
Modèle de couplage de documents structurés et de bases de données

Le projet DRUID

Frédérique Laforest, Youakim Badr

*LIRIS,
INSA, Bat Blaise Pascal
7 avenue Recteur Capelle
69621 Villeurbanne Cedex
frederique.laforest@insa-lyon.fr, youakim.badr@lisi.insa-lyon.fr*

RÉSUMÉ. Dans cet article, nous présentons DRUID, un système de saisie d'information sous forme de documents semi-structurés, lié à une base de données relationnelle. Les documents saisis par l'utilisateur sont des documents centrés paragraphes, c'est-à-dire que les balises encadrent des portions de texte libre en langue naturelle, et non des données unitaires. Le DRUID Core Module utilise un ensemble de règles permettant d'extraire de ces paragraphes les données unitaires prévues dans la base de données. Les règles sont composées d'une extension de la syntaxe XSL pour la manipulation de la structure du document, ainsi que de transducteurs à états finis pour l'appel à un processeur de langue naturelle. Les données ainsi extraites sont tout d'abord déposées dans un document centré données puis dans la base de données. L'intérêt d'un tel système est d'associer à la fois la souplesse des interfaces documentaires, et la puissance des langages d'interrogation des bases de données relationnelles. Chaque requête à la base permet d'avoir une vue tabulaire des informations demandées, ainsi qu'un accès aux documents sources de l'information.

ABSTRACT. In this paper, we present the DRUID system that links document-based information capture to relational database querying. End-users capture paragraph-centric documents where tags embrace paragraphs of free text. The DRUID Core Module employs a set of rules to extract relevant data from free text paragraphs. Some rules are written as XSL extension elements for document structure management, and others invoke finite state transducers for natural language processing and information extraction. Extracted data are stored in both a data-centric document and in the database. This system is interesting because it associates documents flexibility with databases querying efficiency. Queries to the database return the classical data tables as well as links to the source documents.

MOTS-CLES: documents structurés, extraction de données, interface utilisateur, base de données, XML

KEYWORDS: structured documents, data extraction, user interface, database, XML

1. Introduction

Les bases de données et les bases de documents ont été créées pour des applications totalement différentes. Dans le premier cas, il s'agit de fournir des supports de stockage pour traiter et interroger des données dont la création et la mise à jour sont fréquentes. Dans le second cas il s'agit de retrouver des documents à l'aide de mots clés, ces documents contenant des informations stables rarement modifiées. L'arrivée d'Internet et du Web a rapproché ces deux mondes en permettant d'une part de remplir les documents à l'aide de données d'une base et d'autre part d'indexer les documents. Ces deux approches sont opposées en ce sens que l'une (données) relève des informations structurées alors que l'autre (documents) fait partie des approches qui traitent aujourd'hui des données semi-structurées.

Pour la gestion du dossier médical, les systèmes classiquement proposés sont fondés sur des bases de données. Pour le médecin, cette approche se traduit par des saisies dans des grilles. La variété des cas médicaux s'accommode mal de systèmes figés et rigides. Cependant, l'utilisation de bases de données pour l'analyse et le calcul est aujourd'hui la seule approche permettant d'accomplir des recherches et traitements efficaces. D'un autre côté, le document électronique est proche du dossier médical papier. Malgré les travaux entrepris à ce jour, l'utilisation directe du document n'est pas encore possible : les systèmes d'interrogation de bases documentaires sont peu précis.

Les systèmes classiques actuels associent documents et données de la façon suivante : (1) les documents sont rédigés selon les règles de qualité imposées par l'entreprise. (2) des informations concernant tout nouveau document sont stockées dans une base de données, à l'aide d'une grille de saisie. Cette grille contient des informations contextuelles (auteur, date, version...) et des mots clés permettant de classer ce document. La base de données est donc un index complexe permettant de retrouver des documents.

L'objectif de notre système est différent : nous voulons mettre à jour une base de données " classique " représentant un système d'information à partir de documents saisis librement par l'utilisateur, et non fournir un index d'accès à des documents. La différence par rapport aux systèmes classiques est double. Il faut : (1) éliminer l'étape fastidieuse du remplissage de la grille de saisie, (2) intégrer dans la base de données non seulement des informations contextuelles, mais également des portions de contenu permettant une interrogation sur les valeurs contenues dans les documents.

Une manière d'extraire les informations des documents consisterait à imposer un balisage très précis des informations à insérer dans la base de données, comme illustré sur la figure 1. Ces documents sont appelés des documents centrés données. Les recherches actuelles du domaine se fondent principalement sur ce type de document. Pour notre part, nous nous plaçons plutôt dans le cadre de documents où

les informations à recueillir sont balisées de manière plus "lâche", pour assurer une souplesse lors de la rédaction des documents. Nous nous intéressons à des documents où le balisage encadre des paragraphes et non des informations unitaires (figure 1), nous les appelons des documents centrés paragraphes. Le travail du processus d'analyse ne se restreint alors pas à l'extraction d'informations balisées, mais à la recherche d'informations dans un paragraphe en langage pseudo-naturel. Les termes en gras dans le document orienté paragraphes sont les informations que le système doit retrouver afin de remplir la base de données.

