Authoring and Annotation of Web Pages in CREAM

Siegfried Handschuh, Steffen Staab
Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
{sha, sst}@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de/WBS

Abstract

Richly interlinked, machine-understandable data constitute the basis for the Semantic Web. We provide a framework, CREAM, that allows for creation of metadata. While the annotation mode of CREAM allows to create metadata for existing web pages, the authoring mode lets authors create metadata — almost for free — while putting together the content of a page.

As a particularity of our framework, CREAM allows to create *relational metadata*, i.e. metadata that instantiate interrelated definitions of classes in a domain ontology rather than a comparatively rigid template-like schema as Dublin Core. We discuss some of the requirements one has to meet when developing such an ontology-based framework, e.g. the integration of a metadata crawler, inference services, document management and a meta-ontology, and describe its implementation, *viz.* Ont-O-Mat a component-based, ontology-driven Web page authoring and annotation tool.

Keywords: Annotation, Metadata, Semantic Web, RDF
Approximate word count: 6900

1 Introduction

The Semantic Web builds on metadata describing the contents of Web pages. In particular, the Semantic Web requires relational metadata, i.e. metadata that describe how resource descriptions instantiate class definitions and how they are semantically interlinked by properties. In a previous paper [16], we have described a first version of CREAM (CREAting Metadata) as an annotation framework suited to allow for the easy and comfortable creation of such relational metadata and we have provided an annotation tool that implements this framework, called Ont-O-Mat.[1]

However, Ont-O-Mat V0.1 — like other annotation environments — has a major drawback. In order to provide metadata about the contents of a Web page, the author must *first* create the content and

[1] The version then was V0.1. A more updated version V0.5 already implementing some of the extensions described here is downloadable at `http://annotation.semanticweb.org`. The next version V1.0 is currently under development and will correspond in its functionality to what is described in this paper.

second annotate the content in an additional, a-posteriori, annotation step. Thus, rather than saving work it puts an additional work load on the creator.

As a way out of this problem, we propose that an author needs the possibility to easily combine authoring of a Web page *and* the creation of relational metadata describing its content. Scrutinizing the difficulties one encounters for such a web page authoring tool, we found that they are very similar to the ones for annotating web pages with relational metadata. In fact, we found it preferable to hide the border between authoring and annotation as far as possible. Therefore, rather than building a completely new tool, we have extended the framework CREAM and the tool Ont-O-Mat in order to reflect the needs for metadata creation that web page authors and a-posteriori annotators have. This required, in particular, the introduction of

- a **Meta Ontology** that describes how the annotation and authoring modes of Ont-O-Mat interfere with classes and properties of the ontology proper, and
- new **Modes of Interaction** that allow for switching easily back-and-forth between authoring and annotation.

In the following we first sketch two usage scenarios (Section 2), describing some of the requirements we must meet for the creation of metadata. Then, we explain our terminology in more detail and give an example of the metadata we want to create. Then we derive the design of CREAM from the requirements elaborated before. In Section 5, we specify how the meta ontology may modularize the ontology description from the way the ontology is used in CREAM. In Section 6, we explain the major modes of interaction with Ont-O-Mat, our implementation of CREAM. Before we conclude, we give a survey of related work in the categories knowledge markup on the Web, knowledge acquisition, annotation environments and authoring environments.

2 Scenarios and Requirements for CREAM

The origin of our work facing this challenge dates back to the start of the seminal KA^2 intiative [1], *i.e.* the initiative for providing semantic markup on HTML pages for the knowledge acquisition community and its presentation in a Web portal [31]. The KA2 portal provides a view onto knowledge of the knowledge acquisition community. Besides of semantic retrieval as provided by the original KA2 initiative, it allows comprehensive means for navigating and querying the knowledge base and also includes guidelines for building such a knowledge portal. The potential users provide knowledge,

e.g. by annotating their web pages in a decentralized manner. The knowledge is collected at the portal by crawling and presented in a variety of ways.

Similarly, our second scenario here, we have used semantic markup in order to provide knowledge about the TIME (telecommunication, IT, multimedia, e-business) markets in the TIME2Research portal [32]. Thus, we have created a knowledge portal for business analysts. The principal idea is that business analyst review news tickers, business plans and business reports. A considerable part of their work requires the comparison and aggregation of similar or related data, which may be done by semantic queries like"Which companies provide B2B solutions?", when the knowledge is semantically available. At the Time2Research portal they handle different types of documents, annotate them and, thus, feed back into the portal to which they may ask questions.

At the start of these two case studies, we had the intuition that these cases may be easily supported by a simple annotation tool. During the course of the projects, however, we found that we had to face many principal problems. Also, we found that the two scenarios would never succeed in the end if we did not give people the possibility to easily create documents and their metadata in one step.

The principal requirements for *a-posteriori annotation* as well as for the *integration of web page authoring with metadata creation* can be outlined as follows:

- **Consistency**: Semantic structures should adhere to a given ontology in order to allow for better sharing of knowledge. For example, it should be avoided that people confuse complex instances with attribute types.

- **Proper Reference**: Identifiers of instances, *e.g.* of persons, institutes or companies, should be unique. For instance, in KA^2 metadata there existed three different identifiers of our colleague Dieter Fensel. Thus, knowledge about him could not be grasped with a straightforward query.[2]

- **Avoid Redundancy**: Decentralized knowledge provisioning should be possible. However, when annotators collaborate, it should be possible for them to identify (parts of) sources that have already been annotated and to reuse previously captured knowledge in order to avoid laborious redundant annotations.

- **Relational Metadata**: Like HTML information, which is spread on the Web, but related by HTML links, knowledge markup may be distributed, but it should be semantically related.

[2]The reader may see similar effects in bibliography databases. E.g., query for James (Jim) Hendler at the — otherwise excellent — DBLP: `http://www.informatik.uni-trier.de/~ley/db/`.

Current annotation tools tend to generate template-like metadata, which is hardly connected, if at all. For example, annotation environments often support Dublin Core [8, 9, 23], providing means to state, e.g., the name of authors, but not their IDs[3].

- **Maintenance**: Knowledge markup needs to be maintained. An annotation tool should support the maintenance task.

- **Ease of use**: It is obvious that an annotation environment should be easy to use in order to be really useful. However, this objective is not easily achieved, because metadata creation involves intricate navigation of semantic structures.

- **Efficiency**: The effort for the production of metadata is a large restraining threshold. The more efficiently a tool supports metadata creation, the more metadata users tend to produce . This requirement is related to the ease of use. It also depends on the automation of the metadata creation process, e.g. on the preprocessing of the document.

The new version of CREAM presented here targets at a comprehensive solution for metadata creation during web page authoring and a-posteriori annotation. The objective is pursued by combining advanced mechanisms for inferencing, fact crawling, document management, meta ontology definitions, metadata re-recognition, content generation, and (in the future) information extraction — as explained in the following sections.

