Projet Outilex - Rapport technique

IGM
période du 1er avril 2005 au 1er octobre 2006

24 novembre 2006

1 Introduction

Cette dernière période du projet Outilex a été consacrée aux différents points suivants :

- développement d'un analyseur de texte par grammaires locales pondérées
- développement d'applications de cet analyseur
- développement d'une interface graphique
- distribution et finalisation des outils
- publications et démonstrations

Les différentes distributions d'Outilex sont disponibles sur le cvs hébergé par Systran. Elles sont accessibles aux partenaires uniquement et le seront prochainement au public, moyennant une cotisation de 100 euros.

2 Rappels

2.1 Le projet Outilex

Le projet Outilex vise à mettre à la disposition de la recherche, du développement et de l'industrie une plate-forme logicielle de traitement des langues naturelles ouverte et compatible avec l'utilisation d'XML, d'automates finis et de ressources linguistiques. En raison de son ambition internationale, les partenaires du projet ont également participé aux efforts actuels de définition de normes en matière de modèles de ressources linguistiques. Le projet Outilex regroupe 10 partenaires français, dont 4 académiques et 6 industriels. Il est coordonné par l'IGM et financé par le ministère de l'Industrie dans le cadre du Réseau national des technologies logicielles (RNTL). Préparé sous la direction de Maurice Gross, il a été lancé en octobre 2002 et doit se terminer en octobre 2006. Outilex a donc

pour objectif de proposer des modules qui effectuent toutes les opérations fondamentales de traitements de texte écrit : traitements sans lexiques, exploitation des lexiques et des grammaires et gestion des ressources linguistiques. Outre le développement de la plate-forme libre, placée pour l'essentiel sous la responsabilité des partenaires IGM et Systran, le projet comporte la réalisation de démonstrateurs propriétaires. Les données manipulées à toutes les étapes du traitement sont structurées dans des formats XML et également dans des formats binaires plus compactes permettant des traitements plus efficaces ; les convertisseurs entre ces formats sont fournis par la plate-forme. Toutes les opérations sont implémentées dans des modules C++ indépendants qui interagissent entre eux à travers des flux XML. Chaque fonctionnalité est accessible aux programmeurs par une API et aux utilisateurs finaux par des programmes binaires.

2.2 Les développements déjà réalisés par l'IGM

- Opérations de base sur les automates
- Lexiques : formats (XML) et traitements (compression et indexation des lexiques)
- Traitement de textes avec lexiques (applications des lexiques sur des textes et construction de l'automate du texte)
- Traitement de textes au moyen de contraintes d'unification

3 Développement d'un analyseur de texte par grammaires locales pondérées (WRTN)

3.1 formalisme des WRTN

Le formalisme des grammaires utilisées dans Outilex appartient à la famille des réseaux de transitions récursifs (RTN) [Woods, 1970]. Une grammaire se présente ainsi comme une collection d'automates lexicaux récursifs. Chaque automate possède un nom unique ; leurs transitions sont étiquetées en entrée soit par le mot vide, soit par un masque lexical (symbole terminal) soit par le nom d'un automate de la grammaire (symbole non terminal). Les automates constituant la grammaire, peuvent également être des transducteurs c'est-à-dire comporter des sorties sous la forme de chaînes de caractères.

Ce formalisme, qui a été remis au goût du jour avec les systèmes Intex et Unitex, permet de construire des grammaires locales au sens de [Gross, 1993, Gross, 1997] est utilisé dans des situations variées : extraction d'informations [Poibeau, 2001, Nakamura, 2005], reconnaissance d'entités nommées [Krstev et al., 2005], identification de structures grammaticales [Mason, 2004, Danlos, 2005]... avec

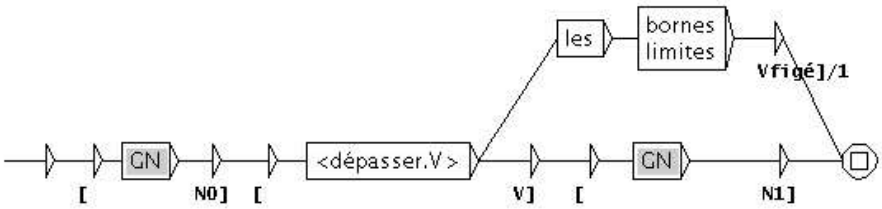

FIG. 1 – Exemple de grammaire WRTN : graphe principal

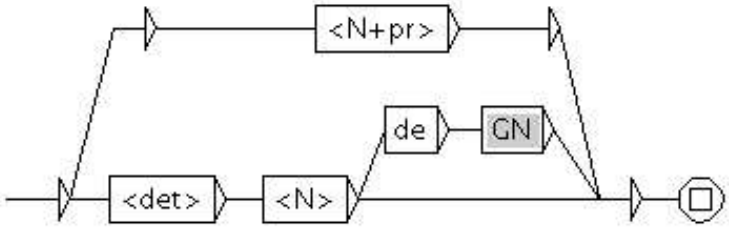

FIG. 2 – sous-graphe GN

pour chacune de ces applications des taux de rappel et de précision qui égalent l'état de l'art du domaine.

Nous avons ajouté au modèle la notion de pondération. Ainsi, chaque transition est également pondérée par un nombre réel, ce qui permet, dans le cas où une grammaire ambiguë fournirait plusieurs analyses, d'assigner des scores aux différents résultats et ainsi de réduire l'ambiguïté en préférant la ou les analyses ayant le score le plus élevé. Le système de pondération devrait permettre également la réalisation de systèmes hybrides utilisant à la fois des méthodes statistiques et des méthodes fondées sur des ressources linguistiques. Nous appelons le formalisme obtenu réseau de transitions récursif pondérées (WRTN).

Les figures 1 et 2 présentent un exemple de grammaire WRTN. Les symboles dans les boîtes grisées sont des symboles non terminaux correspondant à des appels à des sous-graphes. Dans le graphe principal, le poids[1] de 1 sur chemin correspondant à l'analyse avec expression figée, permet de rendre cette analyse prioritaire par rapport l'analyse compositionnelle décrite dans l'autre chemin (de poids 0 par défaut).

Ainsi par exemple, le résultat de l'application de cette grammaire sur les phrases suivantes :

Luc dépasse les limites.

[1]Le poids d'un chemin est la somme des poids de ce chemin. Par défaut, le poids est de 0.

Luc dépasse les limites de son art.
Luc dépasse le cap de Beauregard.

donnera les analyses suivantes :

[Luc N0] [dépasse les limites Vfigé].
[Luc N0] [dépasse V] [les limites de son art N1].
[Luc N0] [dépasse V] [le cap de Beauregard N1].

La première analyse ayant un score de 1 et les deux suivantes un score de 0.

