Annotation automatique de documents XML [1]

Birahim Gueye, Philippe Rigaux, Nicolas Spyratos
Laboratoire de Recherche en Informatique
Université Paris-Sud Orsay
{gueye,rigaux,spyratos}@lri.fr

Résumé. Nous proposons dans cet article un mécanisme automatique d'annotation de documents. Ce mécanisme s'appuie sur une opération de composition permettant de créer de nouveaux documents à partir de documents existants et sur un algorithme permettant d'inférer l'annotation d'un document composé à partir des annotations de ses parties. Notre modèle est illustré par une étude de cas consacrée à la mise en commun de documents pédagogiques au format XML, dans un environnement coopératif d'enseignement à distance. Nous décrivons un prototype permettant d'annoter ces documents, et d'engendrer une description RDF contenant les annotations.

1 Introduction

Les nombreux axes de développement du Web, très actifs ces dernières années, visent à pallier les limites de HTML et de son modèle d'interaction en développant de nouveaux langages et de nouveaux services. Dans cet article nous considérons plus spécifiquement les applications qui consistent à rechercher, consulter et intégrer des ressources documentaires réparties sur le réseau. En particulier nous proposons un modèle pour gérer les *métadonnées* relatives à un corpus de documents XML. Les objectifs de ce modèle sont la description des documents, le support de langages de recherche, et la création de nouveaux documents par intégration de fragments extraits du corpus. Notre approche est basée sur la notion *d'annotation* des documents, et sur un mécanisme permettant la dérivation automatique de nouvelles annotations lors de la création de nouveaux documents.

Les caractéristiques de notre modèle peuvent être brièvement résumées de la manière suivante. Tout d'abord nous représentons simplement un document sous la forme d'un *identifiant* et d'un graphe de *composants*, lesquels sont eux-mêmes d'autres documents disponibles sur le réseau. Ensuite nous associons à chaque document une description, ou *annotation*, basée sur une taxonomie commune aux utilisateurs du système, et permettant à ces derniers de rechercher des documents en vue de les réutiliser. Finalement la principale contribution de cet article est un mécanisme de génération automatique de l'annotation d'un document composite à partir des annotations de ses documents composants. Notre modèle est décrit à un niveau très général, ce qui permet de l'instancier dans divers contextes architecturaux : nous donnons, à titre d'illustration, une brève étude de cas montrant son utilisation pour la gestion de documents pédagogiques dans un environnement coopératif d'enseignement à distance.

1. Ces travaux ont été partiellement financés par le BQR *Hétérodoc* de l'Université Paris-Sud Orsay.

La section 2 développe de manière informelle les motivations de notre travail. Notre modèle d'annotation est ensuite présenté dans la section 3, et notre étude de cas dans la section 4. Nous concluons cet article avec la section 5. Nous avons choisi de concentrer la présentation de notre approche sur la gestion des annotations, en laissant de côté les aspects relatifs à l'interrogation et à la réutilisation des fragments collectés. Les démonstrations de nos résultats sont également omises. Nous renvoyons le lecteur au rapport technique [Rigaux et Spyratos, 2003], disponible en ligne à l'adresse *http://www.lri.fr/~rigaux/DOC/RS03b.pdf*.

2 Motivation

Le travail présenté dans cet article était initialement motivé par un domaine applicatif particulier, à savoir la gestion coopérative de ressources pédagogiques dans un environnement réparti d'enseignement à distance (*eLearning*). Les résultats obtenus sont plus généraux et on peut les mettre en perspective en les rattachant au domaine de recherche récemment labellisé sous le terme de « Web sémantique » [SW, 2003, Decker *et al.*, 2000]. D'une manière générale, ce terme recouvre les recherches et techniques visant à promouvoir une utilisation plus « intelligente » du Web, ce qui passe nécessairement par un enrichissement de la structure et de la description des contenus accessibles sur le réseau. Un des principaux efforts pour atteindre ces objectifs est le développement des utilisations de XML, un (méta)-langage qui résout quelques-unes des principales limitations de HTML, notamment en ce qui concerne les échanges d'information et l'intégration de données hétérogènes.

