On Deep Annotation

Siegfried Handschuh[1], Steffen Staab[1,2], Raphael Volz[1]
[1]Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
{sha, sst, rvo}@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de/WBS
[2]Ontoprise GmbH, 76131 Karlsruhe, Germany
http://www.ontoprise.com/

ABSTRACT

The success of the Semantic Web crucially depends on the easy creation, integration and use of semantic data. For this purpose, we consider an integration scenario that defies core assumptions of current metadata construction methods. We describe a framework of metadata creation when web pages are generated from a database *and* the database owner is cooperatively participating in the Semantic Web. This leads us to the definition of ontology mapping rules by manual semantic annotation and the usage of the mapping rules and of web services for semantic queries. In order to create metadata, the framework combines the presentation layer with the data description layer — in contrast to "conventional" annotation, which remains at the presentation layer. Therefore, we refer to the framework as *deep annotation.*[1]

We consider deep annotation as particularly valid because, *(i)*, web pages generated from databases outnumber static web pages, *(ii)*, annotation of web pages may be a very intuitive way to create semantic data from a database and, *(iii)*, data from databases should not be materialized as RDF files, it should remain where it can be handled most efficiently — in its databases.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*; I.7.1 [**Document and Text Processing**]:

General Terms

Design, Human Factors

Keywords

Annotation, Metadata, Semantic Web, Information Integration, Wrapping, Mapping and Merging

1. INTRODUCTION

One of the core challenges of the Semantic Web is the creation of metadata by mass collaboration, i.e. by combining semantic content created by a large number of people. To attain this objective several approaches have been conceived (e.g. CREAM, MnM, or Mindswap [8, 26, 7]) that deal with the manual and/or the semi-automatic creation of metadata from existing information. These approaches, however, as well as older ones that provide metadata, e.g. for search on digital libraries, build on the assumption that the information sources under consideration are *static*, e.g. given as static HTML pages or given as books in a library.

Nowadays, however, a large percentage of Web pages are not static documents. On the contrary, the majority of Web pages are dynamic.[2] For dynamic web pages (e.g. ones that are generated from the database that contains a catalogue of books) it does not seem to be useful to manually 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 [20] or by learning such definitions from examples [11, 2]. Thus, it has been possible to manually create metadata for a set of structurally similar Web pages. The wrapper approaches come with the advantage that they do not require cooperation by the owner of the database. However, their shortcoming is that the correct scraping of metadata is dependent to a large extent on data layout rather than on 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. Thus, they give their users the possibility to utilize

1. information proper,
2. information structures, and
3. information context.

A user may then exploit these three 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 receives is restricted to information structures [16] and/or information proper alone [5].

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 to query the database underlying a web site in order to retrieve semantic data — combining the capabilities of conventional annotation and databases.

[1]The term "deep annotation" was coined by Carole Goble in the Semantic Web Workshop of WWW 2002.

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM 1-58113-680-3/03/0005..

[2]It is not possible to give a percentage of dynamic to static web pages in general, because a single Web site may use a simple algorithm to produce an infinite number of, probably not very interesting, web pages. Estimations, however, based on web pages actually crawled by existing search engines estimate that dynamic web pages outnumber static ones by 100 to 1.

In the remainder of the paper, we will describe the building blocks for deep annotation. First, we elaborate on the use cases of deep annotation in order to illustrate its possible scope (Section 2). We continue with a description of the overall process in Section 3. Section 4 details the architecture that supports the process, where we find three major requirements that must be provided:

1. A server-side web page markup that defines the relationship between the database and the web page content (cf. Section 5)
2. An annotation tool to actually let the user utilize information proper, information structures and information context for creating mappings (cf. Section 6).
3. Components that let the user investigate the constructed mappings (cf. Section 7), and query the serving database.

Before we conclude with future work, we relate our work to other communities that have contributed to the overall goal of metadata creation and exploitation.

2. USE CASES FOR DEEP ANNOTATION

Deep annotation is relevant for a large and fast growing number of web sites that aim at cooperation, for instance:

Scientific databases. They are frequently built to foster cooperation among researchers. Medline, Swissprot, or EMBL are just a few examples that can be found on the Web. In the bioinformatics community alone current estimations are that 500+ large databases are freely accessible.

