Document inquisitor : un système de validation des structures et d'élicitation de modèles de documents

Florian Evéquoz, Maurizio Rigamonti, Denis Lalanne et Rolf Ingold

DIUF. Département d'Informatique de l'Université de Fribourg
Bd de Pérolles 90
CH-1700 Fribourg, Suisse
{prenom.nom}@unifr.ch

Abstract :

This paper introduces *document inquisitor*, a tool for the validation and correction of the results of document analysis systems. It presents the architecture of the system focusing on its internal data representation and exposes interactive features of the tool which are divided into two categories: (1) validation of document structures and (2) document model elicitation using explicit links and entities.

KEYWORDS : document analysis, document structure, validation, ground truthing, document model.

1. Introduction

Le groupe DIVA de l'Université de Fribourg travaille depuis plusieurs années sur le développement de techniques automatiques ou supervisées visant à reconstruire les structures physique et logique de documents. Des outils ont été développés dans le but de retrouver en plusieurs étapes, ces structures sous-jacentes et sont décrits notamment dans [Rig03], [Rig05b], [Had05] et [Blo06]. Néanmoins, le processus n'est pas trivial et les systèmes utilisés aux différents niveaux d'analyse ne parviennent pas toujours à des résultats optimaux. C'est pourquoi le besoin d'un outil léger permettant de valider et d'éditer facilement les résultats de différents niveaux d'analyse de documents s'est imposé et que l'application *document inquisitor* a été développée. En outre, certains systèmes d'analyse basés sur des règles, par exemple pour la détection de la structure logique d'un journal, requièrent l'appel à des modèles décrivant les caractéristiques fonctionnelles de la classe de document. Cependant, la définition de tels modèles n'est pas aisée et nécessite la prise en compte du jugement de l'utilisateur. *Document inquisitor* utilise un paradigme basé sur des liens explicites pour représenter les modèles de documents, permettant ainsi de les éliciter ou de les faire apparaître, par la manipulation et l'expérimentation.

Un état de l'art récent des outils de validation de structure a été réalisé dans [Yac05]. Celui-ci donne un aperçu général de plusieurs systèmes, dont notamment xmillum [Rig03] qui a directement inspiré *document inquisitor*. PerfectDoc, système présenté dans l'article, est globalement le plus abouti de ces outils. *Document inquisitor* se distingue de ce dernier sur plusieurs points. Tout d'abord, l'utilisation d'un langage interne de représentation permet à *document inquisitor* de supporter différents formats de données issues de l'analyse de documents, moyennant un mécanisme de transformations bidirectionnelles. Ensuite, la manipulation de la structure physique au moyen de *document inquisitor* ne comporte pas d'outils de haut niveau dédiés à la correction d'OCR, car il a été utilisé exclusivement avec des systèmes d'analyse de documents électroniques, tels que XED, qui n'utilise pas d'OCR et assure des résultats de reconnaissances de mots supérieurs à 99% dans le meilleur des cas [Rig05b], et ce sur divers types de documents (journaux, articles scientifiques, présentations, etc.). Enfin, *document inquisitor* assure une plus grande souplesse de représentation de la structure logique que PerfectDoc qui ne semble proposer que deux niveaux de structure logique, représentés de manière ambiguë par des codes de couleur sur les blocs physiques ne donnant pas d'informations quant à leur organisation hiérarchique. Ces différences peuvent être attribuées à une orientation différente des systèmes : alors que PerfectDoc a été optimisé pour une tâche très spécifique, la correction de l'extraction d'archives du magazine « Times », *document inquisitor* vise la généralité.

Cet article décrit l'architecture du système, particulièrement du point de vue du langage interne de représentation des données, et ses fonctionnalités de validation et de création de modèles. Après une présentation de l'héritage que *document inquisitor* doit à xmillum dans la section 2, la section 3 explore l'architecture du système, en mettant l'accent sur le langage interne de représentation des données de *document inquisitor* et ses avantages ainsi que sur le mécanisme de transformations bidirectionnelles. La section 4 décrit ensuite les fonctionnalités de l'application, tant au niveau de la correction et de la validation de la structure physique qu'à celui de la création de structures logiques et de l'élicitation de modèles de documents. La section 5 conclut l'article.

2. Xmillum et *document inquisitor*

Document inquisitor est le successeur de xmillum, dont il révise les concepts et l'architecture. Les deux systèmes proposent la même idée de base, c'est-à-dire de permettre aux chercheurs d'afficher, de valider et d'éditer n'importe quel type de résultats d'analyse de documents exprimés dans un format XML quelconque. Par contre, cette philosophie est adoptée d'une manière différente : xmillum privilégiait une architecture ouverte, permettant de configurer complètement l'affichage et le traitement des données à l'aide du langage xmi (*XMillum Internal*), tandis que *document inquisitor* définit le format IML (*Inquisitor Modeling Language,* voir 3.2) pour une configuration limitée de l'éditeur. Ce choix a l'apparence d'une régression mais est largement justifié par la difficulté de configurer xmillum de la part des utilisateurs, qui étaient obligés d'écrire des transformations XSLT définissant en même temps la configuration du système et le traitement des données en entrée. La figure 1 compare l'architecture des deux systèmes et notamment : (1) la transformation des données en entrée dans le format interne (flèches épaisses) ; (2) leur mani-

pulation par l'utilisateur à l'aide de l'éditeur (flèches fines à deux pointes) ; et (3) leur traitement après les tâches de validation et d'édition (flèches pointillées).