Si on se limite aux applications de gestion de contenu considérées dans le présent article, on peut noter en premier lieu que XML permet la création de documents structurés auto-décrits, dont les « fragments » (les « éléments » pour employer la terminologie du modèle XML) peuvent être facilement référencés avec des langages d'adressage comme XPath [XPath, 1999] et XPointer [XPointer, 2002]. En second lieu, on crée relativement facilement de nouveaux documents en effectuant un assemblage de fragments extraits d'autres documents, ces derniers étant éventuellement localisés sur de lointaines parties du réseau, au prix parfois d'une restructuration effectuée avec des langages spécialisés comme XSLT [XSLT, 1999, Amann et Rigaux, 2002]. Ces techniques sont maintenant bien établies et fournissent un cadre puissant pour la restructuration et la réutilisation de contenus dans de nouveaux contextes.

Tout n'est cependant pas acquis car, au-delà des besoins en extraction et restructuration, nous devons également disposer de langages de *description* des contenus supportant des outils de recherche intelligents. Ce besoin est à l'origine de l'un des axes de recherche du web sémantique, à savoir la qualification des documents du Web par des informations permettant de décrire leur contenu. Ces informations sont communément désignées par le terme de *méta-données*.

De nombreux résultats ont été obtenus récemment pour la représentation, le stockage et l'interrogation de méta-données. On peut citer principalement le langage RDF (*Resource Description Framework*) [RDF, 1999], les schémas RDF [RDFS, 2000], des langages de requêtes pour des bases de données RDF [Karvounarakis *et al.*, 2002, Alexaki *et al.*, 2001] et quelques outils pour aider la production – manuelle – de des-

criptions RDF à partir de documents [Kahan et Koivunen, 2001, Decker *et al.*, 2000]. De nombreux standards de méta-données existent maintenant, comme par exemple le Dublin Core [DublinCore, 1999], ou la recommandation de l'IEEE sur les *Learning Object Metadata* (LOM) [LOM, 2002]. La génération de méta-données reste cependant délicate, et leur utilisation restreinte dans la mesure où elles se limitent souvent à décrire la partie « administrative » d'un document (langue, auteur, date de création et de modification) au détriment de la partie « sémantique » décrivant le contenu du document proprement dit (nous renvoyons au livre [Baeza-Yates et Ribeiro-Neto, 1999] pour une discussion plus approfondie sur cette distinction). De plus cette génération est essentiellement manuelle, ou, disons, interactive, le rôle du logiciel se limitant à l'aide à l'acquisition [Kahan et Koivunen, 2001, Wang et Lochovsky, 2003, Dill *et al.*, 2003].

Dans cet article nous nous intéressons précisément à ces méta-données « sémantiques », et à l'automatisation de leur production. L'idée de base est d'exploiter la structure d'un document pour inférer sa description en fonction de la description de ses parties. Un exemple simple pour comprendre l'intuition de notre approche est la comparaison avec la table des matières d'un livre : connaissant les sujets traités par les différentes sections d'un chapitre, on peut déduire le sujet traité globalement dans le chapitre. Nous formalisons – et généralisons – cette intuition sous la forme d'un mécanisme d'inférence qui s'appuie sur une taxonomie hiérarchique des « sujets », relative au domaine de connaissance considéré (nous prendrons comme exemple l'enseignement en informatique). Les méta-données que nous considérons sont donc des ensembles de « sujets » qualifiant le contenu d'un document. Nous les désignons plus précisement sous le terme *d'annotation* dans ce qui suit [Erdmann *et al.*, 2000, Staab *et al.*, 2001].

Une application représentative de l'intérêt de notre approche est la gestion de documents pédagogiques, au format électronique, dans un environnement d'enseignement « ouvert » où chacun peut construire ses propres documents en réutilisant éventuellement la production des autres. Nous avons réalisé un prototype « auteur » qui permet de créer des annotations sur des documents XML à l'aide des mécanismes issus de notre modèle. Le prototype produit les annotations obtenues sous forme d'une description RDF qui peut alors être utilisée pour informer les autres utilisateurs de la disponibilité du document, et transmise à un médiateur qui se charge de fournir les services de référencement et de support à la recherche. Comme indiqué dans l'introduction, nous nous limitons dans le reste de cet article à la gestion des annotations, et à la description du prototype à titre d'illustration de notre modèle.

3 Le modèle d'annotation

Nous adoptons dans notre modèle un point de vue abstrait selon lequel un document est simplement représenté par un *identifiant* et un *graphe de composition*. En pratique l'identifiant peut-être n'importe quel mécanisme de référencement d'un contenu XML comme, par exemple, XPointer [XPointer, 2002]. Le graphe de composition reflète la structure hiérarchique obtenue par imbrication des éléments XML.

