# Open Ontology Forge: An Environment for Text Mining in a Semantic Web World

**Nigel Collier** and **Koichi Takeuchi** and **Ai Kawazoe**
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku
Tokyo 101-8430 Japan
{collier,koichi,zoeai}@nii.ac.jp

**Abstract**

This paper describes the main features of Open Ontology Forge (OOF), a client-side tool for use in cooperative ontology engineering and Web page annotation. Using OOF allows experts in a domain of knowledge to engineer ontologies in RDF Schema and then create annotated contents as instances of the ontology classes which form a link between the text and the ontology. OOF offers many features to make the life of the annotator easier within a user-friendly drag-and-drop environment. The server-side of Ontology Forge provides a hosting environment for domain ontologies and annotations as well as an information extraction system for reducing the effort of annotations based on supervised machine learning from examples. We discuss the motivations for the design and highlight key features of the software.
Keywords: Ontology, RDF, Annotation, XPointer, Information Extraction, Collaboration

## Introduction

In this paper we describe Open Ontology Forge, a client-side tool for use in cooperative ontology engineering and Web page annotation. Using OOF allows experts in a domain of knowledge to engineer ontologies in RDF Schema and then create annotated contents as instances of the ontology classes which form a link between the text and the ontology. Ontologies and annotations are available in machine understandable form for sharing on the Semantic Web (Berners-Lee, Fischetti, & Dertouzos 1999), i.e. the next generation Web.

The cornerstone of the Semantic Web is machine understandable data that is available in a commonly understood format. However creating the meta-data schemes in the form of ontologies and annotating Web pages risks becoming the occupation of a few highly trained professionals unless we can provide support tools, i.e. the "off the shelf software for writing Semantics" (Berners-Lee, Hendler, & Lassila 2001), and environments that encourage ordinary users to participate and share their knowledge.

Ontology Forge is a client-server system that builds on ideas from several previous projects and tools: ontology editors such as Protege-2000 (Fridman Noy, Fergerson, & Musen 2000) (Noy *et al.* 2001) and Ont-O-Mat (Handschuh, Staab, & Maedche 2001), ontology servers such as Ontolingua (Farquhar, Fikes, & Rice 1997) and annotation servers (clients) such as Annotea (Amaya) (Kahan *et al.* 2000). What characterizes Ontology Forge are two points. Firstly, we believe that since ontologies are primarily community artifacts that they need to be agreed and constructed in a way that is accessible to the community. This led naturally to a hosted development environment for ontology projects. Secondly, to reduce the high cost of annotation we include sophisticated information extraction technology that can be rapidly customized to new domains to enable high quality semi-automatic annotation based on the existing database of expert-made annotations.

In Ontology Forge we adopt a *domain* view (Kashyap & Sheth 1997) of knowledge representation and acquisition. By *domain* we mean a group of people who share a common view on the structure of knowledge in a particular area. Their purpose in constructing an ontology is primarily to formalize the relationships, concepts and properties for the terminology used in that domain and to share this with others. We broadly define terminology according to (de Besse, Nkwenti-Azek, & Sager 1997) as "A lexical unit representing one or more words that represents a concept in a domain". The domain view has two major advantages: the first is that it modularizes knowledge by consent, and secondly it decentralizes knowledge from any particular knowledge scheme. Groups of users are free to get together and form a domain in any area of knowledge that they consider meaningful to themselves.

The key point is that our research brings together work on high level knowledge representation and reasoning in the Semantic Web and Artificial Intelligence communities with surface text-level realization of this knowledge from the information extraction community. Since free texts provide by far the highest proportion of knowledge available on the Web, our vision is that annotated texts will provide semi-structured data that describes proto-typical facts that are of broad interest to users within the domain such as cell signal transductions (Functional Genomics), missile launches (Defense), conference announcements (Academia) or company takeovers (Stock Market). This should enable much greater precision in Web searching, document summarization and question answering for a new generation of 'smart' tools.