3.2 format des grammaires locales

Les WRTN sont construits sous la forme de graphes à l'aide d'un éditeur (cf. section 5) et sont sauvegardés dans un format XML appelé xgrf, élaboré à partir de [Sastre, 2005]. Le graphe de la figure 1 est ainsi encodé de la manière suivante :

```
<?xml version="1.0" encoding="utf-8"?>
<xgrf>
  <header>
    <prop name="width" value="1188"/>
    <prop name="height" value="840"/>
    <prop name="font" value="Times New Roman:  10"/>
    <prop name="ofont" value="Monospaced:B 8"/>
    <prop name="bcolor" value="16777215"/>
    <prop name="fcolor" value="0"/>
    <prop name="acolor" value="13487565"/>
    <prop name="scolor" value="3355443"/>
    <prop name="ccolor" value="255"/>
    <prop name="dframe" value="y"/>
    <prop name="ddate" value="y"/>
    <prop name="dfile" value="y"/>
    <prop name="ddir" value="n"/>
    <prop name="drig" value="n"/>
  </header>
  <boxes number="14">
    <box id="0" x="48" y="200">
      <labels number="1">
        <label value="&lt;E&gt;"/>
      </labels>
      <out value=""/>
      <trans dest="12"/>
    </box>
```

```
<box id="1" x="546" y="200">
  <labels number="1">
    <label value="&lt;E&gt;"/>
  </labels>
  <out value=""/>
  <trans dest=""/>
</box>
<box id="2" x="183" y="200">
  <labels number="1">
    <label value="&lt;E&gt;"/>
  </labels>
  <out value="["/>
  <trans dest="3"/>
</box>
<box id="3" x="221" y="200">
  <labels number="1">
    <label value="&lt;dépasser.V&gt;"/>
  </labels>
  <out value=""/>
  <trans dest="4 5"/>
</box>
<box id="4" x="342" y="200">
  <labels number="1">
    <label value="&lt;E&gt;"/>
  </labels>
  <out value="V]"/>
  <trans dest="8"/>
</box>
<box id="5" x="373" y="126">
  <labels number="1">
    <label value="les"/>
  </labels>
  <out value=""/>
  <trans dest="6"/>
</box>
<box id="6" x="422" y="126">
  <labels number="2">
    <label value="bornes"/>
    <label value="limites"/>
  </labels>
  <out value=""/>
```

```
  <trans dest="7"/>
</box>
<box id="7" x="490" y="126">
  <labels number="1">
    <label value="&lt;E&gt;"/>
  </labels>
  <out value="Vfigé]/1"/>
  <trans dest="1"/>
</box>
<box id="8" x="378" y="200">
  <labels number="1">
    <label value="&lt;E&gt;"/>
  </labels>
  <out value="["/>
  <trans dest="9"/>
</box>
<box id="9" x="419" y="200">
  <labels number="1">
    <label value=":GN"/>
  </labels>
  <out value=""/>
  <trans dest="10"/>
</box>
<box id="10" x="493" y="200">
  <labels number="1">
    <label value="&lt;E&gt;"/>
  </labels>
  <out value="N1]"/>
  <trans dest="1"/>
</box>
<box id="11" x="109" y="200">
  <labels number="1">
    <label value=":GN"/>
  </labels>
  <out value=""/>
  <trans dest="13"/>
</box>
<box id="12" x="77" y="200">
  <labels number="1">
    <label value="&lt;E&gt;"/>
  </labels>
```

```
      <out value="["/>
      <trans dest="11"/>
    </box>
    <box id="13" x="149" y="200">
      <labels number="1">
        <label value="&lt;E&gt;"/>
      </labels>
      <out value="N0]"/>
      <trans dest="2"/>
    </box>
  </boxes>
</xgrf>
```

Ces grammaires sont ensuite compilées dans un format XML appelé wrtn, plus adéquat aux traitements informatiques. La grammaire décrite dans les graphes des figures 1 et 2 est ainsi compilée dans le format suivant :

```
<?xml version="1.0"?>
<wrtn-grammar main="depasser-les-bornes" sz="2">
 <wrtn-pattern sz="12">
  <p>
   <name>depasser-les-bornes</name>
  </p>
  <q id="0">
   <tr to="1">
    <in>
     <epsilon></epsilon>
    </in>
    <out w="0">[</out>
   </tr>
  </q>
  <q id="1">
   <tr to="2">
    <in>
     <scall>GN</scall>
    </in>
    <out w="0"></out>
   </tr>
  </q>
  <q id="2">
```

```
 <tr to="3">
  <in>
   <epsilon></epsilon>
  </in>
  <out w="0">N0]</out>
 </tr>
</q>
<q id="3">
 <tr to="4">
  <in>
   <epsilon></epsilon>
  </in>
  <out w="0">[</out>
 </tr>
</q>
<q id="4">
 <tr to="5">
  <in>
   <lex>
    <lem>dépasser</lem>
    <pos v="verb"/>
   </lex>
  </in>
  <out w="0"></out>
 </tr>
</q>
<q id="5">
 <tr to="6">
  <in>
   <epsilon></epsilon>
  </in>
  <out w="0">V]</out>
 </tr>
 <tr to="7">
  <in>
   <lex>
    <form>les</form>
    <pos v="lex"/>
   </lex>
  </in>
  <out w="0"></out>
```

```
 </tr>
</q>
<q id="6">
 <tr to="10">
  <in>
   <epsilon></epsilon>
  </in>
  <out w="0">[</out>
 </tr>
</q>
<q id="7">
 <tr to="8">
  <in>
   <lex>
    <form>bornes</form>
    <pos v="lex"/>
   </lex>
  </in>
  <out w="0"></out>
 </tr>
 <tr to="8">
  <in>
   <lex>
    <form>limites</form>
    <pos v="lex"/>
   </lex>
  </in>
  <out w="0"></out>
 </tr>
</q>
<q id="8">
 <tr to="9">
  <in>
   <epsilon></epsilon>
  </in>
  <out w="1">Vfigé]</out>
 </tr>
</q>
<q f="1" id="9"/>
<q id="10">
 <tr to="11">
```

```
   <in>
    <scall>GN</scall>
   </in>
   <out w="0"></out>
  </tr>
 </q>
 <q id="11">
  <tr to="9">
   <in>
    <epsilon></epsilon>
   </in>
   <out w="0">N1]</out>
  </tr>
 </q>
</wrtn-pattern>
<wrtn-pattern sz="6">
 <p>
  <name>GN</name>
 </p>
 <q id="0">
  <tr to="1">
   <in>
    <lex>
     <pos v="noun"/>
     <f n="proper" v="true"/>
    </lex>
   </in>
   <out w="0"></out>
  </tr>
  <tr to="2">
   <in>
    <lex>
     <pos v="det"/>
    </lex>
   </in>
   <out w="0"></out>
  </tr>
 </q>
 <q f="1" id="1"/>
 <q id="2">
  <tr to="3">
```

```
    <in>
     <lex>
      <pos v="noun"/>
     </lex>
    </in>
    <out w="0"></out>
   </tr>
  </q>
  <q f="1" id="3">
   <tr to="4">
    <in>
     <lex>
      <form>de</form>
      <pos v="lex"/>
     </lex>
    </in>
    <out w="0"></out>
   </tr>
  </q>
  <q id="4">
   <tr to="5">
    <in>
     <scall>GN</scall>
    </in>
    <out w="0"></out>
   </tr>
  </q>
  <q f="1" id="5"/>
 </wrtn-pattern>
</wrtn-grammar>
```

Au cours de l'opération de compilation, chaque graphe est optimisé par les opérations d'émondation, suppression des transitions vides, déterminisation et minimisation.

Il est également possible de transformer une grammaire en un transducteur fini équivalent, en faisant remonter les sous-graphes dans le graphe principal, éventuellement à une approximation près. Le résultat occupe plus d'espace mémoire mais accélère les traitements.

Outilex offre la possibilité de transcoder les graphes du format grf (Unitex) au format xgrf (et inversement) et de les exporter vers le format dot [Gansner and North, 2000].

3.3 Algorithme d'application d'une grammaire locale pondérée

Notre algorithme d'analyse permet d'appliquer une grammaire locale pondérée sur l'automate du texte ; notre algorithme est inspiré de l'algorithme d'Earley que nous avons adapté de manière à traiter d'une part les grammaires sous la forme de RTN pondérés (au lieu de règles de réécritures hors-contexte) et d'autre part pour accepter une entrée sous la forme d'automate sur les mots étiquetés (au lieu d'une séquence de mots). Notre analyseur produit comme résultat, pour chaque phrase analysée, une forêt partagée d'arbres d'analyse pondérés ; cette forêt peut ensuite être passée à un module de traitement indépendant de notre analyseur permettant de produire différents types de résultats : concordances, texte annoté, automate du texte modifié, formulaires XML, etc.

