Metadata Management and Materialization of Composite Documents in a Digital Library

Anh Tuan Ly[1*], Nicolas Spyratos[2]

[1] Faculty of Computer Science and Engineering, ThuyLoi University, Hanoi, Vietnam.
[2] Laboratoire de Recherche en Informatique, Université Paris Sud, Orsay, France.

* Corresponding author. Tel.: (084) 947537447; email: tuanla@tlu.edu.vn
Manuscript submitted October 25, 2016; accepted April 10, 2017.
doi: 10.17706/ijcce.2017.6.2.91-103

Abstract: In this paper we present a method for the materialization of a (virtual) composite document that is the creation of a paper version of the composite document including a table of contents and an index. First, we introduce a simple model for the creation of composite documents from other, simpler documents and for the metadata management during the creation process; then we present algorithms for generating the table of contents and the index of a composite document to be used for the document's materialization.

Key words: Composite document, digital library, materialization, metadata.

1. Introduction

A Digital Library (DL) is a networked infrastructure supporting the creation and distribution of services over digital content. In order to realize such a vision, suitable models of the information embodied in the digital content, as well as of the interaction between a DL and its users, must be developed.

Our approach assumes that a DL serves a network of providers (such as museums, archives or other cultural or educational institutions), willing to share their documents with other providers and/or consumers (collectively called users). Each document is seen as a complex multimedia object whose components reside in the local repositories of their providers. Therefore all providers' repositories, collectively, can be seen as a database of documents spread over the network. The DL acts as a mediator, indexing all shareable documents so that users can access them transparently.

For a document to become shareable, its author must register it in the library, by providing a description of the document, called the *registration description*. The description of a document is a set of terms from a controlled vocabulary, or *taxonomy*, to which all authors adhere. An example of a taxonomy is the well-known ACM Computing Classification System [1].

From an interaction point of view, a DL supports the following basic operations:

Document *Access*: A user accesses documents transparently, through the library, in order to satisfy an information need (such as learning about a specific topic), or edits documents and reuses them in creating new documents. Accessing documents is done by querying the library.

Document Creation: A user creates a new document either from scratch (such a document is called *atomic*), or by editing and re-using existing documents assembled as a new document (such documents are called *composite*).

Moreover, the DL also provides a number of services to support document access and document

composition. These services include metadata management; document registration, removal and modification; and profile based customization (such as notification, recommendation, ranking of query answers, context based search, document materialization etc.)

In this paper, we focus on metadata management and materialization of composite documents.

Firstly, we define a data model for metadata management based on the one proposed in [2]. The model in [2] mainly describes the metadata model and syndicators, as well as an infrastructure using metadata for sharing information objects. It assumes that a composite document has a tree structure, in which each interior node is a composite document and each leaf node is an atomic document. Moreover, each node (whether interior or leaf) is associated with a description of its content, such that the description of an interior node is automatically inferred from the descriptions of its components while the description of a leaf node is created by its author. In our model, while using the *inferred description* of [2], we also allow the document author to augment it based on a suggestion by the system.

Secondly, we propose definitions and algorithms to manage materializations of a composite document [3]. Indeed, a composite document is a *virtual document* in the sense that we can access its component documents by clicking the nodes representing the components. In contrast, what we call a *materialization* of a (composite) document is the arrangement of the document's components in a linear order (defined by the user), together with the addition of a table of contents (TOC) and an index (as in a traditional book). Materializations have three important characteristics compared to the documents they materialize:

- In general, a document may have one or more materializations (in fact, as many as there are different ways of arranging the document's components in a linear order)

- Each materialization of a document can serve to produce a paper version of the document (i.e. a book in traditional form, including a TOC and an index)

- Each materialization of a document at different points in time might produce different paper versions of the document, as the document's components might have been changed by their authors during the time elapsed. Therefore a composite document can be seen as a *live document* and materialization can be seen as the dynamic process producing *instances* of the live document at different points in time.

In the rest of this paper, after a brief survey of related work (Section 2), we present the metadata management model for (virtual) documents and their descriptions (Section 3); then we describe the materialization process of a composite document, by proposing definitions and algorithms to generate the TOC and the index (Section 4); and finally, we present some conclusions and suggestions for future work (Section 5).

2. Related Work

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 [4], RDF schemas [5], the SPARQL query language for RDF [6], efficient RDF Stores and SPARQL query processors [7]–[10] and tools to produce RDF descriptions from documents [11], [12].

