Metadata Inference for Document Retrieval in a Distributed Repository *

Philippe Rigaux
LAMSADE
Univ. Paris-Dauphine
philippe.rigaux@dauphine.fr

Nicolas Spyratos
LRI
Univ. Paris-Sud Orsay, France
spyratos@lri.fr

Abstract. This paper describes a simple data model for the composition and metadata management of documents in a distributed setting. We assume that each document resides at the local repository of its provider, so all providers' repositories, collectively, can be thought of as a single database of documents spread over the network. Providers willing to share their documents with other providers in the network must register them with a coordinator, or *mediator*, and providers that search for documents matching their needs must address their queries to the mediator. The process of registering (or un-registering) a document, formulating a query to the mediator, or answering a query by the mediator, all rely on document content *annotation*.
Content annotation depends on the nature of the document: if the document is atomic then an annotation provided explicitely by the author is sufficient, whereas if the document is composite then the author annotation should be augmented by an *implied annotation*, i.e., an annotation inferred from the annotations of the document's components.
The main contributions of this paper are:

1. providing appropriate definitions of document annotations;
2. providing an algorithm for the automatic computation of implied annotations;
3. defining the main services that the mediator should support.

1 Introduction

In this paper, we propose a simple data model for the composition and metadata management of documents in a distributed setting where a community of *authors* co-operate in the creation of documents to be used also by other authors. Each author is a "provider" of documents to the network but also a "consumer", in the sense that he creates documents based not only on other documents that he himself has created but also on documents that other authors have created and are willing to share. We envisage several possible domain applications for this framework, and in particular *e-Learning systems* where instructors and learners create and share educational material [16, 19, 15]. In a nutshell, our approach can be described as follows.

* Research supported by the EU DELOS Network of Excellence in Digital Libraries and the EU IST Project (Self eLearning Networks), IST-2001-39045.

We distinguish documents into *atomic* and *composite*. Intuitively, an atomic document is any piece of material (text, image, sound, etc.) that can be identified uniquely; its nature and granularity are entirely up to its author. A composite document consists of a set of *parts*, i.e., a set of other documents that can be either atomic or composite. We assume that each document resides at the local repository of its author, so all authors' repositories, collectively, can be thought of as a database of documents spread over the network. Typically, an author wishing to create a new document will use some of the documents in his local database as components and will also search for relevant documents available over the network.

Authors willing to share their documents with other authors in the network must register them with a coordinator, or *mediator*, and authors that search for documents matching their needs must address their queries to the mediator. The process of registering (or un-registering) a document, formulating a query to the mediator, or answering a query by the mediator, all rely on document content *annotations*.

Such annotations are actually sets of terms from a controlled vocabulary, or *taxonomy*, to which all authors adhere. The well known ACM Computing Classification System [1] is an example of such a taxonomy. In this respect, we distinguish three kinds of annotation: the author annotation, the implied annotation and the registration annotation.

During registration of a document at the mediator, its author is required to submit the following items:

1. The document identifier, say d; this can be a URI allowing to access the document.
2. An annotation of the document content, that we call the *author annotation* of d; if d is atomic then the author annotation must be nonempty, whereas if d is composite then the author annotation can be empty.
3. If d is composite, then registration requires, additionally, the submission of all parts of d (i.e., all documents that constitute the document being registered); using the annotations of these parts, the mediator then computes *automatically* an annotation that "summarizes" the annotations of the parts, and that we call the *implied annotation* of d.

To register a document the mediator uses the author annotation augmented by the implied annotation, after removing all redundant terms (i.e., terms that are subsumed by other terms). The final set of terms used for registration is what we call the *registration annotation*.

The mediator is actually a software module that can be seen as one component of a digital library serving a number of subscribers (the "consumers"). A digital library maintains pointers to documents stored at the repositories of their authors. The mediator allows all subscribers to search for and consult documents of interest. Additionally, it allows authors to reuse documents of the library as components of new documents that they create. Of course, apart from the mediator services, a digital library offers a number of other services to its users, such as profiling, recommandations, personalization, and so on. However, in this paper, we focus only on the mediator services.

The mediator, actually, maintains a *catalogue* of registered documents: during registration of a document with identifier d, the mediator inserts in the catalogue a pair

(t, d), for each term t in the registration annotation of d. Authors searching for documents that match their needs address their queries to the mediator. In turn, the mediator uses the catalogue to answer such queries.

The main issues addressed in this paper are:

1. providing appropriate definitions of document annotations;
2. providing an algorithm for the computation of implied annotations;
3. defining the main services that the mediator should support.

This paper proposes *generic* solutions to the above issues, i.e., solutions that are valid independently of questions concerning network configuration. In other words, the solutions that we provide are still valid whether the network is configured around a central mediator, or whether it is organized in clusters, each cluster being served by a separate mediator, or even whether there is no mediator but each node plays the role of a mediator for all its connected nodes (as in pure peer-to-peer network [11]).

Work in progress aims at:

1. validating our model in the context of a prototype, in which the documents are XML documents;
2. embedding our model in the RDF Suite [2];
3. integrating our annotation generating algorithms into a change propagation module to be integrated in the mediator.

We stress the fact that, in this paper, we do *not* deal with the management of document content, but only with the management of document annotations based on subject areas. We are aware that, apart from subject area, there are several other dimensions of content description such as the format of the document, its date of creation, its author, the language in which the document content is written (if there is text involved), and so on. However, in this paper, we focus only on the subject area dimension, and when we talk of annotation we actually mean subject area description.

We note that a lot of efforts have been devoted recently to develop languages and tools to generate, store and query metadata. Some of the most noticeable achievements are the RDF language [20], RDF schemas [21], query languages for large RDF databases [14, 2] and tools to produce RDF descriptions from documents [13, 5].