3 Relational Metadata

We elaborate the terminology we use in our framework, because many of the terms that are used with regard to metadata creation tools carry several, ambiguous connotations that imply conceptually important decisions for the design rationale of CREAM:

- **Ontology**: An ontology is a formal, explicit specification of a shared conceptualization of a domain of interest [15]. In our case it is constituted by statements expressing definitions of DAML+OIL classes and properties [13].

- **Annotations**: An annotation in our context is a set of instantiations attached to an HTML document. We distinguish *(i)* instantiations of DAML+OIL classes, *(ii)* instantiated properties

[3] In the web context one typically uses the term 'URI' (uniform resource identifier) to speak of 'unique identifier'.

from one class instance to a datatype instance — henceforth called attribute instance (of the class instance), and *(iii)* instantiated properties from one class instance to another class instance — henceforth called relationship instance.

Class instances have unique URIs, e.g. like 'urn:rdf:936694d5ca907974ea16565de20c997a-0'.[4] They frequently come with attribute instances, such as a human-readable label like 'Steffen'.

- **Metadata**: Metadata are data about data. In our context the annotations are metadata about the HTML documents.

- **Relational Metadata**: We use the term relational metadata to denote the annotations that contain relationship instances.

  Often, the term "annotation" is used to mean something like "private or shared note", "comment" or "Dublin Core metadata". This alternative meaning of annotation may be emulated in our approach by modelling these notes with attribute instances. For instance, a comment note "I like this paper" would be related to the URL of the paper via an attribute instance 'hasComment'.

  In contrast, relational metadata also contain statements like 'Siegfried cooperates with Steffen', *i.e.* relational metadata contain relationships between class instances rather than only textual notes.

Figure 1 illustrates our use of the terms "ontology", "annotation" and "relational metadata". It depicts some part of the SWRC[5] (semantic web research community) ontology. Furthermore it shows two homepages, viz. pages about Siegfried and Steffen (`http://www.aifb.uni-karlsruhe.de/WBS/sha` and `http://www.aifb.uni-karlsruhe.de/WBS/sst`, respectively) with annotations given in an XML serialization of RDF facts. For the two persons there are instances denoted by corresponding URIs (`urn:rdf:947794d5ca907974ea16565de21c998a-0` and `urn:rdf:936694d5ca907974ea16565de20c997a-0`). The `swrc:name` of `urn:rdf:947794d5ca907974ea16565de21c998a-0` is "Siegfried Handschuh". In addition, there is a relationship instance between the two persons, *viz.* they cooperate. This cooperation information 'spans' the two pages.

The objective of CREAM is to allow for the easy generation of such a target representation irrespective of whether the major mode of interaction is a-posteriori annotation or web page authoring.

[4] In the Ont-O-Mat implementation we create the URIs with the createUniqueResource method of the RDF-API

[5] `http://ontobroker.semanticweb.org/ontos/swrc.html`

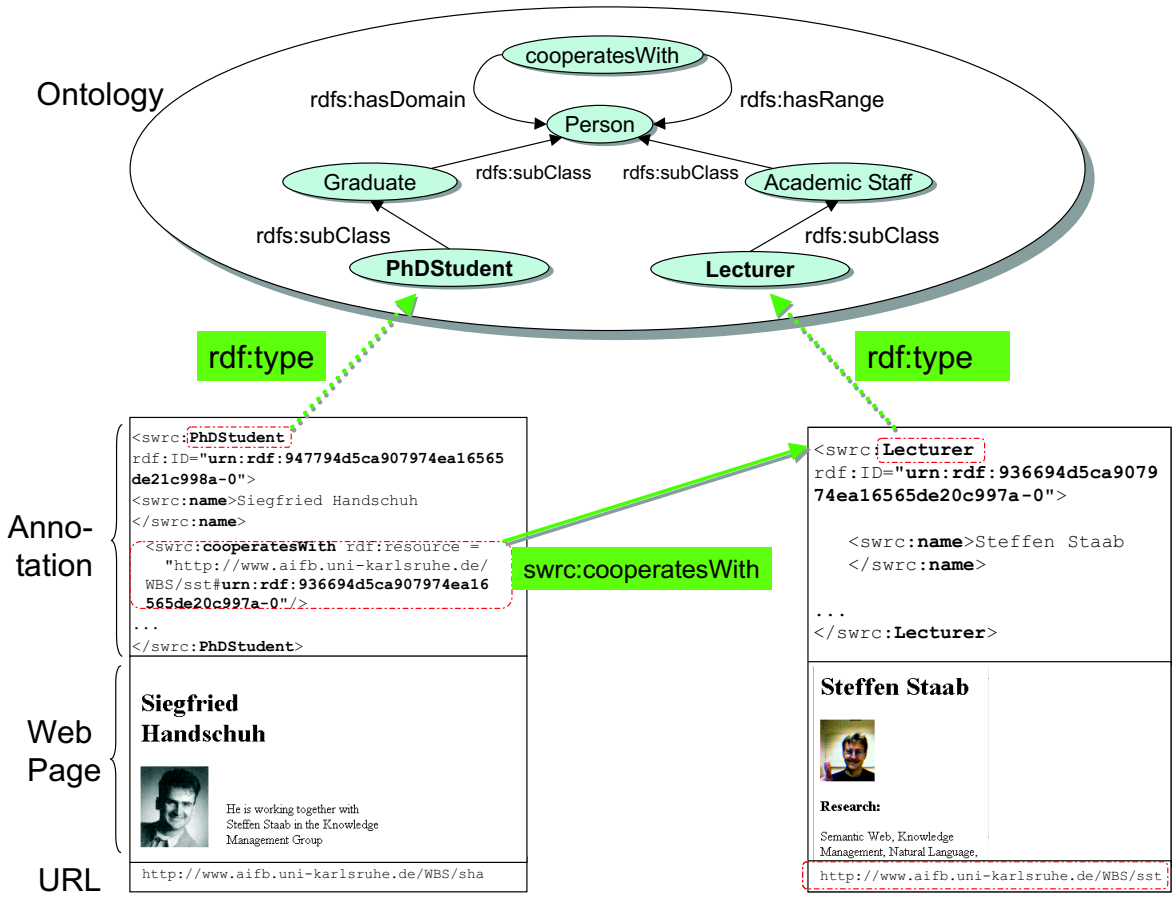

Figure 1: Annotation example

4 Design of CREAM

4.1 CREAM Modules

The requirements and considerations from Sections 1 to 3 feed into the design rationale of CREAM. The design rationale links the requirements with the CREAM modules. This results in a N:M mapping (neither functional nor injective). An overview of the matrix is given in Table 1.

- **Document Editor/Viewer**: The document editor/viewer visualizes the document contents. The metadata creator may easily provide new metadata by highlighting text. The document viewer should support various formats (HTML, PDF, XML, etc.).

- **Content Generation**: The editor also allows the conventional authoring of documents. In addition, instances already available may be dragged from a visualization of the content of the

annotation inference server and dropped into the document. Thereby, some piece of text and/or a link is produced taking into account the information from the meta ontology (cf. Section 5). The newly generated content is already annotated and the meta ontology guides the construction of further information, e.g. further XPointers are attached to instances.