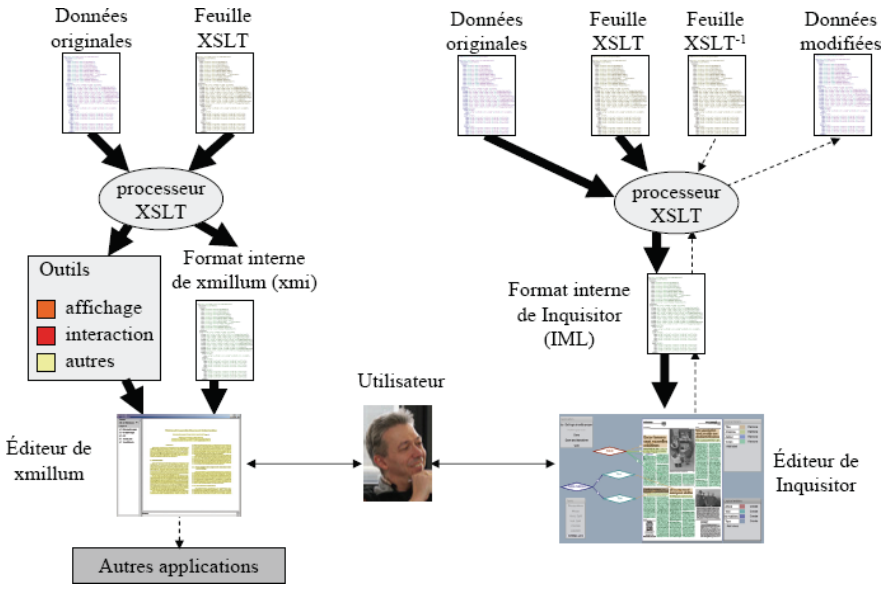

Figure 1 L'architecture de xmillum (gauche) et celle de *document inquisitor* (droite)

La figure 1 met en évidence un autre aspect qui différencie considérablement les deux systèmes : le cycle de vie des données après avoir été manipulées par l'utilisateur. Dans xmillum, les données n'étaient pas retransformées dans leur format d'origine et cette tâche était affectée soit à d'autres systèmes d'analyse de documents, soit à des extensions développées par les utilisateurs et qui étaient spécifiques au document XML en entrée. À l'opposé, *document inquisitor* propose un mécanisme de reconversion du document manipulé dans son format d'origine et résout un problème de transformation bidirectionnelle qui sera présenté à la section 3.4.

3. Architecture

Cette section introduit dans un premier temps les notions de structures et de modèles de document. Elle présente ensuite le langage de représentation interne de *document inquisitor*, IML, et décrit la transformation appliquée pour le générer, puis détaille les mécanismes mis en œuvre pour garantir la bidirectionnalité des transformations.

3.1 Structures physique et logique, modèle de document

Selon notre acception, la *structure physique* d'un document décrit la mise en page. Elle consiste en une hiérarchisation de ses primitives en entités homogènes du point de vue morphologique, topologique, et typographique. Les aspects morphologiques permettent de différencier entre elles des entités de type image, tableau, graphique ou contenu textuel. Une distinction morphologique est donc littéralement une dis-

tinction de forme, ou de « type de donnée ». Les aspects topologiques sont quant à eux issus de la disposition des éléments sur la page. Les aspects typographiques ont trait aux polices de caractères et aux différents choix de mise en forme.

La structure physique se prête à une représentation unique et non ambiguë pour tout document contenant du texte, des images et des composants graphiques. Récemment [Blo06] a proposé un ébauche de format canonique pour représenter cette structure, définissant un langage XML propre nommé XCDF (*eXhaustive Canonical Document Format*), qui a été appliqué à la représentation de documents PDF analysés et restructurés. Plusieurs niveaux hiérarchiques y sont distingués, dont les principaux sont présentés sur la figure 2. La structure physique canonique divise ainsi le document en pages, lesquelles comprennent des images, des graphiques et des blocs de texte. Ces derniers contiennent eux-mêmes des lignes de texte, qui rassemblent à leur tour des entités de type *token*, représentant les unités lexicales. La tâche d'un outil d'extraction automatique de la structure physique est de produire un fichier conforme au format canonique. Pourtant un certain nombre d'erreurs peuvent apparaître au cours du processus, qu'un outil tel que *document inquisitor* permet de corriger.