L'algorithme fonctionne à l'aide d'une charte qui consiste en un tableau dans lequel sont stockées toutes les analyses partielles en cours de traitement. L'utilisation de cette charte est la clef de l'efficacité de notre algorithme puisqu'elle permet, en conservant tous ces résultat partiels, de ne pas recalculer plusieurs fois les mêmes sous-analyses. La charte a la taille de l'automate de phrase analysé. A chaque position se trouve une pile des analyses en cours se terminant à la position correspondante dans l'automate du texte. Une analyse en cours correspond à la reconnaissance partielle d'un segment du texte par un automate de la grammaire ; elle contient les informations telles que l'état de départ dans l'automate du texte à partir duquel la reconnaissance à débuter, les états courants de l'analyse dans l'automate du texte et dans l'automate de la grammaire. Nous conservons également dans une analyse en cours les traces du chemin dans le texte qui a permis la reconnaissance, ainsi que les sorties qui ont été produites lors du franchissement des transitions de la grammaire et le score courant de l'analyse.

L'algorithme général d'analyse consiste à avancer état par état dans l'automate du texte ; pour chaque position, nous parcourons la pile des analyses en cours et nous empilons de nouvelles analyses lors du franchissement de transitions lexicales et syntaxiques dans la grammaire.

4 Développement d'applications

Toutes les applications par grammaires que nous présentons ici utilisent le moteur d'analyse décrit dans la section précédente. Plus précisément, elles ont été implémentées sous la forme de modules qui sont branchés à notre analyseur. Au niveau de l'API C++, la routine d'analyse qui prend en entrée une grammaire et un automate du texte et qui produit la forêt d'arbres d'analyses accepte un troisième argument sous la forme d'un *functor* C++, que l'on peut considérer comme un

procédure externe. Cette procédure est appelée par notre analyseur avec comme paramètre la forêt d'arbres de dérivation obtenue après l'analyse de chaque phrase du corpus en cours de traitement, et peut produire tout type de résultats.

De cette manière nous séparons clairement les routines d'application d'une grammaire sur un texte, des routines qui permettent de formater les résultats. Cette architecture modulaire rend très aisé le développement de différents types d'applications utilisant des grammaires WRTN, puisqu'il suffit d'implémenter la routine qui se charge de transformer le résultat de l'analyse en le résultat escompté et de brancher cette procédure de post-traitement sur notre analyseur qui lui reste non modifié.

De plus, malgré la séparation du moteur d'analyse de la routine de post-traitement, cette dernière n'est pas enclenchée après l'analyse du texte dans sa totalité, mais est appelée par notre analyseur après l'analyse de chaque phrase. Ainsi, lors de leurs traitements, les corpus sont lus séquentiellement phrase par phrase ; chaque phrase est d'abord analysée par la grammaire, puis la routine est appelée et les structures de données sont ensuite libérées avant de passer à la phrase suivante. Cette particularité est très importante, notamment pour le traitement de corpus de textes de grande taille, puisque elle nous assure que la consommation mémoire du programme ne dépend pas de la taille du texte traité, mais simplement de la longueur de chaque phrase (et de la taille des structures produites lors de leurs analyses).

Enfin, l'analyseur ne pose aucune contrainte sur la forme des sorties qui étiquettent la grammaire. Celles-ci sont considérées comme de simples chaînes de caractères et sont reproduites telles quelles dans les arbres de dérivation résultant de l'analyse. Cependant il n'en est pas de même pour la routine de post-traitement, qui peut avoir une interprétation sémantique du contenu des sorties et ainsi leur attacher des actions particulières. Ceci offre la possibilité à tous types de traitements de dépasser largement la simple transduction de chaînes de caractères. Nous avons ainsi pu implémenter, à partir du même moteur d'analyse général, diverses applications, qui à partir de grammaires WRTN, peuvent produire différents types de résultats telles que la transduction, l'annotation de l'automate du texte ou encore la génération de formulaire dans le contexte d'extraction d'informations. Toutes ces applications sont détaillées dans les sections suivantes.

4.1 Sérialisation de la forêt

Nous avons développé un module permettant de sérialiser la forêt, résultat de l'application d'une grammaire locale sur une phrase. Cette forêt présente l'inconvénient d'avoir une grosse taille (environ dix fois la taille de l'automate du texte). Cette fonctionnalité reste néanmoins utile lors de l'application de grammaire complexe, pour lesquelles le temps passé durant l'analyse grammaticale est largement

supérieur au temps passé durant les traitements de la forêt d'arbres d'analyse ou encore lorsque l'on souhaite effectuer différents traitements sur cette forêt. La sauvegarde de la forêt sur le disque permet alors de lancer différentes routines de traitements sur celle-ci, en ne faisant l'analyse du texte qu'une seule fois. Notons que nous avons également implémenté une routine de post-traitement prenant en argument plusieurs routines de post-traitement, et qui, lorsqu'elle est appelée, déclenche les différentes routines qui lui sont passées en argument. Cette technique nécessite néanmoins de compiler un programme pour chaque combinaison de traitements que l'on souhaite appliquer au texte. La technique passant par la sauvegarde du résultat de l'analyse grammaticale dans un fichier intermédiaire ne souffre pas de cette limitation.

4.2 Concordancier

Nous avons également développé un concordancier qui permet de lister dans leur contexte d'apparition les différentes occurrences des motifs décrits par la grammaire. Le concordancier est avant tout une aide précieuse aux linguistes recherchant dans les textes des attestations de formes décrites dans des grammaires locales. Cet outil est également très pratique comme aide à l'écriture des grammaires puisqu'il permet de tester facilement leur couverture lorsque celles-ci sont en construction.

La taille des contextes gauche et droit peut être paramétrée par l'utilisateur et les concordances peuvent être classées soit suivant leur ordre d'apparition dans le texte, soit par ordre lexicographique. De plus, il est possible de paramétrer la mise en forme des concordances, de manière à visualiser ou non les sorties de la grammaire, l'arbre de dérivation (sous forme parenthésée) ainsi que les scores des concordances. Notre concordancier construit les concordances et formate toutes ces information dans un fichier XML ; une feuille de style XSLT permet ensuite de mettre en page ce document sous le format HTML en fonction des paramètres de visualisation.

4.3 Création d'un texte annoté

Nous avons également développé la fonctionnalité d'application d'un transducteur sur le texte qui est proposée dans les systèmes INTEX et Unitex. Cette fonctionnalité produit en sortie un texte brut dans lequel sont insérées les sorties spécifiées dans la grammaire. Ces sorties peuvent au choix être insérées dans le texte d'origine ou remplacer les segments concordant avec la grammaire.

Ce procédé n'est pas complètement trivial puisque, d'une part, il nécessite de projeter la structure sous forme de graphe de l'automate du texte vers la structure linéaire du texte brut produit en sortie. D'autre part, il est également nécessaire de

traiter le problème des chevauchements potentiels des segments du texte concordant avec la grammaire.

Notre algorithme consiste à traverser l'automate de phrase état par état en commençant par l'état initial. Pour chaque état, s'il n'existe pas de concordance commençant à cette position dans la forêt, on franchit la transition étiquetée par la catégorie LEX et on continue l'algorithme à partir de l'état d'arrivée de cette transition. S'il existe plusieurs concordances débutant à cette position du texte, alors nous sélectionnons la *meilleure* d'entre elles. Pour ce faire, nous filtrons ces concordances en appliquant successivement les trois règles de selection suivantes, jusqu'à ce qu'il ne reste plus qu'une seule concordance :

- *Conserver les concordances qui recouvre le segment du texte le plus long*. l'automate du texte étant trié topologiquement, il suffit de conserver les concordances dont l'état de destination est le plus élevé.
- *Conserver les concordances qui franchissent le moins de transitions lexicales*. A cette étape, toutes les concordances recouvrent le même segment du texte, ainsi si une concordance franchit un nombre moins élevée de transitions lexicales, c'est qu'elle comporte plus de transitions qui recouvrent plusieurs tokens.
- *Choisir une transition au hasard.*.

