Automatic and Manual Annotation using Flexible Schemas for Adaptation on the Semantic Desktop

Alexandra Cristea[1], Maurice Hendrix[1,2] and Wolfgang Nejdl[2]

[1] Faculty of Mathematics and Computer Science, Eindhoven University of Technology, (TU/e), PBox 513, 5600 MB Eindhoven, The Netherlands
m.hendrix@student.tue.nl, a.i.cristea@tue.nl
[2] L3S Research Center, University of Hannover, 30539 Hannover, Germany
nejdl@l3s.de

Abstract. Adaptive Hypermedia builds upon the annotation and adaptation of content. As manual annotation has proven to be the main bottleneck, all means for supporting it by reusing automatically generated metadata are helpful. In this paper we discuss two issues. The first is the *integration of a generic AH authoring environment MOT into a semantic desktop environment*. In this setup, the semantic desktop environment provides a rich source of automatically generated meta-data, whilst MOT provides a convenient way to enhance this meta-data manually, as needed for an adaptive course environment. Secondly, we also consider the *issue of source schema heterogeneity, especially during the automatic metadata generation process*, as semantic desktop metadata are generated through a lot of different tools and at different times, so that schemas are overlapping and evolving. Explicitly taking into account all versions of these schemas would require a combinatorial explosion of generation rules. This paper investigates a solution to this problem based on malleable schemas, which allow metadata generation rules to flexibly match different versions of schemas, and can thus cope with the heterogeneous and evolving desktop environment.

1 Introduction

Current desktop file structures are surprisingly poor from a semantic point of view. They merely provide fixed folder hierarchies and use simple names to denote entities. The semantic desktop promises much richer structures: resources can be categorized by rich ontologies and semantic links express various kinds of semantic relationships between these resources. For a document, for example, the semantic desktop stores not only a filename, but also information about where this paper was published, when and by whom, which of my colleagues sent it to me, and how often and in what context I accessed it. All these metadata is generated automatically, by the appropriate applications, and stored in an application independent way as RDF metadata, in the user's personal data store. This rich set of metadata clearly makes it easier for the user to retrieve appropriate material for different contexts: for example, when he wants to

select appropriate materials for a course lecture. Of course, in this context, we still have to add additional metadata information about course structures and other relevant attributes, like pedagogically relevant attributes, or further contents.
In this paper we first describe course authoring on the semantic desktop. Specifically, we show the interaction and exchange of data between the Beagle++ environment, which is an advanced search and indexing engine for the desktop, generating and utilizing metadata information, and the adaptive hypermedia authoring environment MOT, a sophisticated system for authoring personalized e-courses.

Second, we focus on the problem of schema heterogeneity in this environment. Metadata on the Semantic Desktop are generated by different applications at different times, and have to be described by different and usually evolving schemas. Whilst classical schema integration approaches rely on a set of mappings between these schemas, unifying data through one global schema, these approaches are infeasible in our environment: on the Semantic Desktop, there will always be a good reason for adding additional metadata or metadata attributes, which we did not foresee when starting to collect metadata one, two or more years ago. This paper investigates a possible solution for this problem based on malleable schemas [11], which employs flexible matching against those schemas without the need to continuously modify our metadata generation rules.
The paper is structured as follows. First, in section 2 we present a motivating scenario;. In section 3, the basic metadata schemas are introduced. Section 4 describes the transformation workflow between these schemas. Section 5 introduces impreciseness in the representation, via malleable schemas, as well as discusses some solutions. Finally we draw conclusions in section 6.

2 Scenario

The following motivating scenario for adaptive authoring builds upon both automatically and manually generated metadata.

Dr. Van Bos prepares a new on-line course on Adaptive Hypermedia for undergraduate 4th year TU/e students. The university work distribution allocates a limited amount of time for this, equivalent to the creation of a static, linear course. However,

- due to the fact that Dr. Van Bos considers it useful to be able to extend the course in the future with more alternative paths guided by *adaptivity*, but also,
- because he wants to benefit from *automatic help* during the authoring process,

he uses a *concept-based adaptive educational hypermedia authoring environment* with *adaptive authoring support*, MOT [8,15]. This decision costs him slightly more time than the static course creation, as he has to *manually* divide his course into conceptual entities with explicit, independent semantics and semantic labeling.

The advantage is that the *adaptive authoring system* can afterwards *automatically* apply pedagogical strategies. E.g., the system can consider the first, manual version of the course created by Van Bos as the version for *beginner* students, which don't aspire at high grades or deep knowledge. For *advanced* students, wishing to pass the course with high honors, or simply desiring more information for their future professions, the

adaptive authoring system can use *semantic personal desktop search* to automatically find on Dr. Van Bos's desktop any stored scientific papers relevant to the current course. These papers can be used as alternative or additional material to the main storyline of the static course. This mechanism builds upon the following assumptions:

- as Dr. Van Bos is a specialist in the subject taught, his interest is wider than that given by the limitations of the course; he therefore both publishes and reads papers of interest on the subject, which are likely to be stored on his computer;
- these papers can be considered as useful extra resources for the current course, and can therefore be reused in this context;
- as this storing process has taken place over several years, Van Bos may not know exactly where each single article relevant to the current course is on his computer;
- however, Dr. Van Bos has been using Beagle++ Semantic Desktop System [2, 5] to store both papers and all relevant metadata automatically, in RDF format.