Wide adoption of metadata standards and common vocabularies like Dublin Core [13], FOAF [14], and schema.org brings hope for automating data integration tasks (also reasoning, decision support, etc.) at a new level. However, if one considers the full set of metadata that these standards propose to attach to a document, it seems indeed quite difficult to produce them automatically. Generation of metadata still remains mostly a manual process, possibly aided by acquisition software [12], [15], [16]. Some recent efforts for automatically generating metadata mainly focus on text analysis techniques [17]–[19] and metadata propagation to infer metadata of derived contents from those of the original contents [20], [21].

Recently, there have been open textbook systems developed to support the process of management and

materialization of composite documents. For example, the Connexions project [22] funded by Rice university intends to provide a platform allowing textbook authors, educators and students to create and customize *textbooks*. In the Connexions' repository, every textbook is managed as a collection of individual learning objects called *modules*. To make a textbook by composing modules of existing textbooks, authors need to find appropriate modules from textbook repositories. Through Connexions' website, users can read textbooks, customize textbooks by removing and adding modules, and create new textbooks by composing modules taken from existing textbooks. Although such systems are operating effectively, they have some lacks in the ability to automatically synthesize metadata of textbooks from the metadata of textbooks and modules at the lower levels based on a taxonomy, as well as the ability to produce the printable version of a textbook based on its synthesized metadata.

In this paper, we focus only on *semantic metadata*, i.e., the part of metadata which describes the content of the document [23]. We refer to this part as the document's *description* (or *annotation*). Our approach relies on the structure of composite documents to infer new descriptions. The work in [2] which is the basis of our study also proposes a metadata inference model for composite documents. However, the inference model of [2] is mainly intended for document repository management.

This paper is an extension of our work presented in [3], [24]. In the model we introduce here, we use inferred descriptions as well but we also let the document author augment the inferred description based on a suggestion by the system. Moreover, we introduce the concept of document materialization and propose tools for supporting this concept in a DL.

3. Model of Composite Documents and Descriptions

3.1. The Representation of a Document

Our model doesn't consider the actual content of a document. It deals only with the structure and the description of a document. Therefore we view a document as a pair consisting of an identifier (for example a URI) and a set of parts. As we shall see in the following this view is sufficient for metadata management.

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

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

Definition 2 (Components of a document). Let $d = d_1 + d_2 + ... + d_n$. The set of components of d, denoted as $comp(d)$, is defined recursively as follows:

- if d is atomic then $comp(d) = \varnothing$
- else $comp(d) = parts(d) \cup comp(d_1) \cup comp(d_2) \cup ... \cup comp(d_n)$.

We assume that every composite document d is a tree in which d is the root and $comp(d)$ is the set of nodes. This choice is because (1) the tree is the most suitable structure for representing traditional books that are hierarchically organized, and (2) the tree is also a common structure adopted by many existing document composition environments. Based on this assumption, given any (composite) document d and a part $d' \in parts(d)$, d' is called a *child* of d, and d is called the *parent* of d', denoted as $parent(d')$. Note that in our model the ordering of parts in a composite document is ignored.

3.2. Taxonomy and Description

Informally, descriptions in our model are just sets of terms taken from a controlled vocabulary, or

taxonomy. A taxonomy consists of a set of terms together with a subsumption relation between terms.

Definition 3 (Taxonomy). Let T be a set of terms, or *keywords*. A *taxonomy* $\mathcal{T}$ over T is defined to be a pair $(T, \preceq)$ where $\preceq$ is a reflective and transitive binary relation over T, called *subsumption relation*.

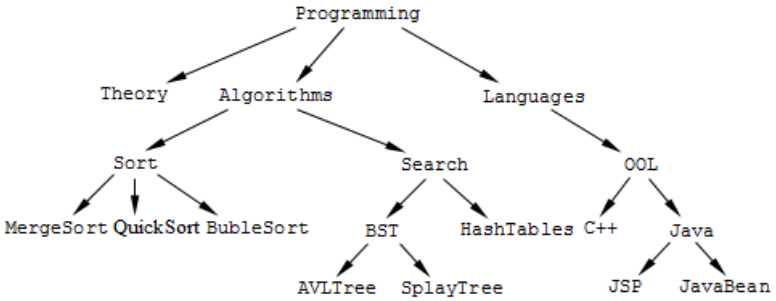


Fig. 1. A taxonomy.