Having placed our research in context we now provide an overview of the system starting with its design in Section and knowledge model in Section . In Section we illustrate

the usage of OOF with a walk through of the construction of small ontologies in the Car Advert and Functional Genomics domains and discuss how a text can be annotated in Section . We conclude in Section  with a comparison of our system against similar projects and plans for future work.

## System Overview

The Ontology Forge design encourages collaborative participation on an ontology project by enabling all communication to take place over the Internet using a standard Web-based interface. After an initial domain group is formed we encourage group members to divide labour according to levels of expertise. This is shown in the following list of basic steps required to create and maintain a domain ontology.

**Domain Setup** a representative for a domain (the *Domain Manager*) will apply to set up an ontology project that is hosted on the Ontology Forge server.

**Community Setup** the *Domain Manager* will establish the domain group according to levels of interest and competence, i.e. *Experts* and *Users*.

**Ontology Engineering** Ontologies are constructed in private by *Domain Experts* through discussion and a series of versions.

**Ontology Publication** When the *Domain Manager* decides that a version of the ontology is ready to be released for public access he/she copies it into a public area on the server.

**Annotation** When a public ontology is available *Domain Users* can take this and annotate Web-based documents according to the ontology. Annotations can be uploaded to the server and stored in the private server area for sharing and discussion with other domain members.

**Annotation Publication** The *Domain Manager* copies private annotations into the public server area.

**Training** When enough annotations are available the *Domain Manager* can ask the Information Extraction system called PIA-Core (Collier & Takeuchi 2002) to learn how to annotate unseen domain texts.

**Improvement Cycle** Annotation and training then proceed in a cycle of improvement so that *Domain Users* correct output from the Information Extraction system and in turn submit these new documents to become part of the pool of documents used in training. Accuracy gradually improves as the number of annotations increases. The amount of work for annotators is gradually reduced.

In reality we expect that the *Ontology Engineering* step will be a complex cyclic stage of discovery with refinement taking place in the *Annotation* and *Improvement Cycle* stages after intuitions are gained through examining terminology and terminological relations in domain texts. We view texts as not only a source of instance data but also as a source of inspiration for discovering the formal classes and relations in the ontology.

The overall system architecture for Ontology Forge is shown in Figure 1. This shows the basic components on the server side including a Web server, a database management system and an application server which will primarily be responsible for information extraction. On the client side we have the Open OntologyForge (OOF) ontology editor and annotation tool. From now we will concentrate on describing the features of OOF.

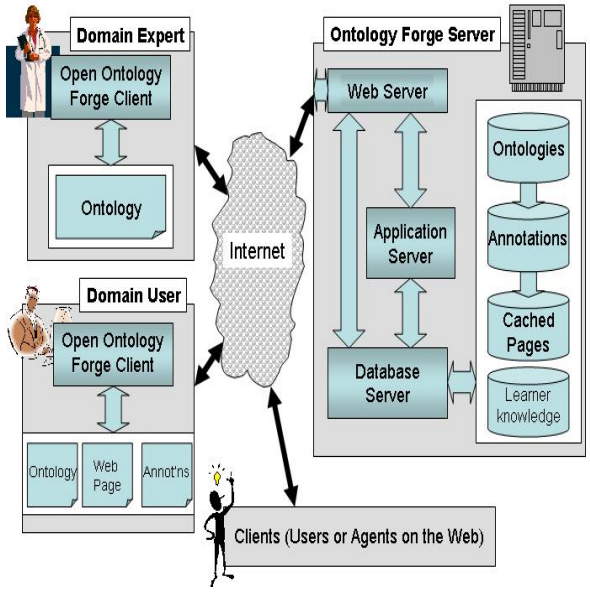


Figure 1: An overview of the Ontology Forge architecture

## Knowledge Model

The advent of the Semantic Web has ushered in a set of standards for the annotation of semantic content on the Web such as XML for document structure, RDF for defining objects and their relations, and RDF Schema for defining basic ontological modelling primitives on top of RDF. In this section we will not be describing RDF/RDF Schema in any detail as both languages are already well documented in the literature, e.g. (Brickley & Guha 2000)(Staab *et al.* 2000). Instead we will focus on the specific characteristics of our annotation scheme.

