Leveraging Metadata Creation for the Semantic Web with CREAM

Siegfried Handschuh[1] and Steffen Staab[1,2] and Rudi Studer[1,2,3]

[1] Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
{sha,sst,rst}@aifb.uni-karlsruhe.de,
http://www.aifb.uni-karlsruhe.de/WBS
[2] Ontoprise GmbH, 76131 Karlsruhe, Germany,
http://www.ontoprise.com/
[3] Forschungszentrum Informatik,
http://www.fzi.de/wim/

Abstract. The success of the Semantic Web crucially depends on the easy creation of ontology-based metadata by semantic annotation. We provide a framework, CREAM, that allows for the creation of semantic metadata about static and dynamic Web pages, i.e. for semantic annotation of the Shallow and the Deep Web. CREAM supports the manual and the semi-automatic annotation of static Web pages, the authoring of new web pages with the simultaneous creation of metadata, and the deep annotation of Web pages defined dynamically by database queries.

1 Introduction

The Semantic Web supports its users to find accurate information, to combine related pieces of information into an overarching picture and to compose new applications without programming knowledge.

To achieve these objectives not only human readers have to understand the content of on a web page, software agents also must be able to interpret existing information. This is only possible when the relevant information is represented in a declarative and semantically precise way and when it is thus understandable for the computer. This need creates the necessity to provide semantically accurate, ontology-based metadata. We describe how the problem is tackled by means of our annotation framework, CREAM — CREAting Metadata for the Semantic Web. CREAM comprises methods for:

– Manual annotation: The transformation of existing syntactic resources (*viz.* textual documents) into interlinked knowledge structures that represent relevant underlying information.
– Authoring of documents: In addition to the annotation of existing documents the authoring mode lets authors create metadata — almost for free — while putting together the content of a document [11].
– Semi-automatic annotation: Efficient semi-automatic annotation based on information extraction that is trained to handle structurally and/or linguistically similar documents [13].

– Deep annotation: Dynamic Web documents results in a semantic mapping to the underlying database if the database owner cooperates in the Semantic Web and allows for direct access to the database [14].

In the following we wrap up several previous contributions ([11, 13, 14]). We start to describe core requirements for semantic annotation. For a more concise description, we first define our terminology and give an example of the kind of metadata the generation of which we support by CREAM (Section 2). In Section 3 we derive the design of CREAM from the requirements elaborated before. In Section 4, we explain the major modes of interaction supported by CREAM. We briefly sketch the integration of CREAM with a learnable information extraction component in Section 5, before we outline the basic building blocks for deep annotation in Section 6.

2 Creating Metadata for the Semantic Web

CREAM is an annotation and authoring framework suited for the easy and comfortable creation of relational metadata. OntoMat-Annotizer (OntoMat for short) is its concrete implementation.

2.1 Requirements for CREAM

Given the problems with syntax, semantics and pragmatics in earlier experiences (e.g. KA2 [1]), we list here a set of requirements. Thereby, the principal requirements apply for *a-posteriori annotation* as well as for the *integration of web page authoring with metadata creation* 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 annotators use an attribute instance, whereas the ontology requires a concept instance.
– **Proper Reference**: Identifiers of instances, *e.g.* of persons, institutes or companies, should be unique. For instance, the metadata generated in the KA2 case study contained three different identifiers for our colleague Dieter Fensel. Thus, knowledge about him could not be grasped with a straightforward query.
– **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. Current annotation tools tend to generate template-like metadata, which is hardly connected, if at all. For example, annotation environments often support Dublin Core [5, 6], providing means to state, e.g., the name of authors of a document, but not their IDs[1]. Thus,

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

the only possibility to query for all publications of a certain person requires the querying for some attribute like `fullname` — which is very unsatisfying for frequent names like “John Smith”.

- **Dynamic Documents**: A large percentage of the Web pages are not static documents. For dynamic web pages (e.g. ones that are generated from a database) it does not seem to be usefull to annotate every single page. Rather one wants to ”annotate the database” in order to reuse it for its own Semantic Web purpose.
- **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, e.g. taxonomies, properties and concepts.
- **Efficiency**: The effort for the production of metadata is an important 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.
- **Multiple Ontologies**: HTML documents in the semantic web may contain information that is related to different ontologies. Therefore the annotation framework should cater for concurrent annotations with multiple ontologies.