Such databases are frequently hard to understand and it is often difficult to evaluate whether a database table named "species" is equivalent to a table named "organism" in another database. Exploiting the information proper found in concrete tuples may help. But whether the "leech" considered as entry to an "organism" is actually the animal or the plant may be much easier to tell from the context in which it is presented than from the concrete database entry, which may resolve to "plant" or "animal" only via several joins.[3]

Syndication. Besides direct access to HTML pages of news stories or market research reports, etc., commercial information providers frequently offer syndication services. The integration of such syndication services into the portal of a customer is typically expensive manual programming effort that could be reduced by a deep annotation process that defines the content mappings.

For the remainder of the paper we will focus on the following use case :

Community Web Portal (cf., [21]). This serves the information needs of a community on the Web with possibilities for contributing and accessing information by community members. A recent example that is also based on Semantic Web technology is[4] [24]. The interesting aspect to such portals lies in the sharing of information, and some of them are even designed to deliver semantic information back to their community as well as to the outside world.[5]

The primary objective of a community setting up a portal will continue to be the opportunity of access for human viewers. However, given the appropriate tools they could easily provide information content, information structures and information context to their members for deep annotation. The way that this process runs is described in the following.

[3]Concrete examples are typically not so easy to understand as the leech example!

[4]http://www.ontoweb.org

[5]Cf., e.g., [23] for an example producing RDF from database content.

3. THE PROCESS OF DEEP ANNOTATION

The process of deep annotation consists of the following four steps (depicted in Figure 1):

Input: A Web site[6] 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 (described in detail in Section 5).

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 (Section 6).

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 (Section 7).

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 (Section 7.1 and 7.2).

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.

To align this with our running example of the community Web portal, the annotator might annotate an organization entry from ontoweb.org according to his own ontology. Then, he may use the ontology and mapping to instantiate his own syndication services by regularly querying for all recent entries the titles of which match his list of topics.

4. ARCHITECTURE

Our architecture 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 2) 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 OntoMat-Annotizer in order to manually create relational metadata, which correspond to a given client ontology, for some Web pages. The extended OntoMat-Annotizer takes into account problems that may arise from generic annotations required by deep annotation (see Section 6). With the help of OntoMat-Annotizer, 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 [25] for those three purposes. In particular, OntoEdit also allows for the investigation, debugging and change of given mapping rules. To that extend, OntoEdit integrates and exploits the Ontobroker [6] inference engine (see Figure 2).

[6]Cf. Section 9 on other information sources.