- **Ontology Guidance and Fact Browser**: The annotation framework needs guidance from the ontology. In order to allow for sharing of knowledge, newly created annotations must be consistent with a community's ontology. If metadata creators instantiate arbitrary classes and properties the semantics of these properties remains void. Of course the framework must be able to adapt to varying ontologies in order to reflect different foci of the metadata creators.

  Furthermore, the ontology and the browser for already given facts are important in order to guide metadata creators towards creating relational metadata. We have done some preliminary experiments and found that subjects have more problems with creating relationship instances than with creating attribute instances (cf. [33]). Without the ontology they would miss even more cues for assigning relationships between class instances.

  Both ontology guidance/fact browser and document editor/viewer should be easy to use: Drag'n'drop helps to avoid syntax errors and typos and a good visualization of the ontology can help to correctly choose the most appropriate class for instances.

- **Crawler:** The creation of relational metadata must take place *within* the Semantic Web. During metadata creation subjects must be aware of which entities exist already in their part of the Semantic Web. This is only possible if a crawler makes relevant entities immediately available. So, metadata creators may look for proper reference, i.e. decide whether an entity already has a URI (e.g. whether the entity named "Dieter Fensel" or "D. Fensel" has already been identified by some other metadata creators) and only by this way metadata creators may recognize whether properties have already been instantiated (e.g. whether "Dieter Fensel" has already been linked to his publications). As a consequence of metadata creators' awareness relational metadata may be created, because class instances become related rather than only flat templates are filled.

  We have built a RDF Crawler[6], a basic tool that gathers interconnected fragments of RDF from the Web and builds a local knowledge base from this data (cf. [16] for a more detailed description).

[6]RDF Crawler is freely available for download at: `http://ontobroker.semanticweb.org/rdfcrawler`.

- **Annotation Inference Server**: Relational metadata, proper reference and avoidance of redundant annotation require querying for instances, i.e. querying whether and which instances exist. For this purpose as well as for checking of consistency, we provide an annotation inference server in our framework. The annotation inference server reasons on crawled and newly created instances and on the ontology. It also serves the ontological guidance and fact browser, because it allows to query for existing classes, instances properties.

  We use Ontobroker's [5] underlying F-Logic [22] based inference engine SilRI [4] as annotation inference server. The F-Logic inference engine combines ordering-independent reasoning in a high-level logical language with a well-founded semantics.

  However, other scheme's like the DAML+OIL FACT reasoner [19, 3] or a fact serving peer [29] may be exploited, too.

- **Document Management**: In order to avoid redundancy of metadata creation efforts, it is not sufficient to ask whether instances exist at the annotation inference server. When a metadata creator decides to capture knowledge from a Web page, he does not want to query for all single instances that he considers relevant on this page, but he wants information, whether and how this Web page has been annotated before. Considering the dynamics of HTML pages on the web, it is desirable to store annotated web pages together with their annotations. When the web page changes, the old annotations may still be valid or they may become invalid. The metadata creator must decide based on the old annotations and based on the changes of the web page.

  A future goal of the document management in our framework will be the semi-automatic maintenance of annotations. When only few parts of a document change, pattern matching may propose revision of old annotations.

  In our current implementation we use a straight forward file-system based document management approach. Ont-O-Mat uses the URI to detect the re-encounter of previously annotated documents and highlights annotations in the old document for the user. Then the user may decide to ignore or even delete the old annotations and create new metadata, he may augment existing data, or he may just be satisfied with what has been previously annotated.

- **Metadata Re-recognition & Information Extraction**: Even with sophisticated tools it is laborious to provide semantic annotations. A major goal thus is semi-automatic metadata creation taking advantage of information extraction techniques to propose annotations to metadata

creators and, thus, to facilitate the metadata creation task. Concerning our environment we envisage three major techniques:

1. First, metadata re-recognition compares existing metadata literals with newly typed or existing text. Thus, the mentioning of the name "Siegfried Handschuh" in the document triggers the proposal that URI, `urn:rdf:947794d5ca907974ea16565de21c998a-0` is co-referenced at this point.
2. "Wrappers" may be learned from given markup in order to automatically annotate similarly structured pages (cf., e.g., [25]).
3. Message extraction like systems may be used to recognize named entities, propose co-reference, and extract some relationship from texts (cf., e.g., [28, 35]).

This component has not yet been integrated in our Ont-O-Mat tool. However, we are near finishing an integration of a simple wrapper approach [24], and we currently work on integrating the Amilcare information extraction system (`http://www.dcs.shef.ac.uk/~fabio/Amilcare.html`).

- **Meta Ontology**: The purpose of the meta ontology is the separation of ontology design and use. It is specifically explained in Section 5.

Besides of the requirements that constitute single modules, one may identify functions that cross module boundaries:

- **Storage**: CREAM supports two different ways of storage. The annotations will be stored inside the document that is in the document management component, but it is also stored in the annotation inference server.
- **Replication**: We provide a simple replication mechanism by crawling annotations into our annotation inference server.

Table 1: Design Rationale — Linking Requirements with CREAM Modules

Requirement / General Problem	Document Editor	Ontology Guidance	Crawler	Storage: Replication	Storage: Annotation Inference Server	Storage: Document Management	Metadata Re-recognition	General Information Extraction	Meta Ontology	Content Generation
Consistency		X			X				X	X
Proper Reference			X		X		X		X	X
Avoid Redundancy			X		X	X				X
Relational Metadata		X	X		X		X			
Maintenance						X	X	X		X
Ease of use	X	X					X	X	X	
Efficiency	X	X	X		X	X	X	X	X	X

4.2 Architecture of CREAM

The architecture of CREAM is depicted in Figure 2. The Design of the CREAM framework pursues the idea to be flexible and open. Therefore, Ont-O-Mat, the implementation of the framework, comprises a plug-in structure, which is flexible with regard to adding or replacing modules.[7] Document viewer/editor and ontology guidance/fact browser together constitute the major part of the graphical user interface. Because of the plug-in structure they can be replaced by alternative viewers. For instance, we are currently working on a PDF viewer plugin capable of writing RDF into PDF documents.

Further capabilities are provided through plugins that establish connections, e.g. one might provide a plug-in for a connection to a commercial document management system.

The core Ont-O-Mat already comes with some basic functionalities. For instance, one may work without a plug-in for an annotation inference server, because the core Ont-O-Mat provides simple means to navigate the taxonomy *per se*. However, he only gets the full-fledged semantic capabilities (e.g. datalog reasoning or subsumption reasoning) when one uses a plug-in connection to a corresponding annotation inference server.

5 Meta Ontology

The core idea behind the meta ontology is the modularization of ontology development and use. It should be possible to define the ontology — or reuse an existing one — rather independently of the purpose of creation of metadata by web page authoring and annotation.