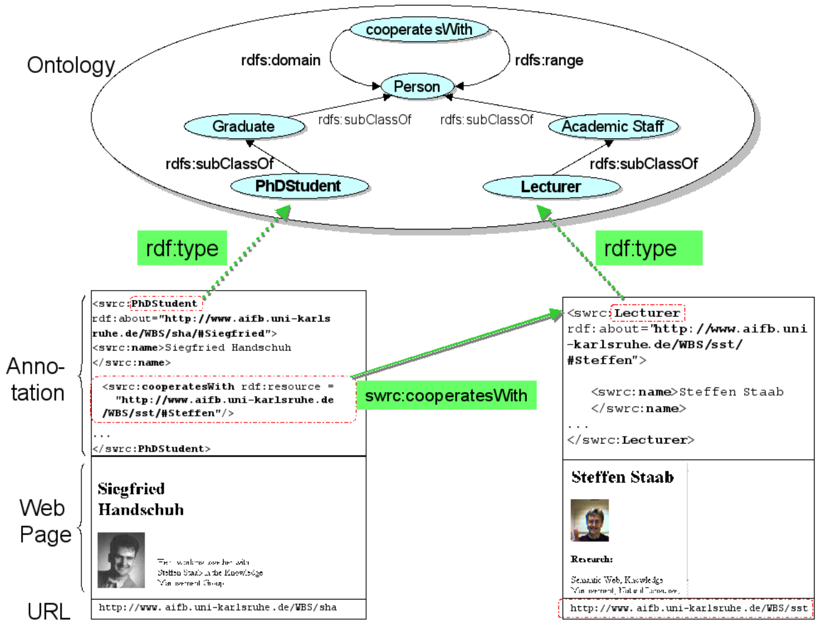
Our framework CREAM that is presented here, targets a comprehensive solution for metadata creation during web page authoring and a-posteriori annotation. The objective is pursued by combining advanced mechanisms for inferencing, fact crawling, document management, meta ontology definitions, metadata re-recognition, content generation, and information extraction. These components are explained in the subsequent sections.

2.2 Relational Metadata

We elaborate the terminology here because many of the terms that are used with regard to metadata creation tools carry several, ambiguous connotations that imply conceptually important differences:

- **Ontology**: An ontology is a formal, explicit specification of a shared conceptualization of a domain of interest [10]. In our case it is constituted by statements expressing definitions of DAML+OIL classes and properties [8].
- **Annotations**: In our context an annotation is a set of instantiations attached to an HTML document. We distinguish *(i)* instantiations of DAML+OIL classes, *(ii)* instantiated properties 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 ’`http://www.aifb.uni-karlsruhe.de/WBS/sst/#Steffen`’. 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.
– **Generic Annotations**: In a *generic annotation*, a piece of text that corresponds to a database field and that is annotated, is only considered to be a place holder. I.e. a variable must be generated for such an annotation. For example, a concept Institute in the client ontology may correspond to one generic annotation for the Organization identifier in the database. As a consequence, we will refer to detailed generic annotations as *generic class instances*, *generic attribute instances*, and *generic relationship instances*.



```
<swrc:PhDStudent
rdf:about="http://www.aifb.uni-karls
ruhe.de/WBS/sha/#Siegfried">
<swrc:name>Siegfried Handschuh
</swrc:name>
  <swrc:cooperatesWith rdf:resource =
    "http://www.aifb.uni-karlsruhe.de
  /WBS/sst/#Steffen"/>
...
</swrc:PhDStudent>
```

```
<swrc:Lecturer
rdf:about="http://www.aifb.uni
-karlsruhe.de/WBS/sst/
#Steffen">

   <swrc:name>Steffen Staab
   </swrc:name>
...
</swrc:Lecturer>
```

http://www.aifb.uni-karlsruhe.de/WBS/sha

http://www.aifb.uni-karlsruhe.de/WBS/sst

Fig. 1. Annotation example.