<pre><rencontre id='245'> <diagnostic> bronchite, 25/5/00 Le patient se plaint d'une toux rauque et de fièvre modérée </diagnostic> <antécédent> père, infarctus du myocarde 1997 </ antécédent > <prescription> Donner 3 comprimés d'aspirine 3 fois par jour pendant 10 jours </prescription> </rencontre></pre>	<pre><rencontre id='245'> <diagnostic> bronchite </diagnostic> <date> 25/5/00 </date> <symptôme> <maladie> toux rauque </maladie> <maladie> fièvre </maladie> </symptôme> <antécédent> <maladie > infarctus du myocarde </maladie> <parenté> père </parenté> <date>1997</date> </antécédent> <prescription> <dosage> 3 </dosage> <unitéDosage> 3 </unitéDosage> <médicament mID='34'>aspirine </médicament> <durée>10</durée> <unitéDurée> jour </ unitéDurée > </prescription> < /rencontre></pre>
--	---

Figure 1: Un document rédigé sous une forme centrée données (à droite) et le même document rédigé sous une forme centrée paragraphes (à gauche)

Dans cet article, nous présentons tout d'abord le système DRUID, d'un point de vue fonctionnel. Nous présentons ensuite de manière détaillée le DRUID Core Module, instance principale de notre architecture. Nous expliquons ensuite les choix d'implantation qui ont été faits pour le prototype que nous avons développé. Avant de conclure, nous comparons notre système avec les études les plus proches que nous avons trouvées dans la littérature.

2. Fonctionnalités du système DRUID

Le système DRUID peut se décrire selon les fonctionnalités suivantes :

Pour l'utilisateur final :

4 Nom de la revue. Volume X – n° X/2002

- Une saisie de documents orientés paragraphes, conformes à un modèle de documents. Cette saisie se fait sous forme de paragraphes de texte libre, qui sont ensuite balisés manuellement. Dans nos essais, on trouve environ 6 à 8 balises par document. La validation du document centré paragraphes entraîne sa transformation automatique en un document centré données.
- Une interrogation d'une base de données relationnelle remplie à partir du document centré données : avec une interface graphique, l'utilisateur construit des requêtes SQL sur la base de données. Les résultats comportent à la fois les données tabulaires classiques, mais également des liens vers les documents sources des données. Ceci permet à l'utilisateur de consulter le contexte des données retournées par le système, et ainsi de mieux interpréter les réponses reçues de la base.
- Une validation des extractions effectuées. Dans notre prototype actuel, nous demandons à l'utilisateur de valider le document centré données. Au besoin, il peut faire des modifications sur ce document. La validation du document centré données entraîne l'alimentation automatique de la base.

Pour l'adaptation au domaine :

DRUID est un système générique permettant la saisie de documents orientés paragraphes, leur transformation en documents orientés données et l'alimentation d'une base de données relationnelle avec ce dernier document. Pour l'utiliser dans un certain domaine, il est nécessaire d'effectuer en amont une étape de définition des éléments suivants :

- Le modèle de données pour la base. Dans le cas où l'informatisation est nouvelle dans l'organisation cible, il faut définir le modèle de la base de données relationnelle. Dans le cas où cette base existe, il faut l'augmenter de colonnes pour stocker les références aux documents sources de données.
- Les modèles de documents adaptés au domaine. Il faut également définir les modèles de documents permettant l'alimentation de la base et la saisie par l'utilisateur final. Pour cela, nous avons défini un algorithme qui, à partir du modèle de la base de données, construit les modèles de documents adéquats [Laforest, 2003].
- Les règles spécifiques au domaine. L'extraction de données depuis les documents centrés paragraphes se fonde sur des règles de transformation et des motifs d'extraction, qu'il faut rédiger manuellement. Nous avons spécifié un langage de haut niveau pour la spécification des motifs.
- La définition des dictionnaires du domaine. Les dictionnaires standards de la langue française sont intégrés à DRUID. Par contre, il est nécessaire de fournir tout dictionnaire supplémentaire spécifique au domaine d'activité concerné (dictionnaire de médicaments par exemple).

Dans notre prototype DRUID, nous avons intégré les modules suivants :

- une interface utilisateur final lui permettant la rédaction de nouveaux documents, la validation des documents centrés données générés, et l'interrogation de la base
- le module principal concerne la transformation de documents centrés paragraphes en documents centrés données et le stockage des données dans une base relationnelle. Il comporte trois sous-modules :
 - 1- Un processeur de transformations dont l'objectif est d'extraire les données de chaque document centré paragraphes saisis et de produire un document centré données pour chacun. L'extraction elle-même est soustraite à un module d'extraction. Ce processeur y ajoute une sélection des paragraphes, un traitement des données extraites et une restructuration du document brut.
 - 2- Un processeur d'alimentation qui crée les tuples pour la base de données à partir des documents centrés données construits par le processeur de transformations.
 - 3- Un processeur d'extraction qui extrait effectivement les données des portions de texte libre. Il se fonde sur une spécification de motifs d'extraction et sur un processeur de langue naturelle (NLP) qui exécute ces motifs.
- un module d'alimentation de la base de données à partir de documents centrés données permet d'inclure les nouvelles données dans la base.
- un système de gestion de bases de données gère la base des données extraites et stocke le fond documentaire. La base de données est relationnelle et structurée. Ce n'est pas un index de mots clés, mais elle est modélisée selon un modèle de données du domaine (tables patient, médecin, médicament, prescription... pour un dossier médical). Nous y avons cependant ajouté des liens vers les documents sources des données par le biais de colonnes contenant des références aux documents. Le fond documentaire contient les documents entiers, sans aucune fragmentation ni indexation. Ce stockage aurait pu être fait sur un système de fichiers ; nous profitons du SGBD pour gérer les droits d'accès aux documents.
- un module de compilation de motifs d'extraction en transducteurs à états finis permet de réécrire les motifs rédigés manuellement dans notre langage de haut niveau en entrées au processeur de langue naturelle.