Several metadata standards exist today, such as the Dublin Core [9], or the IEEE Learning Object Metadata [18]. It seems quite difficult to produce them automatically. In this paper, we focus on taxonomy-based annotations to describe the content of documents [4]. Generation of such annotations remains essentially a manual process, possibly aided by acquisition software (see for instance [13, 22, 5], and [10] for a discussion). The fully automatic generation of metadata is hardly addressed in the literature, with few exceptions [17, 25, 8]. A representative work is the Semtag system described in [8] which "tags" web pages with terms from a standard ontology, thanks to text analysis techniques. This is different – and essentially complementary – to our approach, which relies on the structure of composite documents to infer new annotations. Finally the functionalities presented in Section 4 can be seen as an extension of well-known mediation techniques [26, 6, 3, 23], wih specific features pertaining to the management of structured information.

In summary, the novel aspects of our work concern the creation, automatic annotation, management and querying of composite documents distributed over an information network. We are not aware of other approaches in the literature that provide a similar formal framework for handling these functionalities.

In the rest of this paper we first describe the representation of documents (Section 2) and their annotation (Section 3) and then the functionalities supported by the mediator (Section 4).

2 The Representation of a Document

As mentioned earlier, in our model, a document is represented by an identifier together with a set of parts showing how the document is constructed from other, simpler documents. We do not consider the document content itself, but focus only on its representation by an identifier and a set of parts, as this is sufficient for our metadata management and mediation purposes. Therefore, hereafter, when we talk of a document we shall actually mean its representation by an identifier and a set of parts.

In order to define a document formally, we assume the existence of a countably infinite set $\mathcal{D}$ whose elements are used by all authors for identifying the created documents. For example, the set $\mathcal{D}$ could be the set of all URIs. In fact, we assume that the creation of a document is tantamount to choosing a (new) element from $\mathcal{D}$ and associating it with a set of other documents that we call its parts.

Definition 1 (The Representation of a document) *A document consists of an identifier d together with a (possibly empty) set of documents, called the* parts *of d and denoted as $parts(d)$. If $parts(d) = \emptyset$ then d is called* atomic*, else it is called* composite.

For notational convenience, we shall often write $d = d_1 + d_2 \ldots + d_n$ to stand for $parts(d) = \{d_1, d_2, \ldots, d_n\}$. Based on the concept of part, we can now define the concept of component.

Definition 2 (Components of a document) *Let $d = d_1 + d_2 \ldots + d_n$. The set of components of d, denoted as $comp(d)$, is defined recursively as follows:*
if *d is atomic* ***then*** $comp(d) = \emptyset$
else $comp(d) = parts(d) \cup comp(d_1) \cup comp(d_2) \cup \ldots \cup comp(d_n)$.

In this paper, we assume that a document d and its associated set of components can be represented as a rooted directed acyclic graph (*dag*) with d as the only root. We shall refer to this graph as the *composition graph* of d. Clearly, the choice of parts of a composite document and their arrangement to form a composition graph should be left entirely up to its author. The composition graph of an atomic document consists of just one node, the document identifier itself. We note that the absence of cycles in the composition graph simply reflects the reasonable assumption that a document cannot be a component of itself. Clearly, this does not prevent a document from being a component of two or more distinct documents belonging to the same composition graph, or to different composition graphs.

It is important to note that in our model the ordering of parts in a composite document is ignored because it is not relevant to our purposes. Indeed, as we shall see shortly, deriving the annotation of a composite document from the annotations of its parts does not depend on any ordering of the parts.

3 Annotations of documents

As we mentioned in the introduction, document content annotations are built based on a controlled vocabulary, or *taxonomy*, to which all authors adhere. A taxonomy consists of a set of terms together with a subsumption relation between terms. An example of a taxonomy is the well known ACM Computing Classifi cation System [1].

Definition 3 (Taxonomy) *A taxonomy is a pair* $(T, \preceq)$ *where* T *is a* terminology*, i.e., a finite and non-empty set of names, or* terms*, and* $\preceq$ *is a reflexive and transitive relation over* T*, called* subsumption*.*

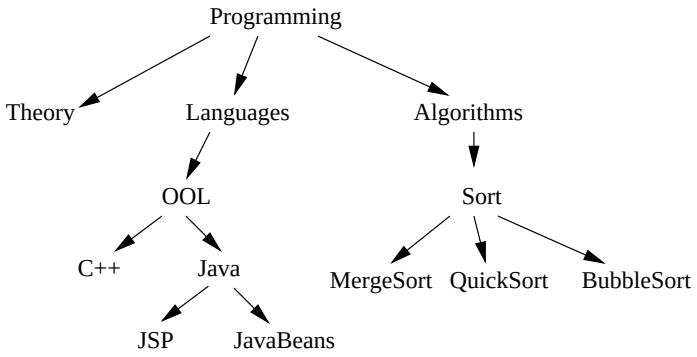

Fig. 1. A taxonomy

If $s \preceq t$ then we say that s is *subsumed* by t, or that t *subsumes* s. A taxonomy is usually represented as a graph, where the nodes are the terms and there is an arrow from term s to term t iff s subsumes t. Figure 1 shows an example of a taxonomy, in which the term `Languages` subsumes the term `OOL`, the term `Java` subsumes the term `JavaBeans`, and so on. We note that the subsumption relation is *not* antisymmetric, i.e., $(s \preceq t)$ and $(t \preceq s)$ does not necessarily imply $s = t$. Therefore, we defi ne two terms s and t to be *synonyms* iff $s \preceq t$ and $t \preceq s$. However, in this paper, we shall not consider synonyms. From a technical point of view, this means that we work with classes of synonym terms, rather than individual terms. Put it differently, we work with just one representative from each class of synonyms. For example, referring to Figure 1, the term `OOL` is the representative of a class of synonyms in which one can also fi nd terms such as `Object-Oriented Languages`, `O-O Languages`, and so on, that are synonyms of `OOL`.

However, even if we work only with classes of synonyms, a taxonomy is not necessarily a tree. Nevertheless, most taxonomies used in practice (including the ACM Computing Classification System mentioned earlier) are in fact trees. In this paper, we shall assume that the taxonomy used by all authors to describe the contents of their documents is in fact a tree. We shall simply refer to this tree-taxonomy as "the taxonomy".