Figure 1 illustrates our use of the terms "ontology", "annotation" and "relational metadata". It depicts some part of the SWRC[2] (semantic web research

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

community) ontology. Furthermore it shows two homepages, viz. pages about Siegfried and Steffen (`http://www.aif b.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 (`http://www.aifb.uni-karlsruhe.de/WBS/sst/#Steffen` and `http://www.aifb.uni-karlsruhe.de/WBS /sha/#Siegfried`). The swrc:name of `http://www.aifb.uni-karlsruhe.de/WBS/sha/#Siegfried` 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.

3 Design of CREAM

3.1 CREAM Modules

The difficulties sketched before directly 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). A tabular overview of the matrix can be found in [11].

- **Document Editor**: The document editor may be conceptually — though not practically — distinguished into a viewing component and the component for generating content:
  - **Document Viewer**: The document viewer visualizes the document contents. The annotator may easily provide new metadata by selecting pieces of text and aligning it with parts of the ontology. The document viewer should support various formats (HTML, PDF, XML, etc.). For some formats the following component for content generation may not be available.
    The document viewer highlights the existing semantic annotation and server-side markup the of web page. It distinguishes visually between semantic annotation and markup that describes the information structure of a underlaying database.
  - **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. module meta ontology).
    The newly generated content is already annotated and the meta ontology guides the construction of further information, e.g. further XPointers (cf. [3], [9]) are attached to instances.
- **Ontology Guidance and Fact Browser**: The 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 multiple ontologies in order to reflect different foci of the metadata creators. In the case of concurrent annotation with multiple ontologies there is an ontology guidance/fact browser for each ontology.

– **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.
– **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. 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 and properties.
– **Meta Ontology**: The purpose of the meta ontology is the separation of ontology design and use. It is needed to describe how classes, attributes and relationships from the domain ontology should be used by the CREAM environment. Thus, the ontology describes how the semantic data should look like and the meta ontology connected to the ontology describes how the ontology is used by the annotation environment to actually create semantic data. It is specifically explained in [11].
– **Deep Annotation Module**: This module enables the deep annotation scenario. It manages the generation of mapping rules between the database and the client ontology. For this purpose, it combines the generic annotation stored in the annotation inference server and the server-side markup provided with the content (cf. Section 6). On demand it publishes the mapping rules derived from the generic annotations.
– **Document Management**: Considering the dynamics of HTML pages on the web, it is desirable to store foreign web pages one has annotated together with their annotations. Foreign documents for which modification is not possible may be remotely annotated by using XPointer (cf. [3], [9]) as a addressing meachanism.
– **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, `http://www.aifb.uni-karlsruhe.de/WBS/sha/#Siegfried` is co-referenced at this point.
  2. "Wrappers" may be learned from given markup in order to automatically annotate similarly structured pages.

3. Message extraction systems may be used to recognize named entities, propose co-reference, and extract some relationship from texts (cf., e.g., [15, 19]).

This component has been realized by using the Amilcare information extraction system (cf. Section 5)[3], but it is not yet available in the download version of OntoMat.

Besides 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. Alternatively or simultaneously it is also possible to store them in the annotation inference server.
– **Replication**: We provide a simple replication mechanism by crawling annotations into our annotation inference server. Then inferencing can be used to rule out formal inconsistencies.

3.2 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, OntoMat, the implementation of the framework, comprises a plug-in structure, which is flexible with regard to adding or replacing modules.

The core OntoMat, which is downloadable, consists of an Ontology Guidance and Fact browser, a document viewer/editor, and a internal memory datastructure for the ontology and metadata. However, one 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.

4 Modes of Interaction

The metadata creation process in OntoMat 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 they differ for generating three types of metadata: i) Generating instances of classes, ii) generating attribute instances, and iii) generating relationship instances.