Given two terms, s and t, if $s \preceq t$ then we say that s is *subsumed* by t, or that t *subsumes* s. We assumes that every taxonomy is a tree in which the nodes are the terms and there is an arrow from term s to term t iff s subsumes t. Fig. 1 shows an example of a taxonomy, in which the term *Algorithms* subsumes *Sort* and *Search*, *OOL* subsumes *Java* and $C++$, and so on. Due to the transitivity of the subsumption relation, the term *Programming* subsumes all terms in the tree including itself.

In order to make a document sharable, a description of its content must be provided, so that users can judge whether the document in question matches their needs. We define such a description to be just a set of terms from the taxonomy.

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

A description can be redundant if some of the terms it contains are subsumed by other terms. For example, the description $\{QuickSort, Java, Sort\}$ is redundant, as *QuickSort* is subsumed by *Sort*. Redundant descriptions are undesirable as they can lead to redundant computations during query evaluation. We shall therefore limit our attention to non-redundant, or *reduced descriptions*, defined as follows:

Definition 5 (Reduced description). Given a taxonomy $(T, \preceq)$, a set of terms D from T is called *reduced* if for any terms s and t in D, $s \not\prec t$ and $t \not\prec s$.

Following the above definition, we can make non-redundant descriptions by removing all but the minimal terms, or removing all but the maximal terms. We adopt the first approach because it produces more accurate descriptions. This should be clear from our previous example, where the description $\{QuickSort, Java\}$ is more accurate than $\{Sort, Java\}$.

Definition 6 (Reduction). Given a description D in taxonomy $(T, \preceq)$, we call *reduction* of D, denoted as $reduce(D)$, the set of minimal terms in D with respect to the subsumption $\preceq$.

A description can be seen both as a summary of the document's content and as a support to find and retrieve the document. The description of an atomic document can be provided either by the author or by the system via a semi-automatic analysis of the document content [16]. Otherwise, the description of a composite document can be derived automatically from the descriptions of the document parts.

We shall refer to such a derived description as the *implied composite description*. To get a feeling of the kind of implied description that we have in mind, let us see an example.

Example 1. Let $d = d_1 + d_2$ be a document with the following descriptions of its parts:

$D_1 = \{QuickSort, Java\}$

$D_2 = \{AVLTree, C++\}$

Then the implied description of $d = d_1 + d_2$ will be $\{Algorithms, OOL\}$, that summarizes what the two parts have in common.

The reason that $\{Algorithms, OOL\}$ is chosen as the implied description of d is because it satisfies the following criteria:

- $\{Algorithms, OOL\}$ is a reduced description;
- the term *Algorithms* summarizes what *QuickSort* and *AVLTree* have in common, and *OOL* summarizes what *Java* and $C++$ have in common;
- it is minimal, as any other description with the above properties will have terms subsuming *Algorithms* or *OOL*.

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

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

In other words, D is finer than D' if every term of D' subsumes some term of D. For example, the description $D = \{QuickSort, Java, AVLTree\}$ is finer than $D' = \{Algorithms, OOL\}$, whereas D' is not finer than D.

Clearly, $\sqsubseteq$ is a reflexive and transitive relation. However, over reduced descriptions, $\sqsubseteq$ becomes antisymmetric as well. So, we can say that $\sqsubseteq$ is a partial order over reduced descriptions, and a set of reduced description has a least upper bound in $\sqsubseteq$. For detailed discussion and proofs of them, see [2].

Theorem 1. *Let* $D = \{D_1, \ldots, D_n\}$ *be any set of reduced descriptions. Let* $\mathcal{U}$ *be the set of all reduced descriptions* S *such that* $D_i \sqsubseteq S$, $i = 1, \ldots, n$, *i.e.,* $\mathcal{U} = \{S \mid D_i \sqsubseteq S,\ i = 1, \ldots, n\}$. *Then* $\mathcal{U}$ *has a least upper bound, that we* shall *denote as* $lub(D, \sqsubseteq)$.