The root class in an Ontology Forge ontology is the annotation, defined in RDF Schema as a name space and held on the server at a URI. The user does not need to be explicitly aware of this and all classes that are defined in OOF will inherit these properties from the parent so that when instances are declared as annotations in a base Web document, the instance will have many of the property values entered automatically by OOF such as linkage information. Basically the user is free to create any classes that help define knowledge in their domain according to the limits of RDF Schema.

We briefly now describe the root annotation class:

**context** This relates an Annotation to the resource to which the Annotation applies and takes on an XPointer (De Rose, Maler, & Daniel, R. (eds) 2000) value.

**language** The name of the language of this Annotation.

**ontology_id** Relates an Annotation to the ontology and class to which the annotation applies.

**conventional_form** The conventional form of the annotation (if applicable) as described in the PIA Annotation Guidelines.

**identity_id** Used for creating coreference chains between annotations where the annotations have identity of reference.

**constituents** This is used to capture constituency relations between annotations if required.

**orphan** This property takes only Boolean values corresponding to 'yes' and 'no'. After the annotation is created, if it is later detected that the annotation can no longer be linked to its correct position in doc_id, then this value will be set to 'yes' indicating that the linkage (in context) needs correcting. The mechanism for detecting broken links is not yet defined by our system but we feel that this is a necessary property for maintaining annotations.

**author** The name of the person, software or organization most responsible for creating the Annotation. It is defined by the Dublin Core (dublincore:1999 1999) 'creator' element.

**created** The date and time on which the Annotation was created. It is defined by the Dublin Core 'date' element.

**modified** The date and time on which the Annotation was modified. It is defined by the Dublin Core 'date' element.

**sure** This property takes only Boolean values corresponding to 'yes I am sure' and 'no I am not sure' about the assignment of this annotation. Used primarily in post-annotation processing.

**comment** A comment that the annotator wishes to add to this annotation, possibly used to explain an unsure annotation. It is defined by the Dublin Core 'description' element.

Due to the lack of defined data types in RDF we make use of several data types in the Dublin Core name space for defining annotation property values. In OOF we also allow users to use a rich set of data type values using both Dublin Core and also XML Schema name space for integers, floats etc. Users can also defined their own enumerated types using OOF.

A partial view of an annotation (shown in shaded ovals) in the Car Advert domain can be seen in Figure 2 showing linkage into a Web document (shown in the boxed text at the bottom) and part of the ontology hierarchy (shown in clear ovals). The annotation shows an instance of a used car called "Audi Coupe 2.3E" and a property called *year* with value 1991. The property value itself appears in the text but there is no linkage. If linkage were needed we would need to create a class called *year* and declare a property whose range was the *year* class.

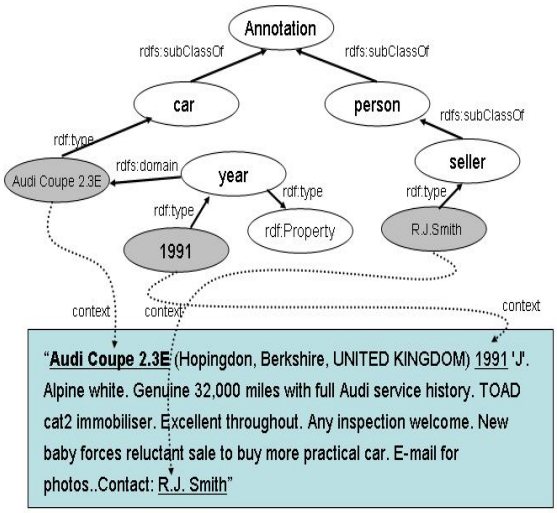


Figure 2: Overview of an annotation.