3. DRUID Core Module

Comme indiqué ci-avant, le DRUID Core Module comporte trois sous-modules, que nous allons détailler dans cette section. Chacun des trois sous-modules (processeurs de transformations, d'extraction et d'alimentation) se base sur des règles pour mener à bien leur mission. Nous verrons donc également les différents

types de règles utilisés par chaque sous-module, ainsi que les langages choisis pour les implanter.

En entrée du DRUID Core Module sont fournis des documents centrés paragraphes. En sortie du DRUID Core Module, on trouve des instructions au SGBD pour mettre à jour la base de données. En interne au DRUID Core Module, des documents centrés données sont construits (voir figure 2). Ces documents sont ajoutés au fond documentaire et pourraient être utilisés comme source de données, et alors être directement interrogés avec les langages d'interrogation dédiés à XML.

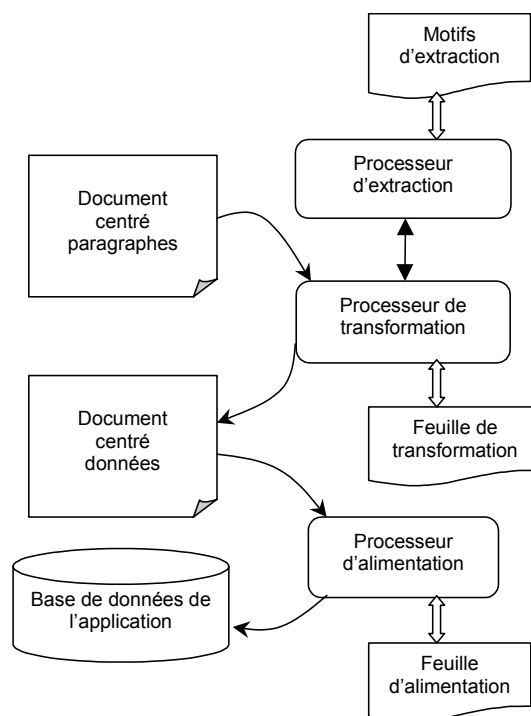


Figure 2 : architecture du DRUID Core Module

3.1. Processeur de transformations

3.1.1. Présentation du processeur de transformations

Le processeur de transformations analyse les paragraphes balisés d'un document centré paragraphes et construit le document centré données correspondant. Le processeur exécute les instructions qui se trouvent dans une feuille de

transformations. Une feuille de transformations contient une séquence de règles qui décrivent comment transformer un document centré paragraphes en un document centré données. Le document centré données ainsi construit est stocké dans le fond documentaire et transmis au processeur d'alimentation.

Pour chaque DTD de documents centrés paragraphes, il existe une feuille de transformation et une DTD de documents centrés données. Dans une feuille de transformations, on distingue quatre types de règles :

- Les règles de sélection permettent de repérer, à l'aide des balises, les paragraphes dans un document centré paragraphes,
- Les règles d'extraction indiquent quels motifs d'extraction utiliser et les types de retour attendus. Ces règles permettent d'activer le processeur d'extraction, et de récupérer ses résultats,
- Les règles de manipulation permettent de manipuler les informations extraites en appliquant des opérations arithmétiques, chaînes de caractères ou booléens,
- Les règles de restructuration permettent de construire un document conforme à une DTD de document centré données en utilisant des opérateurs de manipulation d'arbres.

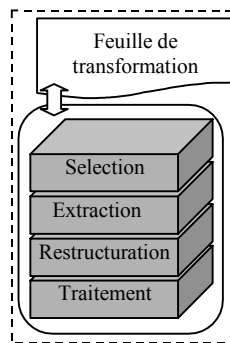


Figure 3 : Processeur de transformations

Nous avons choisi d'implanter le processeur de transformations comme une extension d'un processeur XSLT (Clark, 1999). Ainsi, nous avons pu reprendre un outil existant et ne développer que la partie spécifique supplémentaire.