This situation can be exploited by the authoring tool; a quick search will find some of Dr. Van Bos's own papers on Adaptive Hypermedia, as well as some by his colleagues on the same topic, e.g., the paper of Brusilovsky, called "Adaptive Educational Hypermedia: From generation to generation" [3], or the paper "Adaptive Authoring of Adaptive Hypermedia" [9]. He may have saved these papers by himself, or might have received them by e-mail, from a colleague working in the same field, or may have used his browser's bookmarks to mark their position on the Web.

In order for these retrieved resources to be relevant to the overall Adaptive Hypermedia course, two conditions have to be fulfilled:

- the domain concept in the course where each resource is most relevant has to be found (the *right information*)
- the resource has to be bound to that particular domain concept (in the *right place*).

This means that the first paper can be added in the course at a higher level, somewhere next to the explanation of generic principles of adaptive hypermedia, whereas the second paper should only be placed somewhere in connection with the authoring process in adaptive hypermedia, otherwise its content might be too specific to follow.

How can the system find the right resource and add it in the right place for Van Bos? The search can take place via keywords, labeling both course pieces created by Van Bos, as well as the papers and resources on his desktop (as shown in section 4).

Finally, Van Bos will use several versions of Beagle++ over the years to store his data. He certainly would like the same type of retrieval to work with all the versions of the schemas using during these years.

The following sections will describe in more detail how Dr. Van Bos can enrich his course semi-automatically, without much extra work, as well as keep at all times the overall control and overview.

3 Enriching Metadata

As we have seen in our scenario, both *automatically* generated metadata as well as *manually* added metadata are important. The automatically generated metadata allow description and retrieval of the appropriate articles. The manual annotation step allows

addition of additional content, as well as of attributes like pedagogical weights and labels, which are necessary to build the final adaptive course product. The following two sub-sections describe these two kinds of metadata in more detail.

3.1 Input Metadata Schema

Beagle [1] is a desktop search system implemented for Linux that indexes all documents on the user's desktop. *Beagle++* [2, 5] is an extension of Beagle that generates and stores additional metadata describing these documents, other resources, and their relationships. Additional metadata automatically annotate material the user has read, used, written or commented upon. Three obvious sources of desktop behavior information are: *files* on the desktop, *Internet* browsing and downloaded files, and *mail* exchanges and files stored from mail attachments [5, 13]. Figure 1 shows an instance of this ontology depicting files annotated with their publication metadata, file system attributes, web history, and mail context (e.g., files being attached to specific e-mails).

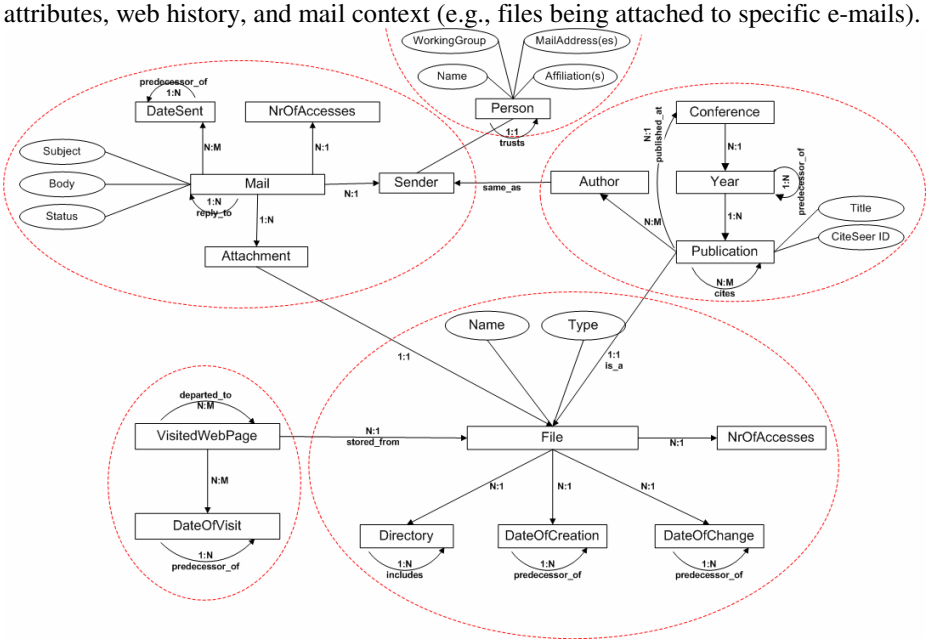

Fig. 1. RDF schema of metadata from Beagle++ [2]

In Figure 1, the upper left ellipse describes emails (with subject, body, sender and status attributes) and their attachments. The upper right ellipse defines publications written by several authors at different conferences (with title, publication year, etc.). The lower left ellipse shows web cache history representation (web pages visited and dates of visits). Finally, the lower right ellipse describes files that are saved on a person's desktop, with their name and specific directory. These files may have been saved via any of the other three processes (from emails, from websites, or from conferences), so an attachment entity and a file entity may refer to the same object.