Definition 1 (Representation d'un document) *Un* document *est représenté par un identifiant d et par un ensemble (éventuellement vide) d'identifiants $d_1, \ldots, d_n$,*

dénommés les parties *de d, et notées parties(d), i.e., parties(d)* $= \{d_1, \ldots, d_n\}$. *Si parties(d)* $= \emptyset$ *alors d est* atomique, *sinon d est* composite.

Pour simplifier, nous assimilerons dans la suite « document » et « représentation d'un document ». Il est entendu qu'en pratique un document est bien plus qu'un identifiant et un graphe, mais ces informations suffisent à nos besoins. Par ailleurs, nous adopterons la notation $d = d_1 + d_2 + \ldots + d_n$ pour signifier qu'un document d a pour parties $parties(d) = \{d_1, d_2, \ldots, d_n\}$.

Un document composite est constitué de parties qui, elles-mêmes, peuvent être des documents atomiques ou composites. D'une manière générale, un document est donc construit récursivement par un processus de composition qui préserve l'identité de chaque composant. On peut le représenter par un graphe doté d'une unique racine, que nous nommerons *graphe de composition* par la suite. Nous supposons que ce graphe est un graphe dirigé acyclique, ce qui correspond à l'hypothèse raisonnable qu'un document n'est pas une partie de lui-même. On peut noter que nous ne considérons pas l'ordre des parties d'un document car notre processus de dérivation d'annotation ne dépend d'aucun ordre particulier.

Comment nous l'avons souligné dans l'introduction, les annotations sont construites à partir d'un vocabulaire structuré, ou *taxonomie*. Cette taxonomie consiste en un ensemble de termes doté d'une relation de subsomption entre les termes. L'ACM fournit par exemple une taxonomie pour représenter les divers sujets d'un enseignement universitaire en informatique, dite *ACM Computing Classification System* [ACCS, 1999].

Definition 2 *Une taxonomie est une paire* $(T, \preceq)$ *où* T *est une* terminologie, *i.e., un ensemble fini non vide de noms, ou* termes, *et* $\preceq$ *est une relation réflexive et transitive sur* T *dite de* subsomption.

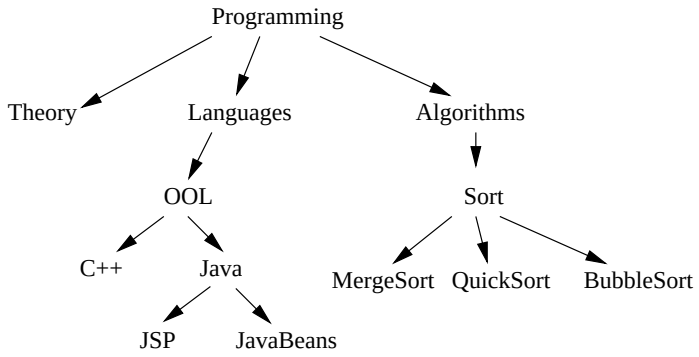

FIG. 1 – *Une taxonomie*

La figure 1 montre une taxonomie que nous utiliserons comme base de tous nos exemples. Le terme `Object-Oriented Languages` (OOL) est subsumé par `Languages` (`OOL` $\preceq$ `Languages`) et `JavaBeans` est subsumé par `Java` (`JavaBeans` $\preceq$ `Java`). Nous supposerons par la suite que le graphe d'une taxonomie est un arbre, ce qui est souvent le cas en pratique.

Quand un auteur crée un document, il choisit un ou plusieurs termes de la taxonomie pour annoter son contenu. Si, par exemple, le document d traite de l'algorithme de tri rapide et de son écriture en Java, l'auteur choisira probablement les deux termes `QuickSort` et `Java`. L'ensemble {`QuickSort, Java`} est l'annotation de d.

Definition 3 (Annotation) *Étant donnée une taxonomie* $(T, \preceq)$*, nous appelons* annotation *dans* T *tout ensemble de termes de* T*.*

Nous considèrerons donc par la suite les annotations *réduites* définies comme suit :

Definition 4 (Annotation réduite) *Une annotation* A *de* T *est dite* réduite *si pour tout terme* s *et tout terme* t *dans* A*,* $s \not\preceq t$ *et* $t \not\preceq s$*.*

Une des manières de réduire l'annotation d'un document d est de ne garder que les termes minimaux : c'est ce que nous désignerons par $\mathcal{A}(d)$.