The least upper bound (*lub*) of a set of descriptions is the most accurate set of terms representing what the descriptions in the set have in common. Therefore, by obtaining the *lub* of descriptions of a set of documents, we can get the most accurate description that summarizes what all documents have in common. By using this theorem, we can now define the implied description of a set of descriptions as follows:

Definition 8 (Implied description). Let $D = \{D_1, \ldots, D_n\}$ be a set of descriptions in T. We call *implied description* of D, denoted $IDescr(D)$, the least upper bound of D in $\sqsubseteq$, i.e., $IDescr(D) = lub(D, \sqsubseteq)$.

We can use the following algorithm for the computation of the implied description. Its proof of correctness follows directly from Theorem 1.

Algorithm 1 IDESCR

Input: A set of descriptions $D_1, D_2, \ldots, D_n$

Output: The implied description

1: Computer $P = D_1 \times D_2 \times \ldots \times D_n$

2: for all $L_k = [t_1^k, t_2^k, \ldots, t_n^k] \in P$ do

3: computer $T_k = lub_{\preceq}(t_1^k, t_2^k, \ldots, t_n^k)$

4: Let $Aux = \{T_1, T_2, \ldots, T_l\}$

5: return $reduce(Aux)$

In the algorithm, the function $lub_{\preceq}(t_1^k, t_2^k, ..., t_n^k)$ returns the least upper bound of the set of terms $t_1^k, t_2^k, ..., t_n^k$ with respect to $\preceq$. The algorithm can be used to automatically compute the implied description of a composite document, based on the descriptions of its parts. To illustrate how this algorithm works, let us see the following example:

Example 2. Consider the document $d = d_1 + d_2$, composed of two parts with the following descriptions (referring to Fig. 1):

$$D_1 = \{QuickSort, Java\}$$

$$D_2 = \{AVLTree, C++\}$$

In order to compute the implied description, first we compute the cross-product $P = D_1 \times D_2$. We find the following set of tuples:

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

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

1. $T_1 = Algorithms$
2. $T_2 = Programming$
3. $T_3 = Programming$
4. $T_4 = OOL$

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

$$Aux = \{Algorithms, Programming, OOL\}$$

Finally we reduce Aux to obtain the implied description:

$$ImpliedDescription = \{Algorithms, OOL\}$$

This result can be interpreted as follows: each part of the document concerns both, algorithms and object oriented languages.

Implied descriptions can be used for term suggestion that allows users to easily define registration descriptions when registering their documents to a DL. If the document is atomic then the author will decide its registration description by selecting one or more appropriate terms from the taxonomy. The registration description is easily seen to be the reduced *author description* of the atomic document. If the document is composite, then the system can assist the user to define the registration description by first computing automatically the implied description and then using it to suggest terms that the author might want to use for the registration description. In default mode the implied description will be the registration description, as in [2]. Yet another possibility is that the system presents to the author the implied description and the author adds extra terms of his liking to create the registration description.

Definition 9 (Registration description). *The registration description* of a document $d = d_1 + ... + d_n$, denoted $RDesc(d)$, is defined as follows:

$RDesc(d) = reduce(ADesc(d))$, where $ADesc(d)$ is *the author description* that can be chosen based on

the implied description $IDescr(d)$. The implied description $IDescr(d)$ is defined recursively as follows:

- if d is atomic, then $IDescr(d) = \varnothing$
- else $IDescr(d) = IDescr(RDescr(d_1),..., RDescr(d_n))$

When a composite document is created, its components are documents selected among those available in the DL and/or documents created by the author. If a component is available in the library then it already has a registration description. If the component is created by the author then the author has to decide what its registration description is (possibly with the aid of the system as described above).

Consequently, for creating a new composite document, the user should follow four steps:

i. create the structure of the document from documents existing in the system or newly created by the author (by specifying a parent-child relationship);

ii. add a description to each node of the composite document possibly based on suggestions by the system;

iii. register the composite document to the DL; and possibly,

iv. materialize the composite document at will (i.e. produce a "paper version" of it).

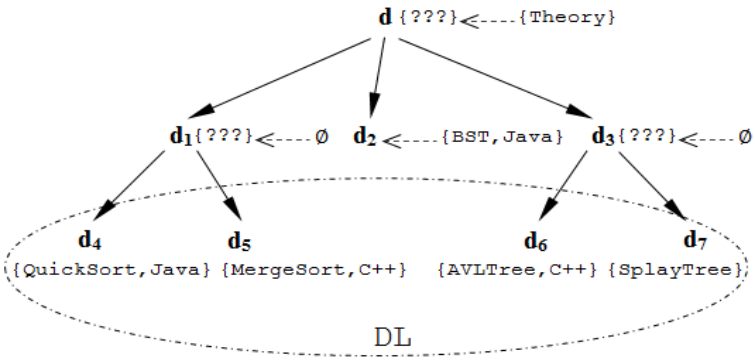


Fig. 2. Registration description of a composite document.

