e.g. synonym may be more familiar to users than equivalence, or broader term more familiar than super class.

Second, the majority of ontology tools reviewed use some form of tree view to explore classes and properties. A tree view lets users hide information when they do not need details, yet quickly drill down to specifics when they do. It is often desirable to have multiple search interfaces to the data being manipulated, rather than solely relying on a tree view: several tools already incorporate this into their UI, such as the tree view and query tab provided by Protégé, or the tree view and fisheye view provided by OntoRama. Tabbed views on the other hand encourage the end user to concentrate on distinct aspects of a task such as class description, class relationship and specifying constraints rather than leaving them overwhelmed by the entire set of functionality available to them within the tool.

Third, any schema tool for use within SIMILE must offer some way of importing vocabulary schemas and controlled vocabularies as done in the MEG client. As we wish to encourage schema re-use, this should be made easy to use, flexible and avoid repetitive searches if at all possible.

Fourth, although librarians differentiate between element sets and controlled vocabularies, in languages like OWL and RDF Schema these are both represented using classes. We need to provide a UI that reflects the fact that classes can be used in different ways e.g. as bundles of properties and as property values.

Fifth, SIMILE may need deal with very large controlled vocabularies (for example 250,000+ in the case of the Getty Thesauri[41]). This has a number of implications: it may not be possible to keep the controlled vocabularies on a client machine so instead some kind of web service is required. Also dealing with such large vocabularies will require a more sophisticated way of browsing than simply listing alphabetically.

Sixth, with some schemas the number of metadata elements that will be captured about an item is potentially very large. Therefore just as it is desirable to simplify a user interface aimed at a naive user, it may also be desirable to think about how schemas and the presentation of schemas can be optimized to make best use of cognitive capabilities of metadata authors. For example, if forms require more than 7 fields then it may be desirable to use multiple forms or use some type of grouping[42]. It may even be possible for schema creation tools to assist schema authors with optimizing their schemas for metadata authors.

Finally we note that existing tools do not assist users in the modelling of their schemas, but rather exist solely to enable the user formally capture an existing model. Unlike entity relationship modelling for relational databases, there is currently no formal approach for creating RDF models. This leaves it to the user to address questions such as:

- When it is appropriate to structure properties by grouping related properties together or when to keep them flat in a 'hedgehog' style model?
- When an object should refer to its constituent parts ('hasPart') or when the constituent parts should refer to the object ('isPartOf')?
- When is it necessary to explicitly type the objects at creation (as done in Protégé) or when is better to leave them untyped and infer type dynamically (as done in OilEd)?

Therefore guidelines on a more methodical approach to data modelling using RDF are highly desirable. Once these guidelines have been agreed on, it may be possible to create tools that assist the user to create models that are compatible with the guidelines.

---

[1] SIMILE, Semantic Interoperability of Metadata and Information in unLike Environments
http://web.mit.edu/simile/www/

[2] DSpace - Durable digital depository, MIT Libraries
www.dspace.org

[3] RDF
http://www.w3.org/RDF/

[4] RDF Vocabulary Description Language 1.0: RDF Schema
http://www.w3.org/TR/rdf-schema/

[5] DAML+OIL
http://www.daml.org/

[6] W3C Web Ontology Language (OWL) Working Group
http://www.w3.org/2001/sw/WebOnt/

[7] Serge Abiteboul, "Querying Semi-Structured Data", ICDT, pages 1-18, 1997,
http://citeseer.nj.nec.com/abiteboul97querying.html

[8] Dan Suciu, "An Overview of Semistructured Data", SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory), V29, 1998,
http://citeseer.nj.nec.com/160105.html

[9] Peter Buneman, "Semistructured data", pages 117--121, 1997,
http://citeseer.nj.nec.com/buneman97semistructured.html

[10] Charles Smith and Mark Butler, "Validating CC/PP and UAProf profiles", HP Technical Report 2002-286, http://www.hpl.hp.com/techreports/2002/HPL-2002-268.html

[11] Dean Jones and Ray Paton, "Some problems in the formal representation of hierarchical knowledge",
http://citeseer.nj.nec.com/jones98some.html

[12] Emmanuel Pietriga, IsaViz, W3C
http://www.w3.org/2001/11/IsaViz/

[13] Damian Steer , RDFAuthor,
http://rdfweb.org/people/damian/RDFAuthor/

[14] A. J. Duineveld, R. Stoter, M. R. Weiden, B. Kenepa, V. R. Benjamins, "WonderTools? A comparative study of ontological engineering tools."
http://sern.ucalgary.ca/KSI/KAW/KAW99/papers/Duineveld1/wondertools.pdf

[15] Michael Denny, "Ontology Building: A survey of editing tools."
http://www.xml.com/pub/a/2002/11/06/ontologies.html

[16] "A survey on ontology tools." OntoWeb Consortium, 2002.
http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OntoWeb_Del_1-3.pdf

[17] Natalya F. Noy and Deborah L. McGuinness, "Ontology Development 101: A guide to creating your first ontology."
http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

[18] Sean Bechhofer and Gary Ng, OilEd, University of Manchester
http://OilEd.man.ac.uk/

[19] Protégé 2000, Stanford Medical Informatics
http://protege.stanford.edu

[20] KAON
http://kaon.semanticweb.org/

[21] Touchgraph Library
http://touchgraph.sourceforge.net/

[22] Ontolingua, KSL Stanford University
http://www.ksl.stanford.edu/software/ontolingua/

[23] OntoRama
http://www.ontorama.org/

[24] Andrew Lum, VIUM, School of Information Technologies – University of Sydney
http://www.it.usyd.edu.au/~alum/demos.html

[25] Ontosaurus, Information Sciences Institute University of Southern California
http://www.isi.edu/isd/ontosaurus.html

[26] MEG Registry project, UKOLN
http://www.ukoln.ac.uk/metadata/education/regproj/

[27] UKOLN
http://www.ukoln.ac.uk/

[28] Dublin Core Element set
http://dublincore.org/documents/dces/

[29] Haystack, MIT Computer Science and AI Laboratory
http://haystack.lcs.mit.edu/

[30] Dennis Quan, David R. Karger and David F. Huynh, RDF Authoring Environments for End Users,
http://haystack.lcs.mit.edu/papers/swfat2003.pdf

[31] SHAME, KM Research group - Centre for user oriented IT design
http://kmr.nada.kth.se/shame/

[32] SHAME, Configurable RDF Editor
http://sourceforge.net/project/showfiles.php?group_id=61349&release_id=177335

[33] Simple Instance Creator, Evren Sirin
http://www.mindswap.org/~evren/SIC/

[34] W3C XForms,
http://www.w3.org/MarkUp/Forms/

[35] Micah Dubinko, What are XForms, XML.Com,
http://www.xml.com/pub/a/2001/09/05/xforms.html

[36] Norm Walsh, RDF Twig,
http://rdftwig.sourceforge.net/

[37] TCS-8, WebChoir
http://www.webchoir.com

[38] Thesaurus Builder
http://www.thesaurusbuilder.com/

[39] MultiTes
http://www.multites.com/

[40] Term Tree, This to That Pty Ltd.
http://www.termtree.com.au/

[41] Getty Art and Architecture Thesaurus, Categories for the Description of Works of Art, Thesaurus of Geographic Names, Union List of Artist Names, J. Paul Getty Trust,
http://www.getty.edu/research/tools/

[42] The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information, George A. Miller, originally published in The Psychological Review, 1956, vol. 63, pp. 81-97, http://www.well.com/user/smalin/miller.html