La première analyse permet d'appliquer le principe du *longest-match*. La seconde heuristique permet de préférer les analyses figées (mots composés) aux analyses compositionnelles. Enfin la dernière heuristique permet de ne conserver qu'une seule analyse. Notons que nous n'utilisons pas les poids associés aux analyses pour sélectionner la meilleure concordance, ceci est dû au fait que la sélection par le poids est déjà faite en amont au moment de l'analyse. Ainsi, lorsque la grammaire décrit plusieurs analyse recouvrant le même segment du texte, seules celles ayant le score le plus élevé apparaissent à l'issue de l'analyse. Ce système de pondération permet ainsi au grammairien d'influer sur le choix des concordances sélectionnées en pondérant de manière appropriée ces grammaires.

Une fois la concordance sélectionnée, nous produisons en sortie le résultat de la transduction de la grammaire par un parcours récursif de l'arbre de dérivation décoré par les sorties. Puis nous continuons le parcours à partir de l'état correspondant à la fin de la concordance. L'algorithme prend fin lorsque nous atteignons l'état final de l'automate de phrase.

4.4 Transduction sur l'automate du texte

La fonctionnalité d'application d'un transducteur à un texte telle que présentée dans la section précédente peut s'avérer très utile dans différents domaines du traitement des langues, telles que la reconnaissance d'entités, l'extraction d'informations ou encore l'analyse syntaxique. De ce fait, elle a été à la base de nom-

breuses applications dans ces domaines par le biais notamment des systèmes Intex et Unitex ou encore Xerox [Karttunen et al., 1997] et AT&T [Mohri, 1997]. Cependant ce système souffre d'inconvénients du fait de la structure du résultat produit qui est sous la forme d'un texte annoté. D'une part la forme linéaire du texte produit en sortie ne permet pas de rendre compte des ambiguïtés d'analyses ainsi que des possibilités de chevauchement entre plusieurs analyses. L'utilisation d'heuristiques (notamment la pondération des grammaires) permet, comme on l'a vu, de réduire les dommages dûs à cette limitation, mais seulement de façon partielle. D'autre part, le fait de produire un texte brut en sortie complique la mise en place de traitements nécessitant l'application de grammaires successives à un texte, puisqu'il est nécessaire à chaque étape, de réeffectuer les opérations coûteuses de pré-traitement (segmentation et étiquetage morpho-syntaxique) sur le texte nouveau texte obtenu comme résultat de l'étape précédente.

Afin de pallier ces limitations, nous avons mis en place la fonctionnalités d'application d'un transducteur sur un automate du texte produisant en sortie un nouvel automate du texte enrichi par de nouveaux chemins produit par la grammaire. Pour ce type de traitement, les sorties présentes dans la grammaire ne peuvent pas être sous la forme de n'importe quelle chaîne de caractères, mais doivent être sous la forme d'une séquence de masques lexicaux. Ce sont ces symboles qui étiquettent les nouveaux chemins parallèles aux chemins reconnus par la grammaire.

Notons que, pour ce type de transduction simple, les nouveaux chemins créés ne remplacent pas les anciens chemins qui ont permis leur reconnaissance mais sont placés dans l'automate en parallèle à ces derniers. Les chemins superflus peuvent néanmoins être supprimés au moyen de grammaires ELAG dédiées, ou encore en utilisant une variante plus puissante de la fonctionnalité d'application d'un transducteur sur un automate du texte que nous présentons dans la section qui suit.

4.5 Annotation de l'automate du texte

L'application d'une grammaire, telle que présentée dans la section précédente, est utile pour certains traitements simples (tels que la normalisation des textes), mais elle nécessite que les étiquettes des transitions créées lors de l'analyse soient écrites de manière atomique dans les sorties de la grammaire, sans tirer partie de la structure composite des masques lexicaux. Pour certains traitements (tels que la découpage en chunks, ou la reconnaissance d'entités nommées par exemple), qui consistent à reconnaître et étiqueter les occurrences des formes décrites dans la grammaire, il est nécessaire d'adapter l'étiquetage des séquences reconnues en fonction des chemins parcourus lors de l'analyse dans la grammaire et dans l'automate du texte. Ce type de procédé peut être très fastidieux à écrire avec le formalisme des transducteurs telles que nous l'avons présenté précédemment puisqu'il

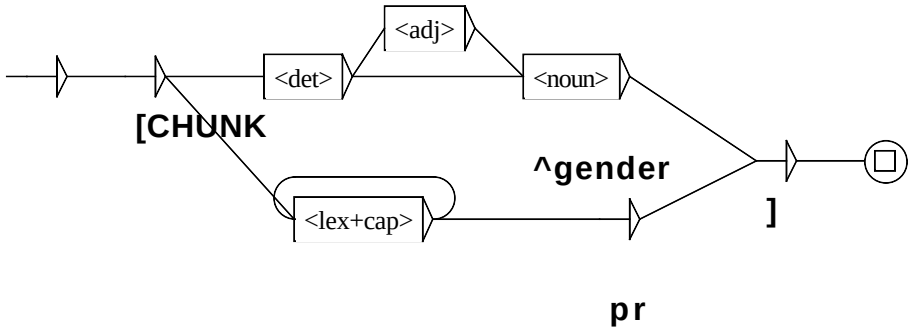

FIG. 3 – grammaire simplifiée de reconnaissance de chunks

oblige le grammairien à démultiplier les chemins de sa grammaire pour chaque combinaison de lexèmes qu'il juge pertinent d'étiqueter différemment. C'est pourquoi nous avons développé une variante plus puissante de la fonctionnalité d'application d'un transducteur sur l'automate du texte, qui donne plus de liberté sur la composition des masques lexicaux qui étiquettent les transitions créées.

Pour ce type d'application, la forme des sorties des grammaires diffère de celle des grammaires utilisées pour la transduction plus classique ; les masques lexicaux qui étiquettent les chemins produits lors de l'application de la grammaire peuvent être décomposés sur plusieurs transitions en différents symboles qui spécifient les différents champs qui composent ces étiquettes.

Par exemple, la figure 3 présente une grammaire simple de reconnaissance de chunks inspirée de [Watrin, 2006]. Cette grammaire reconnaît des séquences telles que

Luc
les grandes idées
l'insatiable Marion

et les étiquette en tant que chunk (étiquette CHUNK), en leur associant des traits spécifiant leur genre, leur nombre ou leur sous-catégorie sémantique. Les symboles qui constituent les sorties de la grammaire dans les chemins délimités par les balises [et] spécifient les différents traits qui composent le masque lexical qui sera créé en sortie. Le premier symbole (CHUNK dans notre exemple), spécifie la catégorie grammaticale du masque (ce symbole doit être déclaré dans la description du jeu d'étiquettes) ; les symboles suivants spécifient les traits syntaxico-sémantiques. Ces derniers peuvent être de deux formes :

– soit ils sont composés de la valeur d'un trait syntaxico-sémantique précédé par un + (par exemple `+pr`). Dans ce cas, l'étiquette créée se verra attribuée cette valeur.