## Ontology Creation

Since text editing of ontologies and instances introduces too many mistakes and frustrates the user it is generally accepted that some level of tool support is required for creating ontologies and instances. In the case of Ontology Forge we have produced the Open OntologyForge client software, colloquially known as OOF.

OOF basically accepts a subset of RDFS ontologies. The major areas of restriction are:

**multiple inheritance of classes** RDFS allows classes to be subclasses of more than one class, OOF does not allow this;

**multiple inheritance of instances** OOF does also not allow instances to belong to more than one class;

OOF has much in common with other ontology editors such as Protege-2000 and Ont-O-Mat. The basic purpose of the editor is to allow users to declare concept classes, properties, property value types and assign class-relations and class-properties. Properties are restricted to having a single value range which must be a pre-defined type or a class in the ontology. In this way properties can specify relations between instances.

One point of difference between OOF and the other editors is due to our current focus on RDFS which means that we do not have a way to specify cardinality on properties or inheritance characteristics of properties as there is no direct provision within RDFS at the metadata level for this. By default we allow all properties of a class to be inherited by its subclasses.

An example of ontology formation in progress using OOF can be see in Figure 3 where the expert has declared a top level ontology for the Car Advert domain. We can see

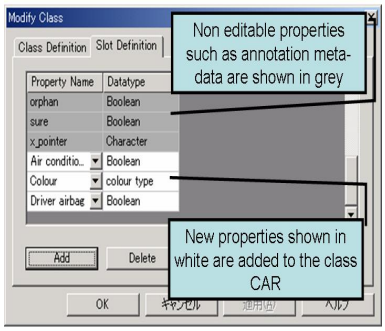


Figure 4: Example of assignment of user defined properties to classes. Here we see the property *Driver airbag* being assigned to the *CAR* class with Boolean values.

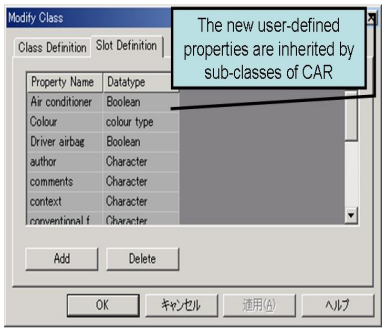


Figure 5: Example of property inheritance from the *CAR* parent class to one of its sub-classes. Note that the inherited property is not editable since cardinality is not implemented within RDFS.

that the expert has defined a number of properties of which *colour* takes values from a new user-defined enumerated type called *colour type*. In Figure 4 we see the expert adding these properties to the *CAR* class. Subclasses of *CAR* such as *Alfa Romeo* automatically inherit these properties as shown in Figure 5.

OOF includes many checking mechanisms that help the user to create and maintain consistent ontologies and annotations such as preventing users from deleting data types, properties and classes if there exist instances which use those definitions.

## Annotating Instances

Although users are free to annotate in any way that they like, one main purpose of Ontology Forge is to reduce the high cost of making annotations using information extraction based on supervised machine learning (Takeuchi & Collier 2002). For this to be effective we need to discipline the user in the way that they declare annotations so that there is a high level of consistency between different annotators in the same domain and for this purpose we are working on a set of guidelines.

There are three basic levels of annotation which conform to the three levels of knowledge in the information extraction task defined by MUC (MUC6:TASK:95 1995). These are *Named Entity*, *Coreference* and *Event Extraction* (incorporating the template element and scenario template task). The first two are supported in the current version of Open Ontology Forge with Coreference annotations considered as a kind of second-class Named Entity which requires the annotator to declare the antecedent Named Entity before the Coreference expression that points to it.