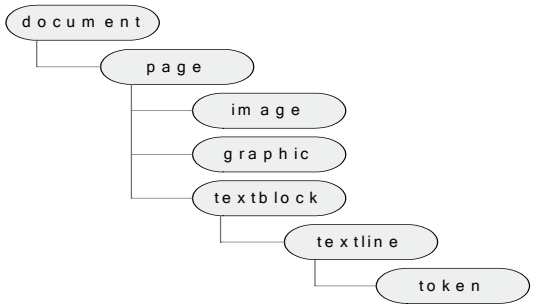

Figure 2 Schéma hiérarchique simplifié des éléments formant la structure physique canonique d'un document statique en XCDF

La *structure logique* d'un document peut être définie comme une hiérarchisation du contenu du document selon un modèle propre à la classe de documents à laquelle il appartient. Alors que la structure physique détermine des entités physiques cohérentes, la structure logique caractérise le rôle de celles-ci et décrit leurs relations, en particulier leur organisation hiérarchique [Sum95]. La reconstruction de la structure logique d'un document à partir de sa forme physique canonique comporte deux niveaux distincts : le premier consiste en un étiquetage des blocs physiques, le second en une projection des blocs étiquetés sur un modèle de document.

Le premier niveau, l'étiquetage, permet de caractériser le rôle logique de chaque entité physique. Chaque bloc de texte reçoit une étiquette, choisie parmi un ensemble déterminé, propre à une certaine classe de documents. Dans le cas d'un journal par exemple, les blocs de texte seront étiquetés comme **titre**, **chapeau**, **corps d'article**, (les caractères gras différencient dans la suite de l'article les entités physiques des entités **logiques**), etc. Le second niveau de reconstruction logique consiste à regrouper les entités liées en éléments logiques cohérents du point de vue de la classe de document considérée. Dans l'exemple du journal, le **titre** de l'article et le **corps d'article** associé constituent une nouvelle entité logique d'ordre supérieur : l'**article**. De même, le **titre de rubrique**, l'**article** qui vient d'être défini et les autres **article**s de la rubrique forment la **rubrique**.

Enfin, si la structure physique de tout document peut être représentée dans un format canonique unique, à l'inverse la structure logique reflète des informations propres à une catégorie de documents aux propriétés similaires, nommée classe de documents. A titre d'exemple, nous pouvons affirmer de manière quelque peu simplificatrice que les journaux forment une classe de documents. Cette classe propose une division en **sujet**s, **rubrique**s, **article**s, **titre**s, **paragraphe**s, etc., hiérarchisés selon des règles précises définies dans un modèle de journal. En toute généralité, la structure logique d'un document sera donc conforme à un modèle propre à la classe de documents à laquelle il appartient.

3.2 Représentation des données : le format IML

La structure de données interne de *document inquisitor* a été définie de manière à être exprimable dans un langage XML propre appelé IML (*Inquisitor Modeling Language*). La tâche première de *document inquisitor* étant de représenter des données visualisables issues de l'analyse de documents, la spécification du langage IML a été conçue de manière à faciliter l'affichage de celles-ci. Ses primitives les plus importantes sont donc les suivantes :

1) boîte (`<box>`)
2) couche (`<layer>`)
3) lien (`<link>`)
4) entités (`<entity>`)

La boîte est l'élément de base de représentation de la structure physique. Elle correspond à un simple rectangle positionné dans le document, doté de coordonnées *(x,y)* et de dimensions *(w,h)*. Elle peut également posséder d'autres propriétés optionnelles, comme un contenu textuel par exemple. Les couches sont des « conteneurs » de boîtes, de liens, d'entités, ou d'autres couches. Elles peuvent être affichées ou masquées à la demande. Les liens permettent de joindre entre elles des boîtes et/ou des entités. Les entités, enfin, représentent des éléments porteurs de propriétés, mais pour lesquels il n'est pas pertinent de spécifier une position physique dans le document. Les éléments appartenant à la structure logique du document seront typiquement spécifiés au moyen d'entités.

Par exemple, pour décrire les structures physique et logique d'un journal dans le langage IML, les différentes primitives sont organisées comme suit. Les entités physiques (unités lexicales, lignes, blocs de texte, images dans la terminologie XCDF) sont représentées par des boîtes. Celles-ci sont regroupées en couches. Toutes les images partagent la même couche, tous les blocs de textes également, et ainsi de suite ; dans une structure hiérarchique analogue à celle du format canonique, la couche des mots est contenue dans celle des lignes, et cette dernière appartient à son tour à celle des blocs (cf. figure 3). Il serait possible de faire de même avec les résultats de la reconnaissance symbolique d'une carte : des boîtes engloberaient les segments, les symboles, les noms de lieux, et seraient regroupées dans différentes couches selon leur type ou leur niveau d'abstraction.

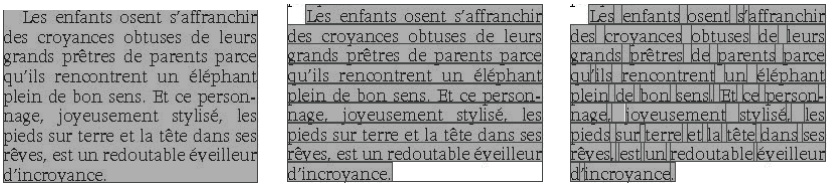

Figure 3 Affichage respectif des couches correspondant aux blocs de texte, aux lignes, et aux mots. La couche parente est visible en filigrane sur la deuxième et la troisième image.