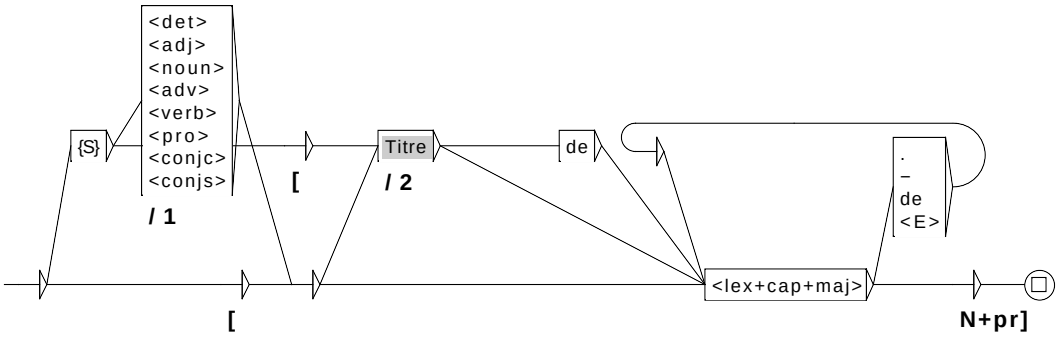

FIG. 4 – Grammaire pour la reconnaissance des noms propres

– soit ils sont composés du nom d'un attribut précédé par +^ (par exemple `+^gender`). Dans ce cas l'étiquette créée hérite de la valeur de cet attribut du lexème dans le texte qui a permis le franchissement de cette transition lors de l'analyse.

De plus, le fait de délimiter par des balises les limites des segments étiquetés par la grammaire permet de faire en sorte que ces limites ne coïncident pas nécessairement avec les limites des segment reconnus de la grammaire. Ceci permet de faire de l'étiquetage de formes en fonction de leur contexte d'apparition, voir même l'étiquetage de plusieurs segments discontinus lorsqu'ils apparaissent dans certains contextes spécifiés par la grammaire.

Par exemple, la figure 4 présente une grammaire simple utilisée pour l'identification des noms propres dans les textes. Celle-ci reconnaît les séquences de mots commençant par une majuscule et les étiquette `N+pr`. Le fait de pouvoir dissocier les limites des segments reconnus par la grammaires des limites des segments étiquetées par celle-ci, en combinaison avec l'utilisation de la pondération, nous a permis d'éviter d'étiqueter comme nom propre tous les mots se trouvant en début de phrase et qui commencent donc par une majuscule du fait des règles typographiques, hormis pour les formes qui sont référencées dans nos dictionnaires. Cette heuristique simple est néanmoins très efficaces, puisqu'elle nous a permis de réduire considérablement le bruit créé lors de la reconnaissance des noms propres, avec un taux de silence faible causé par les cas pathologiques des noms propres homographes avec un mot commun et positionnés en début de phrase.

L'annotation de l'automate du texte par application d'une grammaire WRTN peut servir pour de nombreux traitements à différents niveaux d'analyse TAL. Il peut par exemple être utilisé comme complément à l'étiquetage morpho-syntaxique pour la reconnaissance d'unités lexicales semi-figées dont les variations sont trop complexes pour être énumérées sous forme de liste mais qui peuvent être décrites

dans des grammaires locales.

Ce procédé peut être itéré facilement pour procéder à la reconnaissance et l'étiquetage successifs de segments de plus en plus grands dans l'automate du texte.

Par exemple, la figure 6 présente l'automate de la phrase *164 procès-verbaux jeudi dernier* après l'application de la grammaire des adverbes de temps de M. Gross, sur l'automate de la figure 5.

Il existe par ailleurs une option qui permet de ne garder que l'analyse de la grammaire ; les chemins concurrents dans l'automate du texte sont supprimés. Dans l'automate précédent, la séquence *jeudi dernier* ne serait alors plus analysée que comme un adverbe de date.

5 Développement d'une interface graphique

Une interface graphique a été programmée en Java et intègre les modules principaux d'Outilex. Elle fait appel à leurs fonctionnalités par l'intermédiaire de programmes écrits en C++. L'interface permet de travailler sur différents projets regroupant un ensemble de ressources (textes, dictionnaires et grammaires). L'utilisateur peut y définir sa propre chaîne de traitement et visualiser les entrées, les sorties et les résultats intermédiaires dans un conteneur à onglets dédié à cet effet. Il existe plusieurs intérêts à cette interface. Les linguistes peuvent s'en servir simplement pour construire leurs propres lexiques et grammaires et réaliser des tests préliminaires. Il existe notamment un éditeur de graphes, inspire de celui d'Unitex. Les programmeurs peuvent réaliser des expériences préliminaires sur les modules qu'ils veulent utiliser dans leurs futures applications. C'est également un moyen simple pour apprendre la syntaxe des différents lancement de programmes car chaque opération lancée à partir de l'interface est répertoriée dans un fichier de log.

Ci-dessous, sont données quelques captures d'écrans montrant l'interface (figures 7, 8, 9, 10 et 11).

6 Harmonisation et finalisation des outils

La dernière année du projet a également permis d'accentuer la collaboration entre les partenaires afin de finaliser les différents composants pour leur utilisation dans les démonstrateurs et composants propriétaires des partenaires. En plus de nombreux échanges de courriels d'ordre technique, nous avons mis en place un certain nombre de réunions techniques avec :

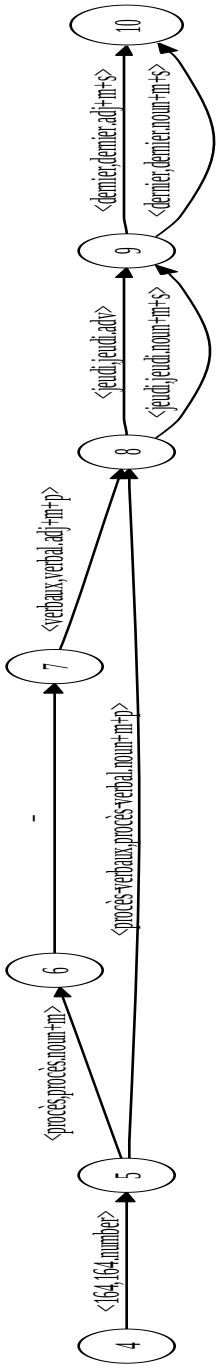

FIG. 5 – Automate du texte initial

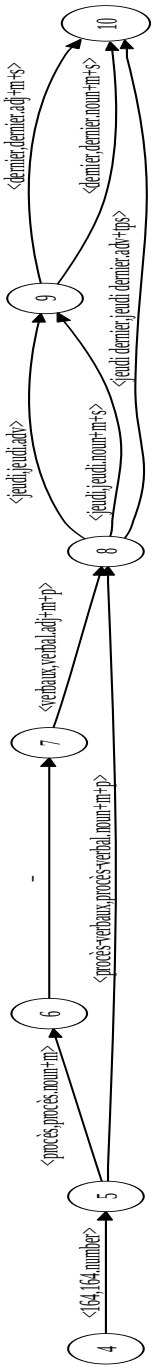

FIG. 6 – Résultat de l’application d’un transducteur sur l’automate du texte

FIG. 7 – Interface graphique (capture d'écran 1)

FIG. 8 – Interface graphique (capture d'écran 2)