Coreference annotation tries to capture the identity relations between surface expressions in the text. We follow a *symmetric* view of coreference whereby each expression contributes equally to the definition of the extension of the instance. This has two major advantages: no expression needs to be nominated as the canonical form, and it is natural to extend coreference relations to expressions in different documents.

A snapshot of an annotation session is shown in Figure 6 for a journal article in the Functional Genomics domain. OOF has many useful features for the annotator, some of them are major ones while others are less obvious and only occurred to us after we had got feedback from users. One example is the need for good memory management and speed of annotation - our prototype version built in Java had many performance problems when the number of annotations exceeded a thousand, consequently the current version has been rebuilt entirely in C++.

Some features that are designed to make the life of the annotator easier are:

- a convenient drag-and-drop annotation environment which allows users to highlight a part of the text that they wish to capture and drag it to the class in the ontology where it becomes an instance;
- the value of the XPointer for **context** is automatically inserted into the property value as are values for other tracking properties such as **created** and **author**;
- a full Web-browser view of the Web page is maintained in the OOF main window including images. Annotations are indicated using highlighted boxes whose colouring scheme is customizable by the user - useful if the page colour is the same as the default box colour.
- if annotations stretch over more than one line the indicating box naturally adapts itself to the Web page;
- annotations are search and replaceable - this was a major feature that users requested from our first version;
- when a user captures an instance OOF automatically searches for similar instances and offers the user the chance to annotate these too, saving the user a lot of time in duplicated work. Property values are automatically adjusted for each new instance;
- removal of leading and trailing white space - frequently the user highlights a text area that includes spaces which should be removed;