La représentation de la structure logique utilise une couche supplémentaire réservée. Les entités logiques, qui peuvent être par exemple de type **article**, **titre**, **date**, etc. dans le cas d'un journal, possèdent des liens vers les blocs physiques correspondants, ou vers d'autres entités (cf. figure 4). Typiquement, une entité logique représentant un **corps d'article** aura des liens vers un certain nombre de blocs physiques englobant son contenu. Par contre, celle représentant un **sujet** pointera vers des **articles** thématiquement proches, qui sont eux-mêmes des entités logiques.

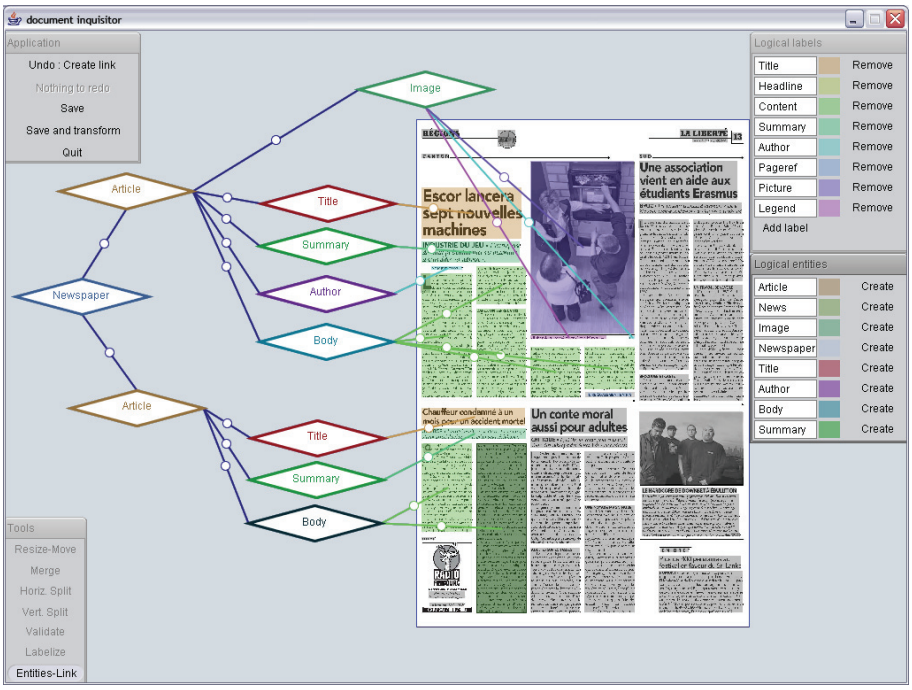

Figure 4 Aperçu de la structure logique partielle d'un document, avec entités et liens

Finalement, il faut encore signaler dans la structure IML interne l'utilité de deux listes globales : la liste des catégories d'étiquettes logiques et la liste des classes d'entités logiques disponibles. En effet, comme la structure logique d'un document s'exprime en deux niveaux, i.e. l'étiquetage des blocs physiques et leur regroupement en entités logiques de plus haut niveau, il faut disposer d'étiquettes et de classes d'entités propres à la classe de document considérée, d'où l'utilité de ces deux listes. Par ailleurs, ces listes sont directement dépendantes du modèle de document courant. Elles apparaissent dans l'application sous la forme de deux panneaux dis-

tincts représentés sur la figure 5. La section 4.3 traite plus particulièrement de la création de modèles de document au moyen de *document inquisitor*.

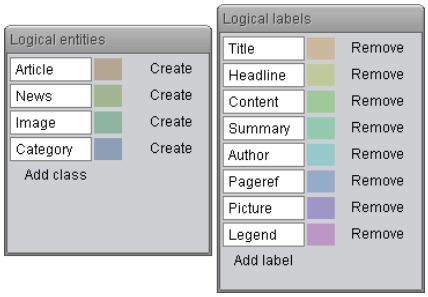

Figure 5 Les listes d'entités et d'étiquettes logiques du modèle de document courant.

3.3 Intérêt du format IML

Document inquisitor aspire à manipuler des résultats d'analyse quelconques. Son format interne doit donc être générique et aussi simple que possible pour faciliter la transformation depuis n'importe quel format de description de document. Pour permettre d'éditer facilement différents formats de données, il est en effet confortable de les transformer dans un langage générique. Xmillum a ouvert cette voie, mais souffre de la trop grande complexité du langage xmi, qui requiert l'écriture de feuilles de style absconses et qui n'est pas prévu pour subir des transformations inverses. IML résout en ce sens les défauts de xmi.

Le premier avantage du langage IML est sa simplicité : il ne contient que les données à manipuler par l'application et non, comme c'est le cas de xmi, des constructions servant à représenter dans le langage-même les outils et le comportement de l'application. Deuxièmement, le langage IML est un langage dédié à la visualisation et à la manipulation de certaines structures : blocs, entités et liens. Il ne représente effectivement que les données utiles à cette fin et évite l'hétérogénéité de représentation pouvant apparaître dans les formats descriptifs originaux. Troisièmement, IML prévoit des constructions servant à isoler les données non éditées sans que le langage ait besoin d'être étendu. Ces constructions, présentées à la section 3.4, permettent de garantir la bidirectionnalité des transformations. Quatrièmement, le paradigme des entités et des liens, présenté en 4.3, permet de rendre explicites les structures logiques d'un document. Enfin, le langage IML assure à *document inquisitor* sa polyvalence : un format quelconque de représentation de données peut être utilisé en entrée et transformé en IML pour être directement interprété par l'application sans que l'utilisateur ait besoin de modifier l'implémentation du programme. L'écriture de l'adaptateur est déléguée à une feuille de style XSLT. Cette généricité du langage IML a été appréciée durant le développement de *document inquisitor*. Grâce à de simples adaptations au niveau des feuilles de style, il n'a pas été nécessaire de répercuter les changements du langage XCDF, développé en parallèle, ni dans les spécifications du langage IML ni dans l'implémentation de *document inquisitor*.