[3] `http://www.dcs.shef.ac.uk/~fabio/Amilcare.html`

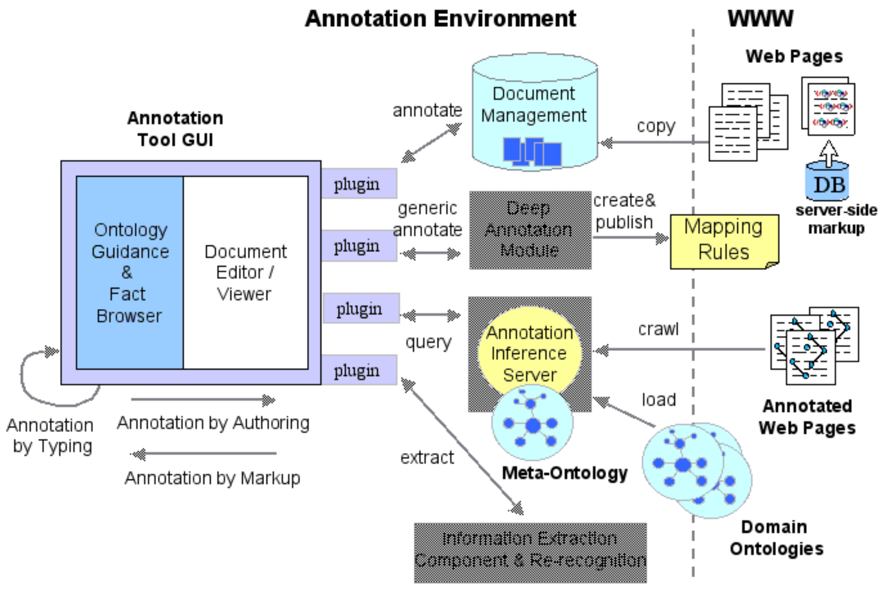


Fig. 2. Architecture of CREAM.

4.1 Annotation by Typing

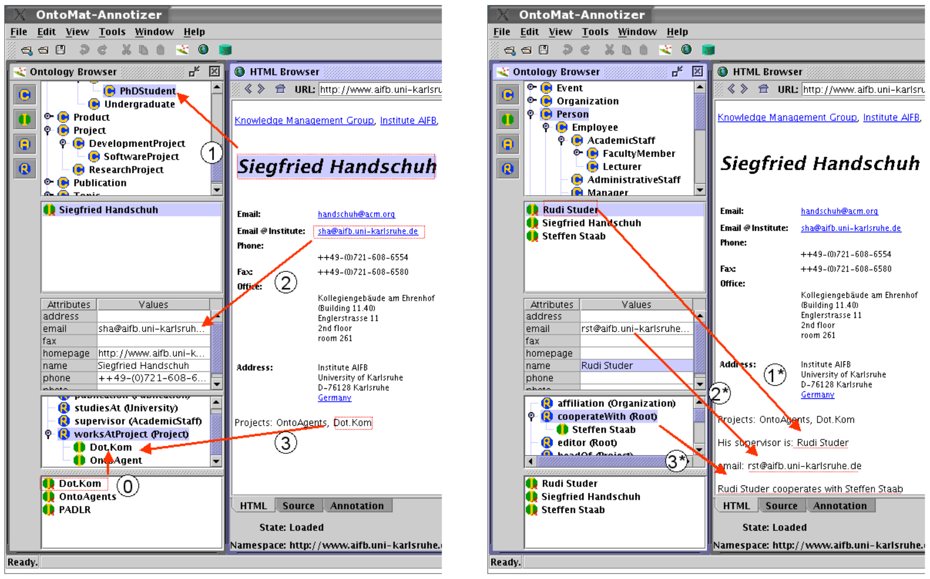
Annotation by typing is almost purely based on the ontology guidance/fact browser (cf. Section 3). The user generates metadata (class instances, attribute instances, relationship instances) that are completely independent from the Web page currently viewed. In addition, the user may drag-and-drop around instances that are already in the knowledge base in order to create new relationship instances (cf. arrow #0 in Figure 3).

4.2 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. A new URI is generated and a corresponding property is assigned the marked up text (cf. arrow #1 in Figure 3). For instance, marking "Siegfried Handschuh" and dropping this piece of text on the concept PhDStudent creates a new URI, instantiates this URI as belonging to PhDStudent and assigns "Siegfried Handschuh" to the swrc:name slot of the new URI.
2. Generating attribute instance: In order to generate an attribute instance the user drops the marked up content into the corresponding table entry (cf. arrow #2 in Figure 3). Depending on the setting the corresponding XPointer or the content itself is filled into the attribute.