3.1.2. Règles des feuilles de transformation

Les règles sont écrites dans une extension du langage de description des règles XSL. La figure 4 donne l'extension de la feuille de style XSL pour les feuilles de transformation. Une feuille de transformations contient un élément *transform* par type de paragraphe. L'attribut *paragraph* de cet élément définit la balise XML des paragraphes concernés, et constitue donc la règle de sélection. Le corps de chaque élément *transform* peut contenir une ou plusieurs règles de restructuration pour

ajouter des éléments XML et attributs, une ou plusieurs règles de manipulation pour traiter les informations balisées, et des règles d'extraction associées au paragraphe en question. La figure 5 montre la syntaxe formelle des règles d'extractions (*extract*) et ses sous-éléments (*slot*). Il existe deux variantes de l'élément *extract* ; dans le cas où l'extraction retourne une seule valeur, l'élément *extract* est réduit à un élément XML vide ; dans le cas où l'extraction retourne plusieurs valeurs, l'élément *extract* contient un élément *slot* par information retournée. Un élément *extract* possède une liste d'attributs : l'attribut *pattern* spécifie le nom du slot à retourner, l'attribut *lookupkey* retourne l'ID du slot (au lieu de sa valeur) en consultant le dictionnaire, l'attribut *taggable* permet de baliser ou non la valeur retournée, et enfin l'attribut *att* met la valeur d'un slot comme attribut dans l'élément XML qui l'engendre.

```

<!-- Category: top-level-element -->
<ext:transform paragraph = qname >
    un ou plusieurs règles d'extractions, règles de manipulation, règles de
    restructuration
</ext:transform>

<!-- Category: instruction -->
<ext:extract pattern = qname lookupkey = ('yes' | 'no') taggable = ('yes' | 'no') att = qname>
    un ou plusieurs éléments slot
</ext:extract>

<ext:slot name = qname    lookuokey = ('yes' | 'no')    taggable = ('yes' | 'no') />

<ext:construct frame = qname type = (' fregment' | 'defragment') >
    un ou plusieurs éléments extract
</ext:construct>

```

Figure 4 : extension de XSL pour la rédaction de feuilles de transformation

Le dernier élément, parmi les extensions proposées à XSL, est l'élément *construct*. Il permet soit de mettre dans le document centré données uniquement les données extraites, soit de conserver l'intégralité du texte du paragraphe en y ajoutant des balises repérant les données extraites.

La figure 5 illustre cette extension par un exemple de feuille de transformation, incluant règles de sélection et d'extraction.

```

....
< ext:transform paragraph="prescription">
  < ext:extract pattern="R_prescription">
    < ext:slot name=" dosage " />
    < ext:slot name=" unitéDosage " />
    < ext:slot name=" médicament " att="mID" />
    < ext:slot name=" médicament " />
    < ext:slot name=" durée " />

```

```

    < ext:slot name=" unitéDurée " />
  </ ext:extract>
</ ext:transform>
< ext:transform paragraph =" antécédent ">
  < ext:antécédent >
    < ext:extract pattern="R_ maladie" />
    < ext:extract pattern="R_ parenté" />
    < ext:extract pattern="R_ date" />
  </ ext:antécédent >
</ ext:transform >
.....

```

Figure 5 : Règles d’extractions extraites de la feuilles de transformation pour les paragraphes prescription et antécédent.

3.2. Processeur d’extraction

3.2.1. Présentation du processeur d’extraction

Nous avons choisi de réutiliser le moteur de traitement de la langue naturelle (NLP) Intex (Silberztein, 2000). Notre processeur d’extraction est une interface entre le processeur de transformations et le NLP. Il reçoit un paragraphe rédigé en langue naturelle et doit retourner des termes extraits du paragraphe grâce à l’application d’un motif d’extraction. Avant de faire effectuer l’extraction par le NLP, le processeur d’extraction lance des étapes préliminaires : il demande au NLP de faire (i) une validation du texte (que du texte et pas de caractères de contrôle), (ii) un pré-traitement pour identifier les phrases, délimiter les mots composés non ambigus, et marquer les mots contactés ou élidés, (iii) une analyse lexicale pour appliquer les dictionnaires de la langue et du domaine, et (iv) une désambiguïsation des mots correspondant à plusieurs entrées lexicales dans les dictionnaires. Après ces étapes préliminaires, le processeur d’extraction fournit au NLP le paragraphe à analyser et le motif d’extraction correspondant pour effectuer l’extraction proprement dite.

Les motifs d’extraction sont rédigés par un expert du domaine. Pour aider à la gestion de ces motifs, nous avons défini un langage de spécification à quatre niveaux que nous détaillons ci-après. Les motifs d’extraction sont très dépendants du domaine d’application, et nécessitent un travail préalable non négligeable. Ils intègrent des références à des dictionnaires de la langue française ainsi qu’à des dictionnaires de termes spécifiques au domaine. Ces motifs ne sont pas directement compréhensibles par le processeur de langue naturelle, mais doivent être transformées en transducteurs à états finis (FST). Il est difficile pour un humain de rédiger des FSTs pour des cas complexes, et le nombre de FST nécessaires est important. C’est pourquoi nous avons préféré définir un langage de spécification de

plus haut niveau et développer un compilateur permettant de traduire les motifs définis en FST. Les motifs d'extraction sont conservés pour réutilisation, les FSTs correspondants sont stockés dans un dictionnaire.

3.2.2. Langage de haut niveau pour la spécification de motifs d'extraction

Le langage de haut niveau que nous avons défini permet à l'utilisateur d'écrire simplement des motifs d'extraction d'information. En regard des classiques expressions régulières, ce langage ajoute les fonctionnalités suivantes :

- définir des méta-mots comme <maladie> pour faire référence à des dictionnaires de termes
- faire référence à un motif existant dans la définition d'un nouveau motif
- rédiger en plusieurs couches les motifs, et donc fournir un résultat plus lisible à l'utilisateur qui rédige ou réutilise des motifs.