3.4 Transformations bidirectionnelles : contraintes et exemple

La généralité donnée par la transformation d'un langage XML dans un autre, comme IML pour *document inquisitor*, est contrebalancée par la difficulté de retrouver un document conforme au format original à partir du document transformé. Cette réversibilité est pourtant nécessaire. En effet, si *document inquisitor* est utilisé pour valider ou corriger les résultats d'une extraction automatique de la structure physique, le fichier corrigé doit être disponible dans son format original pour un usage futur par exemple comme entrée d'un système de reconstruction de la structure logique ou en vue d'une intégration dans un système d'analyse tel que DocMining [Cla04]. Pour permettre une telle réversibilité, il faut néanmoins s'assurer que les processus de transformation dans un sens et dans l'autre n'induisent aucune perte d'informations. Deux contraintes d'ordre syntaxique sont à respecter pour garantir la bidirectionnalité des transformations :

1) Représentation, même implicite, de tous les éléments et attributs du langage original.
2) Représentation, même implicite, de la structure hiérarchique originale.

Pour résoudre la première contrainte, le langage IML met à disposition deux constructions *ignore* et *original* permettant respectivement d'ignorer des sous-arbres entiers du format original et de conserver le nom et les attributs d'éléments qui ne sont pas transformés dans un des quatre éléments principaux du langage IML décrits à la section 3.2. Pour que la deuxième contrainte de bidirectionnalité soit respectée, les éléments du format original doivent être pourvus d'un attribut *id* unique. Ainsi, lors de la transformation dans le langage IML, les éléments IML pointent vers leur parent dans la structure originale par le biais d'un attribut *parent*.

Pour illustrer le fonctionnement d'une transformation en IML et les exigences posées par les contraintes syntaxiques, nous allons prendre comme exemple la transformation d'un document XCDF en IML. Celle-ci procède selon les principes généraux suivants :

1) Chaque type d'élément XCDF amené à être représenté par des boîtes en IML (à savoir les éléments *textblock*, *textline*, *token* et *image*) se voit attribuer une couche propre. Les couches sont imbriquées de manière à représenter la structure hiérarchique.
2) Les éléments cités du format canonique sont transformés en boîtes. A chaque transformation, l'élément original est encapsulé au moyen de la construction *original*.
3) Les sous-arbres non traités sont isolés dans des nœuds *ignore*.
4) L'attribut *parent* garde la trace de la structure hiérarchique originale.

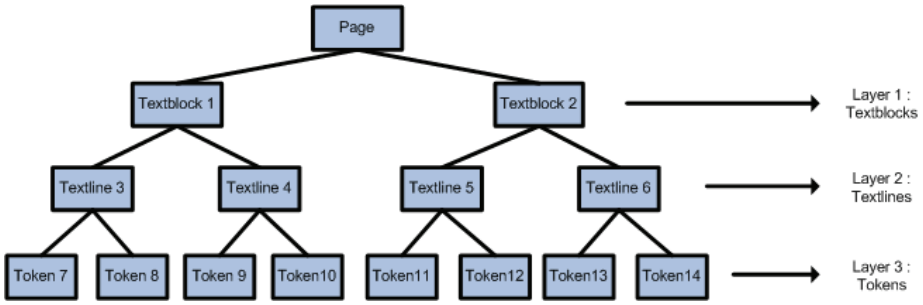

Figure 6 Un exemple de structure physique simple représentée dans le format canonique XCDF

La figure 6 montre un exemple de structure au format XCDF. Le résultat de la transformation de cette structure en IML est présenté sur la figure 7, qui indique par des traits pointillés la valeur de l'attribut *parent*. La structure canonique originale de la figure 6 apparaît visuellement par le biais des lignes pointillées.