Now, in order to make a document sharable, an annotation of its content must be provided, so that users can judge whether the document in question matches their needs. We define such an annotation to be just a set of terms from the taxonomy. For example, if the document contains the quick sort algorithm written in java then we can choose the terms `QuickSort` and `Java` to describe its content. In this case the set of terms {`QuickSort, Java`} can be used as an annotation of the document.

Definition 4 (Annotation) *Given a taxonomy $(T, \preceq)$ we call* annotation *in T any set of terms from T.*

However, a problem arises with annotations: an annotation can be redundant if some of the terms it contains are subsumed by other terms. For example, the annotation {`QuickSort, Java, Sort`} is redundant, as `QuickSort` is subsumed by `Sort`. If we remove either `Sort` or `QuickSort` then we obtain a non-redundant annotation: either {`QuickSort, Java`} or {`Sort, Java`}, respectively. As we shall see later, redundant annotations are undesirable as they can lead to redundant computations during query evaluation. We shall therefore limit our attention to non-redundant, or *reduced annotations*, defined as follows:

Definition 5 (Reduced Annotation) *An annotation A in T is called* reduced *if for any terms s and t in A, $s \not\prec t$ and $t \not\prec s$.*

Following the above definition one can reduce an annotation in (at least) two ways: removing all but the minimal terms, or removing all but the maximal terms. In this paper we adopt the first approach, i.e., we reduce an annotation by removing all but its minimal terms. The reason for our choice lies in the fact that by removing all but minimal terms we obtain a more accurate annotation. This should be clear from our previous example, where the annotation {`QuickSort, Java`} is more accurate than {`Sort, Java`}.

Definition 6 (Reduction) *Given an annotation A in T we call* reduction *of A, denoted $reduce(A)$, the set of minimal terms in A with respect to the subsumption $\preceq$.*

An annotation can be seen both as a summary of the document content and as a support to find and retrieve the document. In the case of an atomic document the annotation can be provided either by the author or by the system via a semi-automatic analysis of the document content [12]. In the case of a composite document, though, apart from the author annotation, we would like to derive to derive also a second annotation, *automatically*, from the annotations of the document parts. We shall refer to such a derived annotation as the *implied annotation* of the composite document. To get a feeling of the kind of implied annotation that we have in mind, let us see an example.

Example 1 *Let $d = d_1 + d_2$ be a composite document with the following annotations of its parts:*

$$A_1 = \{\texttt{QuickSort}, \texttt{Java}\}$$
$$A_2 = \{\texttt{BubbleSort}, \texttt{C++}\}$$

Then the implied annotation of $d = d_1 + d_2$ will be {`Sort, OOL`*}, that summarizes what the two parts have in common.*

We shall come back to this example after the formal definition of implied annotation.

Now, the question is: how can one define the implied annotation of a composite document so as to best reflect the contents of its parts. Roughly speaking, what we propose in this paper is that the implied annotation should satisfy the following criteria:

- it should be reduced, for the reasons explained earlier;
- it should summarize what the parts have in common;
- it should be minimal.

To illustrate points 2 and 3 above, suppose that a composite document has two parts with annotations {`QuickSort`} and {`BubbleSort`}. The term `Sort` is a good candidate for being the implied annotation, as it describes what the two parts have in common. Moreover, as we can see in Figure 1, `Sort` is the minimal term with these properties. On the other hand, the term `Algorithms` is not a good candidate because, although it summarizes what the two parts have in common, it is not minimal (as it subsumes the term `Sort`).

Coming back to Example 1, following the above intuitions, we would like the implied annotation of d to be {`Sort,OOL`}. Indeed,

- {`Sort,OOL`} is a reduced annotation;
- the term `Sort` summarizes what `QuickSort` and `BubbleSort` have in common, and `OOL` summarizes what `Java` and `C++` have in common;
- it is minimal, as any other annotation with the above properties will have terms subsuming either `Sort` or `OOL`.

In order to formalize these intuitions, we introduce the following relation on annotations.

Definition 7 (Refinement Relation on Annotations) *Let A and A' be two annotations. We say that A is* finer *than A', denoted $A \sqsubseteq A'$, iff for each $t' \in A'$, there exists $t \in A$ such that $t \preceq t'$.*

In other words, A is finer than A' if every term of A' subsumes some term of A. For example, the annotation $A =$ {`QuickSort, Java,BubbleSort`} is finer than $A' =$ {`Sort,OOL`}, whereas A' is not finer than A. To gain some more insight into this ordering, let us see another example. Referring to Figure 1, consider the following reduced annotations:

– $A = \{$`JSP,QuickSort,BubbleSort`$\}$
– $A' = \{$`Java,Sort`$\}$

Then $A \sqsubseteq A'$, as each term t' of A' subsumes some term t of A. Indeed, `Java` subsumes `JSP` and `Sort` subsumes `QuickSort` (of course, `Sort` also subsumes `BubbleSort`, but the existence of one term in A subsumed by `Sort` is sufficient).

Note that, according to this ordering, once we have verified that $A \sqsubseteq A'$ we may add to A as many extra terms as we wish *without* destroying the ordering.

Clearly, $\sqsubseteq$ is a reflexive and transitive relation, thus a pre-ordering over the set of all descripions. However, $\sqsubseteq$ is *not* antisymmetric, as the following example shows. Consider $A_1 =$ `{OOL, Java, Sort}` and $A_2 =$ `{Java, Sort, Algorithms}`. It is easy to see that $A_1 \sqsubseteq A_2$ and $A_2 \sqsubseteq A_1$, although $A_1 \neq A_2$. However, as we have explained earlier, for the purposes of this paper, we restrict our attention to reduced annotations only; and, as stated in the following proposition, for reduced annotations, the relation $\sqsubseteq$ becomes also antisymmetric, thus a partial order.