L'annotation d'un document peut être vue à la fois comme un « résumé » de son contenu et comme un outil pour référencer le document. Dans le cas d'un document atomique l'annotation peut être produite soit par l'auteur, soit éventuellement par un mécanisme d'extraction automatique. Dans le cas d'un document composite d, en revanche, on souhaiterait dériver automatiquement l'annotation de d à partir des annotations de ses parties. Cela nous amène à la question principale de cet article : étant donné un document composite d, et en supposant connues les annotations des parties de d, qu'est-ce qu'une « bonne » annotation dérivée pour d, et comment la déterminer automatiquement ? La réponse à la première question relève de choix conceptuels que nous allons tout d'abord introduire par un exemple. Considérons un document $d = d_1 + d_2$, composé de deux parties dont les annotations sont les suivantes :

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

Alors l'annotation dérivée de d sera {`Sort, OOL`}, car elle résume de la manière la plus complète et la plus exacte possible le contenu de *l'ensemble* des composants de d : tous parlent de tri et de programmation objet. Comme le montre (intuitivement) cet exemple, une annotation dérivée retient tout ce qui est *commun* à l'ensemble des composants d'un document. De plus, elle doit être minimale, autrement dit aussi proche que possible de l'annotation de chaque partie de d. L'annotation {`Programming`} (la racine de la taxonomie) est par exemple toujours un candidat possible pour décrire ce que des parties ont en commun, mais, le plus souvent, il existe d'autres possibilités, plus précises. Nous allons maintenant formaliser ces intuitions en commençant par introduire la relation suivante sur les annotations :

Definition 5 (Relation sur les annotations) *Soit* A *et* A' *deux annotations. Nous dirons que* A *est* plus fine *que* A'*, et nous le noterons* $A \sqsubseteq A'$*, si et seulement si pour chaque terme* $t' \in A'$*, il existe un terme* $t \in A$ *tel que* $t \preceq t'$*.*

Par exemple l'annotation $A =$ {`QuickSort, Java, BubbleSort`} est plus fine que $A' =$ {`Sort, Java`}. Il est clair que $\sqsubseteq$ est une relation réflexive et transitive, et donc un pré-ordre sur l'ensemble des annotations. En revanche $\sqsubseteq$ n'est pas antisymétrique :

prenons A_1 = {OOL, Java, Sort} et A_2 = {Java, Sort, Algorithms}. Il est facile de constater que $A_1 \sqsubseteq A_2$ et $A_2 \sqsubseteq A_1$, bien que $A_1 \neq A_2$. Si on ne considère que les annotations réduites, alors $\sqsubseteq$ devient antisymétrique, et donc une relation d'ordre.

Proposition 1 *La relation $\sqsubseteq$ est un ordre partiel sur l'ensemble des annotations réduites.*

On peut montrer, de plus, que le graphe de $\sqsubseteq$ est un semi-treillis supérieur sur l'ensemble des annotations réduites.

Proposition 2 *Soit $\mathcal{D} = \{A_1, .., A_n\}$ un ensemble d'annotations réduites. Soit $\mathcal{U}$ l'ensemble de toutes les annotations réduites S telles que $A_i \sqsubseteq S, i = 1, 2, ..., n$, i.e., $\mathcal{U} = \{S | A_i \sqsubseteq S, i = 1, \ldots, n\}$. Alors $\mathcal{U}$ a un plus petit élément, que nous dénoterons $lub(\mathcal{D}, \sqsubseteq)$.*

Nous sommes maintenant en mesure de donner la définition formelle de l'annotation dérivée d'un document composite.

Definition 6 (Annotation dérivée) *Soit un document $d = d_1 + d_2 \ldots + d_n$ et soient $A_1, .., A_n$ les annotations réduites de ses parties.* L'annotation dérivée *de d, dénotée $\mathcal{A}_{der}(d)$, est le plus petit élément supérieur commun à $\{A_1, .., A_n\}$ dans $\sqsubseteq$, i.e., $\mathcal{A}_{der}(d) = lub(\{A_1, .., A_n\}, \sqsubseteq)$*

L'algorithme suivant calcule l'annotation dérivée d'un document d. Il s'appuie sur la fonction $lub_{\preceq}(t_1, \ldots, t_n)$ qui renvoie le plus petit élément supérieur commun aux termes $t_i, i = 1, \ldots, n$ dans $(T, \preceq)$. Cet élément existe toujours puisque $(T, \preceq)$ est un arbre.