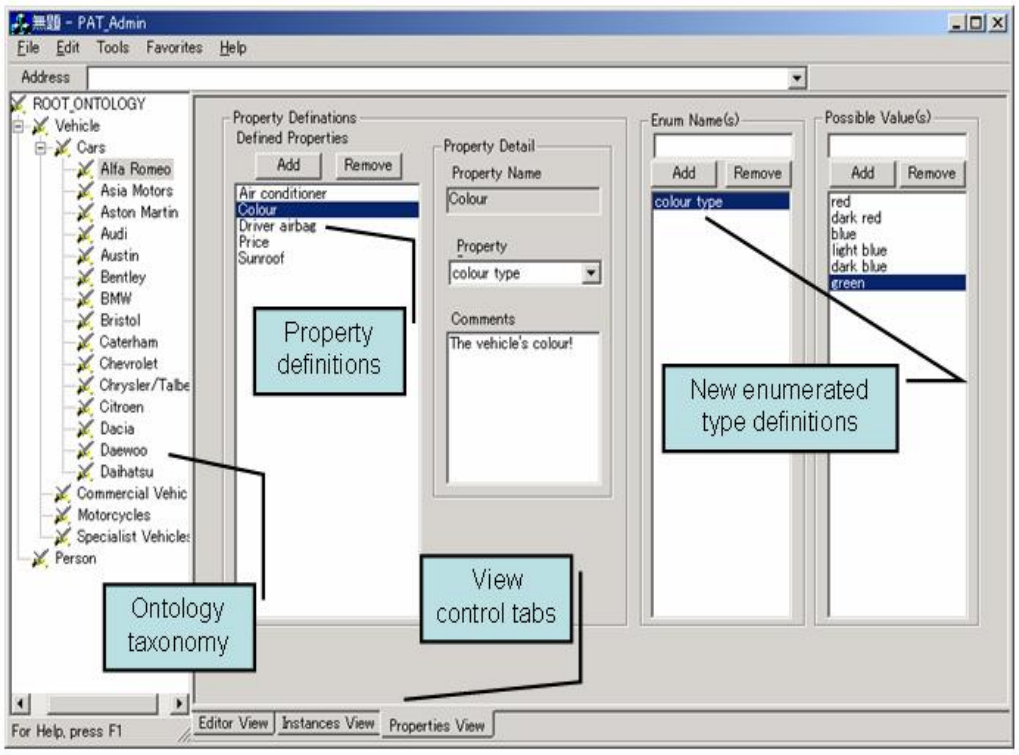


Figure 3: A snapshot from the formation of a simple Car Advert ontology. The tree on the left shows the class hierarchy. Along the bottom are various 'tabs' which allow the user to focus on a different aspect of the knowledge base such as the Property View or the Instances View. In the Property View experts can define new properties. The expert can choose predefined property types such as 'character' or 'Boolean' and is free to define new types such as the *colour type* being created here. Here we see that *colour type* is assigned to the *Colour* property. Once a property is defined the expert can create a new class such as *Alfa Romeo* and assign the new property to it. At annotation time the user will instantiate the expert-defined property values.

- automatic detection of overlapping annotations when a user deletes or annotates;
- automatic detection of coreference dependencies when a user deletes an annotation;

Our long term goal is to improve the quality of information extraction in order to reduce the cost of human annotation. For this to be effective there are two important points: the first is a certain number of annotations, dependent on the linguistic characteristics of the instances and the number of classes, and also the *consistency* between annotations. We hope that by providing a convenient environment like Ontology Forge that users will be encourage to cooperate so that this number will be achieved. More important perhaps is consistency which is the long term key to obtaining high accuracy, and since annotation is a community task we must strive to obtain a high level of *inter-annotator agreement*, which we can measure by asking all the annotators to annotate the same text according to the same ontology. To achieve this goal an important part of our research is the production of general annotation guidelines (Collier *et al.* 2002) which try to cover the majority of general uncertain cases such as whether or not an article should be included at the start of an annotation (*the* {*RAS protein*} vs. {*the Big Apple*}), or whether embedded annotations should be used ({*Leicester Square*}$_{PLACE}$ vs {{*Leicester*}$_{PLACE}$ *Square*}$_{PLACE}$).

## Related Work

There were several starting points for our work. Perhaps the most influential was the Annotea project (Kahan *et al.* 2000) which provided an annotation hosting service based on RDF Schema. Annotations in Annotea can be considered as a kind of electronic 'Post-it' for Web pages, i.e. a remark that relates to a URI. Annotea has several features which our design has used including:

- use of XLink/XPointer linkage from annotations to the place in the document where a text occurs;
- use of generic RDF schema as the knowledge model enhanced with types from Dublin Core;
- physical separation of ontology, annotation and base document data as well as the people who create them;
- a consideration of annotations as first class objects;

However, there are several major differences between our work and Annotea including various levels of support for cooperative development of annotations and ontologies, domain groups and explicit roles and access rights for members, and a focus on aiding consistency of annotation for the purpose of information extraction.

Also influential in our development has been Protege-2000 and other ontology editors such as Ont-O-Mat which provide many of the same features as OOF including ontology editing and instance capturing as well as saving in a variety of languages such as XML and RDF. In fact the first version of OOF was built as a Java plug-in to Protege-2000 and allowed users to capture instances in texts and HTML documents. In user trials however this earlier version suffered from a number of drawbacks. The most serious we found was that memory management in Java was taken out of the control of the programmer and consequently become slow and expensive as the number of instances captured exceeded several hundred. We therefore decided to continue development using a Visual C++ client which could take advantage of the Windows user interface and tools such as Explorer and Microsoft Agent for the help system. The server side remains fairly platform independent requiring a Web server and an SQL-based database management system. In our case we settled on Apache and Oracle on RedHat Linux 7.3.

While Protege-2000 provides a rich environment for creating ontologies it provides limited support for large-scale instance capturing and collaborative development. Ont-O-Mat (Handschuh, Staab, & Maedche 2001) and OntoEdit from the University of Karlsruhe provide similar functionality to OOF but does not link instances back to the text or have the built-in design focus on collaborative development and information extraction.