The meta ontology describes how classes, attributes and relationships from the ontology should be used by the CREAM environment. In particular, we have recognized the urgent need for the meta ontology characterizations elaborated in Sections 5.1 to 5.3.

The reader may note that the descriptions of how the meta ontology influences the interaction with the ontology, which are given in this section, depend to some extent on the modes of interaction described in Section 6 — and vice versa.

[7] the Ont-O-Mat architecture serves also as basis for the KAON-IDE, see
`http://kaon.aifb.uni-karlsruhe.de`

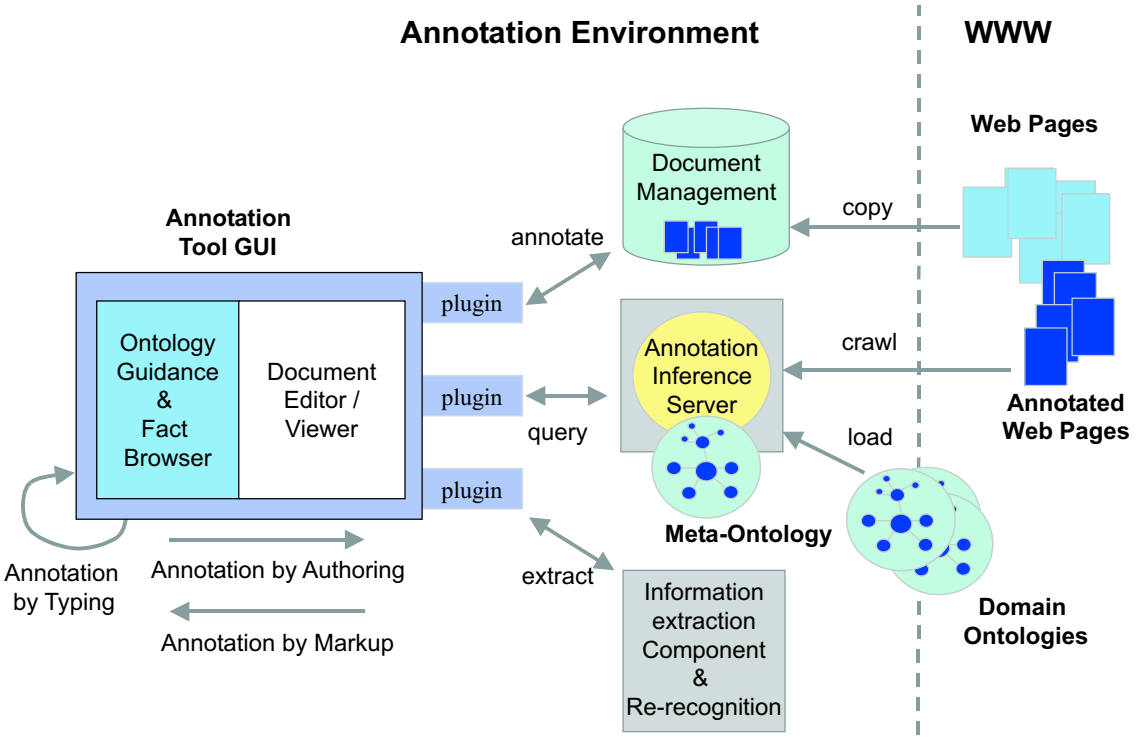

Figure 2: Architecture of CREAM.

5.1 Label

The specification of RDFS [2] provides a RDFS:LABEL as a human-readable version of a resource name. Analogously, one wants to assign an instance with a human-readable name even if it instantiates a class from a given ontology that does not use the property RDFS:LABEL *per se*.

For instance, assume that (part of) the ontology definition is as follows:

```
<rdf:Property ID="ssn">
    <rdfs:comment>Social Security Number</rdfs:comment>
    <rdfs:range rdf:resource="http://www.w3.org/2000/03/example/classes#Integer"/>
    <rdfs:domain rdf:resource="#Person"/>
  </rdf:Property>
<rdf:Property ID="fullname">
    <rdfs:comment>Last Name, First Name, Middle Initial</rdfs:comment>
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
    <rdfs:domain rdf:resource="#Person"/>
  </rdf:Property>
```

Then, one might want to state that the property FULLNAME rather than the property SSN takes the

role of RDFS:LABEL for class PERSON. We may link the meta ontology (the relevant piece here is RDFS:LABEL) with the ontology proper by:

```
<rdf:Property ID="fullname">
    <rdf:subPropertyOf rdf:resource="http://www.w3.org/TR/rdf-schema/#rdfs:label"/>
</rdf:Property>
```

Now, the authoring and annotation environment may exploit this additional piece of information at (at least) two points of interaction:

1. Instance Generation: When a *new instance* is about to be created and some piece of text has been chosen to represent the name, CREAM uses the RDF-API to create a new URN and automatically assigns this piece of text to the attribute recorded as RDFS:LABEL, i.e. here FULLNAME.

2. Content Generation: When an instance is selected for generating content of a Web page, the generated text is produced by the RDFS:LABEL attribute. By this way we may produce text that is meaningful to humans with little interaction, because the author need not specify the attribute that should be used but he may just refer to the instance.

One may note that another person, e.g. from administration, could rather choose:

```
<rdf:Property ID="ssn">
    <rdf:subPropertyOf rdf:resource="http://www.w3.org/TR/rdf-schema/#rdfs:label"/>
</rdf:Property>
```

Thus, this person would create web page content refering to social security numbers when authoring with existing instances and she would create new instances of PERSON from given social security numbers and not from names.

The reader may note from this small example that

- The difference between more or less metadata creation effort is often just one click.[8]

- The connection between ontology and meta ontology is not an objective one. Rather their linkage depends on the way an ontology is used in a particular metadata creation scenario.

Statements equivalent to the last text passage hold for the following meta ontology descriptions.

[8] We conjecture that the one click difference may distinguish between success and failure of a tool.

5.2 Default Pointing

For many instances that are to be created it is desirable to point to the Web page from which they "originated". Analogously to the way that RDFS:LABEL is used, we use four types of definitions in order to specify the default pointing behavior for class instances, exploiting the XPointer candidate recommendation [7].

Consider the four meta ontology properties CREAM:DPOINTER, CREAM:UNIQUEDPOINTER, CREAM:AUTODPOINTER, and CREAM:AUTOUNIQUEDPOINTER. If a property is RDFS:SUBPROPERTYOF one of these four the following interactions take place during annotation/authoring:

1. Instance Generation: When a new instance is generated and a property of that instance is of type CREAM:AUTODPOINTER or CREAM:AUTOUNIQUEDPOINTER, an XPointer to the current (part of the) web page will be automatically added into the corresponding slot of the instance.

2. Content Generation: When an instance is used for generating web page content, the attribute containing the XPointer is offered for link generation. When the attribute is of type CREAM:UNIQUEDPOINTER or CREAM:AUTOUNIQUEDPOINTER indicating uniqueness, a link with text corresponding to RDFS:LABEL and HRef corresponding to the XPointer will be automatically generated .