Proposition 1 *The relation $\sqsubseteq$ is a partial order over the set of all reduced annotations.*

Proof. Indeed, assume $A \sqsubseteq A'$ and $A' \sqsubseteq A$, and consider a term t' of A'. Then there is a term t in A such that $t \preceq t'$. We claim that $t' \preceq t$ as well, and therefore that $t = t'$. Otherwise, as $A' \sqsubseteq A$ and t is in A, there is a term t'' (different than t') such that $t'' \preceq t$, and thus $t'' \preceq t'$. Assuming $t'' \neq t'$, we have a contradiction to the fact that A' is a reduced annotation. □

Now, using this ordering, we can define formally the implied annotation of a composite document so as to satisfy the criteria for a "good" implied desription, given earlier. First, we need the following result:

Theorem 2 *Let $\mathcal{A} = \{A_1, .., A_n\}$ be any set of reduced annotations. Let $\mathcal{U}$ be the set of all reduced annotations S such that $A_i \sqsubseteq S, i = 1, 2, ..., n$, i.e., $\mathcal{U} = \{S | A_i \sqsubseteq S, i = 1, \ldots, n\}$. Then $\mathcal{U}$ has a unique minimal element, that we shall denote as $lub(\mathcal{A}, \sqsubseteq)$.*

Proof. Let $P = A_1 \times A_2 \times \ldots \times A_n$ be the cartesian product of the annotations in $\mathcal{A}$, and suppose that there are k tuples in this product, say $P = \{L_1, L_2, ..., L_k\}$.

Let $A = \{lub_{\preceq}(L_1), lub_{\preceq}(L_2), \ldots, lub_{\preceq}(L_k)\}$, where $lub_{\preceq}(L_i)$ denotes the least upper bound of the terms in L_i, with respect to $\preceq$. As $(T, \preceq)$ is a tree, this least upper bound exists, for all $i = 1, 2, ..., n$. Now, let R be the reduction of A, i.e., $R = reduce(A)$. We shall show that R is the smallest element of $\mathcal{U}$.

Indeed, it follows from the definition of R that $A_i \sqsubseteq R$, for $i = 1, 2, ..., n$. Moreover, let S be any annotation in $\mathcal{U}$, and let t be a term in S. It follows from the definition of $\mathcal{U}$ that there is a term v_i in each annotation A_i such that $v_i \preceq t$. Consider now the tuple $v =< v_1, v_2, ..., v_n >$. By the definition of least upper bound, $lub_{\preceq}(v) \preceq t$, and as $lub_{\preceq}(v)$ is in R, it follows that $R \sqsubseteq S$, and this completes the proof. □

With this theorem at hand, we can now define the annotation implied by a set of annotations $\mathcal{A} = \{A_1, \ldots, A_n\}$.

Definition 8 (Implied Annotation) *Let $\mathcal{A} = \{A_1, .., A_n\}$ be a set of annotations in T. We call* implied annotation *of $\mathcal{A}$, denoted $IAnnot(\mathcal{A})$, the least upper bound of $\mathcal{A}$ in $\sqsubseteq$, i.e., $IAnnot(\mathcal{A}) = lub(\mathcal{A}, \sqsubseteq)$*

Theorem 2 suggests the following algorithm for the computation of the implied annotation. Its proof of correctness follows directly from Theorem 2.

Algorithm IANNOT
Input: A set of annotations $A_1, A_2, \ldots, A_n$
Output: The implied annotation
begin
  Compute $P = A_1 \times A_2 \times \ldots \times A_n$
  for each tuple $L_k = [t_1^k, t_2^k, \ldots, t_n^k]$ in P,
    compute $T_k = lub_{\preceq}(t_1^k, t_2^k, \ldots, t_n^k)$
  Let $Aux = \{T_1, \ldots, T_l\}$
  return $reduce(Aux)$
end

In the algorithm IANNOT, the function $lub_{\preceq}(t_1^k, \ldots, t_n^k)$ returns the least upper bound of the set of terms $t_1^k, \ldots, t_n^k$ with respect to $\preceq$. In Section 4 we shall use this algorithm to compute automatically the implied annotation of a composite document, based on the annotations of its parts. More precisely, given a composite document $d = d_1, + \ldots + d_n$, the annotations $A_1, \ldots, A_n$ in the above algorithm will be those of the parts $d_1, \ldots, d_n$, respectively, and the implied annotation will then be the implied annotation of d.

We end this section by working out a few examples illustrating how this algorithm works (always referring to the taxonomy of Figure 1).

Example 2 *Consider the document $d = d_1 + d_2$, composed of two parts with the following annotations:*

$$A_1 = \{\texttt{QuickSort, Java}\}$$
$$A_2 = \{\texttt{BubbleSort, C++}\}$$

In order to compute the implied annotation, first we compute the cross-product $P = A_1 \times A_2$. We find the following set of tuples:

$$P = \begin{cases} L_1 = \texttt{<QuickSort, BubbleSort>} \\ L_2 = \texttt{<QuickSort, C++>} \\ L_3 = \texttt{<Java, BubbleSort>} \\ L_4 = \texttt{<Java, C++>} \end{cases}$$

Next, for each tuple L_i, $i = 1, \ldots, 4$, we compute the least upper bound T_i of the set of terms in L_i:

1. $T_1 = \texttt{Sort}$
2. $T_2 = \texttt{Programming}$
3. $T_3 = \texttt{Programming}$
4. $T_4 = \texttt{OOL}$

We then collect together these least upper bounds to form the set Aux:

$$Aux = \{\texttt{Sort, Programming, OOL}\}$$

Finally we reduce Aux to obtain the implied annotation:

$$\textit{Implied Annotation} = \{\texttt{Sort}, \texttt{OOL}\}$$

In view of our discussions so far, this result can be interpreted as follows: each part of the document concerns both, sorting and object-oriented languages.