Algorithme AnnotationDérivée
Entrée : Un document composite $d = d_1 + d_2 + \ldots + d_n$
Sortie : L'annotation dérivée $\mathcal{A}_{der}(d)$
début
  Calculer $P = \mathcal{A}(d_1) \times \mathcal{A}(d_2) \times \ldots \times \mathcal{A}(d_n)$
  pour chaque n-uplet $L_k = [t_1^k, t_2^k, \ldots, t_n^k]$ dans P, calculer $T_k = lub_{\preceq}(t_1^k, t_2^k, \ldots, t_n^k)$
  Soit $A' = \{T_1, \ldots, T_l\}$ [l'ensemble des T_k calculés ci-dessus]
  Réduire A' [ne conserver que les termes minimaux de A']
  retourner A'
fin

Pour comprendre comment fonctionne cet algorithme, prenons une nouvelle fois l'exemple du document $d = d_1 + d_2$ étudié précédemment, composé de deux parties avec les annotations suivantes :

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

Le produit cartésien $\mathcal{A}(d_1) \times \mathcal{A}(d_2)$ donne l'ensemble de n-uplets suivant :

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

On calcule ensuite le plus petit élément supérieur commun aux termes de chaque tuple (voir figure 1), ce qui donne respectivement $T_1 =$ `Sort`, $T_2 =$ `Programming`, $T_3 =$ `Programming` et $T_4 =$ `OOL`. On obtient l'ensemble $A' = \{$`Sort, Programming, OOL`$\}$, qui après réduction, devient $\{$`OOL, Sort`$\}$. L'interprétation est simple : le document composite traite, dans toutes ses parties, à la fois de tri et de langages orientés-objet.

Voici un second exemple, presque similaire, qui montre en quoi l'annotation dérivée ne retient que ce que les documents ont en commun. Prenons le document $d' = d_1 + d_3$, avec les annotations suivantes pour les deux parties :

$$\mathcal{A}(d_1) = \{\texttt{QuickSort, Java}\} \qquad \mathcal{A}(d_3) = \{\texttt{BubbleSort}\}$$

En appliquant l'algorithme, on obtient tout d'abord le résultat suivant pour le produit cartésien :

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

Donc on a $A' = \{$`Sort, Programming`$\}$ dans lequel on élimine, au cours de la phase de réduction, le terme `Programming` qui subsume `Sort`. L'annotation dérivée est donc $\{$`Sort`$\}$, qui représente le sujet commun aux deux parties du document. Le fait que l'une des parties traite également de Java apparaît secondaire, *dans le contexte du document composite*, et n'est donc pas représenté. Deux remarques sont ici nécessaires :

1. Ne pas avoir conservé `Java` dans l'annotation dérivée ne signifie pas que l'on a perdu de l'information. Si on effectue une recherche des documents qui traitent de Java, on trouvera d_1, et pas d', ce qui constitue une réponse cohérente.

2. Placer `Java` au niveau de l'annotation du document composite d' soulèverait en revanche le problème suivant : toute recherche avec un terme (disons donc, `Java`) ramènerait tous les documents dont une partie, même minime, serait relative à ce terme, et cette partie serait présente autant de fois qu'elle intervient dans la composition de documents.

Notre choix de placer dans l'annotation dérivée les sujets communs aux parties d'un document composite d revient à capturer la motivation de l'auteur de d. En choisissant les parties de d, ce dernier obéit à une logique précise consistant à sélectionner certains aspects des documents dont il dispose, et à les regrouper en fonction de ces aspects. En fonction du contexte, un même document pourra d'ailleurs engendrer des annotations dérivées très différentes selon l'aspect de son annotation qui aura été privilégiée.

4 Application : le prototype XANNOT

Nous décrivons dans cette section un prototype, XANNOT (pour *XML Annotation Tool*) qui montre en pratique une application de notre modèle et illustre concrètement son intérêt. L'architecture de XANNOT est résumée dans la figure 2. Les documents composites sont représentés dans cette instanciation de notre modèle par des documents XML au format DocBook [Walsh et Muellner, 1999]. Un programme est utilisé du côté client pour parcourir et annoter les parties et sous-parties d'un document. Les annotations produites sont alors représentées en RDF et transmises à un médiateur qui s'appuie sur une base RDF pour fournir des services de recherche aux utilisateurs du système.