Fig. 2 shows an example of providing the registration description for a composite document when registering it to a DL (referring to the taxonomy of Fig. 1). As shown in the figure, four atomic documents, d_4, d_5, d_6 and d_7 have been registered in the DL. The author descriptions of all four atomic documents are also shown in the figure. Note that the registration descriptions of all four atomic documents coincide with their author descriptions (since all four documents are atomic and their author descriptions happen to be reduced). The composite document d_1 has two parts are d_4 and d_5, its implied description is easily seen to be $\{Sort, OOL\}$. Similarly, d_3 has two parts are d_6 and d_7, and its implied description is easily seen to be $\{BST\}$. Suppose that the author decides to choose default mode (i.e. not to modify the implied descriptions) when registering d_1 and d_3 to the DL. In this case, the registration descriptions of d_1 and d_3 will coincide with their implied descriptions.

Now, suppose that an author wishes to reuse both d_1 and d_3 (and their parts) in order to create a new composite document, composed of three parts d_1, d_2 and d_3, where d_2 is an atomic document from the author's local database. Suppose now that in order to register d, the author provides the author description for d_2, as shown in the figure. Based on the author description of d_2 and the registration description of d_1 and d_3 (computed above), the system will compute the implied description of d, which is easily seen to be $\{Algorithm\}$, and use it for term suggestion. Suppose that the author decides to

choose and augment it by the term *Theory*. The registration description of d will then be $\{Algorithm, Theory\}$.

In the following section, we will focus on the materialization process of a composite document.

4. Materialization of Composite Documents

4.1. Document Materialization and Related Concepts

After it has been created, a composite document will consist of a set of nodes and their descriptions structured in a hierarchy. These nodes are actually identifiers (for example, URIs) that refer to the identified resources. Materialization simply puts the contents that can be accessed through the nodes in a sequence (i.e. in a linear order). This can be done easily by showing to the user all the nodes, level by level in the hierarchy, and asking the user to mark (for each level) the desired linear order of nodes.

We note that, in general, there are several different *linearizations* of the nodes, therefore there might be several different materializations of the same composite document.

The objective of materialization is to produce a usual document, i.e. a version of the document that is printable (in the form of a book). To facilitate the assembly of all node contents in the form of a unique book, we assume that all node contents are in the same pre-defined format.

Another important issue of document materialization is that we must associate with it a TOC and an index at the time the document is materialized. In our context, the *TOC* of a composite document is a data structure in tabular format that lists all node descriptions, and for each node description, its location relatively to other nodes of the composite document.

Definition 10 (Table of contents). *The table of contents (TOC)* of a composite document is defined as follows:

1/ let $descr_1, ..., descr_n$ be the set of node descriptions in the whole tree, in which each $descr_i$ associates with the node dd_i to create the *i–the* line of the TOC.

2/ the set of all lines thus created is the TOC.

Similarly, in our context, the *index* of a composite document is a data structure in tabular format that lists all terms existing in the whole composite document written in alphabetical order. Each term is followed by the list of nodes of the composite document in whose descriptions it appears.

Definition 11 (Index). *The index* of a composite document is a table defined in three steps as follows:

1/ let $k_1, ..., k_m$ be the set of terms each of which appears in one or more node descriptions.

2/ associate each k_i with the list of nodes in whose description k_i appears, to create the *i–the* line of the index.

3/ the set of all lines thus created is the index.

In what follows, we describe algorithms to generate the TOC and the index of a composite document at the time of its materialization.

4.2. Generation of the TOC and the Index

As we have seen, a materialization of a composite document consists of the tree structure of the document together with a linearization of its nodes as specified by the user. Fig. 3 shows the tree structure of the composite document of Fig. 2 after it has been registered in the DL. Each node of the tree is associated with a description that is a set of terms taken from the taxonomy of Fig. 1. Note that the descriptions of different nodes can have terms in common because: 1) the user is allowed to create the description of each node by choosing terms from the taxonomy without any restrictions; and 2) the inferred description, which is synthesized from the descriptions of nodes at lower levels and is used for term suggestion, might still

contain some terms from the descriptions of the nodes at lower levels.