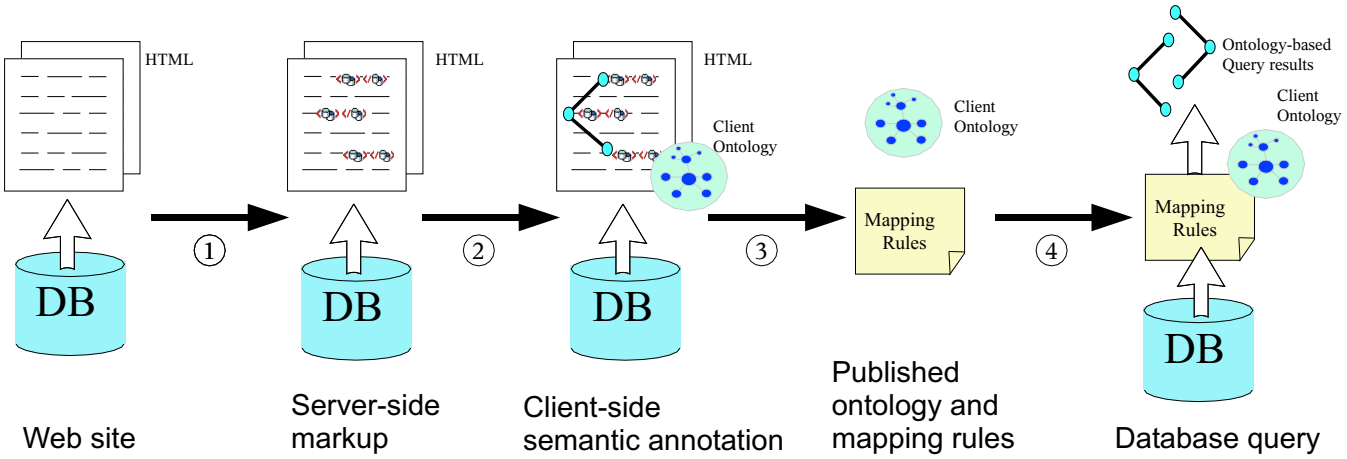

Figure 1: The Process of Deep Annotation

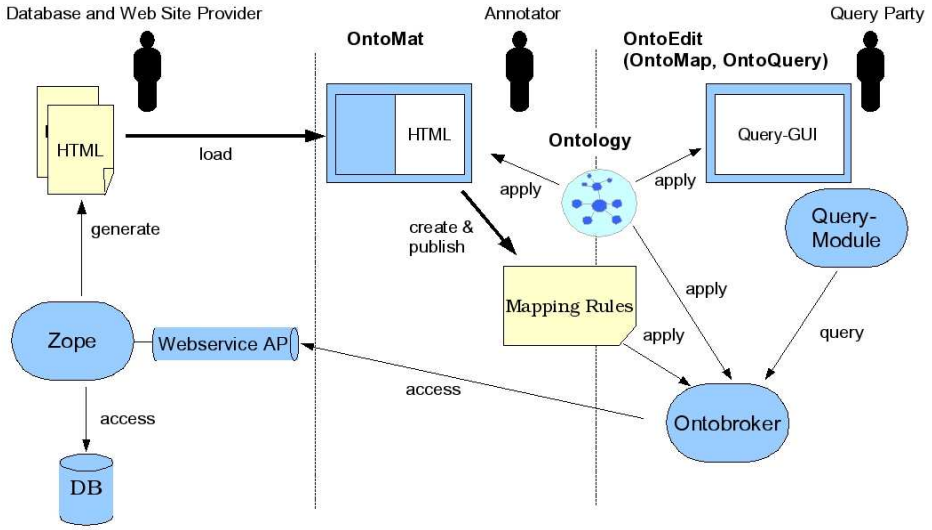

Figure 2: An Architecture for Deep Annotation

5. SERVER-SIDE WEB PAGE MARKUP

The goal of the mapping process is to allow interested parties to gain access to the source data. Hence, the content of the underlying database is not materialized, as proposed in [22]. Instead, we provide pointers to the underlying data sources in the annotations, e.g. we specify which database columns provide the data for certain attributes of instances. Thus, the capabilities of conventional annotation and databases are combined.

5.1 Requirements

All required information has to be published, so that an interested party can use this information to retrieve the data from the underlying database. The information which must be provided is as follows: *(i)* which database is used as a data source and how this database can be accessed, *(ii)* which query is used to retrieve data from the database, and *(iii)* which elements of the query result are used to create the dynamic web page. Those three components are detailed in the remainder of this section.

5.2 Database Representation

The database representation is specified using a dedicated deep annotation ontology, which is instantiated to describe the physical structure of the part of the database which may facilitate the understanding of the query results. Thereby, the structure of all tables/views involved in a query can be published. For example the following representation is part of the HTML head of the web page presented in Figure 3.

```
<!--

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:da="http://annotation.semanticweb.org#deepanno">
  <da:DB rdf:ID="OntoSQL">
    <da:accessService
        rdf:resource="www.ontoweb.org/database_access.wsdl"/>
  </da:DB>
  <da:Table rdf:ID="Person">
    <da:name>Person</da:sqlName>
    <da:inDatabase rdf:resource="#OntoSQL" />
    <da:hasColumns rdf:parseType="Collection">
```

```
        <da:PrimaryKey rdf:ID="Person.ID"
            da:name="ID" da:type="int" />
        <da:Column da:name="FIRSTNAME" da:type="varchar"/>
        <da:Column da:name="LASTNAME" da:type="varchar"/>
      </da:hasColumns>
    </da:Table>
    <da:Table rdf:ID="Organization">
      <da:name>Organization</da:name>
      <da:inDatabase rdf:resource="#OntoSQL" />
      <da:hasColumns rdf:parseType="Collection" />
          <da:PrimaryKey rdf:ID="Organization.ID"
              da:name="ID" da:type="int" />
          <da:Column da:name="ORGNAME" da:type="varchar"/>
          <da:Column da:name="LOCATION" da:type="varchar"/>
          ...
      </da:hasColumns>
    </da:Table>
    <da:Table rdf:ID="PersonOrg">
      <da:name>Person_Org<da:name>
      <da:inDatabase rdf:resource="#OntoSQL" />
      <da:hasColumns rdf:parseType="Collection" />
          <da:PrimaryKey da:name="PERSONID" da:type="int">
              <references rdf:resource="#Person.ID"/>
          </da:PrimaryKey>
          <da:PrimaryKey da:name="ORGID" da:type="int">
              <references rdf:resource="#Organization.ID"/>
          </da:PrimaryKey>
      </da:hasColumns>
    </da:Table>
</rdf:RDF>
-->
```

The property *accessService* of the <DB> class represents the link to a service which allows anonymous database access, consequently additional security measures can be implemented in the service. Usually, anonymous users should only have read-access to public information. As we rely on a web service to host the database access we avoid protocol issues (database connections are usually made via sockets on proprietary ports).