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



Attributes	Values
address	
email	sha@aifb.uni-karlsruh...
fax	
homepage	http://www.aifb.uni-k...
name	Siegfried Handschuh
phone	++49-(0)721-608-6...

Knowledge Management Group, Institute AIFB,

Siegfried Handschuh

Email: handschuh@acm.org
Email @ Institute: sha@aifb.uni-karlsruhe.de
Phone: ++49-(0)721-608-6554
Fax: ++49-(0)721-608-6580
Office: Kollegiengebäude am Ehrenhof (Building 11.40) Englerstrasse 11 2nd floor room 261

Address: Institute AIFB University of Karlsruhe D-76128 Karlsruhe Germany

Projects: OntoAgents, Dot.Kom

Attributes	Values
address	
email	rst@aifb.uni-karlsruhe...
fax	
homepage	
name	Rudi Studer
phone	

Knowledge Management Group, Institute AIFB,

Siegfried Handschuh

Email: handschuh@acm.org
Email @ Institute: sha@aifb.uni-karlsruhe.de
Phone: ++49-(0)721-608-6554
Fax: ++49-(0)721-608-6580
Office: Kollegiengebäude am Ehrenhof (Building 11.40) Englerstrasse 11 2nd floor room 261

Address: Institute AIFB University of Karlsruhe D-76128 Karlsruhe Germany

Projects: OntoAgents, Dot.Kom

His supervisor is: Rudi Studer

email: rst@aifb.uni-karlsruhe.de

Rudi Studer cooperates with Steffen Staab

Fig. 3. Screenshot Annotation by Markup (left) and Annotation by Authoring (right).

4.3 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. As far as authoring is concerned, we have only implemented *(i)* so far. However, we want to point out that already very simple information extraction mechanisms, i.e. metadata re-recognition (cf. Section 3) may help the author to produce consistent metadata.

Authoring with Content Generation By inverting the process of markup (cf. Figure 3), 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 3).
2. Attribute instances: Dropping attribute instances from the fact browser in the document (cf. arrow #2* in Figure 3) generates the corresponding text or even linked text.

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 3). 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.

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

5 Semi-automatic Creation of Metadata

Providing plenty of relational metadata by manual annotation, i.e. conceptual mark-up of text passages, is a laborious task. In Section 2 we described the idea that wrappers and information extraction components could be used to facilitate the work. Hence, we have developed S-CREAM (Semi-automatic CREAtion of Metadata), an annotation framework (cf. [13]) that integrates a learnable information extraction component (viz. Amilcare [2]). Amilcare is a system that learns information extraction rules from manually marked-up input. S-CREAM aligns conceptual markup, which defines relational metadata, (such as provided through OntoMat-Annotizer) with semantic and indicative tagging (such as produced by Amilcare).
Synthesizing S-CREAM: In order to synthesize S-CREAM out of the existing frameworks CREAM and Amilcare, we consider their core processes in terms of input and output, as well as the process of S-CREAM. Figure 4 surveys the three processes. The first process is indicated by a circled M. It is manual annotation of metadata, which turns a document into relational metadata that corresponds to the given ontology. The second process is indicated by a circled A1. It is information extraction, e.g. provided by Amilcare. When comparing the desired relational metadata from manual markup and the semantic tagging provided by information extraction systems, one recognizes that the output of this type of systems is underspecified for the purpose of the Semantic Web. In particular, the nesting of relationships between different types of concept instances is undefined and, hence, more comprehensive graph structures may not be produced. In order to overcome this problem, we introduce a new processing component, viz. a lightweight module for discourse representation. This third process is indicated by the composition of A1, A2 and A3. It bridges from the tagged output of the information extraction system to the target graph structures via an explicit discourse representation. Our discourse representation is based on a very lightweight version of Centering.