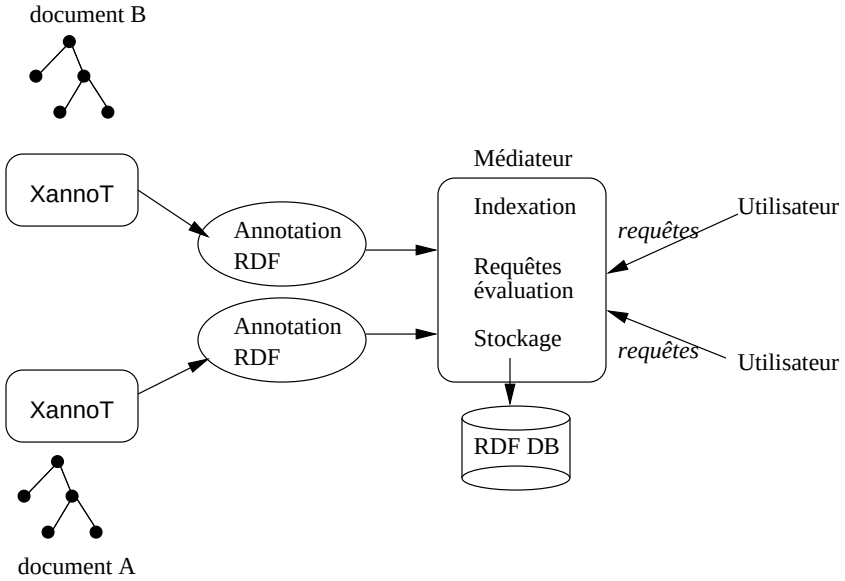

FIG. 2 – *Architecture de* XANNOT

Nous nous limitons dans ce qui suit à la partie relative au programme client générateur d'annotations, et laissons de côté les fonctionnalités du serveur/médiateur.

Les documents

DocBook est une DTD destinée à normaliser, en SGML ou XML, la réprésentation de gros documents techniques. Elle est bien adaptée aux livres ou aux documentations, notamment informatiques, et très largement utilisée dans le domaine de l'édition. Nous avons donc choisi de le prendre comme base pour la représentation de nos documents. Il est bon cependant de souligner que n'importe quelle autre DTD ferait l'affaire, la seule contrainte étant que tous les auteurs utilisent un format commun. Cette contrainte découle de nécessités pratiques : l'échange de fragments et leur intégration est grandement facilitée par l'homogénéité de la représentation au départ. Il est assez simple d'insérer

un fragment DocBook dans un document DocBook tout en préservant la validité du résultat par rapport à la DTD.

Notre prototype construit une représentation arborescente d'un document en se basant sur les éléments structurels suivants : `book`, `chapter` et `section`. Les éléments de type `section` de plus bas niveau – qui n'ont pas eux-mêmes de fils de type `section`[2] – sont considérés comme les feuilles du graphe de composition du document (voir section 3), et donc comme les documents atomiques pour lesquels l'auteur doit fournir une annotation. L'annotation des éléments supérieurs dans la hiérarchie est alors automatiquement inférée.

Parcours et annotation des documents

Notre outil, XANNOT, propose une interface graphique pour consulter le document et pour créer des annotations. Quand un document est lu par un auteur, sa structure est analysée, et les nœuds de l'arborescence sont affichés dans la fenêtre gauche de l'interface, tandis que le contenu lui-même est affiché dans la fenêtre principale (voir figure 3).