For instance, one may model in the ontology that a PERSON comes with properties HASHOMEPAGE and FULLNAME and in the instantiation of the meta ontology that HASHOMEPAGE is a subPropertyOf CREAM:UNIQUEDPOINTER and FULLNAME a subPropertyOf RDFS:LABEL. During annotation of people homepages, the label and pointer mechanisms automate, *(i)*, the generation of unique IDs with reasonable labels, *(ii)*, the creation of pointers to people's homepages, and, *(iii)*, the correct linking between people mentioned on the different homepages. Like with RDFS:LABEL, the linkage between meta ontology and ontology proper may depend on the current usage scenario.

5.3 Property mode

The property mode distinguishes between different roles, which correspond to different ways the property should be treated by the metadata creation environment:

1. **Reference**: In order to describe an object, metadata may simply point to a particular place in a resource, e.g. a piece of text or a piece of multimedia. For instance, one may point to a particular place at `http://www.whitehouse.gov` in order to refer to the current U.S. president. Even when the presidency changes the metadata may remain up-to-date.

   References are particularly apt to point to parts of multimedia, e.g. to a part of a scalable vector graphics.

   The reader may note that every default pointer is a reference, but not vice versa.

2. **Quotation**: In order to describe an object, metadata may copy an excerpt out of a resource. In contrast to the mode "reference", a quotation does not change when the corresponding resource changes. A copy of the string "Bill Clinton" as president of U.S. in 1999 remains unchanged even if its original source at `http://www.whitehouse.gov` changes or is abandoned.

3. **Unlinked Fact**: An unlinked fact describes an object, but is not in any way stemming or depending on a resource. Unlinked facts are typical for comments. The are also very apt to be combined with references in order to elucidate the meaning or name of a graphics or piece of multimedia.

   For instance, there may be a reference pointing to the picture "Guernica" (`http://www.grnica.swinternet.co.uk/guernica.jpg`) and attributes that are specified to be unlinked facts. The unlinked fact-attributes may be filled by someone who knows Picasso's paintings, e.g. with specifications like "Guernica" or "Spanish Civil War".

The meaning of the property mode may slightly overlap with the definition of the range of a property. The reason is that a pointer may be used as a URI (e.g. [14]) and a URI should typically not appear in a literal (though this is not forbidden). We separate the two aspects, because not every URI is a pointer (e.g., a URI may be a URN) and it sometimes makes sense to specify the value of a literal by a pointer. Thus, the definition of the range of a property as reference, quotation or unlinked fact may be considered orthogonal to the range of a property being a literal or a resource.

5.4 Further Meta Ontology Descriptions

Concluding this section, we want the reader to note that the list of possibly useful meta ontology descriptions sketched here is not closed by far. Rather, we envision (and partially support) the use of meta ontology descriptions for purposes such as

- Knowledge acquisition from templates: For instance, in SWOBIS (`http://tools.semanticweb.org/`) we describe software tools with metadata (cf. Figure 3). For each instance, there are a number of attributes required to specify a software tool. The meta ontology allows the definition of attribute instances being required attribute instances. This information is used to automatically generate a template like interface for Ont-O-Mat — one that is similar in its structure to a Dublin Core template. This approach is akin to the way that Protege allows to construct knowledge acquisition interfaces [30].

- Authoring of dynamic ontology and metadata-based Web pages (also cf. OntoWebber [20]).

- Provisioning of metametadata, e.g. author, date, time, and location of an annotation. Though this may appear trivial at first sight, this objective easily clashes with several other requirements, *e.g.* ease of use of metadata generation *and* usage. Eventually, it needs a rather elaborate meta ontology, containing not only static, but also dynamic definitions, *i.e.* rules.

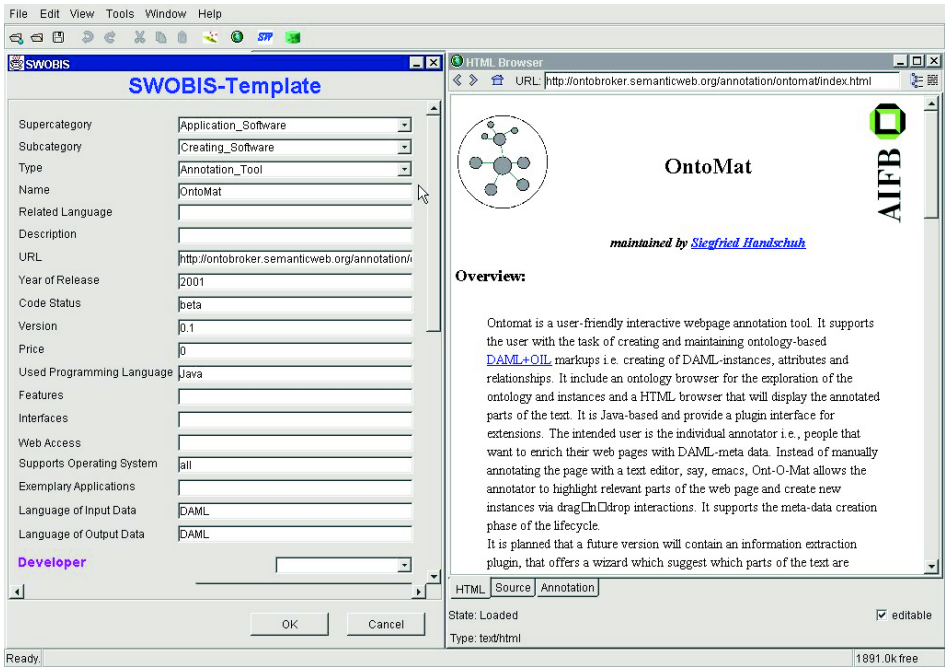

Figure 3: SWOBIS template.

6 Modes of Interaction with Ont-O-Mat

The metadata creation process in Ont-O-Mat is actually supported by three types of interaction with the tool (also cf. Figure 2):

1. Annotation by Typing Statements: This involves working almost exclusively within the ontology guidance/fact browser.
2. Annotation by Markup: This mostly involves the reuse of data from the document editor/viewer in the ontology guidance/fact browser.
3. Annotation by Authoring Web Pages: This mostly involves the reuse of data from the fact browser in the document editor.

In order to clarify the different role of the three types of interaction, we here describe how the differ for generating three types of metadata:

1. Generating instances of classes
2. Generating attribute instances
3. Generation relationship instances

6.1 Annotation by Typing

Annotation by typing is almost purely based on the ontology guidance/fact browser (cf. Section 4) and the generated templates (cf. Section 5.4). Basically, the more experienced user navigates the ontology and browses the facts, the less experienced user should rather use templates. The user generates metadata (class instances, attribute instances, relationship instances) that are completely independent from the Web page currently viewed.