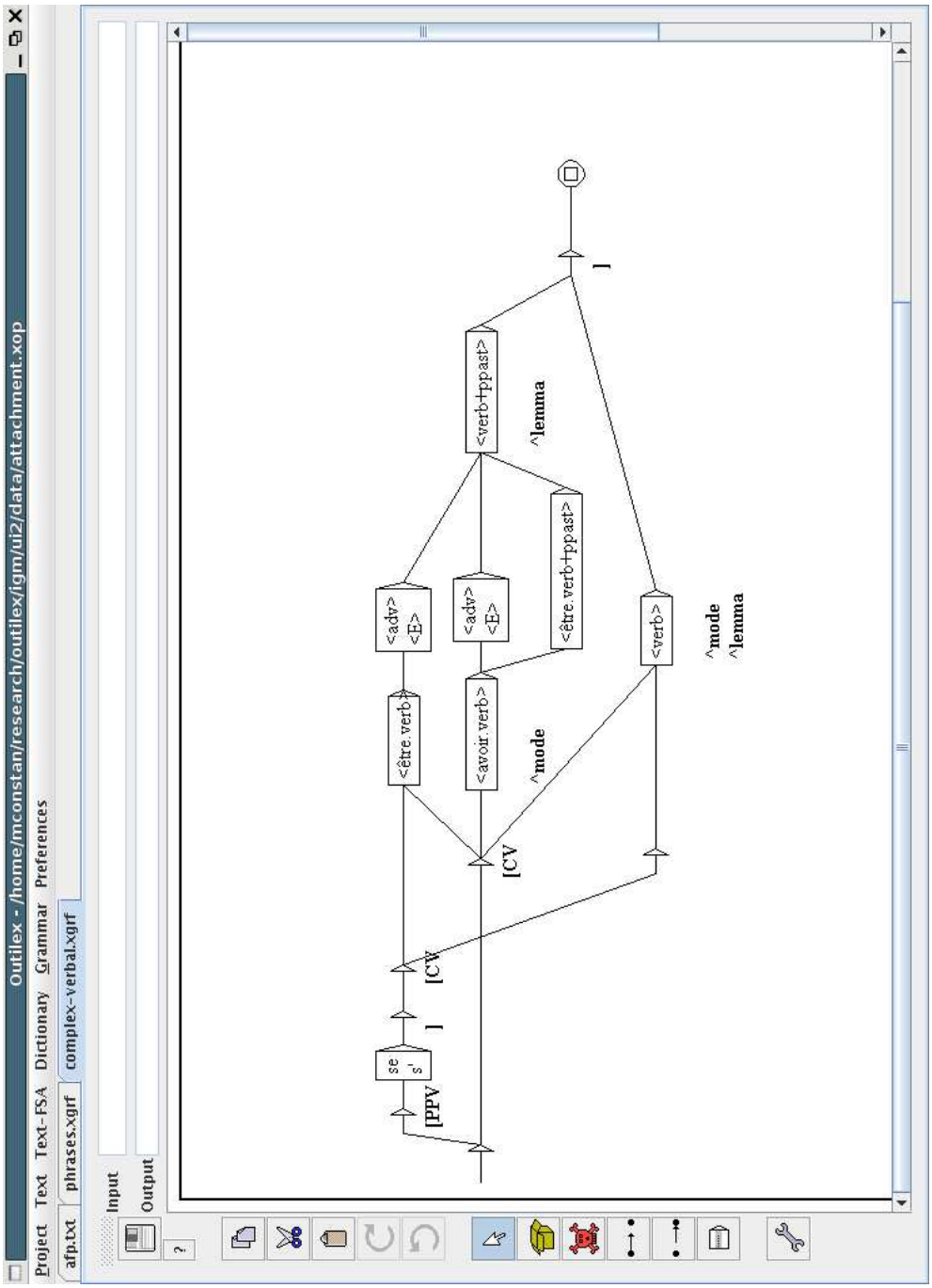

FIG. 9 – Interface graphique (capture d'écran 3)

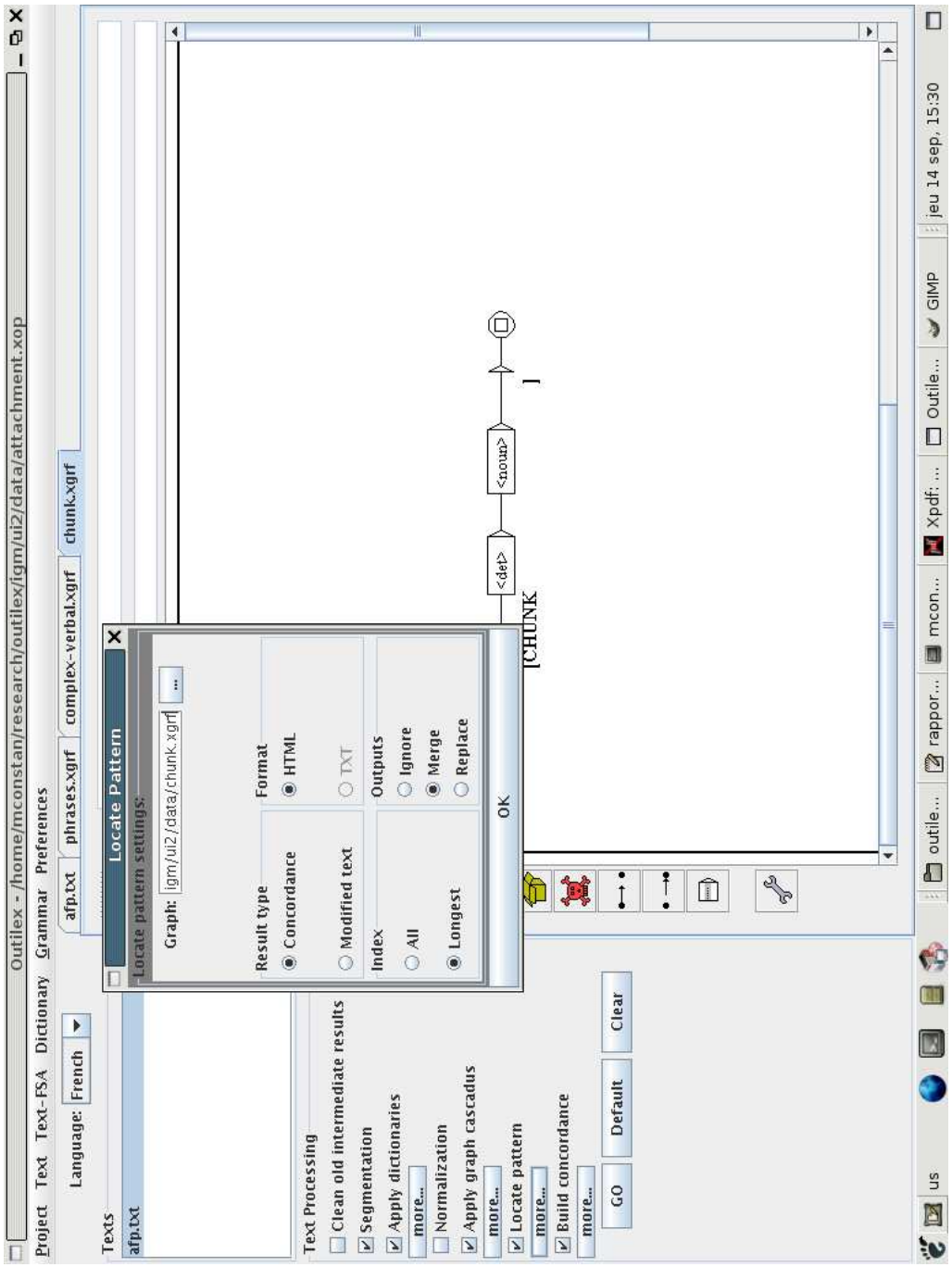

FIG. 10 – Interface graphique (capture d'écran 4)