3.2 Output Metadata Schema

These files and metadata are however not enough to generate a complete course. Specifically, information is needed about the hierarchical structure and order of the material in the context of a lesson, as well as additional pedagogical annotations describing which students the material is best suited for (e.g., *beginner* versus *advanced*). Figure 2 shows this target schema, as defined in MOT [8], an adaptive hypermedia authoring system. The schema describes two models used in the adaptive hypermedia authoring paradigm: the *domain map* (left side of Figure 2) and the *lesson* (right side).

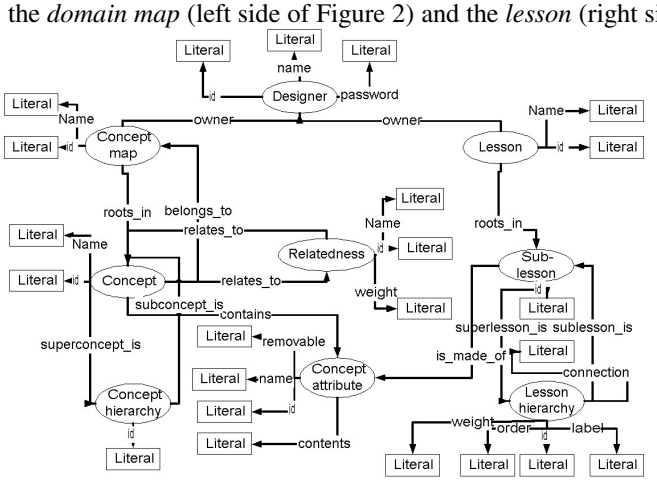

Fig. 2. RDF Schema of MOT [8]

A *domain map* is composed of a hierarchy of domain concepts. Each concept has attributes, containing or linking to e-learning content alternatives describing this concept. In addition to the hierarchy, concepts can also connect to related concepts.
A *lesson* is composed of a hierarchy of sub-lessons. The lesson represents a filtered, pedagogically labeled, weighted and ordered version of the concept attributes (see the relation between sub-lessons and concept attributes).

4 Transformation workflow

Beagle++ stores all metadata in the Sesame RDF database [18]. All Beagle++ components, which *generate metadata* (e.g., the email, publication, web cache and file metadata generators) add the metadata to this database. All Beagle++ components which *use metadata* (for example, the search and indexing module, the ranking module or the browsing module) retrieve their data from this repository, and, in some cases, write back new data (such as the PageRank value for documents or other resources).

It is easy to accommodate additional modules in this environment by writing appropriate interface components, which read and write from the repository. This is what we have done for *MOT* [8, 15]. In the scenario, we have described semi-automatic addition of articles stored on the user's desktop to a MOT lesson [8]. In MOT, this addition is done to an existing lesson. Based on pedagogic goals, the author can proc-

ess the data by changing labels and weights and adding other information on the article. After this enrichment, the lesson can be imported back into the RDF store. We use CAF (Common Adaptation Format [7], a system-independent XML exchange format) to simplify the transformation process from RDF to the MOT MySQL storage format.

4.1 RDF2CAF

In this step, we transform available articles into a CAF sub-lesson. This is a semi-automatic step, where the author selects a lesson in MOT and then the system searches (via keywords and title) for related articles in the Sesame data store. This is done by a Java utility that takes the metadata in the Sesame store and the lesson in CAF format as input and generates an updated lesson as CAF file. As articles are stored locally on the desktop, they have to be than physically transported to the supporting MOT server. The question is now where in the lesson the new article information is placed? As MOT is mainly a tool for authoring *educational* (adaptive) material, internal information structures are based on strict *hierarchies* (Figure 2). When enriching the domain-model and lesson, we of course want to get the *right information* in the *right place* in this hierarchy. To achieve this, the program first queries the Sesame database, using as search terms title and keywords of each domain concept found in the current lesson:

```
select x from x {p} y where y like "*keyword"
```

This query may result in one file being relevant in many places within the hierarchy. To ensure adding every resource only once, the place 'relevance' is computed:

$$rank\,(a,c) = \frac{card\,(k(c) \bigcap k(a))}{card\,(k(a))}\text{ ; where:}$$

rank(a,c) is the rank of article *a* with respect to the current domain concept *c*;
k(c) is the set of keywords belonging to the current domain concept *c*;
k(a) is the set of keywords belonging to the current article *a*;

If a resource is ranked equally for two domain concepts in the hierarchy, we add it to the topmost concept. The number of articles to be added per concept is limited to 3, as adding too many articles to one concept could result in confusing the learner.