Pour la rédaction des motifs d'extraction, nous avons défini quatre couches :

La couche 1 est composée d'un ensemble fini de *termes*. Un terme est soit une séquence de lettres délimitée par des séparateurs, soit un méta-mot décrivant un nombre <NB>, un mot <MOT>, un mot vide <E> ou une référence à un dictionnaire <dico>. Un terme peut être nommé, c'est-à-dire que le terme identifié doit être retourné balisé par le nom donné : medic [<medicamentDic>] indique que le terme du dictionnaire medicamentDic doit être balisé par le mot medic.

La couche 2 est composée d'un ensemble fini d'*expressions*. Une expression est une concaténation de termes séparés par des espaces ou des tabulations. On dit qu'une expression est valide dans un paragraphe si on y trouve une séquence de termes correspondant à cette expression.

La couche 3 est constituée d'un ensemble fini de *segments*. Un segment est une liste d'expressions alternatives, exprimant les différentes formes de rédaction d'un tel segment. On dit qu'un segment est valide dans un paragraphe si une de ses expressions est valide dans ce paragraphe.

La couche 4 est constituée d'un ensemble fini de *motifs*. Un motif est un ensemble fini non ordonné de segments. Certains segments sont obligatoires, d'autres sont optionnels. Des contraintes sur les segments peuvent être indiquées, pour réduire le nombre factoriel de combinaisons possibles (cardinalité, ordre entre n segments...). On dit qu'une règle est valide pour un paragraphe si l'ensemble de ses segments obligatoires est valide dans ce paragraphe.

Par exemple, pour définir un motif d'extraction de prescription (figure 6), on définit (i) un motif (pattern) indiquant qu'un paragraphe de prescription contient les segments (slots) dosage, médicament, et optionnellement les segments fréquence, durée et unité pour la durée, (ii) une ou plusieurs expressions indiquant la définition de chaque segment avec ses différentes formes d'écriture possibles. Le segment dosage indique qu'il y a trois expressions possibles pour repérer le dosage tel que

'½ comprimé', '1 cachet', '1 et ½ gélule' etc. Les termes comprimé, cachet et gélule sont recensés dans un dictionnaire et repérés par le meta-mot <unité>. De même, le meta-mot <période> regroupe les termes tels que jour, semaine, mois, etc.

```

expression expDosage1= dosage [<NB> "/" <NB>] <unité> ;
expression expDosage2= dosage [<NB> "et" <NB> "/" <NB>] <unité> ;
expression expDosage3= dosage [<NB>] <unité> ;
slot dosage= expDosage1 | expDosage2 | expDosage3 ;
expression expuDosage1= <NB> "/" <NB> unitéDosage[<unité>];
expression expuDosage2= <NB> unitéDosage[<unité>] ;
slot unitéDosage = expuDosage1| expuDosage2;
slot médicament = médicament[<medical>] ;
slot durée = 'pendant' durée [<NB>] <période> ;
slot unitéDurée = 'pendant' <NB> unitéDurée[<période>] ;
...
pattern prescription = dosage, médicament, durée?, unitéDurée?, fréquence? ;

```

Figure 6 : modèle d'un motif d'extraction pour le paragraphe prescription

3.2.3. Transformation des motifs en FST

Le NLP que nous avons choisi pour l'implantation de notre système utilise les transducteurs à états finis (FST) comme format d'entrée des motifs d'extraction. Ces FST sont abrupts à lire pour des cas non triviaux, et c'est pourquoi nous avons préféré fournir à l'utilisateur concepteur un langage de haut niveau pour la rédaction des motifs. La traduction de motifs en FST nécessite l'utilisation d'un compilateur. Nous avons développé un tel compilateur. Il prend en entrée des motifs dans le langage de spécification et retourne en sortie les mêmes motifs traduits en FST. Ce compilateur suit les étapes classiques d'analyse lexicale, syntaxique et sémantique. Il lit les scripts contenant les spécifications des motifs et compile chaque motif d'extraction en un FST en utilisant une procédure récursive bottom up et ceci en incluant à chaque couche les FSTs de couche inférieure : une FST est écrite pour chaque terme, puis pour chaque expression (les FSTs des termes inclus sont mis en série), puis pour chaque segment (les FSTs des expressions correspondantes sont mis en parallèle), et enfin pour chaque motif (on met en parallèle tous les ordres possibles de segments). La figure 7 présente le FST correspondant à la traduction du segment dosage rédigé dans notre langage de spécification à quatre couches.