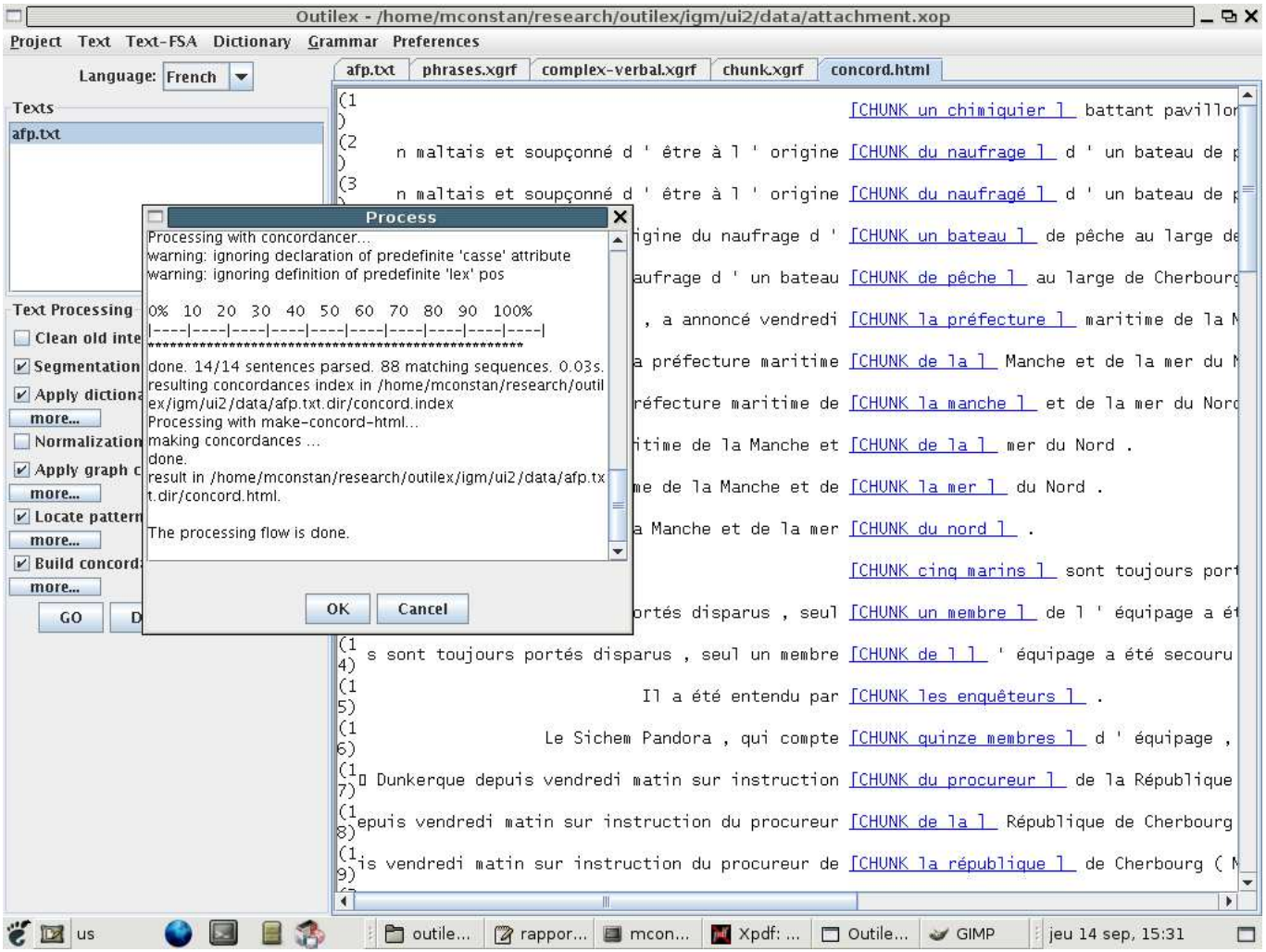

FIG. 11 – Interface graphique (capture d'écran 5)

– Julien Lemoine (Thales) : discussion autour des fonctionnalités nécessaires au démonstrateur Thales ;
– Jean Senellart (Systran) : définition précise des besoins de l'IGM en matière de lexiques et de segmentation ;
– Olivier Houard, Amine Rezrazi, Raphaël Tabary (Univ. Rouen) : présentation de l'interface graphique ;
– Romaric Besançon (CEA) : harmonisation de l'API C++ avec les réels besoins de CEA ;
– Benjamin Piwowarski (LIP6) : aide pour l'utilisation de la plate-forme

Ces différentes réunions nous ont amenés à modifier certaines spécifications de la plate-forme, que ce soit au niveau de l'API, qu'au niveau des fonctionnalités.

7 Publications et démonstrations

7.1 Publications

Un gros effort a également été fait pour publier et promouvoir la plate-forme au sein de la communauté du TAL à la fois en France et à l'étranger. En plus de différents séminaires présentant un certain nombre de points d'Outilex ou des applications (réunions du projet lexsynt – ILF , par exemple), nous avons présenté des articles à différentes conférences internationales sur :

– le formatage XML des données linguistiques [Laporte, 2005, Sastre, 2005]
– la présentation de la plate-forme [Blanc et al., 2006, Blanc and Constant, 2006]

Nous donnons en annexe ces deux derniers articles, l'un en français, l'autre en anglais. Nous avons également produit une petite documentation de l'interface graphique pour les utilisateurs, qui est aussi en annexe. Ce document est complété d'une succinte documentation de l'API, produite automatiquement à l'aide du logiciel $doxygen$ (cf annexe pour l'exemple de la classe `wrtn_grammar`).

7.2 Démonstrations

Dans le cadre de [Blanc and Constant, 2006], nous avons réalisé une démonstration de la plate-forme Outilex, présentant les principaux composants ainsi qu'un démonstrateur d'extraction d'informations biographiques implémenté à partir de l'API C++ d'Outilex.

Cette application ne fait pas, à proprement parlé, partie de la plate-forme, mais nous l'avons implémentée en réutilisant les fonctionnalités fournies par celle-ci de manière à démontrer les facilités d'extensions d'Outilex dont l'architecture permet de réutiliser aisément certains de ses composants dans le cadre du développement d'application TAL plus spécialisées.

Dans ce contexte, nous avons développé un module simple d'extraction d'informations en suivant les méthodes indiquées dans [Nakamura, 2005]. Etant donné un type fixé d'information que l'on souhaite extraire des textes, nous postulons que les différentes réalisations syntaxiques permettant de formuler des énoncés contenant ce type d'information n'admettent qu'un degré restreint de liberté permettant de décrire toutes ces formes dans des grammaires locales. Les différentes entités qui participent à l'information extraite peuvent être identifiées à l'aide de balises incluses dans les sorties de la grammaire. Suivant ce principe, nous avons développé un module qui, étant donné le résultat de l'application d'une grammaire de ce type sur un texte, extrait de la forêt d'arbres d'analyse les informations reconnues par la grammaire et les formate dans des formulaires XML aux champs préétablis.

Afin de valider cette approche, nous avons écrit une grammaire simple spécialisée dans l'extraction des informations biographiques des personnes dans les textes écrits en anglais. Les figures 12 et 13 présentent des extraits de cette grammaire concernant les informations sur les postes qu'occupent les personnes citées dans les énoncés. En particulier, nous nous intéressons à identifier, lorsque ces informations sont disponibles, le nom de la personne, l'organisation pour laquelle elle travaille, la position qu'elle y occupe, ainsi que les dates de début et de fin pour cet emploi.