5.3 Query Representation

Additionally, the query itself, which is used to retrieve the data from a particular source is placed in the header of the page. It contains the intended SQL-query and is associated with a name as a means to distinguish between queries and operates on a particular data source.

```
<!--

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:da="http://annotation.semanticweb.org#deepanno">
    <da:Query rdf:ID="Q1">
      <da:source rdf:resource="#OntoSQL" />
      <da:hasResultColumns rdf:parseType="Collection">
        <ColumnGroup rdf:about="#g1" />
        <ColumnGroup rdf:about="#g2" />
      </da:hasResultColumns>
      <da:sql>
      SELECT Person.*, Person_Org.Orgid, Organization.*
      FROM   Person, Organization, Projekt_Org
      WHERE  Person.ID = Projekt_Org.PERSONID
             AND Organization.ID = Projekt_Org.ORGID
      </da:sql>
    </da:Query>
    <da:Columngroup rdf:ID="#g1">
        <da:prefix
            rdf:resource="http://www.ontoweb.org/person/">
        <da:hasColumns rdf:parseType="Collection">
            <Identifier da:name="Id" />
            <Column da:name="Firstname" />
            <Column da:name="Lastname" />
        </da:hasColumns>
    </da:Columngroup>
    <da:Columngroup rdf:ID="#g2">
        <da:prefix
            rdf:resource="http://www.ontoweb.org/org/">
        <da:hasColumns rdf:parseType="Collection">
            <Identifier da:name="OrganizationId" />
            <Column da:name="Orgname" />
            <Column da:name="Location" />
        </da:hasColumns>
    </da:Columngroup>
  </rdf:RDF>
-->
```

The structure of the query result must be published by means of column groups. Each column group must have at least one identifier, which is used in the annotation process to distinguish individual instances and detect their equivalence. Since database keys are only local to the respective table, but the Semantic Web has a global space of identifiers, appropriate prefixes have to established. The prefix also ensures that the equality of instance data generated from multiple queries can be detected, if the web application maintainer chooses the same prefix for each occurrence of that *id* in a query. Eventually, database keys are translated to instance identifiers (cf. Section 7.2) via the following pattern:

$$< prefix > [key_i - name = key_i - value]$$

For example: http://www.ontoweb.org/person/id=1

5.4 Result Representation

Whenever parts of the query results are used in the dynamically generated web page, the generated content is surrounded by a tag, which carries information about which column of the result tuple delivered by a query represents the used value. In order to stay compatible with HTML, we used the tag as an information carrier. The actual information is represented in attributes of :

```
<table>
<tr>
<td>
<span da:qresult="q1" da:column="Orgname">AIFB</span>
</td>
<td>
<span da:qresult="q1" da:column="Location">Karlsruhe</span>
</td>
...
<td>
<span da:qresult="q1" da:column="Firstname">Steffen</span>
</td>
...
</tr>
</table>
```

Such span tags are then interpreted by the annotation tool and are used in the mapping process.

6. ANNOTATION

An annotation in our context is a set of instantiations related to an ontology and referring 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.

In addition, for the deep annotation one must distinguish between a *generic annotation* and a *literal annotation*. In a *literal annotation*, the piece of text may stand for itself. 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. For example, a concept Institute in the client ontology

may correspond to one generic annotation for the Organization identifier in the database.

Consequential to the above terminology, we will refer to generic annotation in detail as *generic class instances*, *generic attribute instances*, and *generic relationship instances*.

6.1 Annotation Process

An annotation process of server-side markup (generic annotation) is supported by the user interface as follows:

1. In the browser the user opens 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 along with 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.

6.2 Creating 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 (cf. arrow #1 in Figure 3). The application displays a dialog for the selection of the instance name and the attributes to which the database value is to be mapped. Attributes which resemble the column name are preselected (cf. dialog #1a in Figure 3). 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.

Each generic instance stores the information about the database query and the unique identifier pattern. This information is resolved from the markup. A server-side markup contains the reference to the query, the column, and the value. The identifier pattern is obtained from the reference to the query description and the according column group (cf. Section 5.3). The markup used to create the instance, defines the identifier pattern for the generic instance. The identifier pattern will be used when instances are generated from the database (cf. Section 7.2). For example, one selects the server-side markup "AIFB" and drops it on the concept Institute. The content of the markup is 'AIFB'. This creates a new generic instance with a reference to the query *q1* (cf. Section 5.3). The dialog-based choice for the instance name "AIFB" assigns the generic attribute name with the database column "Orgname". This defines the identifier pattern of the generic instance as "http://www.ontoweb.org/org/OrganizationID=$OrganizationID". OrganizationID is the name of the database column in query *q1* that holds the database key.