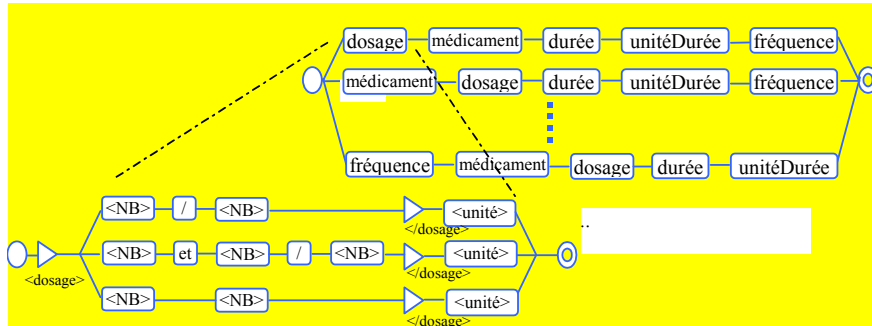


Figure 7 : FST correspondant au segment Dosage de la figure 6

3.3. Processeur d'alimentation

Le processeur d'alimentation permet d'enregistrer les données des documents centrés données dans la base de données relationnelle. Nous avons réutilisé le travail de R. Bourret (XML-DBMS). Ce processeur utilise des règles d'alimentation écrites dans un format XML pour construire une vue objet du document centré données et d'autres pour relier cette vue au modèle de la base de données. Pour plus d'informations, voir (Bourret, 2000). La figure 8 donne un exemple de feuille d'alimentation pour l'exemple des prescriptions.

```

<ClassMap>
  <ElementType Name= "prescription" />
  <ToClassTable> <Table Name="PrescriptionTable" /> </ToClassTable>
  <RelatedClass >
  <PropertyMap>
    <Attribute Name=" dosage "/>
    <ToColumn> <Column Name="dosage-column"/> </ToColumn>
  </PropertyMap>
  <PropertyMap>
    <Attribute Name=" unitéDosage "/>
    <ToColumn> <Column Name="udosage-column"/> </ToColumn>
  </PropertyMap>
  <PropertyMap>
    <Attribute Name=" médicamernt "/>
    <ToColumn> <Column Name="Med-column"/> </ToColumn>
  </PropertyMap>
  ....
  
```

Figure 8 : Exemple de feuille d'alimentation pour les prescriptions

L'extrait de la feuille d'alimentation montre que l'élément XML *prescription* est relié à la table *PrescriptionTable* dans la base de données et que les éléments XML *dosage*, *unitéDosage*, *médicament...* sont reliés respectivement aux colonnes *dosage-column*, *udosage-column* et *Med-column* dans la table *PrescriptionTable*. Les données extraites du document centré données de la figure 1 alimentent ces colonnes avec les valeurs (3, comprimé, aspirine).

4. Prototype

Nous avons développé un prototype qui intègre la saisie, la transformation et l'alimentation des données. Il est écrit en java et se fonde sur les standards XML. Dans cette section, nous présentons rapidement l'implantation de chacun des modules.

- *Interface graphique documentaire* : L'interface utilisateur est écrite en java et utilise la bibliothèque Swing pour les interactions. Elle est écrite avec la technologie Enterprise Java Beans et fonctionne sur une plate-forme weblogic. Les opérations XML de manipulation et validation de documents utilisent le standard DOM, et plus particulièrement l'implantation Apache Xerces (Xerces). Pour plus d'informations, voir (laforest et al., 2002, Badr et al., 2001).

- *Processeur de transformations* : Le processeur de transformations est construit sur XSLT. Le processeur Saxon (Kay, 2002) a été utilisé. Nous l'avons étendu pour intégrer les spécifications des règles des feuilles de transformation. Ce processeur de transformations utilise également JAXP1.1 (JAXP) pour manipuler les documents centrés paragraphes et les documents centrés données. Les fonctionnalités de parsing sont effectuées à l'aide de l'implantation Apache Xerces.

- *Processeur d'extraction et processeur de langue naturelle* : Nous avons choisi le processeur de langue naturelle INTEX (Silberztein, 1993) car il permet de créer des dictionnaires personnels et donc d'intégrer des mots techniques du domaine cible, mais également parce qu'il permet de spécifier des règles d'extraction contenant des termes nommés. De plus, il utilise les FST aussi bien pour décrire les motifs d'extraction que pour modéliser les dictionnaires. Le processeur d'extraction est rédigé dans le langage Java.

- *Compilateur de spécifications de motifs en FSTs* : Nous avons développé un compilateur en java pour transformer les spécifications de motifs d'extractions en quatre couches en finite state transducers. Ce compilateur se fonde sur la bibliothèque JavaCC (Sun, 1997).

- *Processeur d'alimentation* : Comme indiqué précédemment, nous avons réutilisé le middleware XML-DBMS (Bourret, 2000) pour effectuer l'alimentation de la base de données à partir d'un document centré données.

Le prototype a été testé sur une base documentaire constituée de 18 documents centré paragraphes dans le domaine de la cardiologie. Nous avons conçu 60 motifs d'extraction qui couvrent la recherche des informations vitales telles que : les

résultats des examens cliniques, les symptômes dans les ECG, échographies, l'historique du patient, la prescription, et les coordonnées du patient, de la clinique et du médecin traitant. Les règles d'extraction écrites selon le langage de spécification génèrent après compilation 463 FST (expressions et slots).

5. Positionnement par rapport à l'état de l'art

Les recherches les plus proches de notre travail concernent les wrappers. De nombreuses études ont été menées sur le sujet pour la construction de wrappers de pages HTML (Crescent *et al.*, 2001), de documents XML (Muslea, 1999), ou pour les textes en langue naturelle (Agichtein *et al.*, 2000). Cependant, la plupart des wrappers ne parviennent pas à effectuer une extraction efficace des données dans le cas où la structure n'est pas complètement régulière. Or dans les applications réelles, il est difficile de trouver des formes syntaxiques régulières, comme des délimiteurs de portions de texte ou de définir des connaissances linguistiques complètes pour extraire correctement des données.