Each node is also associated with a natural number that specifies the birth order of the node among its siblings. It is generated when the user linearizes each set of nodes having the same parent in the tree. It equals 0 if the node is the root of the tree, equals 1 if the node is the oldest child, equals 2 if the node is the second oldest child, and so on. The linear order of the node is shown by the full path of the node that can be easily calculated from the full path of the parent node (omitted if the parent is the root) and the birth order of the node. For example, the *path* of the node d_6 of the tree in Fig. 3 has value 3.1.

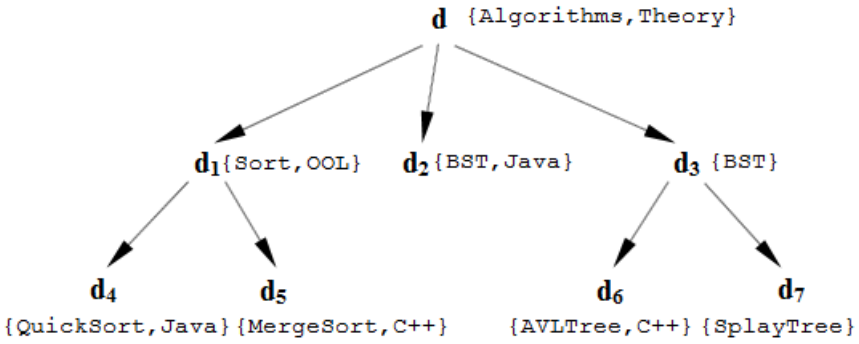


Fig. 3. The structure of a composite document.

Therefore, a node in the tree consists of the following fields: *URI*, *description*, *parent*, *child*, *birth_order*, and *path*, in which:

- *URI*: a uniform resource identifier used to identify the related resource of the node.
- *description*: the description associated with the node.
- *parent*: the parent node of the node.
- *child*: a set of nodes in the tree that are children of the current node.
- *birth_order*: the birth order of the node.
- *path*: the full path of the node.

We are now ready to present the two algorithms in detail.

4.2.1. The algorithm for generating the TOC

The Algorithm 2 is called at the top level with the root as an argument. It visits every node in the tree in pre-order traversal. When visiting a node, it prints out the full path of the node and the description of the node, thus creating one line in the TOC. When the algorithm terminates, we obtain a complete TOC. Fig. 4a presents the TOC that is generated by implementing the algorithm on the tree of Fig. 3.

Algorithm 2 $\mathrm{TOC}(\mathrm{T}, \mathrm{v})$

Input: Tree T, node $v \in T$

Output: The TOC

1: Print *v.path*, *v.description*, new line

2: for all $w \in v.child$ do

3: $TOC(T, w)$

4: return The TOC

4.2.2. The algorithm for generating the INDEX

The Algorithm 3 is called at the top level with the root as an argument. The algorithm uses an array, in which each element of the array consists of two members: *term*, for storing one term in the tree, and *lspath*, for storing the full paths of the nodes that contain the *term*. The algorithm prints the index in three steps: create the index array (by calling to the Algorithm 4); sort the index array; and print the index array.

Algorithm 3 Index(T, v)

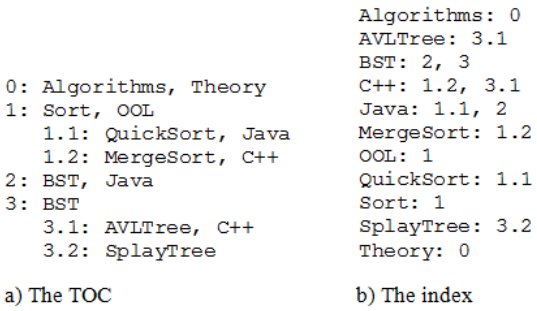
Input: Tree T, node $v \in T$, array M
Output: The index
1: Initialize M is empty
2: $IndexArr(T, v, M)$
3: Sort M in alphabet order on $M.term$
4: for all $m \in M$ do
5: Print $m.term$, $m.lspath$, new line
6: return The index

For creating the index array, the Algorithm 4 is called at the top level with the root as an argument. It visits every node in the tree in post-order traversal. When visiting a node, it compares each term in the description of the node with the term in *term* of each array element. If they are the same, the path of the node will be attached to the list *lspath* of the array element. If not (similar to the case when the array is empty), a new array element that consists of two elements: the term and the path of the node will be attached to the end of the array. When the algorithm terminates, it will return an index array.