6.3 Creating Generic Attribute Instances

In order to create a generic attribute instance the user simply drops the server-side markup into the corresponding table entry (cf. arrow #2 in Figure 3). 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. Database definition integrity is checked.
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, non-matching information (e.g. publication titles and countries) could be queried.
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.

6.4 Creating Generic Relationship Instances

In order to create a generic relationship instance the user simply drops the selected server-side markup onto the relation of a pre-selected instance (cf. arrow #3 in Figure 3). As in Section 6.2 a new generic instance is generated. In addition, the new generic instance is connected with the pre-selected generic instance.

7. MAPPING AND QUERYING

The results of the annotation are mapping rules between the database and the client ontology. The annotator publishes[7] 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 semantic that is defined in the ontology. The user of this mapping description might be a software agent or a human user.

7.1 Investigating Mappings

The querying party uses a corresponding tool to visualize the client ontology, to investigate the mapping and to compile a query from the client ontology. In our case, we used the OntoEdit plugins OntoMap and OntoQuery.

OntoMap visualizes the database query, the structure of the client ontology, and the mapping between them (cf. figure 4). The user can control and change the mapping and also create additional mappings.

7.2 Querying the Database

OntoQuery is a Query-by-Example user interface. One creates a query by clicking on a concept and selecting the relevant attributes and relationships. The underlying Ontobroker system transforms the ontological query into a corresponding SQL query. Ontobroker uses the mapping descriptions, which are internally represented as F-Logic Axioms, to transform the query. The SQL query will be sent as an RPC call to the web service, where it will be answered in the form of a set of records. These records are changed back into an ontological representation. This task will be executed automatically, so that no interaction with the user is necessary.

[7]Here, we used the Ontobroker OXML format to publish the mapping rules.