4.2 CAF2MOT

The import of CAF files into MOT is implemented as a PHP script and done in two steps. First, the domain map is processed. During this process, an associative array is created, as lookup table. In it, every created concept attribute is stored as follows:

```
[name_of_domain_map\name_of_concept_1\...\
  name_of_concept_N\attribute_name] => attribute_id
```

This allows a faster lookup in the next step than via a direct query on the database. In the second step, the lesson (parts) are generated. While importing, conflicts between lesson or domain map *names* may occur. There are currently 3 ways of handling this:

- The author *ignores* it, allowing one name for multiple MOT domain maps /lessons.
- The author *renames* domain maps /lessons with conflicting names (after the import, from a list of conflicting names).

- The system *merges* (automatically) lessons and domain maps with the same name (handling them as extensions of each other). For domain maps, merging means that two domain maps with the same name are compared, concept by concept. Domain concepts and attributes that were not present before are added.

In the current implementation, a lesson is exported, then articles are added to the CAF file, and then the CAF file is imported back into MOT with the *merge* option. This is essential, as it allows additions made to lessons between export and import not to be deleted, so other lessons using one of the adapted domain maps are not affected.

4.3 Working in MOT

In MOT, data and metadata imported from Sesame can be manually reordered and edited. Figure 3 shows an extract of editing in the *lesson* environment (the import from Beagle++ generates extra information both for the *domain map* as well as for the *lesson*). The paper called 'Adaptive Educational Hypermedia: From generation to generation' and all its relevant metadata (title, authors, references, conference where presented) have been added automatically during extraction to the 'Adaptive Hypermedia' domain concept, at the highest level in the hierarchy. The paper 'Adaptive Authoring of Adaptive Educational Hypermedia' has been added to the domain concept 'Authoring of Adaptive Hypermedia', in a lower position in the hierarchy, as it is a more specific paper. The author can now manually add pedagogical labels to this new material, e.g., labeling the material 'adv' for advanced learners.

Fig. 3. Adding manual metadata in a MOT lesson

4.3 MOT2RDF

MOT2CAF

Importing MOT into CAF is also implemented as a PHP script. First, the lesson is constructed, based on the lesson hierarchy. During this process, a list of the mappings

between lesson objects and domain objects is kept. Next, the corresponding domain map is generated. Every domain map used in the lesson is added to the domain model.

CAF2RDF

For more flexibility, MOT lessons should also be exported as RDF; we are currently implementing this step. These RDF data could be re-used by Beagle++[2]. The RDF metadata schema is described in [9]. Since the CAF file is in XML format, we can use an XSLT style sheet and do an XSLT transformation to convert into an RDF file describing the lesson. The RDF MOT output then looks as shown in Figure 2. If we export the changed information to RDF so that it can again be used by Beagle++, the problem of duplicates arises again. Exact duplicates that overwrite existing information are obviously not a problem. However, information contradicting existing information is less desirable. A solution is to keep track of objects already present, and not export those. However, in this case, changes made to data in MOT [8] would not be reflected. Therefore, if an author manually corrects some error, this would be omitted in the target RDF. On the other hand, an author may make changes only valid in a particular context (e.g., for a given domain map), that do not apply to the resource in general. Therefore, automatic updating, as well as automatic discharging of new information are both undesirable. There are two possible solutions to this problem:

- *manual:* search for existence of resources to be exported and ask the author whether to overwrite the current resources or not.
- *automatic:* save information together with its context as a new information item.

5 Handling Flexible Schemas

The previous sections implicitly assumed fixed schemas. I.e., all schema elements, relationships and attributes are defined and fixed. In reality, however, our Sesame data store contains data based on different schemas, and different versions of these schemas, as metadata and metadata schemas continuously evolve on the Semantic Desktop. Although this does not create great problems in our transformation process (we store all schemas and schema version together with our metadata in RDF/RDFS format), it can lead to problems for metadata generation rules. E.g., rules that specifically refer to certain elements or attributes in a given schema are not viable (cannot be reused) if the schema evolves. The solution we propose in this paper is based on malleable schemas, which allow us to flexibly describe our metadata as well as employ imprecise matching over these schemas, to flexibly refer to the appropriate attributes.

5.1 Extended Malleable Schema

Malleable Schemas (introduced in [11]) are a new concept in the database and web communities. Although the problems with integration of information on the web have been recognized early (various schemas, non-structured data, etc.), solutions proposed often involve either individual *mediating or merging of two or more schemas* [4], or

mapping to a higher level schema [13, 18] or *ontology* [12]. More advanced solutions deal with *gradually evolving schemas* [23].