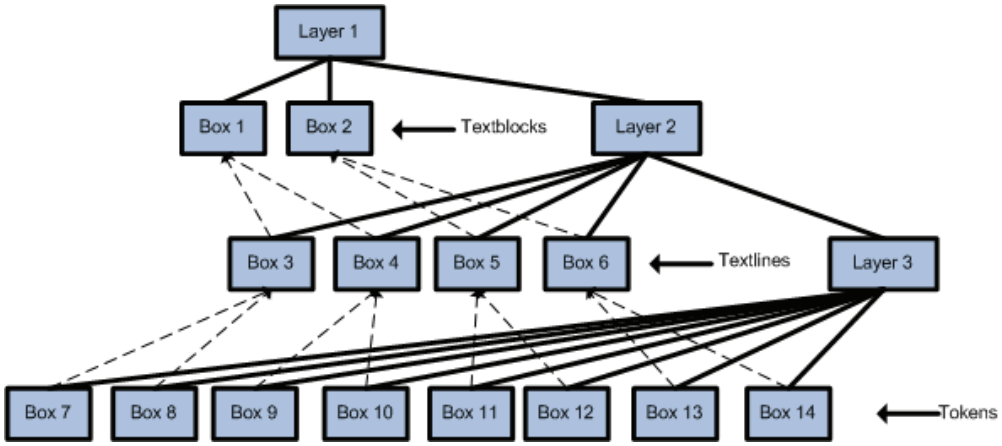

Figure 7 Structure interne correspondant à la structure au format canonique représentée à la figure 6. Les traits pointillés marquent la valeur de l'attribut *parent* du langage IML.

Nous avons vu que les unités lexicales sont représentées sous forme de boîtes dans le langage IML. Les seuls attributs requis pour une boîte définissent son placement physique (coordonnées et dimensions). Nous avons également indiqué qu'une boîte pouvait posséder un contenu textuel optionnel. Les autres attributs d'une unité lexicale ainsi que le nom de l'élément sont, eux, encapsulés dans un nœud de type *original*. Ainsi, l'élément suivant :

```
<token x=… y=… w=… h=… content=… fontsize="…"/>
```

est représenté dans le langage IML de la sorte :

```
<box x=… y=… w=… h=… content=…>
      <original>
```

```
            <element name="token"/>
            <attributes fontsize="…"/>
      </original>
</box>
```

Cette construction isole dans l'élément boîte lui-même les attributs directement requis par l'interface. Le noeud *original* garde quant à lui toutes les informations relatives à l'élément original, qui a été transformé en cette boîte, pour permettre de le reconstituer lors de la transformation inverse.

Enfin, les documents au format canonique XCDF contiennent des éléments tels que la définition des polices de caractères qui ne sont pas visualisés par *document inquisitor*. Néanmoins, pour respecter la première contrainte de bidirectionnalité, il est nécessaire qu'ils apparaissent dans la structure interne IML. A cet effet, les sous-arbres XML concernés (les nœuds *font* et leurs enfants dans le format canonique) sont encapsulés dans des nœuds de type *ignore* et représentés à l'identique dans le langage IML, de la sorte :

```
<ignore>
      <font …>
            <glyph …>
                  <name …/>
                  <code …/>
            </glyph>
      </font>
</ignore>
```

4. Fonctionnalités couvertes par *document inquisitor*

Document inquisitor est un outil de validation et de correction des résultats d'une extraction de structure, un outil dédié à la création de fonds de vérité et un outil permettant de découvrir et de créer des modèles de document. Cette section examine les mécanismes mis en œuvre pour remplir ces objectifs. Elle présente tout d'abord les opérations applicables au niveau de la structure physique du document et les contraintes que celles-ci doivent respecter pour garantir le respect de l'intégrité du document. Enfin, elle décrit les opérations relevant de la structure logique du document et permettant la mise en lumière de modèles de document.

4.1 Opérations sur la structure physique

La première fonctionnalité couverte par *document inquisitor* est une tâche de validation et de correction des résultats d'analyse de documents. Différents types d'erreurs peuvent en effet survenir lors d'une extraction automatique de structures de documents électroniques, en particulier des erreurs de sur-segmentation et de sous-segmentation. Pour corriger manuellement ces erreurs, *document inquisitor* met à disposition un certain nombre d'opérations sur les boîtes, qui se divisent en opérations simples (ne touchant qu'une boîte par opération) ou complexes (touchant plus d'une boîte). Cette section décrit brièvement les différentes opérations possibles.

Les opérations simples sur la structure physique n'ont à faire qu'à un seul élément, et ne modifient pas la structure arborescente des données, mais seulement les propriétés ou attributs existants. Elles sont au nombre de quatre. (1) La première opération simple fournie par *document inquisitor* est le redimensionnement et le déplacement d'une boîte. Du point de vue de la structure physique du document, cela revient à changer les coordonnées et/ou les dimensions d'un bloc mal reconnu. (2) L'opération de validation permet d'attribuer une valeur de validité à un bloc, qui peut être utilisée pour entraîner un système dans le cas d'un apprentissage supervisé. L'opération de validation peut être considérée comme un étiquetage de boîte pour lequel les seules étiquettes possibles sont « valide », « non valide » ou « indéterminé » tant que la validation n'a pas été effectuée. (3) L'édition manuelle des propriétés d'une boîte permet de modifier les propriétés optionnelles des boîtes, telles que le contenu textuel ou d'autres attributs directement hérités des éléments originaux dont elles sont issues. Cette opération permet d'éditer la valeur d'une propriété existante, mais pas d'ajouter de nouvelles propriétés ni d'en supprimer. (4) Finalement, la dernière opération simple est l'étiquetage des blocs physique. Il s'agit d'une opération d'annotation logique sur la structure physique d'un document. Elle sert à attribuer une étiquette à un bloc, spécifiant sa fonction logique. Les étiquettes peuvent être définies dans un modèle de document chargé au préalable (comme **titre** ou **contenu** peuvent être prédéfinies dans le modèle de document *journal*), ou peuvent avoir été définies manuellement par l'utilisateur.