Example 3 *Consider now the composite document $d' = d_1 + d_3$, with the following annotations of its parts:*

$$A_1 = \{\texttt{Java}, \texttt{QuickSort}\}$$
$$A_3 = \{\texttt{BubbleSort}\}$$

Proceeding similarly, as in Example 2, we find successively:

1. *The cross-product:*

$$P = \begin{cases} L_1 = <\texttt{QuickSort, BubbleSort}> \\ L_2 = <\texttt{Java, BubbleSort}> \end{cases}$$

2. *The least upper bounds: $Aux = \{\texttt{Sort}, \texttt{Programming}\}$*
3. *The implied annotation: $reduce(Aux) = \{\texttt{Sort}\}$*

The following comments are noteworthy:

1. The term `Java` is *not* reflected in the implied annotation of Example 3, as it is not something that both parts share.
2. The fact that `Java` has disappeared in the implied annotation means no loss of information: if a user searches for documents related to java, then d_1 will be in the answer and d' will not, which is consistent.
3. If we had put `Java` in the implied annotation of d', this would give rise to the following problem: when one searches for documents related to java, the system will return both d_1 and d'. Clearly, this answer is redundant (because d_1 is part of d'), and also somehow irrelevant as only a part of d' concerns java.

Finally we note that the same document will generate different implied annotations, depending on what its "companion" parts are in a composite document. This is illustrated by our last example.

Example 4 *Consider the composite document $d'' = d_1 + d_4$, with the following annotations of its parts:*

$$A_1 = \{\texttt{Java}, \texttt{QuickSort}\} \quad A_4 = \{\texttt{C++}\}$$

Proceeding similarly, as in Example 2, we find successively:

1. *The cross-product:*

$$P = \begin{cases} L_1 = <\texttt{Java}, \texttt{C++}> \\ L_2 = <\texttt{QuickSort}, \texttt{C++}> \end{cases}$$

2. $Aux = \{\texttt{OOL}, \texttt{Programming}\}$
3. $reduce(Aux) = \{\texttt{OOL}\}$

Note that, in the two previous examples, d_1 is part of a composite document, but each time with a different "companion part": first with d_3 in d', then with d_4 in d''. It is interesting to note that, depending on the companion part, either the "Sort-aspect" of d_1 or the "OOL-aspect" appears in the implied annotation.

4 The Mediator

As we mentioned in the introduction, we consider that a community of authors cooperate in the creation of documents to be used by other authors. Each author is a "provider" of documents to the community but also a "consumer", in the sense that he creates documents based not only on other documents that he himself has created but also on documents that other authors have created and are willing to share. Each document resides at the local repository of its author, so all authors' repositories, collectively, can be thought of as a database of documents spread over the network. Typically, an author wishing to create a new document will use as components some of the documents from his local database, and will also search for relevant documents that reside at the local databases of other authors – provided that those other authors are willing to share them.

Authors willing to share their documents with other authors in the network must register them with a coordinator, or *mediator*, and authors that search for documents matching their needs must address their queries to the mediator. The mediator is actually a software module supporting the sharing of documents. It provides a set of services, among which the following basic services:

- query evaluation
- registration of a document
- un-registration of a document
- annotation modification

In this section, we discuss these basic services and outline their interconnections.

The implementation of all the above services relies on document annotations that are provided to the mediator during document registration. Indeed, during registration of a document, its author is required to submit the document identifier, say d, and an annotation of d that we call the *author annotation* of d, denoted as $AAnnot(d)$. If d is atomic then the author annotation must be nonempty, whereas if d is composite the author annotation can be empty. However, if d is composite the author is also required to submit all parts of d. Based on the annotations of the parts, the mediator then computes (automatically) the implied annotation of d. Finally, to register d, the mediator uses the author annotation augmented by the implied annotation, after removing all redundant terms. The final set of terms used for registration is what we call the *registration annotation* of d.

Definition 9 (Registration Annotation) *The Registration Annotation of a document $d = d_1 + \ldots + d_n$, denoted $RAnnot(d)$, is defined recursively as follows:*

- *if d is atomic, then $RAnnot(d) = reduce(AAnnot(d))$*
- *else $RAnnot(d) = reduce(AAnnot(d) \cup IAnnot(RAnnot(d_1), \ldots, RAnnot(d_n)))$*

One may wonder why the author annotation is not sufficient for document registration, and why we need to augment it by the implied annotation. The answer is that the author of a composite document d may not describe the parts of d in the same way as the authors of these parts have done. Let us see an example. Suppose that two documents, d_1 and d_2, have been created by two different authors, with the following author annotations:

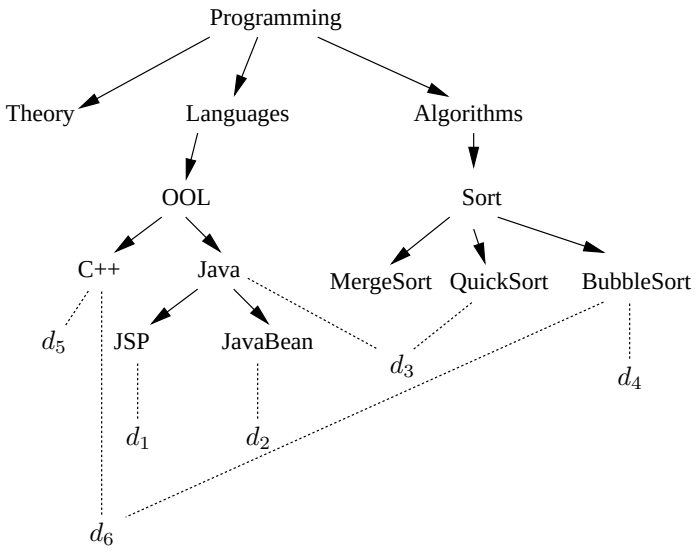

Fig. 2. A catalogue

$$AAnnot(d_1) = \{\texttt{QuickSort,Java}\}$$
$$AAnnot(d_2) = \{\texttt{BubbleSort,C++}\}$$

Assume now that, after browsing their content, a third author considers d_1 and d_2 as examples of good programming style, and decides to use them as parts of a new, composite document $d = d_1 + d_2$. Consequently, the author of d provides the following author annotation:

$$AAnnot(d) = \{\texttt{GoodProgrammingStyle}\}$$

Although this author annotation might be accurate for the author's own purposes, the document d still can serve to teach (or learn) java and sorting algorithms. This information will certainly be of interest to users searching for documents containing material on java and sorting algorithms. Therefore, before registration, the author annotation should be completed, or augmented by the implied annotation of d, i.e., {`OOL,Sort`}, to obtain the following registration annotation:

$$\{\texttt{GoodProgrammingStyle,OOL,Sort}\}$$

This annotation expresses *all* annotations, i.e., the one given by the author of d *and* those given by the authors of the parts of d.