6 On Deep Annotation

A large percentage of Web pages are not static documents. Rather the majority of Web pages are dynamic. For dynamic web pages (e.g. ones that are generated from the database that contains a catalogue of books) it does not seem

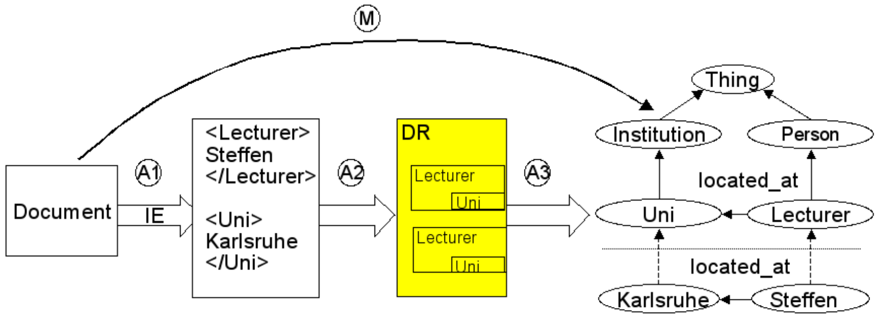


Fig. 4. Manual and Automatic Annotation

to be useful to annotate every single page. Rather one wants to "annotate the database" in order to reuse it for one's own Semantic Web purposes.

For this objective, approaches have been conceived that allow for the construction of wrappers by explicit definition of HTML or XML queries [17] or by learning such definitions from examples [2]. Thus, it has been possible to manually create metadata for a set of structurally alike Web pages. The wrapper approaches come with the advantage that they do not require cooperation by the owner of the database. However, their disadvantage is that the correct scraping of metadata is dependent to a large extent by data layout rather than by the structures underlying the data.

While for many web sites, the assumption of non-cooperativity may remain valid, we assume that many web sites will in fact participate in the Semantic Web and will support the sharing of information. Such web sites may present their information as HTML pages for viewing by the user, but they may also be willing to describe the structure of their information on the very same web pages.

Dynamic web sites with an cooperative owner may present their information as HTML pages for viewing by the user, but they may also be willing to describe the structure of their information on the very same web pages. Thus, they give their users the possibility to utilize: i) information proper, ii) information structures, and ii) information context.

A user may then exploit these three types of information in order to create mappings into his own information structures (e.g., his ontology) — which may be a lot easier than if the information a user gets is restricted to information structures [16] and/or information proper only [4].

We define "deep annotation" as an annotation process that utilizes information proper, information structures and information context in order to derive mappings between information structures. The mappings may then be exploited by the same or another user in order to query the database underlying a web site in order to retrieve semantic data — combining the capabilities of conventional annotation and databases.

6.1 Deep Annotation Process

The process of deep annotation consists of the following four steps:

Input: A Web site[4] driven by an underlying relational database.
Step 1: The database owner produces server-side web page markup according to the information structures of the database.
Result: Web site with server-side markup.
Step 2: The annotator produces client-side annotations conforming to the client ontology and the server-side markup.
Result: Mapping rules between database and client ontology
Step 3: The annotator publishes the client ontology (if not already done before) and the mapping rules derived from annotations.
Result: The annotator's ontology and mapping rules are available on the Web
Step 4: The querying party loads second party's ontology and mapping rules and uses them to query the database via the web service API.
Result: Results retrieved from database by querying party.

Obviously, in this process one single person may be the database owner and/or the annotator and/or the querying party. For example, the annotator might annotate an organization entry from ontoweb.org according to his own ontology. Then, he may use the ontology and corresponding mapping to instantiate his own syndication services by regularly querying for all recent entries the titles of which match to his list of topics.

6.2 Configuration and Roles

Our scenario for deep annotation consists of three major pillars corresponding to the three different roles (database owner, annotator, querying party) as described in the process.
Database and Web Site Provider. At the web site, we assume that there is an underlying database (cf. Figure 5) and a server-side scripting environment, like Zope, JSP or ASP, used to create dynamic Web pages. Furthermore, the web site may also provide a Web service API to third parties who want to query the database directly.
Annotator. The annotator uses an extended version of the OntoMat in order to manually create relational metadata, which correspond to a given client ontology, for some Web pages. The extended OntoMat takes into account problems that may arise from generic annotations required by deep annotation (see Section 6.3). With the help of OntoMat, we create mapping rules from such annotations that are later exploited by an inference engine.
Querying Party. The querying party uses a corresponding tool to visualize the client ontology, to compile a query from the client ontology and to investigate the mapping. In our case, we use OntoEdit [18] for those three purposes. In particular, OntoEdit also allows for the investigation, debugging and change of given mapping rules. For this purpose, OntoEdit integrates and exploits the Ontobroker [7] inference engine (see Figure 5).