Malleable schemas provide a mechanism by which the modeler can capture the imprecise aspects of the domain during the modeling phase in a well principled fashion [11]. This is done based on keyword matching; however, these keywords are elements of the schema, instead of arbitrary keyword fields. Unlike earlier solutions, malleable schemas are an answer to more than one problem when integrating information on the web and especially on the desktop, including:

- multiple original schemas with varying semantics;
- evolution of schemas and semantics;
- need of only partial integration (as opposed to full integration): often, only a part of the existing schema is relevant for a particular application or user; integrating the whole schema can be both an arduous as well as a superfluous exercise.

Malleable schemas help in capturing the important (relevant) aspects of the domain at modeling time, without having to commit to a strict schema. More importantly, the vague parts of the schema can, in principle, evolve later towards more structure, or can just as well remain as they are.

The data model of malleable schemas as described in [11] is an object oriented one, which fits nicely the RDF/RDFS data model [17]. RDF/RDFS represents everything as triples <*subject, predicate, object*>; where subject is represented by *class* elements of the schema (or instances thereof); the predicate represents a *relationship* or an *attribute*, and object can be another class element, an instance thereof, or a value. Relationships have a *domain*, where the domain is a *set of classes*. Attributes have a *range*, specifying *allowed values* for this attribute. Malleable schemas are composed of similar elements as regular schemas. The major differences to conventional schemas are:

- classes and relationships (properties) do not have to have precise naming; the names can be keywords or phrases, as well as regular expressions; a distinction is made between precisely and imprecisely named elements.
- structure can vary; a distinction is made between strict structure and variable structure elements. The latter allow us to flexibly refer to classes in paths composed of more than one property.

Malleable schemas thus provide a simplified, unified view on a collection of related and overlapping base schemas. The relationships and overlaps of the base schemas are described by mappings. An example schema is presented in Figure 4.

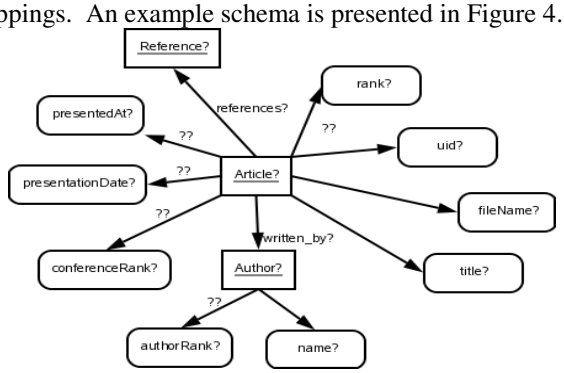

Fig. 4. RDF malleable schema of directory data.

This type of malleable schema represents the schema on which queries on several partially known (or evolving) schemas can be based, as discussed in sub-section 5.2. Figure 4 shows the schema on which queries between MOT and the Semantic Desktop data in Beagle++ are made.

Term impreciseness: Flexible class and attribute names

In Figure 4, impreciseness is denoted via *single question marks* '***?***'. For instance, the entity '*Article'* is imprecise (or malleable) and is written as '*Article?'*. In other words, a malleable schema allows impreciseness in the naming of classes. E.g., '*File'* can sometimes appear as '*Article'*, sometimes as '*Artikel'*. In the latter, the language has to be identified, whilst in the former, synonyms have to be searched for. Such an uncertainty about class names can appear, as listed above, when the same class has different names in two different schemas, but also, when the target schema that is queried is not known. This malleable feature of schemas is based on the ones proposed in [11]. In the following, the malleability definition in [11] is extended, based on the needs we have found in our transformation and conversion exercise between schemas.

Term impreciseness: Flexible relations or property names

Just as classes can be imprecise, properties of classes can also be only partially known. For instance, we can expect that the malleable class '*Author?'* can have a property '*name'*, but it could also be '*firstName'* and '*lastName'*, or be expressed in another form. Therefore, we depict this expectation by adding to our schema the malleable property '*name?'*. Again, the *question mark* '*?*'denotes the impreciseness.

Furthermore, composed words can appear, such as in '*presentedAt'* versus '*conference'*. In such a case, synonyms are not enough. First, decomposition into simple words has to be performed, and then, the expression identified.

Flexible paths

Beside the naming differences, schemas can also have different structures. A class can be connected to another class or property via a direct connection, or via an imprecise, indirect connection. In our example (Figure 4), the imprecise attribute '*presentedAt?*' can be directly or indirectly connected to the imprecise object '*Article?*'. This would correspond in Figure 1 to the relationship between the (not displayed) property '*Name'* of the class '*Conference'*, to the entity '*Publication'*. '*Name'* in Figure 1 is not a direct attribute of the class '*Publication'*. Therefore, in Figure 4, the property '*presentedAt?'* (equivalent to the above mentioned property '*Name'* in Figure 1), appears as an imprecisely linked property of the class '*Article?'* (equivalent to the above mentioned class '*Publication'* in Figure 1). In Figure 4, such indirect, imprecise connections (of undetermined length) are marked by *double question marks '**??**'*.