La figure 8 montre une opération d'étiquetage, une opération de redimensionnement et une opération de validation. Le redimensionnement s'effectue par manipulation directe de l'enveloppe d'un bloc, alors que les opérations de validation et d'étiquetage sont effectuées par le biais de menus contextuels circulaires.

Figure 8 Opérations d'étiquetage logique (gauche), de redimensionnement (centre) et de validation (droite)

Les opérations complexes manipulent plusieurs éléments et modifient l'arborescence IML interne. (1) La fusion consiste à regrouper deux blocs en un seul. Elle est nécessaire pour corriger les problèmes de sur-segmentation. (2) La division consiste à partager un bloc en deux, horizontalement ou verticalement. Elle sert à corriger les cas de sous-segmentation.

4.2 Contraintes sur les opérations

Les opérations décrites dans la sous-section précédente ne doivent pas engendrer d'états dans lesquels la structure physique du document ne serait plus cohérente. Pour garantir l'intégrité de la structure physique certaines contraintes particulières doivent être respectées dans le langage IML :

1) Contrainte d'inclusion : un bloc fils est géométriquement contenu dans son bloc parent.
2) Contrainte de respect de l'ordre de lecture : l'ordre des blocs fils d'un bloc donné respecte l'ordre de lecture.

La première contrainte est toujours respectée. Ainsi, l'interface ne permet pas de redimensionner ou déplacer un bloc de telle manière que cette contrainte soit bafouée. Similairement, la division n'est possible qu'aux endroits compatibles avec le respect de cette contrainte.

Le respect de la deuxième contrainte en toute généralité est soumis à une analyse dédiée à la découverte de l'ordre de lecture qui n'a pas sa place dans un outil de validation tel que *document inquisitor*. Aussi, pour résoudre les conflits apparaissant lors d'opérations complexes de fusion ou de division de blocs, *document inquisitor* demande à l'utilisateur de spécifier explicitement l'ordre de lecture des blocs physiques concernés. Il faut enfin noter que les contraintes citées ici ne concernent que la structure physique du document, et que *document inquisitor* n'impose aucune contrainte sur la structure logique dont le fonctionnement est présenté dans la section suivante.

4.3 Le paradigme du lien, les entités et les structures logiques

Une des particularités de *document inquisitor* est la possibilité de créer des structures logiques en utilisant des liens explicites et des entités logiques. Ces dernières groupent différentes composantes du document dans un même ensemble logique, défini par une étiquette. La relation d'appartenance à un même ensemble logique est mise en évidence par les liens explicites, à leur tour de deux types : (1) les liens entre entités physiques et entités logiques et (2) les liens entre différentes entités logiques. Par exemple, dans le cas d'un journal, des liens de la première sorte regroupent les régions textuelles d'une page de journal en **article**s et des liens de la deuxième sorte permettent par exemple de définir une **table des matières** pointant vers les **titre**s des **articles** définis auparavant. Ces deux exemples sont explicités par la figure 9.

Figure 9 Exemple d'une entité article**, avec des liens vers la structure physique du document (haut), et d'une entité** table des matières**, dont les liens pointent vers deux entités** titre **(bas).**

Document inquisitor permet de créer de nouveaux liens et entités, ainsi que de définir une classe d'entités qui n'existe pas encore, pouvant être édités ou effacés dans un deuxième temps. L'intérêt majeur de cette fonctionnalité réside dans les tâches qui peuvent être accomplies :

- **validation de structures logiques** : les résultats de systèmes d'analyses logiques supervisés ou automatiques peuvent être validés ou corrigés à l'aide de *document inquisitor*, en éditant les étiquettes des entités et des blocs logiques et en modifiant leurs liens. Par exemple, l'utilisateur pourrait soit changer l'étiquette d'un **titre** en **chapeau**, soit affecter un bloc de texte à un **article** en déplaçant le lien qui pointait précédemment vers un autre article ;
- **création de données de fonds de vérités** : la rapidité de création de liens et de nouvelles instances d'entités permet à l'utilisateur de créer manuellement des archives de données hiérarchisées et étiquetées à utiliser comme fonds de vérités pour des systèmes d'apprentissage. Par exemple, l'ensemble d'entités attribuées par un système d'analyse à un type de document particulier pourrait être réutilisé en chaîne sur d'autres documents similaires afin d'incrémenter la consistance de la base de donnée ;
- **création de nouveaux modèles de document** : la possibilité de définir ses propres entités, de les intégrer dans une structure d'arbre et enfin de les lier à des blocs physiques implique que l'utilisateur puisse créer ses propres modèles de documents, à utiliser ensuite avec des systèmes de reconnaissance de structures logiques. Un utilisateur peut donc utiliser *document inquisitor* comme un langage graphique pour décrire la structure d'une page

de journal en définissant une hiérarchie d'**article**s, composée de **titre**s, **auteur**s, etc. Les processus de création et de regroupement d'entités permettent ainsi, à partir de la structure physique d'un document, de découvrir progressivement, ou d'éliciter, un modèle de structure logique pouvant par la suite s'appliquer à tous les documents de sa classe et pouvant également être partagé avec la communauté de recherche ou avec des utilisateurs du même métier.