[4] Cf. Section 7 on other information sources.

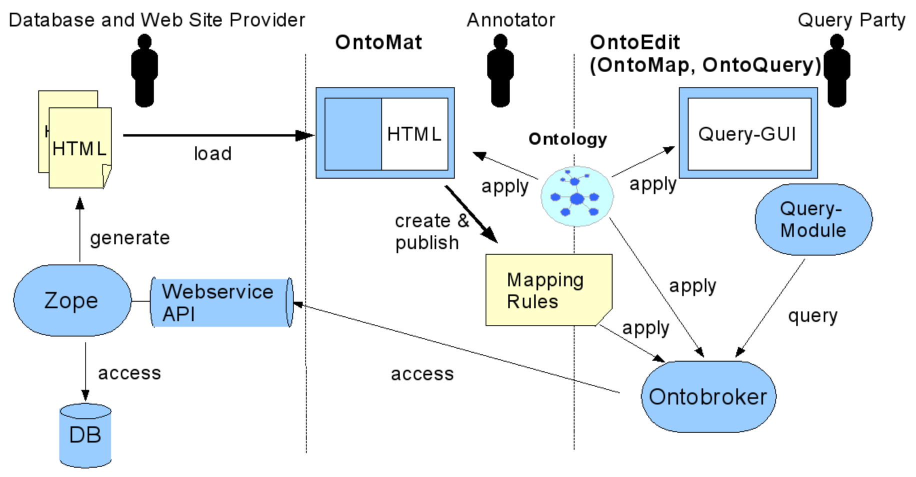


Fig. 5. Configuration for Deep Annotation

6.3 Annotation

To enable deep annotation one must consider an additional kind of annotation, viz. *generic annotation*. In a *generic annotation*, a piece of text that corresponds to a database field and that is annotated is only considered to be a place holder. I.e. a variable must be generated for such an annotation and the variable may have multiple relationships allowing for the description of general mapping rules.
Annotation Process: An annotation process of server-side markup (generic annotation) is supported by the user interface as follows:

1. The user opens in the browser a server-side marked up web page.
2. The server-side markup is handled individually by the browser, e.g. it provides graphical icons on the page wherever a markup is present, so that the user can easily identify values which come from a database.
3. The user can select one of the server-side markups to either create a new *generic instance* and map its database field to a generic attribute, or map a database field to a*generic attribute* of an existing *generic instance*.
4. The database information necessary to query the database in a later step is stored alongwith the *generic instance*.

The reader may note that *literal annotation* is still performed when the user drags a marked up piece of content, that is not a server-side markup.
Create Generic Instances of Classes: When the user drags a server-side markup onto a particular concept of the ontology, a new generic class instance is generated. The application displays a dialog for the selection of the instance name and the attributes to map the database value to. Attributes which resemble the column name are preselected. If the user clicks OK, database concept and instance checks are performed and the new generic instance is created. Generic instances will appear with a database symbol in their icon.

Create Generic Attribute Instances: In order to create a generic attribute instance the user simply drops the server-side markup into the corresponding table entry. Generic attributes which are mapped to database table columns will also show a special icon and their value will appear in italics. Such generic attributes cannot be modified, but their value can be deleted. When the generic attribute is filled the following steps are performed by the system:

1. Checking database definition integrity.
2. All attributes of the selected generic instance (except the generic attribute to be pasted to) are examined. The following conditions apply to each attribute:
   - The attribute is empty or
   - The attribute does not hold server-side markup or
   - The attribute holds markup, the database name and the query ID of the content on the current selection must be the same. This must be checked to ensure that result fields come from the same database and the same query. If this is not checked, unmatching information (e.g. publication titles and countries) could be queried for.
3. The generic attribute contains the information given by the markup, i.e. which column of the result tuple delivered by a query represents the value.