Algorithm 4 IndexArr(T, v, M)

Input: Tree T, node $v \in T$, array M
Output: array M
1: for all $w \in v.child$ do
2: $IndexArr(T, w, M)$
3: for all $k \in v.description$ do
4: if $M \neq \varnothing$ then
5: for all $m \in M$ do
6: if $k = m.term$ then
7: Add $v.path$ to $m.lspath$
8: if $M = \varnothing$ or $k \notin M.term$ then
9: $m' = \langle k, v.path \rangle$
10: Add m' to M
11: return array M

The index array that has been created will be sorted in alphabetic order on the field *term*. Finally, the Algorithm 3 prints the index by traversing the array and printing the information consisting of the terms and the associated full paths. Fig. 4b shows an index that is generated after running the algorithm on the tree in Fig. 3.



a) The TOC b) The index

Fig. 4. The TOC and the index of a composite document.

5. Conclusions and Future Work

We have seen a data model for the composition of documents and their metadata management, and we have proposed a method for the materialization of a composite document (i.e. the creation of a paper version including a TOC and an index). Regarding document creation, a user can assemble together existing and/or newly created documents in the form of a tree. How the component documents are assembled together in a tree is totally at the discretion of the author. To create the registration description of a newly created document, the author can select terms from a taxonomy and/or be assisted by the system. To assist the author, the system recommends terms from the implied description of the document (calculated automatically by the system). Regarding document materialization, we have presented algorithms for the automatic generation of the TOC and the index.

One basic assumption of our work is that all atomic documents (i.e. leaf nodes), are in the same format and they can be assembled together easily without any mismatches. Following this assumption, we need to define a unique format for all atomic documents that can be reused in our system. For example, if the contents of atomic documents are all text-based, then we can use XML format. Otherwise, if the contents are of different formats, some other solution must be found.

Another important issue that needs to be considered is the format of the results of materializations. One solution is to have the resulting documents created in EPUB format. EPUB (short for Electronic Publication) is an open e-book standard by the International Digital Publishing Forum (IDPF) [25]. It supersedes the Open eBook standard and its files have the extension EPUB.

In future work, an urgent task is validating our model by developing a prototype, in which we plan to: have composite documents in the form of XML documents; embed our model in the RDF4J (formerly Sesame) framework [8]; and integrate our description generating algorithms, our TOC and index generating algorithms into modules of document creation and document materialization. After that, we need to design a coordinator to assist the user in managing and storing composite documents, as well as in wrapping an existing document (plain text, text with markup, image, sound, etc.) with descriptions and operations in order to create a composite document. The coordinator should also support the sharing of composite documents and offer a query language for searching and retrieving composite documents.

References

[1] ACM Computing Classification System. Retrieved from http://www.acm.org/about/class/2012, visited on October 23, 2016.

[2] Rigaux, P., & Spyratos, N. (2004). Metadata inference for document retrieval in a distributed repository. *Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making* 418-436. Springer Berlin Heidelberg.

[3] Ly, A. T. (2013). *Accessing and Using Complex Multimedia Documents in a Digital Library*. Dissertation, Universite Paris Sud, France.

[4] Cyganiak, R., Wood, D., & Lanthaler, M. (2014). RDF 1.1 concepts and abstract syntax. *W3C Recommendation*, *25*. 1-8. Retrieved from http://www.w3.org/TR/rdf11-concepts/

[5] Brickley, D., & Guha, R. (2014, February 25). RDF schema 1.1. W3C recommendation. *World Wide Web Consortium*. Retrieved from http://www.w3.org/TR/rdf-schema/

[6] Harris, S., Seaborne, A., & Prud'hommeaux, E. (2013). SPARQL 1.1 query language. *W3C Recommendation*, *21*. Retrieved from https://www.w3.org/TR/sparql11-query/

[7] Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., & Wilkinson, K. (2004, May). Jena: Implementing the semantic web recommendations. *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters* 74-83. ACM.

[8] Broekstra, J., Kampman, A., & Van Harmelen, F. (2002, June). Sesame: A generic architecture for storing and querying rdf and rdf schema. *International Semantic Web Conference* 54-68. Springer Berlin Heidelberg.