The specification of the RDFS:LABEL property allows to create (or re-discover) instances by typing where the URI is given as a new URN and the RDFS:LABEL property is filled with the text. The specification of a default pointer by the meta ontology may associate newly created instances with the currently marked passage in the text.

In addition, the user may drag-and-drop around instances that are already in the knowledge in order to create new relationship instances (cf. arrow #0 in Figure 4).

6.2 Annotation by Markup

The basic idea of annotation by markup is the usage of marked-up content in the document editor/viewer for instance generation.

1. Generating class instances: When the user drags a marked up piece of content onto a particular concept from the ontology, a new class instance is generated. If the class definition comes with a meta ontology description of a RDFS:LABEL a new URI is generated as a URN and the corresponding property is assigned the marked up text (cf. arrow #1 in Figure 4).

   For instance, marking "Siegfried Handschuh" and dropping this piece of text on the concept PHDSTUDENT creates a new URN, instantiates this URN as belonging to PHDSTUDENT and assigns "Siegried Handschuh" to the SWRC:NAME slot of the new URN. In addition, default pointers may be provided.

2. Generating attribute instance: In order to generate an attribute instance the user simply drops the marked up content into the corresponding table entry (cf. arrow #2 in Figure 4). Depending on whether the attribute is specified as reference or quotation the corresponding XPointer or the content itself is filled into the attribute.

3. Generating relationship instance: In order to generate an attribute instance the user simply drops the marked up content onto the relation of a pre-selected instance (cf. arrow #3 in Figure 4). Like in "class instance generation" a new instance is generated and connected with the preselected instance.

6.3 Annotation by Authoring

The third major process is authoring Web pages and metadata together. There are two modi for authoring: *(i)*, authoring by using ontology guidance and fact browser for content generation and, *(ii)*, authoring with the help of metadata re-recognition or — more general — information extraction. So far we have only implemented means for content generation. Concerning the integration of information extraction and annotation by markup we refer the reader to [35]. Hence, we want to point out that already very simple information extraction mechanisms, i.e. metadata re-recognition (cf. Section 4) may help the author to produce consistent metadata.

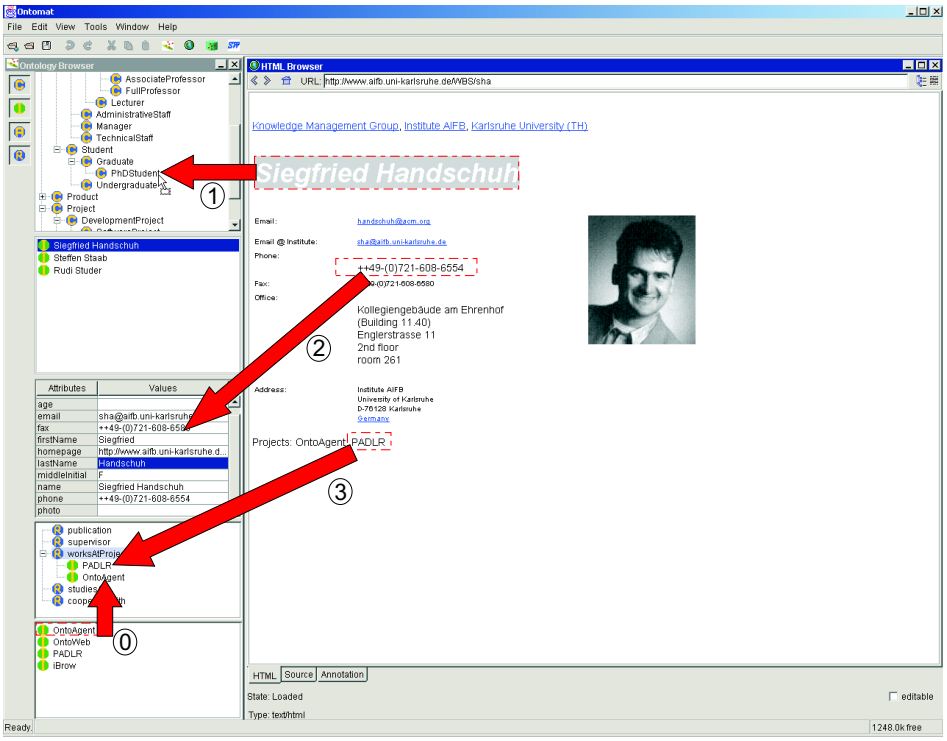

Figure 4: Annotation example

Authoring with Content Generation. By inverting the process of markup (cf. Figure 2), we may reuse existing instance description, like labels or other attributes:

1. Class instances: Dropping class instances from the fact browser into the document creates text according to their labels and — if possible — links (cf. arrow #1 in Figure 5).

2. Attribute instances: Dropping attribute instances from the fact browser in the document (cf. arrow #2 in Figure 5) generates the corresponding text (for quotations or unlinked facts) or even linked text (for references).

3. Relationship instances: Dropping relationship instances from the fact browser in the document generates simple "sentences". For instance, the dropping of the relationship COOPERATESWITH between the instances corresponding to Rudi and Steffen triggers the creation of a small piece of text (cf. arrow #3 in Figure 5). The text corresponds to the instance labels plus the label of the relationship (if available), e.g. "Rudi Studer cooperates with Steffen Staab". Typically, this piece of text will require further editing.

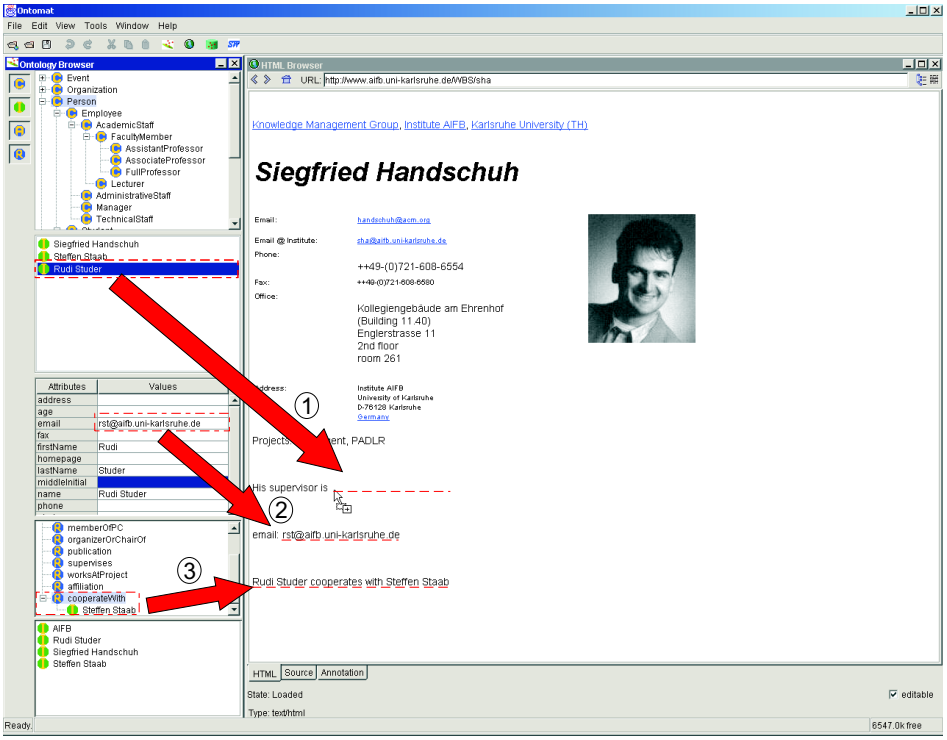

Figure 5: Annotation by Authoring example

Further mechanisms, like the creation of lists or tables from selected concepts (e.g. all PERSONS) still need to be explored.