During document registration, the registration annotation of d is what is actually stored by the mediator in a repository. Conceptually, the repository can be thought of as a set of pairs constructed as follows: during registration of a document d, the mediator stores a pair (t, d) for each term t appearing in the registration annotation of d. The set of all such pairs (t, d), for all documents that are (currently) registered is what we call the *Catalogue*.

Definition 10 (Catalogue) *A* catalogue $\mathcal{C}$ *over* $(T, \preceq)$ *is a set of pairs* (t, d)*, where* t *is a term of* T *and* d *is a document identifier.*

Figure 2 shows a catalogue over the taxonomy of Figure 1. The dotted lines indicate the pairs (t, o) of the catalogue, relating terms with documents. Roughly speaking, the catalogue is a "shopping list" in which users look for documents that match their needs. As such, the catalogue is the main conceptual tool for "integrating" all documents repositories. In what follows, we discuss in more detail how the mediator uses the catalogue to support the basic services listed earlier.

Query Language

In our model, a query is either a single term or a boolean combination of terms, as stated in the following definition.

Definition 11 (Query Language) *A* query *over the catalogue is any string derived by the following grammar, where* t *is a term:*

$$q ::= t | q \wedge q' | q \vee q' | q \wedge \neg q' | (q)$$

Roughly speaking, the answer to a query is computed as follows. If the query is a single term, then the answer is the set of all documents related either to t or to a term subsumed by t. If the query is not a single term then we proceed as follows. First, for each term appearing in the query, replace the term by the set of all documents computed as explained above; then replace each boolean combinator appearing in the query by the corresponding set-theoretic operator; finally, perform the set-theoretic operations to find the answer. These intuitions are reflected in the following definition of answer, where the symbol $tail(t)$ stands for the set of all terms in the taxonomy strictly subsumed by t, i.e., $tail(t) = \{s \backslash s \preceq t\}$.

Definition 12 (Query Answer) *The* answer *to a query* q *over a catalogue* $\mathcal{C}$*, denoted by* $ans(q)$*, is a set of documents defined as follows, depending on the form of* q *(refer to Definition 11):*

Case 1*:* q *is a single term* t *from* T*, i.e.,* $q = t$

$ans(t)$ = ***if*** $tail(t) = \emptyset$
then $\{d \backslash (t, d) \in \mathcal{C}\}$
else $\bigcup\{ans(s) | s \in tail(t)\}$

Case 2*:* q *is a general query*

$ans(q)$ =
if $q = t$ ***then*** $ans(t)$
else
begin
if $q = q_1 \wedge q_2$, $ans(q) = ans(q_1) \cap ans(q_2)$
if $q = q_1 \vee q_2$, $ans(q) = ans(q_1) \cup ans(q_2)$
if $q = q_1 \wedge \neg q_2$, $ans(q) = ans(q_1) \backslash ans(q_2)$
end

Example 5 *Consider the query* q = `C++` $\vee$ `Sort`. *Referring to Figure 2 and applying the above definition, we find* $ans(q) = \{d_5, d_6\} \cup \{d_3, d_4, d_6\} = \{d_3, d_4, d_5, d_6\}$. *Similarly, for the query* q = `C++` $\wedge\neg$ `BubbleSort` *we find* $ans(q) = \{d_5\}$.

Registration

An author wishes to make a document available to other users in the network.

To make a document available to other users in the network, its author must submit the following three items to the mediator:

1. the document identifier, say d;
2. the author annotation of d, which must be nonempty if d is atomic;
3. the identifiers of the parts of d, if d is composite.

The mediator then computes the registration annotation of d on which the actual registration will be performed. To do this, the mediator uses the following algorithm, whose correctness is an immediate consequence of Definition 9. The algorithm takes as input the above items, and updates the catalogue (i.e., the old catalogue is augmented by a set of pairs (t, d), one for each term t in the registration annotation of d).

Algorithm REGISTRATION
Input: A document d, the author annotation $AAnnot(d)$,
the parts $\{d_1, d_2, \ldots, d_n\}$ of d
begin
$\mathcal{A} = \emptyset$
for each $d_i \in parts(d)$ **do**
if d_i is registered **then**
$R_i = \{t | (t, d_i) \in \mathcal{C}\}$
else
Input the author annotation $AAnnot(d_i)$
R_i = *Registration* (d_i, $AAnnot(d_i)$, $parts(d_i)$)
endif
$\mathcal{A} = \mathcal{A} \cup \{R_i\}$
end for
Let R = $reduce(IAnnot(\mathcal{A}) \cup AAnnot(d))$
for each t in R, insert the pair (t, d) in the catalogue
end

Note that, if d is atomic then its registration annotation reduces to the reduction of its author annotation (because $parts(d) = \emptyset$, and thus $IAnnot(\emptyset) = \emptyset$). From a practical point of view, the following scenarios can be envisaged for providing the inputs to the algorithm REGISTRATION; they depend on the nature of the parts of d, as well as on whether these parts have been registered beforehand or not:

– if a part d_i of d has already been registered then its registration annotation is taken from the catalogue, independently of whether d_i is atomic or composite.

– else if d_i is composite and not yet registered, then its registration annotation is recursively computed from the registration annotations of the parts of d_i;
– else if d_i is atomic then its author annotation is required as input, and its registration annotation is the reduction of its author annotation.