The above three types of impreciseness can be resolved in various ways. T*erm similarity* [11] ,e.g., can be resolved via Wordnet [24] or by other similarity measures. We can rely on extensive research regarding similarity measures between individual schemas (*instance similarity* [11], *structural similarity*: e.g., editing distance [25],

approximate nearest neighbor search [20], unordered tree matching [18], *schema-corpus similarity* [14]).

It is important to note that, for malleable schemas, complete resolution of the impreciseness and schema mapping is not desirable. The resolution is only interesting and important when specific queries are posed against the malleable schema. The elements of the malleable schema which do not appear in the queries do not need to be bound to concrete schema instances or schema alternatives. For malleable schemas, the resolution process of the impreciseness is an *iterative discovery process*.

5.2 Queries and Rules with malleable schema elements

The next issue is how the query process takes place in presence of malleable schemas, and how end-user (e.g., learner) adaptation (personalization) based on malleable schemas can be implemented. As hinted in [11], there are two main query targets: *instances* and *schemas*. Whilst querying based on regular schemas usually targets the instances, for malleable schemas, asking questions about the structure and composition of the schema itself can also be interesting, as it is also, to some degree, unknown.

Instance Querying

Instance querying is the type of querying usually performed: a schema instance is queried about the instances of elements that it contains. For instance, we want to search for articles written by certain authors. We base our instance query system on [11], expanding it, though, as we want to express the impreciseness of *terms* (classes or properties), as well as the impreciseness of *structure* in our queries.

In the following, we exemplify these three types of impreciseness in a query. Let us suppose we want to find an article written by author “Cristea”, which was sent to the current user by “Nejdl” via email.

First, in Example 1, we look at how we write this query if we knew the exact structure of the RDFS base schema (as depicted in Figure 1). As we are querying RDF data, the language used a simplified form of SeRQL from Sesame; moreover, this was the implementation language used for our transformation steps described in the first part of the paper.

Example 1 - Standard query:

```
SELECT Title FROM { bplus:Publication} art:title {bplus:Title},
     { bplus:Publication} art:authored_by { bplus:Author},
     { bplus:Publication} rdfs:subClassOf { bplus:File},
     { bplus:Attachment}  rdfs:subClassOf { bplus:File},
     {Mail} bplus: Attachment{ bplus:Attachment};
            bplus: email_reply_to { bplus:Sender},
     {Sender} bplus:name {“Nejdl”}
WHERE bplus:Publication = bplus:Attachment
      AND { bplus:Author} like "*#Cristea"
using namespace
 rdfs = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>,
 art = < http://www.win.tue.nl/~acristea/
             schemas/articleSchema.rdf#>
 bplus = <http://www.l3s.de/domain_l3s#>
```

The example returns the desired article(s), if the terminology, as well as the relations used in these query, are correct. We will now gradually relax this query towards a malleable query.

In Example 2, we rephrase this query, based *partially* on the malleable schema in Figure 4. In this case, we are not aware of the precise naming and terminology of the classes of the basis schemas; moreover, we cannot bind the terminology to a given namespace.

Example 2 - Term impreciseness - class impreciseness:

```
SELECT Title? FROM {Article?} title {Title?},
     {Article?} authored_by {Author?},
     {Mail?} Attachment {Article?};
           Sender {Sender?},
     {Sender?} name {"Nejdl"}
WHERE {Author?} like "*#Cristea"
```

As can be seen, Example 2 still assumes some knowledge of the correct terminology of properties, as well as knowledge about the structure, in order to return the desired article(s). Impreciseness is introduced via the imprecise term '*Article?'*, which can be mapped to several classes of the original schema(s): '*Publication'*, '*File'* and '*Attachment'*. As we can see, the impreciseness of terms has the side-effect that queries can be written in a more compact way. Explicit reference to equivalent classes is not necessary in the query.

The following relaxation allows for properties to be also imprecise (Example 3).

Example 3 - Term impreciseness - property impreciseness:

```
SELECT Title? FROM {Article?} title? {Title?},
     {Article?} written_by? {Author?},
     {Mail?} has_attached? {Article?};
           has_sender? {Sender?},
     {Sender?} name {"Nejdl"}
WHERE {Author?} like "*#Cristea"
```

In Example 3, properties such as *'written_by?'* are imprecise. This property will have to be connected during the resolution process with the property '*authored_by'* in Figure 1. This step allows all terms of the query to be vague. This corresponds to the situation where either the actual names of the classes, attributes and properties are forgotten, or they are unknown. The structure of the RDF schema, however, still has to be known.