Create Generic Relationship Instances: To create a generic relationship instance the user drops the selected server-side markup onto the relation of a pre-selected instance. As in Section 6.3 a new generic instance is generated. The new generic instance is connected with the preselected generic instance.

6.4 Mapping and Querying

The results of the annotation are mapping rules between the database and the client ontology. The annotator publishes the client ontology and the mapping rules derived from annotations. This enables third parties (querying party) to access and query the database on the basis of the semantics that is defined in the ontology. The user of this mapping description might be a software agent or a human user.

7 Conclusion

CREAM is a comprehensive framework for creating semantic metadata, relational metadata in particular — the foundation of the Semantic Web. CREAM supports the annotation on the Shallow and the Deep Web. In order to avoid problems with syntax, semantics and pragmatics, CREAM employs a rich set of modules including inference services, crawler, document management system, ontology guidance/fact browser, and document editors/viewers.

For a more elaborate comparison of related work the interested reader may turn to [11, 14] as well as to the corresponding contributions in [12].

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):687–713, 1999.
2. F. Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *Proc. of the 17th IJCAI*, Seattle, Usa, August 2001.
3. S. DeRose, E. Maler, and R. Daniel. XML Pointer Language (XPointer). Technical report, W3C, 2001. Working Draft 16 August 2002.
4. A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proc. of the WWW 2002*, pages 662–673. ACM Press, 2002.
5. Dublin core metadata initiative, April 2001. http://purl.oclc.org/dc/.
6. Dublin Core Metadata Template, 2001. http://www.ub2.lu.se/metadata/DC_creator.html.
7. 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 *Proc. of WebNet 99, Honolulu, Hawaii, USA*, pages 366–371, 1999.
8. Reference description of the DAML+OIL (March 2001) ontology markup language, March 2001. http://www.daml.org/2001/03/reference.html.
9. C. Goble, S. Bechhofer, L. Carr, D. De Roure, and W. Hall. Conceptual Open Hypermedia = The Semantic Web? In S. Staab, S. Decker, D. Fensel, and A. Sheth, editors, *The Second International Workshop on the Semantic Web*, CEUR Proceedings, Volume 40, http://www.ceur-ws.org, pages 44–50, Hong Kong, May 2001.
10. T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 6(2):199–221, 1993.
11. S. Handschuh and S. Staab. Authoring and annotation of web pages in cream. In *Proc. of the 11th WWW 2002, Honolulu, Hawaii, May 7-11, 2002*, pages 462–473. ACM Press, 2002.
12. S. Handschuh and S. Staab, editors. *Annotation in the Semantic Web.* IOS Press, 2003.
13. S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM — Semi-automatic CREAtion of Metadata. In *Proc. of EKAW02*, LNCS/LNAI 2473, pages 358–372, Sigüenza, Spain, October 2002. Springer.
14. S. Handschuh, S. Staab, and R. Volz. On deep annotation. In *Proc. of the WWW2003*, Budapest, HUNGARY, May 2003.
15. *MUC-7 — Proc. of the 7th Message Understanding Conference*, 1998. http://www.muc.saic.com/.
16. N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proc. of AAAI-2000*, pages 450–455, 2000.
17. A. Sahuguet and F. Azavant. Building intelligent Web applications using lightweight wrappers. *Data and Knowledge Engineering*, 3(36):283–316, 2001.
18. Y. Sure, J. Angele, and S. Staab. Guiding Ontology Developement by Methodology and Inferencing. In K. Aberer and L. Liu, editors, *Proc. of ODBASE-2002. Irvine, CA, USA, Oct. 29-31, 2002*, LNCS, pages 1025–1222. Springer, 2002.
19. M. Vargas-Vera, E. Motta, J. Domingue, S. Buckingham Shum, and M. Lanzoni. Knowledge Extraction by using an Ontology-based Annotation Tool. In *Proc. of the Knowledge Markup and Semantic Annotation Workshop 2001 (at K-CAP 2001)*, pages 5–12, Victoria, BC, Canada, October 2001.