7 Comparison with Related Work

CREAM can be compared along four dimensions: First, it is a framework for mark-up in the Semantic Web. Second, it may be considered as a particular knowledge acquisition framework very vaguely similar to Protégé-2000[11]. Third, it is certainly an annotation framework, though with a different focus than ones like Annotea [21]. And fourth it is an authoring framework with emphasis on metadata creation.

7.1 Knowledge Markup in the Semantic Web

We know of three major systems that intensively use knowledge markup in the Semantic Web, viz. SHOE [17], Ontobroker [5] and WebKB [27]. All three of them rely on knowledge in HTML pages. They all started with providing manual mark-up by editors. However, our experiences (cf. [10]) have

shown that text-editing knowledge mark-up yields extremely poor results, viz. syntactic mistakes, improper references, and all the problems sketched in the scenario section.

The approaches from this line of research that are closest to *CREAM* is the *SHOE Knowledge Annotator*[9] and the WebKB annotation tool.

The SHOE Knowledge Annotator is a Java program that allows users to mark-up webpages with the SHOE ontology. The SHOE system [26] defines additional tags that can be embedded in the body of HTML pages. The SHOE Knowledge Annotator is rather a little helper (like our earlier OntoPad [12], [5]) than a full fledged annotation environment.

WebKB uses conceptual graphs for representing the semantic content of Web documents. It embeds conceptual graph statements into HTML pages. Essentially they offer a Web-based template like interface like knowledge acquisition frameworks described next.

7.2 Comparison with Knowledge Acquisition Frameworks

The CREAM framework allows for creating class and property instances and for populating HTML pages with them. Thus, it targets a roughly similar target like the instance acquisition phase in the Protégé-2000 framework [11] (the latter needs to be distinguished from the ontology editing capabilities of Protégé). The obvious difference between CREAM and Protégé is that the latter does not (and has not intended to) support the particular Web setting, *viz.* managing and displaying Web pages — not to mention Web page authoring. From Protégé we have adopted the principle of a meta ontology that allows to distinguish between different ways that classes and properties are treated.

7.3 Comparison with Annotation Frameworks

There are a lot of — even commercial — annotation tools like ThirdVoice[10], Yawas [6], CritLink [36] and Annotea (Amaya) [21]. These tools all share the idea of creating a kind of user comment about Web pages. The term “annotation” in these frameworks is understood as a remark to an existing document. For instance, a user of these tools might attach a note like ”A really nice professor!” to the name “Studer” on a Web page. In CREAM we would design a corresponding ontology that would allow to type the comment (an unlinked fact) “A really nice professor” into an attribute instance belonging to an instance of the class COMMENT with a unique XPointer at “Studer”.

[9] http://www.cs.umd.edu/projects/plus/SHOE/KnowledgeAnnotator.html

[10] http://www.thirdvoice.com

Annotea actually goes one step further. It allows to rely on an RDF schema as a kind of template that is filled by the annotator. For instance, Annotea users may use a schema for Dublin Core and fill the author-slot of a particular document with a name. This annotation, however, is again restricted to attribute instances. The user may also decide to use complex RDF descriptions instead of simple strings for filling such a template. However, he then has no further support from Amaya that helps him providing syntactically correct statements with proper references.

One of the systems most similar to CREAM is the annotation tool cited in [35]. It uses information extraction components (Marmot, Badger and Crystal) from the University of Massachusetts at Amherst (UMass). It allows the semi-automatic population of an ontology with metadata. So far, they have not dealt with relational metadata or authoring concerns.

7.4 Comparison with Authoring Frameworks

An approach related to the CREAM authoring is the Briefing Associate of Teknowledge [34]. The tool is an extension of Microsoft PowerPoint. It pursues the idea to produce PowerPoint documents with the metadata coding as a by-product of the document composition. For each concept and relation in the ontology, an instantiation button is added to the PowerPoint toolbar. Clicking on one of these buttons, allows the author to insert an annotated graphical element into his briefing. Thus, a graphic element in the briefing corresponds to an instance of a concept and arrows between the elements correspond to relationship instances. In order to be able to use an ontology in PowerPoint one must have assigned graphic symbols to the concepts and relations, which is done in the beginning by the visual-annotation ontology editor (again a kind of meta ontology assignment). The Briefing Associate is available for PowerPoint documents. In contrast, CREAM does not provide graphic support like the Briefing Associate, but it supports both parts of the annotation process, i.e. it permits the simultaneous creation of documents and metadata and in addition the annotation of already existing documents. However, the Briefing Associate has shown very interesting ideas that may be of future value to CREAM.

The authoring of hypertexts and the authoring with concepts are topics in the COHSE project [14]. They allow for the automatic generation of metadata descriptions by analysing the content of a Web page and comparing the tokens with concept names described in a lexicon. They support ontology reasoning, but they do not support the creation of relational metadata. It is unclear to what extent COHSE considers the synchronous production of document and metadata by the author.

Somewhat similar to COHSE concerning the metadata generation, Klarity [23] automatically fills Dublin Core fields taking advantage of statistic methods to allocate values based on the current document.

In [18] Jim Hendler states that "semantic markup should be a by-product of normal computer use". He requires that "In an easy, interactive way a user would be assisted in creating a page and get the markup created for free". We think that the CREAM framework can offer a solid start for this vision to become true.

8 Conclusion

CREAM is a comprehensive framework for creating annotations, relational metadata in particular — the foundation of the future Semantic Web. The new version of CREAM presented here supports metadata creation during Web page authoring as well as by a-posteriori annotation. CREAM comprises inference services, crawler, document management system, ontology guidance/fact browser, document editors/viewers, and a meta ontology.

Ont-O-Mat is the reference implementation of the CREAM framework. It is Java-based and provides a plugin interface for extensions for further advancements, e.g. information extraction, collaborative metadata creation, or integrated ontology editing and evolution.

We are already dealing with many different issues and through our practical experiences we could identify problems that are most relevant in several settings, KA2 and Time2Research. Nevertheless our analysis of the general problem is far from being complete. In particular, we are now investigating how different tools may be brought together, e.g. to allow for the creation of relational metadata in PDF, SVG, or SMIL with Ont-O-Mat.