[9] Erling, O., & Mikhailov, I. (2009). RDF support in the virtuoso DBMS. *Networked Knowledge-Networked Media* 7-2. Springer Berlin Heidelberg.

[10] Neumann, T., & Weikum, G. (2010). The RDF-3X engine for scalable management of RDF data. *The VLDB Journal*, *19(1)*, 91-113.

[11] Hsueh, H. Y., Chen, C. N., & Huang, K. F. (2013). Generating metadata from web documents: a systematic approach. *Human-Centric Computing and Information Sciences*, *3(1)*, 1.

[12] Kiyavitskaya, N., Zeni, N., Cordy, J. R., Mich, L., & Mylopoulos, J. (2009). Cerno: Light-weight tool support for semantic annotation of textual documents. *Data & Knowledge Engineering*, *68(12)*, 1470-1492.

[13] Kunze, J. A., & Baker, T. (2007). The Dublin core metadata element set. Retrieved from http://dublincore.org/

[14] Brickley, D., & Miller, L. (2012). FOAF vocabulary specification 0.98. *Namespace Document*, *9*. Retrieved from http://xmlns.com/foaf/spec/

[15] Tudorache, T., Nyulas, C., Noy, N. F., & Musen, M. A. (2013). WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web. *Semantic Web*, *4(1)*, 89-99.

[16] Handschuh, S., Staab, S., & Volz, R. (2003, May). On deep annotation. Proceedings of the 12th International *Conference on World Wide Web* 431-438. ACM.

[17] Kiryakov, A., Popov, B., Terziev, I., Manov, D., & Ognyanoff, D. (2004). Semantic annotation, indexing, and retrieval. *Web Semantics: Science, Services and Agents on the World Wide Web*, *2(1).*

[18] Dill, S., Eiron, N., Gibson, D., Gruhl, D., Guha, R., Jhingran, A., & Tomlin, J. A. (2003). A case for automated large-scale semantic annotation. *Web Semantics: Science, Services and Agents on the World Wide Web*, *1(1)*, 115-132.

[19] Ciravegna, F., Chapman, S., Dingli, A., & Wilks, Y. (2004, May). Learning to harvest information for the semantic web. *European Semantic Web Symposiu* 312-326. Springer Berlin Heidelberg.

[20] Pastorello Jr, G. Z., Daltio, J., & Medeiros, C. B. (2008, December). Multimedia semantic annotation propagation. *Multimedia, 2008. ISM 2008. Tenth IEEE International Symposium on*, 509-514. IEEE.

[21] Leung, M. K., Mandl, T., Lee, E. A., Latronico, E., Shelton, C., Tripakis, S., & Lickly, B. (2009, October). Scalable semantic annotation using lattice-based ontologies. *International Conference on Model Driven Engineering Languages and Systems*, 393-407. Springer Berlin Heidelberg.

[22] The Connexions Project. Retrieved October 23, 2016 from http://cnx.rice.edu/.

[23] Sicilia, M. A. (2006). Metadata, semantics, and ontology: Providing meaning to information resources. *International Journal of Metadata, Semantics and Ontologies*, *1(1),* 83-86.

[24] Sugibuchi, T., Ly, A. T., & Spyratos, N. (2012, February). Metadata inference for description authoring in a document composition environment. *Italian Research Conference on Digital Libraries*, 69-80. Springer

Berlin Heidelberg.,
[25] International Digital Publishing Forum. Retrieved October 23, 2016 from http://idpf.org/

Anh Tuan Ly is currently a permanent lecturer in the department of software engineering, faculty of computer science and engineering at ThuyLoi University, Vietnam. He did his PhD in computer science from the University of Paris South, France. His research interests include digital library, semantic web and software engineering. He has 7 years of experience in teaching and guiding projects for undergraduate students. He has to his credit 7 publications in national/international conferences.

Nicolas Spyratos is currently professor emeritus at the University of Paris South, scientific advisor of the Japan Science and Technology agency (JST) and adjunct senior researcher at the FORTH Institute of Computer Science in Greece. His research interests include databases, digital libraries, conceptual modeling and big data analytics. He has published over 200 papers in refereed international journals and conferences and has participated in over 20 European and international research projects. He has supervised 24 doctoral theses and has been evaluator for the European programs Esprit and Esprit-Bra as well as for the National Science Foundation (NSF).