We assume that a document d, whether atomic or composite, can be registered only if its registration annotation is nonempty. This assumption is justified by the fact that search for documents of interest is based on annotations. As a consequence, if we allow registration of a document with an empty annotation, then such a document would be inaccessible. Therefore, the mediator needs at least one term t, in order to insert the pair (t, o) in the catalogue, and make d accessible. This is ensured by the following constraint.

Constraint 1 (Registration) *A document can be registered only if its registration annotation is nonempty*

For atomic documents this is tantamount to requiring that the author decription be nonempty.

Constraint 2 (Atomic documents registration) *An atomic document can be registered only if its author annotation is nonempty*

If the document is composite, then Constraint 1 implies that either the author annotation must be nonempty or the implied annotation must be nonempty. A sufficient condition for the implied annotation to be nonempty (and thus for Constraint 1 to be satisfied) is that *all* parts of the document be registered, or (reasoning recursively) that all atomic components of the document be registered. Indeed, if all atomic components have already been registered, then the mediator will be able to compute a nonempty implied annotation, and thus a nonempty registration annotation, independently on whether the author annotations of one or more components are missing. Therefore the following sufficient condition for the registration of a composite document:

Constraint 3 (Composite Document Registration)
If every atomic component of a composite document is registered then the document can be registered

Figure 3 shows an example of composite document registration. As shown in the figure, two atomic documents, d_3 and d_4 have already been registered in the catalogue $\mathcal{C}$, and so has the composite document d_2, whose parts are d_3 and d_4. The author annotations of all three documents are shown in the figure. Although the author annotation of d_2 is empty, its registration was possible as both its parts have nonempty author annotations. Note that the registration annotations of d_3 and d_4 coincide with their author annotations (since both documents are atomic and their author annotations happen to be reduced). The registration annotation of d_2 is easily seen to be $\{\texttt{OOL}\}$.

Now, suppose that an author wishes to reuse d_2 (and its parts) in order to create a new document d, composed of two parts: d_1 and d_2, where d_1 is an atomic document from the author's local database. Suppose now that in order to register d, the author

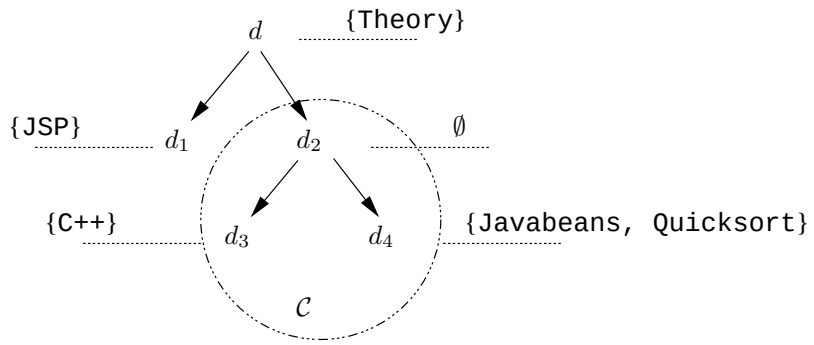

Fig. 3. Registration annotation of a composite document

provides to the mediator author annotations for d and d_1, as shown in the figure. Based on the author annotations of d and d_1, and the registration annotation of d_2 (computed above), the mediator will then compute the registration annotation of d – which is easily seen to be {OOL,Theory}. Finally, the mediator will enter in the catalogue the two pairs (OOL, d) and (Theory, d).

Unregistration:

An author wishes to remove from the catalogue one of his registered documents

To unregister a document, its author must submit to the mediator the document identifier, say d. The mediator then performs the following tasks:

- notify all users using d as a component in their composite documents;
- remove from the catalogue each pair (t, d);
- re-compute the registration annotations of all composite documents affected by the removal of d;
- use the re-computed registration annotations to maintain the catalogue.

We note that notification can be done either by broadcasting the removal of d to all users, or by first finding the users concerned and then notifying only those concerned. The first solution is probably cheaper but may create inconvenience to those users not concerned, whereas the second avoids inconvenience but requires searching the catalogue for finding the users concerned (assuming that the mediator keeps track of the "foreign documents" used by each user). In any case, once notified of the (pending) unregistration of d, the users concerned have the option of first creating (in their local database) a copy of d and then proceeding to re-register all composite documents in which d appears as a component. Otherwise, the registration annotation of such documents might become empty.

Annotation modification:

An author wishes to modify the annotation of one of his registered documents

To modify the annotation of a document, its author must submit to the mediator the document identifier, say d, and the new author annotation, say A. The mediator then performs the following tasks:

- notify all users using d as a component in some of their composite documents;
- remove from the catalogue each pair (t, d);
- using the new author annotation A, compute the new registration annotation $RIannot(d)$ from the catalogue;
- re-compute the registration annotations of all composite documents affected by the modification;
- use the re-computed registration annotations to maintain the catalogue.

As in the case of un-registration, notification can be done either by broadcasting the modification in the annotation of d to all users or by first finding the users concerned and then notifying only those concerned. However, now, there is no need for any action on the part of the user: all modified annotations can be obtained by querying the catalogue.

5 Concluding Remarks

We have presented a model for composing documents from other simpler documents and we have seen an algorithm for computing implied annotations of composite documents based on the annotations of their parts.

In our model, a document is represented by an identifier together with a composition graph which shows how the document is composed from other, simpler documents. The annotation of each document is a set of terms from the taxonomy. We have distinguished three kinds of annotation:

1. the author annotation, i.e., the annotation provided to the mediator explicitly by the author;
2. the implied annotation, i.e., the annotation implied by the annotations of the parts (and computed by the mediator automatically during registration);
3. the registration annotation, i.e., the annotation computed from the previous two annotations and used by the mediator to register the document.

We have also outlined the main functionalities of the mediator, a software module that acts as a central server, registering or unregistering sharable documents, notifying users of changes, maintaining the catalogue and answering queries by authors.

Work in progress aims at:

1. validating our model in the context of a prototype, in which the documents are XML documents;
2. embedding our model in the RDF Suite [2];

3. integrating our annotation generating algorithms into a change propagation module to be integrated in the mediator.