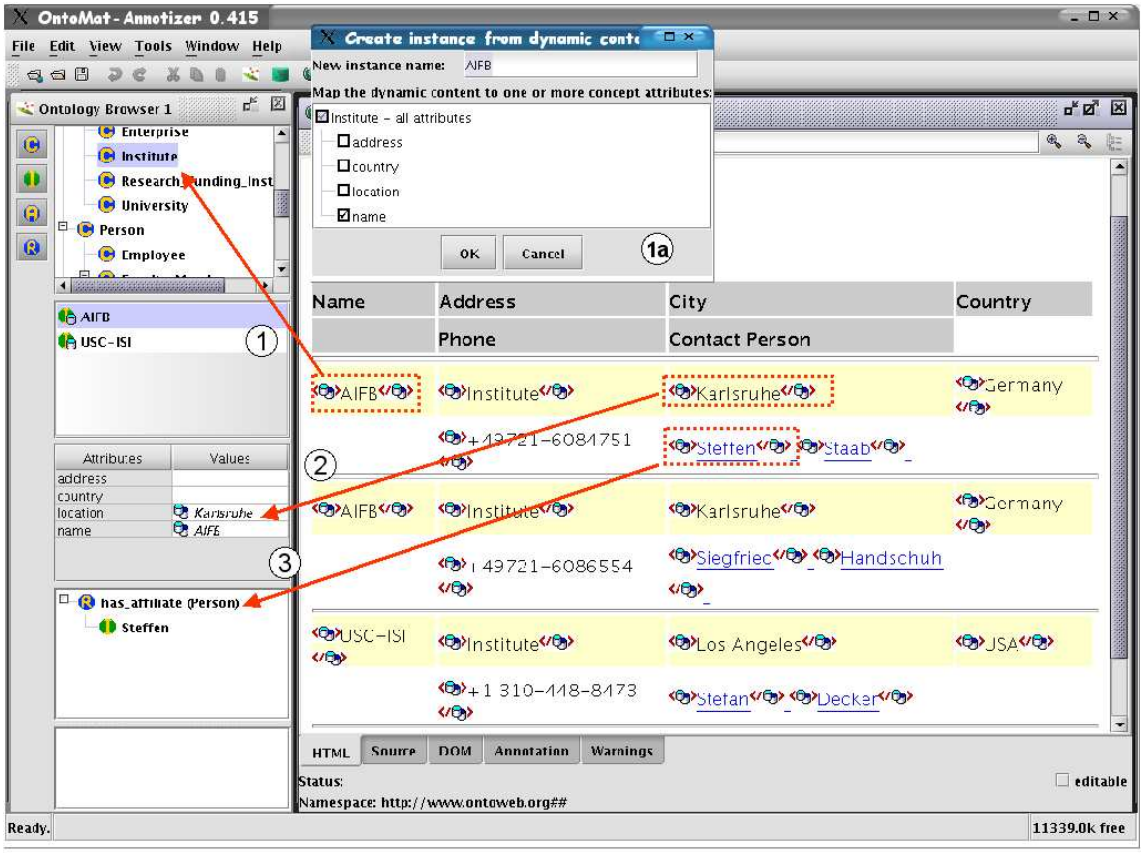

Figure 3: Screenshot of Providing Deep Annotation with OntoMat-Annotizer

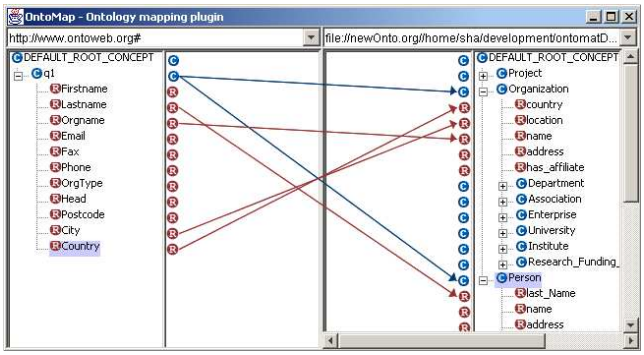

Figure 4: Mapping between Server Database (left window) and Client Ontology (right window)

The data migration will be executed in two separate steps. In the first step, all the required concept instances are created without considering relationships or attributes. The instances are stored together with their identifier. The identifier is translated from the database keys using the identifier pattern (see Section 5.2). For example, the instance with the name "AIFB" of the concept Institute, which is a subconcept of Organization, has the identifier: http://www.ontoweb.org/org/OrganizationID=3.

After the creation of all instances the system starts computing the values of the instance relationships and attributes. The way the values are assigned is determined by the mapping rules. Since the values of an attribute or a relationship have to be computed from both the relational database and the ontology, we generate two queries per attribute/relationship, one SQL query and one Ontobroker query. Each query is invoked with an instance key value (corresponding database key in SQL-queries) as a parameter and returns the value of the attribute/relationship.

Note that the database communication takes place through bind variables. The corresponding SQL query is generated, and if this is the first call, it is cached. A second call would try to use the same database cursor if still available, without parsing the respective SQL statement. Otherwise, it would find an unused cursor and retrieve the results. In this way efficient access methods for relations and database rules can be maintained throughout the session.

8. RELATED WORK

Deep annotation as we have presented it here is a cross-sectional enterprise.[8] Therefore there are a number of communities that have contributed towards reaching the objective of deep annotation. So far, we have identified communities for information integration (Section 8.1), mapping frameworks (Section 8.2), wrapper construction (Section 8.3), and annotation (Section 8.4).

8.1 Information Integration

The core idea of information integration lies in providing an algebra that may be used to translate information proper between dif-

[8]Just like the Semantic Web overall!

ferent information structures. Underlying algebras are used to provide compositionality of translations as well as a sound basis for query optimization (cf., e.g., a commercial system as described in [17] with many references to previous work — much of the latter based on principal ideas issued in [27].

Unlike [17], our objective has not been the provisioning of a flexible, scalable integration platform *per se*. Rather, the purpose of deep annotation lies in providing a flexible framework for *creating the translation descriptions* that may then be exploited by an integration platform like EXIP (or Nimble, Tsimmis, Infomaster, Garlic, etc.). Thus, we have more in common with the approaches for creating mappings with the purpose of information integration described next.

8.2 Mapping and Merging Frameworks

Approaches for mapping and/or merging ontologies and/or database schemata may be distinguished mainly along the following three categories: discovery, [19, 3, 5, 1, 16, 14], mapping representation [12, 1, 15, 18] and execution [4, 15].

In the overall area, closest to our own approach is [13], as it handles — like we do — the complete mapping process involving the three process steps just listed (in fact it also takes care of some more issues like evolution).

What makes deep annotation different from all these approaches is that for the initial discovery of overlaps between different ontologies/schemata they all depend on lexical agreement of part of the two ontologies/database schemata. Deep annotation only depends on the user understanding the presentation — the information within an information context — developed for him anyway. Concerning the mapping representation and execution, we follow a standard approach exploiting Datalog giving us many possibilities for investigating, adapting and executing mappings as described in Section 7.