A l'opposé, notre processeur d'extraction permet d'un part la recherche d'informations dans des paragraphes narratifs ou de style télégraphique et d'autre part a recours à des dictionnaires linguistiques et techniques, ce qui augmente le taux d'extraction des données.

Un autre domaine de recherche ayant trait à notre problématique concerne l'interrogation et la gestion de données dans des documents XML, et plus précisément dans des documents centrés données. Cela consiste la plupart du temps en la mise en correspondance de données irrégulières avec un schéma traditionnel de base de données. Ce type de mapping est proposé par de nombreux systèmes, tels Stored (Deutch *et al.*, 1999), UnQL (Buneman *et al.*, 1996), Lorel (Abiteboul *et al.*, 1997), struQL (Fernandez *et al.*, 1998) ou OQL-Doc (Abiteboul *et al.*, 1996).

Dans notre approche, nous utilisons une base de données relationnelle et pouvons donc profiter de la maturité de SQL pour manipuler les données avec des requêtes complexes (étude statistique, analytique, mining, etc) et pour sélectionner les documents correspondants.

De nos jours, les documents centrés paragraphes sont de plus en plus utilisés. La structure de tels documents définit des régions d'intérêt et les données pertinentes extraites de ces régions doivent être conformes à une sémantique précise. Peu de systèmes ont été conçus pour la gestion de tels documents. NoDoSE (Adelberg, 1998) fournit un système semi-automatique pour l'extraction de données dans des instances d'un type de document. Il supporte complètement les documents en texte libre, les documents HTML et en partie les documents XML centrés paragraphes. Les données extraites sont stockées dans un schéma relationnel traditionnel. Sur la base d'ontologies, (Embley *et al.*, 1998) formule des règles permettant l'extraction

de données pertinentes et applique un système de reconnaissance pour construire des valeurs de tuples d'attributs qui sont insérés dans un schéma de base de données. Cette approche est limitée à des ontologies relativement petites et à des ensembles de mots clés petits.

Notre système partage avec NoDoSE la même vision basée sur l'extraction des données à partir de documents pour alimenter une base de données. Mais les deux systèmes restent différents : nous utilisons les documents comme interface de saisie et alimentons une application existante, alors que NoDoSE n'alimente pas une base de données, il présente les données extraites sous forme structurée dans des fichiers. Bien que notre travail soit basé sur des dictionnaires pour extraire les informations, le travail d'Embly est plus ambitieux : il utilise des ontologies pour modéliser une application et ensuite pour générer automatiquement les motifs d'extraction. Ces motifs ne couvrent pas bien la richesse du langage humaine ; l'intervention de l'être humain est indispensable pour lever les ambiguïtés.

Enfin, nous mentionnerons l'utilisation d'automates à états finis pour les processeurs de langue naturelle (Gross, 1989) et pour l'extraction d'information. De nombreuses propositions d'outils ont été faites dans la communauté linguistique, tels FS5 (Karttunen *et al.*, 2001) et Xerox (Karttunen, 2000). Dans la communauté française, INTEX (Silberstein, 1993) semble être l'outil le plus utilisé.

6. Conclusion

Le système que nous proposons associe une base documentaire à une base de données. Il définit un outil permettant à l'utilisateur de travailler sur des documents, sans se préoccuper de la manière dont les données sont intégrées dans la base de données. Ce système permet d'analyser des documents semi-structurés, afin d'en extraire les informations qui doivent être stockées dans la base de données. Pour cela, il s'appuie sur des règles de transformation, des règles d'alimentation et des motifs d'extraction. Des requêtes sur la base de données permettent de retrouver les documents, ou d'effectuer des traitements statistiques sur les données. La technique proposée dans ce travail est générique et peut s'appliquer à de nombreux domaines, dès lors qu'ils manipulent des documents centrés paragraphes, et qu'une interrogation plus fine que par mots-clés est nécessaire. Les règles et les motifs nécessaires sont très dépendants de l'application cible et doivent être rédigés pour chaque implantation.

Les documents centrés données sont ajoutés au fond documentaire et pourraient être utilisés comme source de données, et alors être directement interrogés avec les langages d'interrogation dédiés à XML. Nous n'avons pas aujourd'hui implanté ceci, mais il semble qu'aucune difficulté technique n'est à redouter. Il serait cependant intéressant de le faire pour comparer l'utilisation d'une base de données relationnelle à l'utilisation d'une base de documents structurés.

Dans l'état actuel de notre travail, les règles d'alimentation sont rédigées manuellement par le concepteur des règles d'extraction. Nous travaillons à la rédaction automatique de ces règles. Ceci est relié à la génération automatisée des DTD de documents centrés paragraphes et centrés données à partir du schéma de la base de données, pour laquelle nous avons développé un algorithme (Laforest *et al.*, 2003).

Dans un avenir proche, nous voulons étudier plus précisément les possibilités d'automatisation de la construction des feuilles de transformation et d'alimentation. Des squelettes de ces feuilles pourraient certainement être produits automatiquement à partir du schéma de la base de données sous-jacente et ainsi réduire le travail de l'utilisateur chargé de l'adaptation du système à un domaine particulier.