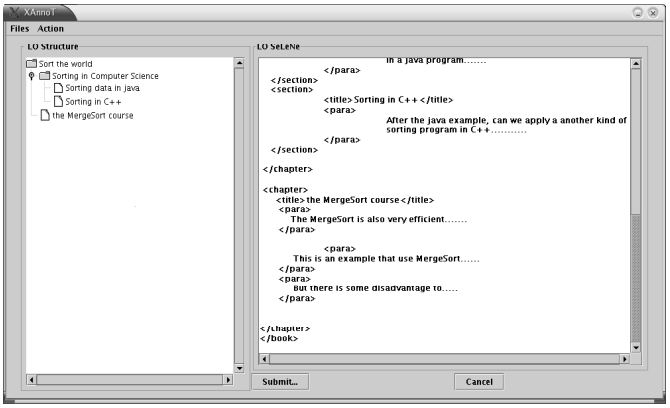

FIG. 3 – *Interface graphique de* XANNOT

Le rôle de l'auteur, avant de soumettre son document au médiateur, est d'annoter les éléments `section` situés au plus bas niveau de l'arbre avec des termes de la taxonomie commune à tous les utilisateurs du système. Pour faciliter cette tâche, les termes de la taxonomie peuvent être sélectionnés graphiquement dans une fenêtre qui affiche l'arbre de la taxonomie avec une représentation hyperbolique (autrement dit la représentation est « focalisée » sur le terme sélectionné) : voir figure 4.

Quand tous les éléments constituant les feuilles sont annotés, l'annotation des éléments composites (correspondant aux *documents* composites dans le modèle) est inférée, XANNOT appliquant l'algorithme ANNOTATIONDÉRIVÉE décrit dans la section 3. Cette inférence est effectuée à chaque fois que l'annotation d'un composant d'un élément composite est modifiée. Un exemple est donné dans la figure 5 : deux éléments ont

2. Ce type d"élément est récursif dans DocBook.

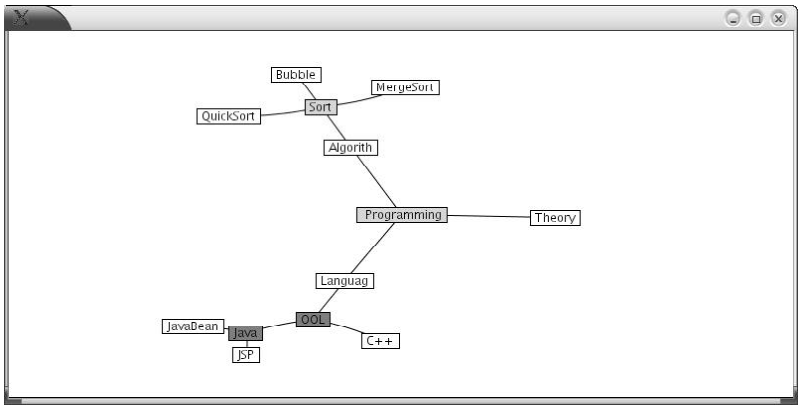

FIG. 4 – *Sélection des termes dans la taxonomie*

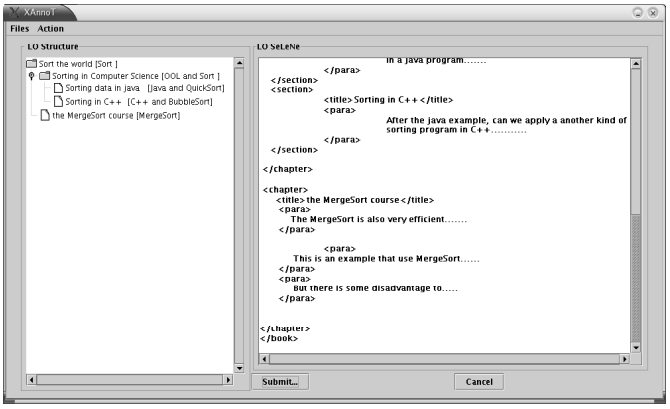

FIG. 5 – *Dérivation d'annotation avec* XANNOT

été annotés avec, respectivement, {`Java, QuickSort`} et {`C++, BubbleSort`}. L'annotation dérivée {`OOL, Sort`} est alors calculée et associée à l'élément composite. Ajoutons que XANNOT permet également à l'auteur d'insérer des termes au niveau d'un élément composite, l'annotation finale étant alors calculée à partir de l'annotation « auteur » et de l'annotation dérivée. Cette extension (et quelques autres) s'intègre facilement au modèle et est présentée en détail dans [Rigaux et Spyratos, 2003].

Finalement le graphe de composition comprenant les annotations des nœuds du document est représenté en RDF pour être transmis au médiateur qui stocke, pour chaque terme t de la taxonomie, la liste des chemins menant aux éléments XML dont l'annotation comprend le terme t.