8.3 Wrapper Construction

Methods for wrapper construction achieve many objectives that we pursue with our approach of deep annotation. They have been designed to allow for the construction of wrappers by explicit definition of HTML or XML queries [20] or by learning such definitions from examples [11, 2]. Thus, it has been possible to manually create metadata for a set of structurally similar Web pages. The wrapper approaches come with the advantage that they do not require cooperation by the owner of the database. However, their shortcoming is that the correct scraping of metadata is dependent to a large extent on data layout rather than on the structures underlying the data.

Furthermore, when definitions are given explicitly [20], the user must cope directly with querying by layout constraints and when definitions are learned, the user must annotate multiple web pages in order to derive correct definitions. Also, these approaches do not map to ontologies. They typically map to lower level representations, e.g. nested string lists in [20], from which the conceptual descriptions must be extracted, which is a non-trivial task. In fact, we have integrated a wrapper learning method, *viz.* Amilcare [2], into our OntoMat-Annotizer. How to bridge between wrapper construction and annotation is described in detail in [9].

8.4 Annotation

Finally, we need to consider annotation proper as part of deep annotation. There, we "inherit" the principal annotation mechanism for creating relational metadata as elaborated in [8]. The interested reader finds an elaborate comparison of annotation techniques there as well as in a forthcoming book on annotation [10].

9. CONCLUSION

In this paper we have described *deep annotation*, an original framework to provide semantic annotation for large sets of data. Deep annotation leaves semantic data where it can be handled best, *viz.* in database systems. Thus, deep annotation provides a means for mapping and re-using dynamic data in the Semantic Web with tools that are comparatively simple and intuitive to use.

To attain this objective we have defined a deep annotation process and the appropriate architecture. We have incorporated the means for server-side markup that allows the user to define semantic mappings by using OntoMat-Annotizer[9]. An ontology and mapping editor and an inference engine are then used to investigate and exploit the resulting descriptions. Thus, we have provided a complete framework and its prototype implementation for deep annotation.

For the future, there is a long list of open issues concerning deep annotation — from the more mundane, though important, ones (top) to far-reaching ones (bottom):

1. Granularity: So far we have only considered atomic database fields. For instance, one may find a string "Proceedings of the Eleventh International World Wide Web Conference, WWW2002, Honolulu, Hawaii, USA, 7-11 May 2002." as the title of a book whereas one might rather be interested in separating this field into title, location and date.

2. Automatic derivation of server-side web page markup: A content management system like Zope could provide the means for automatically deriving server-side web page markup for deep annotation. Thus, the database provider could be freed from *any* workload, while allowing for participation in the Semantic Web. Some steps in this direction are currently being pursued in the KAON CMS, which is based on Zope[10].

3. Other information structures: For now, we have built our deep annotation process on SQL and relational databases. Future schemes could exploit XQuery[11] or an ontology-based query language.

4. Interlinkage: In the future deep annotations may even link to each other, creating a dynamic interconnected Semantic Web that allows translation between different servers.

5. Opening the possibility to directly query the database, certainly creates problems such as new possibilities for denial of service attacks. In fact, queries, e.g. ones that involve too many joins over large tables, may prove hazardous. Nevertheless, we see this rather as a challenge to be solved by clever schemes for CPU processing time (with the possibility that queries are not answered because the time allotted for one query to one user is up) than for a complete "no go".

We believe that these options make *deep annotation* a rather intriguing scheme on which a considerable part of the Semantic Web might be built.

[9]The methodology "CREAM" and its implementation "OntoMat-Annotizer" have been intensively tested by authors of ISWC-2002 when annotating the summary pages of their papers with RDF metadata; see http://annotation.semanticweb.org/iswc/documents.html.

[10]see http://kaon.aifb.uni-karlsruhe.de/Members/rvo/kaon_portal

[11]http://www.w3.org/TR/xquery/

Acknowledgements.

Research for this paper has been funded by the projects DARPA DAML OntoAgents, EU IST Bizon, and EU IST WonderWeb. We gratefully thank Leo Meyer and Dirk Wenke, Ontoprise, for implementations that contributed toward the deep annotation prototype described in this paper.

10. REFERENCES

[1] S. Bergamaschi, S. Castano, D. Beneventano, and M. Vincini. Semantic Integration of Heterogeneous Information Sources. In *Special Issue on Intelligent Information Integration, Data & Knowledge Engineering*, volume 36, pages 215–249. Elsevier Science B.V., 2001.