Finally, we make another relaxation step, as depicted in Example 4a,b. Example 4a is based on the malleable schema in Figure 4, extended with mail information.

Example 4a: Relation impreciseness:

```
SELECT Article? FROM {Article?} written_by? {Author?};
                                  ??sent_by? {"Nejdl"},
                          {Author?} name? {"Cristea"}
```

The relation *'??sent_by?'* is both imprecise in *terminology* (it should be equivalent to *'email_reply_to'* in Figure 1); this is why it has a question mark on its right side: *'sent_by**?**'* ; as well as imprecise in *structure* (it actually represents a path between '*Attachment'* and '*Sender'* from Figure 1, i.e., '*Attachment**<**has_attached**-**Mail**-**email_reply_to**>**Sender'*; this is the reason why it also has two question marks on its left side: *'**??**sent_by'*).

The query can be relaxed even further, as in Example 4b. This corresponds to a simplified malleable schema, a reduced version of the schema from Figure 4. In Example 4b, the connection between '*Article*' and author name is imprecise as well.

Example 4b - Relation impreciseness:

```
SELECT Article? FROM {Article?} ??written_by? {"Cristea"};
                                ??sent_by? {"Nejdl"}
```

Notice how in queries 4a,b the query has become more compact, and almost like natural language. This resemblance is due to the fact that the malleable query is similar to the way people ask questions: they only remember partially terminology and relations, and some information about the structure. Moreover, the individual representations remembered by different people differ. Still, individuals are capable of communicating with each other despite these different models (schemas), which provides additional motivation to express uncertainties in the world around us through mechanisms like malleable schemas and malleable queries.

Schema Querying

Schema querying is only necessary when dealing with imprecise schemas [11]. As our malleable schemas are imprecise both in the *naming* of the elements (classes and properties) and in their *structure*, schema queries can serve to identify the actual naming and structure of the base schemas. This is useful in resolving the impreciseness of our malleable schema.

In the following, we exemplify the three types of impreciseness defined above, for a proposed schema query.

In our first example, we want to find the correct term for a *class* or an *attribute*. Here, we want to know how the class '*Article?'* is called in the base schema(s), which is connected via a '*written_by?'*-like property with a class similar to '*Author?'*.

Example 1a - Term impreciseness - class impreciseness:

```
SELECT Name(Article?)
FROM {Article?} written_by? {Author?}
```

For the base schemata in Figure 1, the answer would be '*Publication'*. If we would want to obtain also 'Attachment' and 'File', we would need to relax the triple conditions, as in Example 1b.

Example 1b - Term impreciseness - class impreciseness:

```
SELECT Name(Article?)
FROM {Article?} * {*}
```

In our second example, we want to find the correct term for a *property*. The property name we're looking for is '*written_by?'*, from the same triple as above.

Example 2 - Term impreciseness - property impreciseness:

```
SELECT Name(written_by?)
FROM {Article?} written_by? {Author?}
```

The query result is the property '*authored_by'* from the schema in Figure 1.

In our third and last example, we want to find the *correct path* between two objects (two classes, or a class and an attribute). We exemplify this by looking for the conference rank of a given article (Example 3).

Example 3 - Relation impreciseness:

```
SELECT Path(??conferenceRank?)
FROM {Article?} ??conferenceRank? {ConferenceRank?}
```

The result will retrieve the path between '*Publication'* and '*rank'*, i.e., *'published_at* **-** *Conference* **-** *has_a'* (from Figure 1).

The two types of queries (*instance* and *schema* query) defined and exemplified above can be performed automatically by search systems, in order to automatically enrich, in our current application, the authored learning material for the adaptive, personalized learning environment of the end-user, the student. The author is expected to use malleable schemas transparently; i.e., he can leave the processing to a query system, and concentrate only in the extra manual annotation and extension of the material.

6 Summary

The semantic desktop is a rich source of metadata about resources as well as relationships between these resources. While many of these metadata can be generated automatically, some additional metadata necessary for adaptive courses still have to be added manually. In this paper we have described how the semantic desktop, building on Semantic Web techniques, can be interfaced with a sophisticated adaptive hypermedia authoring environment. The semantic desktop system provides automatically generated metadata describing publications and other materials relevant for an advanced course. The MOT authoring system adds context and pedagogical information.

Moreover, we have looked at flexible ways to describe metadata in this environment. Conventional schemas are not suited for that purpose, as they cannot cope with changing or evolving schemas, which are very important for the semantic desktop (new tasks make new metadata attributes necessary, additional data sources with additional schemas will be added over time, etc.) The solution we are proposing relies on malleable schemas [11] to flexibly describe our metadata, and on fuzzy querying and matching, based on these schemas, to flexibly retrieve metadata.

In future work we will concentrate on automatic query expansion for malleable schema queries – we have already formulated appropriate expansion rules which we are currently validating and extending -, and on adaptive navigation rules relying on these query expansion mechanisms. Furthermore, we have started to investigate the use of malleable schema-based querying in a search context, and are working on integrating these algorithms into the next Beagle++ version.