5 Conclusion

Nous avons présenté dans cet article un modèle basé sur la composition de documents dans un environnement distribué comme l'Internet, et comprenant un système d'annotation automatique des documents composites en fonction de l'annotation des documents composants. Ce travail est une étape dans un projet qui vise à définir les moyens de produire le plus riche ensemble possible de méta-données pour décrire des bibliothèques numériques réparties comme, par exemple, des documents pédagogiques en ligne fournis aux étudiants.

À ce titre nous étudions la possibilité de produire l'équivalent des systèmes d'aide à la lecture que nous sommes habitués à trouver dans les livres « papier » : table des matières bien sûr (c'est la proposition du présent article) mais également index, ordres possibles de parcours des différentes parties, etc. Le travail en cours consiste également à concevoir et implanter le médiateur gérant nos documents composites. Nous travaillons en coopération avec l'institut ICS-FORTH (Crète) pour utiliser leur plateforme *RDFSuite* [Alexaki *et al.*, 2001] dans ce sens.

Références

[ACCS, 1999] The ACM Computing Classification System. ACM, 1999. http://www.acm.org/class/.

[Alexaki *et al.*, 2001] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, et K. Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *Proc. Intl. Conf. on Semantic Web*, 2001.

[Amann et Rigaux, 2002] B. Amann et P. Rigaux. *Comprendre XSLT*. Éditions O'Reilly, 2002.

[Baeza-Yates et Ribeiro-Neto, 1999] R. Baeza-Yates et B. Ribeiro-Neto, editors. *Modern Information Retrieval*. Addison-Wesley, 1999.

[Decker *et al.*, 2000] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, et I. Horrocks. The Semantic Web: The Roles of XML and RDF. *IEEE Expert*, 15(3), 2000.

[Dill *et al.*, 2003] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, et 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.

[DublinCore, 1999] Dublin Core Metadata Element Set. Technical Report, 1999. http://dublincore.org/.

[Erdmann *et al.*, 2000] M. Erdmann, A. Maedche, H. Schnurr, et 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.

[Kahan et Koivunen, 2001] J. Kahan et M. Koivunen. Annotea: an Open RDF Infrastructure for Shared Web Annotations. In *Proc. Intl. World Wide Web Conference (WWW)*, pages 623–632, 2001.

[Karvounarakis *et al.*, 2002] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, et M. Scholl. RQL: A Declarative Query Language for RDF. In *Proc. Intl. World Wide Web Conference (WWW)*, pages 623–632, 2002.

[LOM, 2002] Draft Standard for Learning Objects Metadata. IEEE, 2002.

[RDF, 1999] Resource Description Framework Model and Syntax Specification. World Wide Web Consortium, 1999.

[RDFS, 2000] Resource Description Framework Schema (RDF/S). World Wide Web Consortium, 2000.

[Rigaux et Spyratos, 2003] P. Rigaux et N. Spyratos. Generation and Syndication of Learning Object Metadata. Technical Report 1371, Laboratoire de Recherche en Informatique, 2003.

[Staab *et al.*, 2001] S. Staab, A. Maedche, et S. Handschuh. An Annotation Framework for the Semantic Web. In *Proc. Intl. Workshop on Multimedia annotation*, 2001.

[SW, 2003] The Semantic Web Community Portal. Web site, 2003. http://www.semanticweb.org.

[Walsh et Muellner, 1999] N. Walsh et Leonard Muellner. *DocBook, the definitive guide*. O'Reilly, 1999.

[Wang et Lochovsky, 2003] J. Wang et F.H. Lochovsky. Data extraction and label assignment for web databases. In *Proc. Intl. World Wide Web Conference (WWW)*, pages 187–196, 2003.

[XPath, 1999] The XPath language recommandation (1.0). World Wide Web Consortium, 1999. http://www.w3.org/TR/xpath.

[XPointer, 2002] The XML Pointer Language. World Wide Web Consortium, 2002. http://www.w3c.org/TR/xptr/.

[XSLT, 1999] The Extensible Stylesheet Language Family (XSL). World Wide Web Consortium, 1999. http://www.w3.org/Style/XSL.

Summary

In this paper we propose an automatic mechanism for annotating XML documents. This mechanism relies on a *composition* operator to create new documents from existing ones, and on an inference algorithm for automatically deriving the annotation of composite documents from the annotations of their components. We illustrate the features of the model with an application to eLearning resources. We also describe a prototype which allows to create a new document from eLearning fragments collected over the Web, and generates an RDF-based annotation of the document's content.