[2] F. Ciravegna. Adaptive Information Extraction from Text by Rule Induction and Generalisation. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, pages 1251–1256, San Francisco, CA, August 2001. Morgan Kaufmann Publishers, Inc.

[3] W. Cohen. The WHIRL Approach to Data Integration. *IEEE Intelligent Systems*, pages 1320–1324, 1998.

[4] T. Critchlow, M. Ganesh, and R. Musick. Automatic Generation of Warehouse Mediators Using an Ontology Engine. In *Proceedings of the 5th International Workshop on Knowledge Representation Meets Databases (KRDB'98)*, pages 8.1–8.8, 1998.

[5] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the World-Wide Web Conference (WWW-2002)*, pages 662–673. ACM Press, 2002.

[6] 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 *Proceedings of the World Conference on the WWW and Internet (WebNet 99), Honolulu, Hawaii, USA*, pages 366–371, 1999.

[7] J. Golbeck, M. Grove, B. Parsia, A. Kalyanpur, and J. Hendler. New Tools for the Semantic Web. In *Proceedings of EKAW 2002*, LNCS 2473, pages 392–400. Springer, 2002.

[8] S. Handschuh and S. Staab. Authoring and Annotation of Web Pages in CREAM. In *Proceedings of the 11th International World Wide Web Conference, WWW 2002, Honolulu, Hawaii, May 7-11, 2002*, pages 462–473. ACM Press, 2002.

[9] S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM – Semi-automatic CREAtion of Metadata. In *Proceedings of EKAW 2002*, LNCS, pages 358–372, 2002.

[10] Siegfried Handschuh and Steffen Staab, editors. *Annotation in the Semantic Web*. IOS Press, 2003.

[11] Nicholas Kushmerick. Wrapper Induction: Efficiency and Expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[12] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *Proceedings of the 27th International Conferences on Very Large Databases*, pages 49–58, 2001.

[13] A. Maedche, B. Motik, N. Silva, and R. Volz. MAFRA - A Mapping Framework for Distributed Ontologies. In *Proceedings of EKAW 2002*, LNCS 2473, pages 235–250. Springer, 2002.

[14] D. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera Ontology Environment. In *Proc. of AAAI-2000*, pages 1123–1124, 2000.

[15] P. Mitra, G. Wiederhold, and M. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proceedings of Conference on Extending Database Technology (EDBT 2000)*. Konstanz, Germany, 2000.

[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] Y. Papakonstantinou and V. Vassalos. Architecture and Implementation of an XQuery-based Information Integration Platform. *IEEE Data Engineering Bulletin*, 25(1):18–26, 2002.

[18] J. Y. Park, J. H. Gennari, and M. A. Musen. Mappings for Reuse in Knowledge-based Systems. In *Technical Report, SMI-97-0697, Stanford University*, 1997.

[19] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[20] A. Sahuguet and F. Azavant. Building intelligent Web applications using lightweight wrappers. *Data and Knowledge Engineering*, 3(36):283–316, 2001.

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

[22] L. Stojanovic, N. Stojanovic, and R. Volz. Migrating data-intensive Web Sites into the Semantic Web. In *Proceedings of the ACM Symposium on Applied Computing SAC-02, Madrid, 2002*, pages 1100–1107. ACM Press, 2002.

[23] N. Stojanovic, A. Maedche, S. Staab, R. Studer, and Y. Sure. SEAL: a framework for developing SEmantic PortALs. In *Proceedings of K-CAP 2001*, pages 155–162. ACM Press, 2001.

[24] R. Studer, Y. Sure, and R. Volz. Managing User Focused Access to Distributed Knowledge. *Journal of Universal Computer Science (J.UCS)*, 8(6):662–672, 2002.

[25] Y. Sure, J. Angele, and S. Staab. Guiding Ontology Developement by Methodology and Inferencing. In K. Aberer and L. Liu, editors, *ODBASE-2002 – Ontologies, Databases and Applications of SEmantics. Irvine, CA, USA, Oct. 29-31, 2002*, LNCS, pages 1025–1222. Springer, 2002.

[26] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology Driven Semi-automatic and Automatic Support for Semantic Markup. In *Proceedings of EKAW 2002*, LNCS 2473, pages 379–391. Springer, 2002.

[27] G. Wiederhold. Intelligent integration of information. *Proceedings of the ACMSIGMOD International Conference on Management of Data*, pages 434–437, 1993.