9 Acknowledgements.

The research presented in this paper would not have been possible without our colleagues and students at the Institute AIFB, University of Karlsruhe, and Ontoprise GmbH. We thank Kalvis Apsitis (now: RITI Riga Information Technology Institute), Stefan Decker (now: Stanford University), Michael Erdmann (now: Ontoprise GmbH), Alexander Maedche (now: FZI Research Center), Mika Maier-Collin, Leo Meyer, Tanja Sollazzo and Sichun Xu (Stanford University).

Research for this paper was partially financed by US Air Force in the DARPA DAML project "OntoAgents" (01IN901C0).

References

[1] R. Benjamins, D. Fensel, and S. Decker. KA2: Building Ontologies for the Internet: A Midterm Report. *International Journal of Human Computer Studies*, 51(3):687, 1999.

[2] D. Brickley and R.V. Guha. Resource description framework (RDF) schema specification. Technical report, W3C, 1999. W3C Proposed Recommendation. http://www.w3.org/TR/PR-rdf-schema/.

[3] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. Enabling knowledge representation on the web by extending rdf schema. In *Proc. of WWW 2001*, pages 467–478. ACM Press, 2001.

[4] S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for rdf. In *Proceedings of the W3C Query Language Workshop (QL-98), Boston, MA, December 3-4*, 1998. http://www.w3.org/TandS/QL/QL98/.

[5] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al., editors, *Database Semantics: Semantic Issues in Multimedia Systems*, pages 351–369. Kluwer Academic Publisher, 1999.

[6] L. Denoue and L. Vignollet. An annotation tool for web browsers and its applications to information retrieval. In *In Proceedings of RIAO2000*, Paris, April 2000. http://www.univ-savoie.fr/labos/syscom/Laurent.Denoue/riao2000.doc.

[7] S. DeRose, E. Maler, and R. Daniel. Xml pointer language (xpointer) version 1.0. Technical report, W3C, 2001. Candidate Recommendation 11 September 2001.

[8] Dublin core metadata initiative, April 2001. http://purl.oclc.org/dc/.

[9] Dublin core metadata template, 2001. http://www.ub2.lu.se/metadata/DC_creator.html.

[10] M. Erdmann, A. Maedche, H.-P. Schnurr, and Steffen Staab. From manual to semi-automatic semantic annotation: About ontology-based text annotation tools. In *P. Buitelaar & K. Hasida (eds). Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content*, Luxembourg, August 2000.

[11] H. Eriksson, R. Fergerson, Y. Shahar, and M. Musen. Automatic generation of ontology editors. In *Proceedings of the 12th Banff Knowledge Acquisition Workshop, Banff, Alberta, Canada*, 1999.

[12] D. Fensel, J. Angele, S. Decker, M. Erdmann, H.-P. Schnurr, S. Staab, R. Studer, and Andreas Witt. On2broker: Semantic-based access to information sources at the www. In *In Proceedings of the World Conference on the WWW and Internet (WebNet 99), Honolulu, Hawaii, USA*, 1999.

[13] Reference description of the daml+oil (march 2001) ontology markup language, March 2001. http://www.daml.org/2001/03/reference.html.

[14] C. Goble, S. Bechhofer, L. Carr, D. De Roure, and W. Hall. Conceptual open hypermedia = the semantic web? In *The Second International Workshop on the Semantic Web*, pages 44–50, Hong Kong, May 2001.

[15] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 6(2):199–221, 1993.

[16] S. Handschuh, S. Staab, and A. Maedche. CREAM — Creating relational metadata with a component-based, ontology driven framework. In *In Proceedings of K-Cap 2001*, Victoria, BC, Canada, October 2001.

[17] J. Heflin and J. Hendler. Searching the web with shoe. In *Artificial Intelligence for Web Search. Papers from the AAAI Workshop. WS-00-01*, pages 35–40. AAAI Press, 2000.

[18] James Hendler. Agents and the semantic web. *IEEE Intelligent Systems Journal*, 16(2):30–37, 2001.

[19] I. Horrocks. Using an expressive description logic: Fact or fiction? In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998*, pages 636–649. Morgan Kaufmann, 1998.

[20] Y. Jin, S. Decker, and G. Wiederhold. OntoWebber: Model-Driven Ontology-Based Web Site Management. In *Semantic Web Working Symposium (SWWS)*, Stanford, California, USA, August 2001.

[21] J. Kahan, M. Koivunen, E. Prud'Hommeaux, and R. Swick. Annotea: An Open RDF Infrastructure for Shared Web Annotations. In *Proc. of the WWW10 International Conference. Hong Kong*, 2001.

[22] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.

[23] Klarity – automatic generation of metadata based on concepts within the document, 2001. Klarity white paper. http://www.klarity.com.au.

[24] J. Klotzbuecher. Ontowrapper. Master's thesis, University of Karlsruhe, to appear 2001.

[25] N. Kushmerick. Wrapper Induction: Efficiency and Expressiveness. *Artificial Intelligence*, 118(1), 2000.

[26] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based Web Agents. In *Proceedings of First International Conference on Autonomous Agents*, 1997.

[27] P. Martin and P. Eklund. Embedding Knowledge in Web Documents. In *Proceedings of the 8th Int. World Wide Web Conf. (WWW'8), Toronto, May 1999*, pages 1403–1419. Elsevier Science B.V., 1999.

[28] *MUC-7 — Proceedings of the 7th Message Understanding Conference*, 1998. http://www.muc.saic.com/.

[29] W. Nejdl and B. Wolf et.al. EDUTELLA: A P2P Networking Infrastructure Based on RDF. WWW 11 submission.

[30] N. Fridman Noy, W. E. Grosso, and M. A. Musen. Knowledge-acquisition interfaces for domain experts: An empirical evaluation of protege-2000. In *Proceedings of the 12th Internal Conference on Software and Knowledge Engineering. Chicago, USA, July, 5-7, 2000*, 2000.

[31] S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer, and Y. Sure. Semantic community web portals. *Proc. of WWW9 / Computer Networks*, 33(1-6):473–491, 2000.

[32] S. Staab and A. Maedche. Knowledge portals — ontologies at work. *AI Magazine*, 21(2), Summer 2001.

[33] S. Staab, A. Maedche, and S. Handschuh. Creating metadata for the semantic web: An annotation framework and the human factor. Technical Report 412, Institute AIFB, University of Karlsruhe, 2001.

[34] M. Tallis, N. Goldman, and R. Balzer. The briefing associate: A role for cots applications in the semantic web. In *Semantic Web Working Symposium (SWWS)*, Stanford, California, USA, August 2001.

[35] M. Vargas-Vera, E. Motta, J. Domingue, S. Buckingham Shum, and M. Lanzoni. Knowledge Extraction by using an Ontology-based Annotation Tool. In *K-CAP 2001 workshop on Knowledge Markup and Semantic Annotation*, Victoria, BC, Canada, October 2001.

[36] Ka-Ping Yee. CritLink: Better Hyperlinks for the WWW, 1998. http://crit.org/~ping/ht98.html.