## Conclusion and Future Plans

Open Ontology Forge will be released in late-December 2003. There will then follow a period for testing which will be finished in March 2003. The software for the project will be freely available to download and groups are encouraged to set up their own Ontology Forge servers using OOF as the client. We will maintain the latest full featured version of OOF and a working server at NII for hosting a limited number of academic ontologies. The project Web site will be available for general information on the project from January 2003 at http://ontology-forge.ex.nii.ac.jp.

OOF contains many features that we believe will be useful to users but there are several services that we want to implement from now. These include customizable output file formats such as SQL, XML, DAML+OIL, OWL, and allowing users to annotate non-text objects such as images. One limitation of OOF is the currently it does not dynamically download and link or verify RDF schemas from remote sites outside Ontology Forge, although this is on our list of work to do. So far we have looked at providing supporting guidelines for the annotation of *named entity* and *coreferences* as instances but we have yet to develop guidelines or a policy on what constitutes an *event*. This is a priority area for our work in the coming months.

## Acknowledgements

We wish to express our gratitude to the team at Axiohelix for implementing our ideas and to Ken Satoh (NII) for helping to support development on the first version of the client software. This work was supported from various sources including the Japanese Ministry of Education and Science (grant no. 14701020) and an NII Leadership Fund grant.

## References

Berners-Lee, T.; Fischetti, M.; and Dertouzos, M. 1999. *Weaving the Web: The Original Design and Ultimate Des-*