References

1. Beagle, http://beaglewiki.org/Main_Page
2. Beagle++, http://www.kbs.uni-hannover.de/beagle++/wiki/pmwiki.php
3. Brusilovsky, P. (2004) Adaptive Educational Hypermedia: From generation to generation. Proceedings of 4th Hellenic Conference on Information and Communication Technologies in Education, Athens, Greece, September 29 - October 3, 2004, pp.19-33.
4. Buneman P., Davidson, S., and Kosky, A. Theoretical Aspects of Schema Merging, Proc. 3rd Int. Conf. Extending Database Technology, Vienna, Austria (March 1992) 152-167.
5. Chirita, P.-A., Costache, S., Nejdl, W., and Paiu, R. Beagle++: Semantically Enhanced Searching and Ranking on the Desktop. Proceedings of the 3rd European Semantic Web Conference, Budva, Montenegro (to appear 11-14 June 2006).

6. Chirita, P.-A., Gavriloaie, R., Ghita, S., Nejdl, W., and Paiu, R. Activity-Based Metadata for Semantic Desktop Search. Proceedings of the 2nd European Semantic Web Conference, Heraklion, Crete, May (2005).
7. Cristea, A.I., Smits, D., and De Bra, P. Writing MOT, Reading AHA! - converting between an authoring and a delivery system for adaptive educational hypermedia -, A3EH Workshop, AIED'05, Amsterdam, The Netherlands (July, 2005)
8. Cristea, A. I. & De Mooij, A. Adaptive Course Authoring: My Online Teacher. Proceedings of ICT'03, Papeete, French Polynesia (2003).
9. Cristea, A., De Mooij, A. LAOS: Layered WWW AHS Authoring Model and its corresponding Algebraic Operators. In Proceedings of WWW'03, Alternate Education track. (Budapest, Hungary 20-24 May 2003). ACM.
10. Cristea, A. I. and Aroyo, L. Adaptive Authoring of Adaptive Educational Hypermedia. AH 2002, Adaptive Hypermedia and Adaptive Web-Based Systems, LNCS 2347, Springer, (2002) 122-132.
11. Dong, X., and Halevy, A., Malleable Schemas, WebDB, Baltimore Maryland (June 16-17, 2005), ACM, webdb2005.uhasselt.be/papers/P-8.pdf.
12. Gardarin, G., and Dang-Ngoc, T-T. Mediating the Semantic Web. 4th journal on extraction and management of knowledge, Clermont Ferrand, Université Blaise Pascal (20-23 January 2004).
http://georges.gardarin.free.fr/Articles/MediatingtheSemanticWeb.pdf
13. Ghita, S., Nejdl, W., and Paiu, R., Semantically Rich Recommendations in Social Networks for Sharing, Exchanging and Ranking Semantic Context, International Semantic Web Conference, Galway, Ireland (6 November 2005) 6 – 10.
14. He, B., and Chang, C.-C., Statistical schema integration across the deep web. In Proc. of SIGMOD (2003).
15. MOT homepage, http://wwwis.win.tue.nl/~acristea/mot.html
16. Pottinger, R. A., Processing Queries and Merging Schemas in Support of Data Integration, PhD thesis, University of Washington, US (2004),
www.cs.ubc.ca/~rap/publications/thesis.pdf.
17. RDF, http://www.w3.org/RDF/
18. Schlieder, T., Naumann, F., Approximate tree embedding for quering cml data, ACM SIGIR Workshop on CML and Information Retrieval.
19. Sesame, http://www.openrdf.org/
20. Shasha, D., Wang, J., Shan, H., Zhang, K., Atreegrep: Approximate searching in unordered trees, in Proc. of International Conference on Scientific and Statistical DB Management (2002) 89-98.
21. Semantic Desktop, http://www.semanticdesktop.org/
22. Simon, B., Dolog, P., Miklós, Z., Olmedilla, D. and Sintek, M. Conceptualising Smart Spaces for Learning. *Journal of Interactive Media in Education*, 2004 (9). Special Issue on the Educational Semantic Web. ISSN: 1365-893X (2004). http://www-jime.open.ac.uk/2004/9
23. Velegrakis, Y., Miller, R.J., and Popa, L. Mapping Adaptation under Evolving Schemas, with Yannis Velegrakis and Renee J.. VLDB'03, Berlin, Germany, September (2003) 584-595. (full version in VLDB Journal)
http://www.almaden.ibm.com/cs/people/lucian/MappingAdaptation_VLDB03.pdf
24. Wordnet. http://www.cogsci.princeton.edu/wn/
25. Zhang, K., Wang, J., Shasha, D., On the editing distance between undirected acyclic graphs and related problems, in Proc. of International Symposium on Combinatorial pattern matching (1998).