5. Conclusion

Document inquisitor est un système permettant de valider et d'éditer les résultats d'analyses de documents. Il a été employé avec succès dans la correction d'une cinquantaine de documents au format XCDF extraits de journaux et de présentations PowerPoint, destinés à être utilisés dans des navigateurs multimédia [Rig05]. Cet article a présenté le système et ses spécificités, dont en particulier la capacité d'accepter en entrée divers formats XML transformés en un format interne nommé IML, de permettre à l'utilisateur d'éditer le document dans ce format et de restituer en sortie un document modifié conforme au format original. L'éditeur de *document inquisitor* permet de manipuler des résultats d'analyse et de créer de nouvelles données de fonds de vérité, un système de contraintes garantissant l'intégrité de la structure interne. D*ocument inquisitor* utilise par ailleurs des primitives simples pour la visualisation des données, en superposant des couches et des boîtes à l'image du document. En outre, la création de liens et d'entités au moyen de l'interface facilite l'élicitation et la définition de nouveaux modèles de documents.

Actuellement, *document inquisitor* est composé d'un processeur XSL pour la transformation dans le format IML et d'un éditeur spécialisé pour des documents contenant une majorité de régions textuelles. Les prochaines extensions prévues touchent à l'intégration d'outils d'édition dédiés aux graphiques et aux images, à l'élaboration des contraintes d'intégrité adaptées, et à la mise à jour du format IML pour le rendre plus flexible quant à la gestion des nœuds *ignore* et pour lui associer un espace de nommage adapté. Enfin, l'utilisation actuelle du système dans le cadre de la préparation de données destinées à des navigateurs multimédia suggère une évaluation de l'ergonomie de l'éditeur, de l'efficacité des contraintes dans la détection d'éventuelles d'erreurs et de la validité des transformations bidirectionnelles appliquées à de nouveaux formats de représentation.

6. Bibliographie

[Blo06] J.-L. Bloechle, M. Rigamonti, K. Hadjar, D. Lalanne, R. Ingold, *XCDF: A Canonical and Structured Document Format.* In Horst Bunke, A. Lawrence Spitz (eds.), LNCS: *"7th International Workshop, DAS 2006, Nelson, New Zealand, February 13-15, 2006, Proceedings"*, Springer-Verlag, vol. 3872, pp. 141-152.

[Cla04] E. Clavier, G. Masini, M. Delalandre, M. Rigamonti, K. Tombre, J. Gardes, *DocMining: A Cooperative Platform for Heterogeneous Document Interpretation According to User-Defined Scenarios.* In Josep Lladós, Young-Bin Kwon (eds.), LNCS: *"Graphics Recognition, Recent Advances and Perspectives, 5th International Workshop, GREC 2003, Barcelona, Spain,*

July 2003, Revised Selected Papers", Springer-Verlag Berlin Heidelberg, vol. 3088, ISBN:3-540-22478-5, Barcelona (Spain), 2004 , pp. 13-24.

[Had05] K. Hadjar, R. Ingold, *Logical Labeling of Arabic Newspapers using Artificial Neural Nets.* In proc. of 8th International Conference on Document Analysis and Recognition (ICDAR'05), Seoul (Korea), August 09 - September 01 2005 , pp. 426-430.

[Rig03] M. Rigamonti, O. Hitz, R. Ingold, *A Framework for Cooperative and Interactive Analysis of Technical Documents.* In proc. of 5th IAPR International Workshop on Graphics Recognition (GREC'03), Barcelona (Spain), July 30 - 31 2003 , pp. 407-414.

[Rig05] M. Rigamonti, D. Lalanne, F. Evéquoz, R. Ingold, *Browsing Multimedia Archives Through Intra- and Multimodal Cross-Documents Links.* In Steve Renals, Samy Bengio (eds.), LNCS: *"Machine Learning for Multimodal Interaction: Second International Workshop, MLMI 2005, Edinburgh, UK, July 11-13, 2005, Revised Selected Papers"*, Springer-Verlag, vol. 3869, pp. 114-125.

[Rig05b] M. Rigamonti, J.-L. Bloechle, K. Hadjar, D. Lalanne, R. Ingold, *Towards a Canonical and Structured Representation of PDF Documents through Reverse Engineering.* In proc. of 8th International Conference on Document Analysis and Recognition (ICDAR'05), Seoul (Korea), August 29 - September 01 2005 , pp. 1050-1054.

[Sum95] K. Summers, *Towards a Taxonomy of Logical Document Structures*, In *Electronic Publishing and the Information Superhighway*: proc. of the Dartmouth Institute for Advanced Graduate Studies (DAGS), 1995, pp 124-133.

[Yac05] S. Yacoub, V. Saxena, S. Nusrulla Sami, *PerfectDoc: A Ground Truthing Environment for Complex Documents.* In proc. of 8th International Conference on Document Analysis and Recognition (ICDAR'05), Seoul (Korea), August 29 - September 01 2005, pp. 452-457.