The basic assumption underlying our work is the existence of a network-wide taxonomy according to which documents are described and queries are formulated. As a result, the local repositories can be seen as peers served by a central catalogue (a "super-peer").

Future work aims at relaxing this assumption, in order to arrive at a pure peer-to-peer network. This will be done in two steps, as follows.

First, we will assume each author, or group of authors to have their own (possibly non-standard) taxonomy, for describing their documents locally and for formulating their queries to the mediator. This will require the establishment of *articulations*, i.e., semantic mappings between each local taxonomy and the central taxonomy. Work in that direction will be based on previous work on mediation [23, 24].

Second, we will assume that the role of the mediator can be played by any local taxonomy. Indeed, in principle, any local taxonomy can play the role of a mediator for all other local taxonomies that are articulated to it.

Another line of future research concerns a personalized interaction with the network. Indeed, from a conceptual point of view, all one has to do is to let the network user express his needs in terms of a set of named queries, or *views* of the form:

```
<term-name> = <query-to-the mediator>
```

The set of terms thus declared (plus, eventually, a user-defined subsumption relation) will then constitute the user-defined taxonomy, that will serve as the *personalized* interface to the network. Queries to this personalized taxonomy can be answered by simple substitution, based on the user declarations defining the terms of the personalized taxonomy. Work on the personalization aspects is ongoing and will be reported later.

References

1. The ACM Computing Classification System. ACM, 1999. http://www.acm.org/class/.
2. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *Proc. Intl. Conf. on Semantic Web*, 2001.
3. J. Ambite, N. Ashish, G. Barish, C. Knoblock, S. Minton, P. Modi, I. Muslea, A. Philpot, and S. Tejada. ARIADNE: a System for Constructing Mediators for Internet Sources. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 561–563, 1998.
4. R. Baeza-Yates and B. Ribeiro-Neto, editors. *Modern Information Retrieval*. Addison-Wesley, 1999.
5. F. Ciravegna, A. Dingli, D. Petrelli, and Y. Wilks. User-System Cooperation in Document Annotation based on Information Extraction. In V. R. B. A. Gomez-Perez, editor, *Proc. of the Intl. Conf. on Knowledge Engineering and Knowledge Management (EKAW02)*, Lecture Notes in Artificial Intelligence 2473, Springer Verlag, 2002.
6. S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your Mediators need Data Conversion. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 1998.

7. S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The Semantic Web: The Roles of XML and RDF. *IEEE Expert*, 15(3), 2000.
8. S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, and A. Tomkins. SemTag and seeker: bootstrapping the semantic web via automated semantic annotation. In *Proc. Intl. World Wide Web Conference (WWW)*, pages 178–186, 2003.
9. Dublin Core Metadata Element Set. Technical Report, 1999. http://dublincore.org/.
10. M. Erdmann, A. Maedche, H. Schnurr, and S. Staab. From Manual to Semi-automatic Semantic Annotation: About Ontology-based Semantic Annotation Tools. In *Proc. COLING Intl. Workshop on Semantic Annotation and Intelligent Context*, 2000.
11. H. Garcia-Molia. Peer-to-peer Data Management. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, 2002.
12. S. Handschuh, S. Staab, and R. Volz. On deep annotation. In *Proc. Intl. World Wide Web Conference (WWW)*, pages 431–438, 2003.
13. J. Kahan and M. Koivunen. Annotea: an Open RDF Infrastructure for Shared Web Annotations. In *Proc. Intl. World Wide Web Conference (WWW)*, pages 623–632, 2001.
14. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. In *Proc. Intl. World Wide Web Conference (WWW)*, pages 623–632, 2002.
15. K. Keenoy, G. Papamarkos, A. Poulovassilis, D. Peterson, and G. Loizou. Self e-Learning Networks – Functionality, User Requirements and Exploitation Scenarios. Technical report, SeLeNe Consortium, 2003. www.dcs.bbk.ac.uk/selene/.
16. B. Kieslinger, B. Simon, G. Vrabic, G. Neumann, J. Quemada, N. Henze, S. Gunnersdottir, S. Brantner, T. Kuechler, W. Siberski, and W. Nejdl. ELENA Creating a Smart Space for Learning. In *Proc. Intl. Semantic Web Conference*, volume 2342 of *LNCS*. Springer Verlag, 2002.
17. E. D. Liddy, E. Allen, S. Harwell, S. Corieri, O. Yilmazel, N. E. Ozgencil, A. Diekema, N. McCracken, J. Silverstein, and S. Sutton. Automatic Metadata Generation and Evaluation. In *Proc. ACM Symp. on Information Retrieval*, Tempere, Finland, 2002. Poster session.
18. Draft Standard for Learning Objects Metadata. IEEE, 2002.
19. W. Nejdl, B. Worlf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. EDUTELLA: a P2P networking Infrastruture Based on RDF. In *Proc. Intl. World Wide Web Conference (WWW)*, page 604:615, 2002.
20. Resource Description Framework Model and Syntax Specification. World Wide Web Consortium, 1999.
21. Resource Description Framework Schema (RDF/S). World Wide Web Consortium, 2000.
22. S. Staab, A. Maedche, and S. Handschuh. An Annotation Framework for the Semantic Web. In *Proc. Intl. Workshop on Multimedia annotation*, 2001.
23. Y. Tzitzikas, N. Spyratos, and P. Constantopoulos. Mediators over Ontology-based Information Sources. In *Proc. Intl. Conf. on Web Information Systems Engineering (WISE'01)*, 2001.
24. Y. Tzitzikas, N. Spyratos, and P. Constantopoulos. Query Evaluation for Mediators over Web Catalogs. In *Proc. Intl. Conf. on Information and Communication Technologies and Programming*, Primorsko, Bulgaria, 2002.
25. J. Wang and F. Lochovsky. Data extraction and label assignment for web databases. In *Proc. Intl. World Wide Web Conference (WWW)*, pages 187–196, 2003.
26. G. Wiederhold. Mediators in the Architecture or Future Information Systems. *IEEE Computer*, 25, 1992.