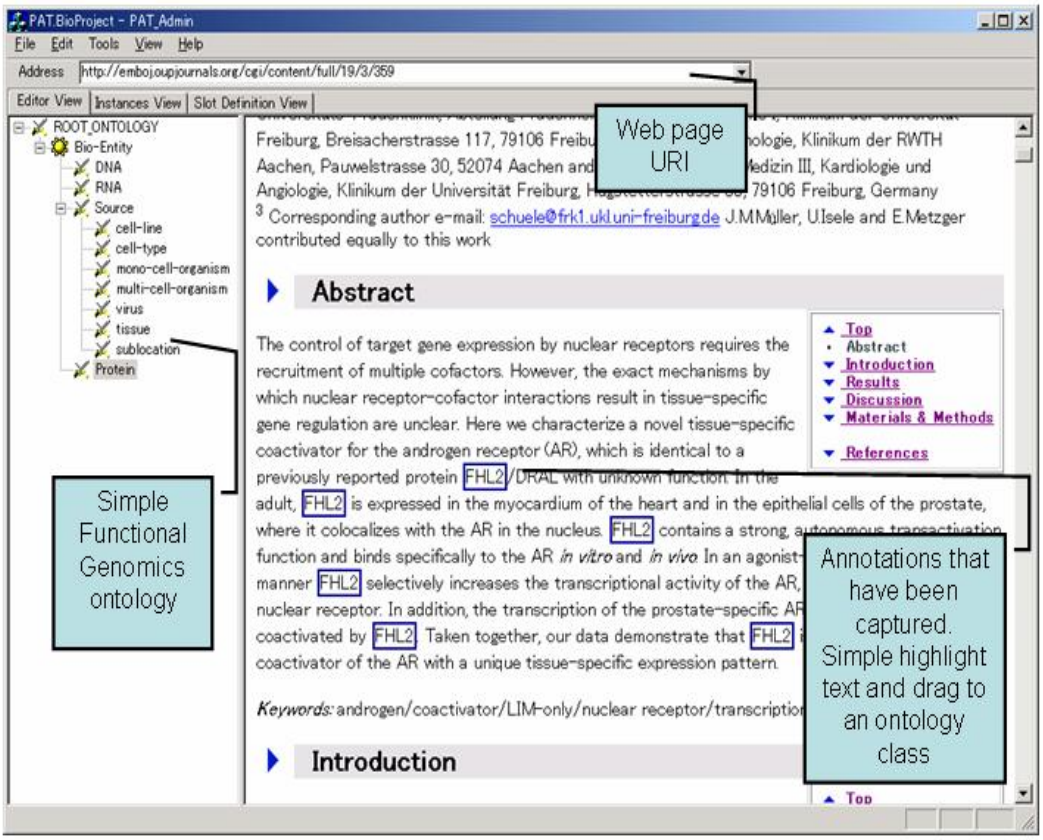


Figure 6: A snapshot of an online journal article being annotated using the classes defined in a simple Functional Genomics ontology. Annotations appear in blue boxes.

*tiny of the World Wide Web*. San Francisco: Harper. ISBN: 0062515861.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The Semantic Web. *Scientific America* 29–37.

Brickley, D., and Guha, R. V. 2000. Resource Description Framework (RDF) schema specification 1.0, W3C candidate recommendation. http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

Collier, N., and Takeuchi, K. 2002. PIA-Core: Semantic annotation through example-based learning. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'2002), Las Palmas, Spain*.

Collier, N.; Takeuchi, K.; Nobata, C.; Fukumoto, J.; and Ogata, N. 2002. Progress on multi-lingual named entity annotation guidelines using RDF(S). In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'2002), Las Palmas, Spain*, 2074–2081.

de Besse, B.; Nkwenti-Azek, B.; and Sager, J. C. 1997. Glossary of terms used in terminology. *Terminology* 4(1):117–156.

De Rose, S.; Maler, E.; and Daniel, R. (eds). 2000. XML pointer language (XPointer) version 1.0, w3c candidate recommendation, 11th september 2001. http://www.w3.org/TR/xptr.

1999. Dublin core metadata element set, version 1.1: Reference description. Technical Report, Dublin Core Metadata Initiative, http://purl.org/DC/documents/rec-dces-19990702.htm.

Farquhar, A.; Fikes, R.; and Rice, J. 1997. The Ontolingua server: a tool for collaborative ontology construction. *The International Journal of Human-Computer Studies* 46:707–727.

Fridman Noy, N.; Fergerson, R. W.; and Musen, M. A. 2000. The knowledge model of Protégé-2000: combining interoperability and flexibility. In *Proceedings of the 2nd International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France*, 1–20.

Handschuh, S.; Staab, S.; and Maedche, A. 2001. CREAM - creating relational metadata with a component-based, ontology-driven annotation framework. In *First International Conference on Knowledge Capture (K-CAP 2001), Victoria, B.C., Canada*.

Kahan, J.; Koivunen, M.; Prud'Hommeaux, E.; and Swick, R. 2000. Annotea: An open RDF infrastructure for shared web annotations. In *the Tenth International World Wide Web Conference (WWW10)*, 623–630.

Kashyap, V., and Sheth, A. 1997. Semantic heterogeneity in global information systems: The role of metadata, context and ontologies. In Papzoglou, M., and Schlageter, G., eds., *Cooperative Information Systems: Current Trends and Directions*. Academic Press.

DARPA. 1995. *Information Extraction Task Definition*, Columbia, MD, USA: Morgan Kaufmann.

Noy, N. F.; Sintek, M.; Decker, S.; Crubezy, M.; Fergerson, R. W.; and Musen, M. A. 2001. Creating semantic web contents with Protégé-2000. *IEEE Intelligent Systems* 16(2):60–71.

Staab, S.; Erdmann, M.; Maedche, A.; and Decker, S. 2000. An extensible approach for modeling ontologies in RDF(S). In *Proceedings of Workshop on the Semantic Web, held at the Fourth European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2000), Lisbon, Portugal*.

Takeuchi, K., and Collier, N. 2002. Use of support vector machines in extended named entity recognition. In *Proceedings of the 6th Conference on Natural Language Learning 2002 (CoNLL-2002), Roth, D. and van den Bosch, A. (eds)*, 119–125.