Notre système ne permet pas de gérer des images, du son ou de la vidéo dans la base de données, mais se restreint pour l'instant à la gestion d'informations textuelles. Il permet d'intégrer une analyse des commentaires associés aux objets multimédia, mais ne permet pas une indexation du contenu de ces objets. Il serait cependant intéressant de coupler notre approche à des analyses d'images et de vidéo, et permettre ainsi une indexation par le contenu et par les commentaires. Une première étude a été menée dans ce sens (Badr *et al.*, 2003).

D'autres pistes d'étude peuvent également être explorées, comme la spécification de règles sémantiques pour la validation des données saisies (ne pas accepter une prescription du style « 3 cachets de sirop ») ou l'interrogation de la base par l'exemple, en fournissant un document modèle (il faudrait alors définir une distance entre documents pour retourner les documents les plus proches du document modèle).

7. Bibliographie

- Abiteboul S., Cluet S., Christophides V., Milo T., Moerkotte G., and Simeon J.. Querying Documents in Object Databases (OQL-Doc), 1996.
- Abiteboul S., Quass D., Widom J., and Wiener J.L.. The lorel query language for semi-structured data. International Journal on Digital Libraries 1997.
- Adelberg B. NoDoSE: a tool for semi-Automatically Extracing Structured and Semistructured Data from Text Documents. Proceeding of ACM SIGMOD international conference on Management of data, 1998, pp 283-94.
- Agichtein E., Eskin E. and Gravano L. Combining Strategies for Extracting Relations from Text. Collections, in the Proceedings of the 2000 ACM SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD 2000)
- Badr Y., Chbeir R., Flory A. Toward an automatic image description. Proc. Int conf. on computer science, software engineering, information technology, e-business and applications CSITeA'03 pp. 242-7, 2003
- Badr Y., Sayah M., Laforest F., Flory A. Transformation rules from semi-structured XML documents to database model. AICSSA 2001, Beirut, Lebanon. June 26-29, 2001

- Bourret, R., Bornhövd, C., Buchmann, A.P.: A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases. 2nd Int. Workshop on Advanced Issues of EC and Web-based Information Systems (WECWIS), San Jose, California, June, 2000
- Buneman P., Davidson S., Hillebrand G., and Succiu D. A query language and optimization techniques for unstructured data (UnQL). In Proc. SIGMOD Conf., 1996.
- Clark J. (ed.) "XML transformations (XSLT) version 1.0" W3C, November 1999, <http://w3c.org/TR/xslt>
- Crescenzi V., Mecca G., Merialdo P. RoadRunner: Towards Automatic Data Extraction from Large Web Sites, 27th International Conference on Very Large Databases 2001.
- Deutsch A., Fernandez M., Suciu D. Storing semistructured data with STORED, In Proc of the ACM SIGMOD International Conference on Management of Data , 1999.
- Embley D., Campbell D., Smith R., Liddle S.: Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents. CIKM 1998: 52-59
- Fernandez M., Florescu D., Kang J., Levy A., Suciu D. Catching the boat with Strudel: experiences with a web-site management system. In Proc of ACM-SIGMOD Int. Conference on Management of Data , 1998
- Gross M. The Use of Finite Automata in the Lexical Representation of Natural Language. Electronic Dictionaries and Automata in Computational Linguistics. In Lecture Notes in Computer Science, pages 34-50, Springer-Verlag, Berlin, Germany, 1989.
- JAXP Java Community ProcessSM. Java APIs for XML parsing (JAXP). Project home page. http://java.sun.com/xml/xml_jaxp.html.
- Karttunen L., Koskenniemi K., Van Noord G. (editors), Finite State Methods in Natural Language Processing. FSMNLP 2001. Extended Abstracts. ESSLLI Workshop, Helsinki 2001
- Karttunen L.. Applications of Finite-State Transducers in Natural Language Pr of CIAA-2000. Lecture Notes in Computer Science. Springer Verlag.
- Kay M. SAXON XSLT Processor 7.2, <http://saxon.sourceforge.net/> , August 2002.
- Laforest F., Boumediene M. Study of the automatic construction of XML documents models from a relational data model. WebS workshop, DEXA'2003, Prague, sept 2003
- Laforest F., Flory A. Using Weakly Structured Documents at the User-Interface Level to Fill in a Classical Database. Chapter in: Advanced Topics in Database Research, K. Siau (ed.) Idea Group Publishing, 2002.
- Muslea I. Extraction patterns for information extraction tasks: A survey. In AAAI-99 Workshop on machine learning for information extraction, 1999.
- Robie J., Lapp J., Schach D. XML Query Language (XQL). <http://www.w3.org/TandS/QL/QL98/pp/xql.html>, 1998
- Silberztein M. Dictionnaires Electroniques et Analyse Lexicale du Français--Le Système INTEX, Masson, Paris, France. 1993.
- Silberztein M. INTEX: an FST toolbox. Theoretical Computer Science, (234)33-46. 2000.
- SunTest JavaCC - A Java Parser Generator, 1997
- Xerces Apache XML Project. Xerces XML Parser for Java. <http://xml.apache.org/xerces-j>