L'utilisation de notre extracteur permet d'extraire automatiquement toutes les informations indentifiées par la grammaire et de les formater dans des formulaires XML de la forme suivante :

```
<infos>
  <job>
    <who>Michael Mina</who>
    <position>executive chef</position>
    <organization>Aqua in San Francisco</organization>
    <date>since 1993</date>
    <segment>Michael Mina , the executive chef at Aqua
     in San Francisco since 1993</segment>
  </job>
</infos>
```

8 Perspectives

La survie de la plate-forme Outilex et de ses composants est intrinsèquement liée avec sa future maintenance. L'IGM est prêt à la prendre en charge. Cependant, la coexistence d'Outilex avec le logiciel Unitex au sein de l'IGM, ces deux

logiciels ayant des fondamentaux similaires, nous semble difficilement gérable sur le long terme par manque de moyens et par souci de cohérence. En accord avec l'auteur principal d'Unitex, nous nous dirigeons vers une fusion des deux logiciels dans les deux années à venir. En attendant, la version actuelle d'Outilex sera maintenue, étendue à d'autres fonctionnalités, documentée et distribuée.

Références

[Blanc and Constant, 2006] Blanc, O. and Constant, M. (2006). Outilex, a linguistic platform for text processing. In *Proc. of ACL 2006*.

[Blanc et al., 2006] Blanc, O., Constant, M., and Laporte, E. (2006). Outilex, plate-forme logicielle de traitement de textes écrits. In Fairon, C. and Mertens, P., editors, *Actes de TALN 2006 (Traitement automatique des langues naturelles)*, pages 83–92, Louvain. ATALA, Univ. Louvain-la-Neuve.

[Danlos, 2005] Danlos, L. (2005). Automatic recognition of French expletive pronoun occurrences. In *Companion Volume of the International Joint Conference on Natural Language Processing, Jeju, Korea*, page 2013.

[Gansner and North, 2000] Gansner, E. R. and North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software — Practice and Experience*, 30(11) :1203–1233.

[Gross, 1993] Gross, M. (1993). Local grammars and their representation by finite automata. In Hoey, M., editor, *Data, Description, Discourse, Papers on the English Language in honour of John McH Sinclair*, pages 26–38. HarperCollins, London.

[Gross, 1997] Gross, M. (1997). The construction of local grammars. In Roche, E. and Schabès, Y., editors, *Finite-state language processing*, Language, Speech, and Communication Series, pages 329–354. MIT Press, Cambridge (Mass.).

[Karttunen et al., 1997] Karttunen, L., Gaál, T., and Kempe, A. (1997). Xerox finite-state tool. Technical report, Centre de recherche Xerox de Grenoble.

[Krstev et al., 2005] Krstev, C., Vitas, D., Maurel, D., and Tran, M. (2005). Multilingual ontology of proper names. In *Proc. of the Language and Technology Conference, Poznan, Poland*, pages 116–119.

[Laporte, 2005] Laporte, É. (2005). Lexicon management and standard formats. In *Proc. of the Language and Technology Conference, Poznan, Poland*, pages 318–322.

[Mason, 2004] Mason, O. (2004). Automatic processing of local grammar patterns. In *Proc. of the 7th Annual CLUK (the UK special-interest group for computational linguistics) Research Colloquium*.

[Mohri, 1997] Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2) :269–311.

[Nakamura, 2005] Nakamura, T. (2005). Analysing texts in a specific domain with local grammars : The case of stock exchange market reports. In *Linguistic Informatics - State of the Art and the Future*, pages 76–98. Benjamins, Amsterdam/Philadelphia.

[Poibeau, 2001] Poibeau, T. (2001). Extraction d'information dans les bases de données textuelles en génomique au moyen de transducteurs à états finis. In Maurel, D., editor, *Actes de TALN 2001 (Traitement automatique des langues naturelles)*, pages 295–304, Tours. ATALA, Université de Tours.

[Sastre, 2005] Sastre, J. M. (2005). XML-based representation formats of local grammars for NLP. In *Proc. of the Language and Technology Conference, Poznan, Poland*, pages 314–317.

[Watrin, 2006] Watrin, P. (2006). *thèse de Patrick Watrin*. PhD thesis, CENTAL, Université Catholique de Louvain-la-Neuve.

[Woods, 1970] Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10) :591–606.

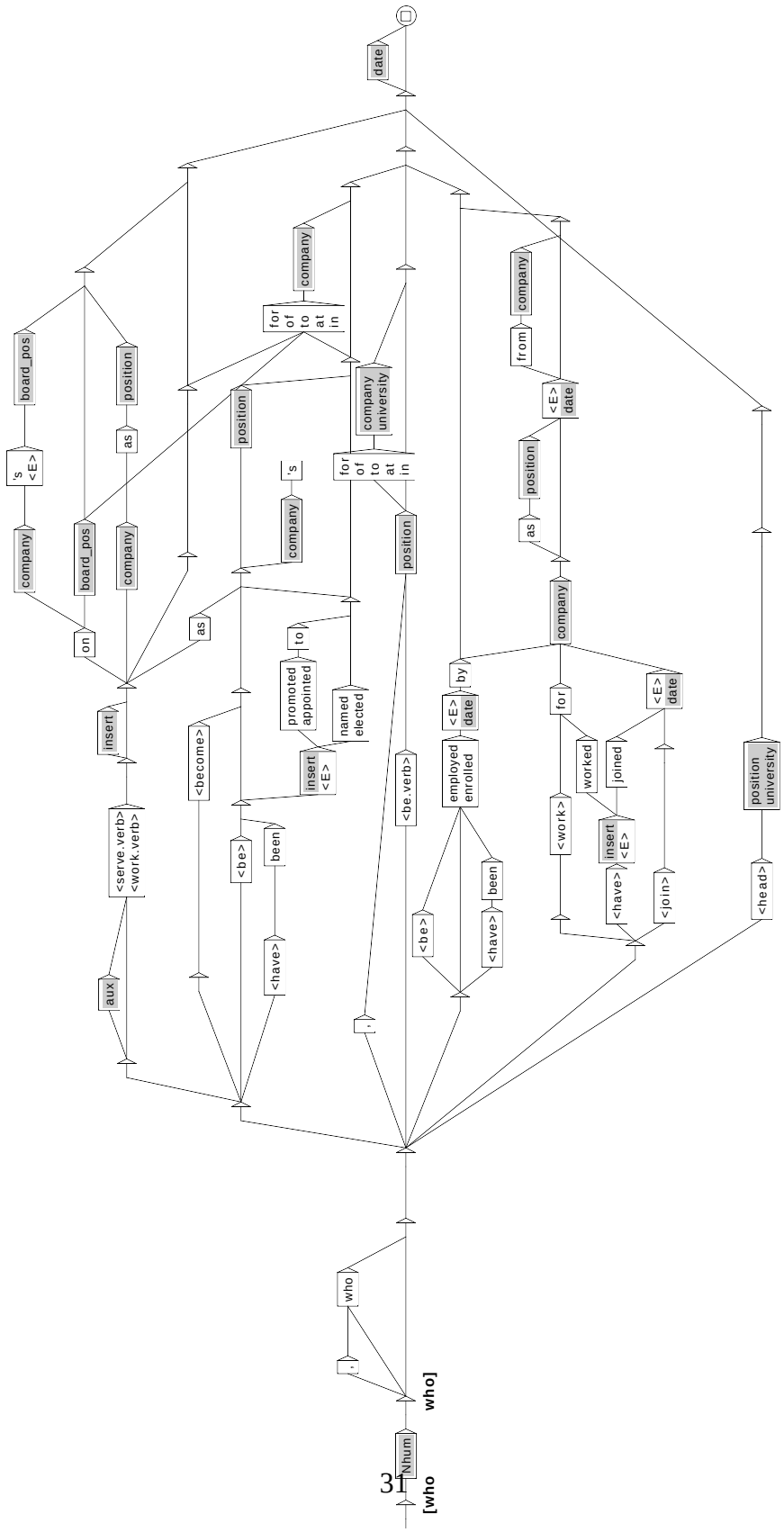

FIG. 12 – Extrait de la grammaire d'extraction d'informations biographiques

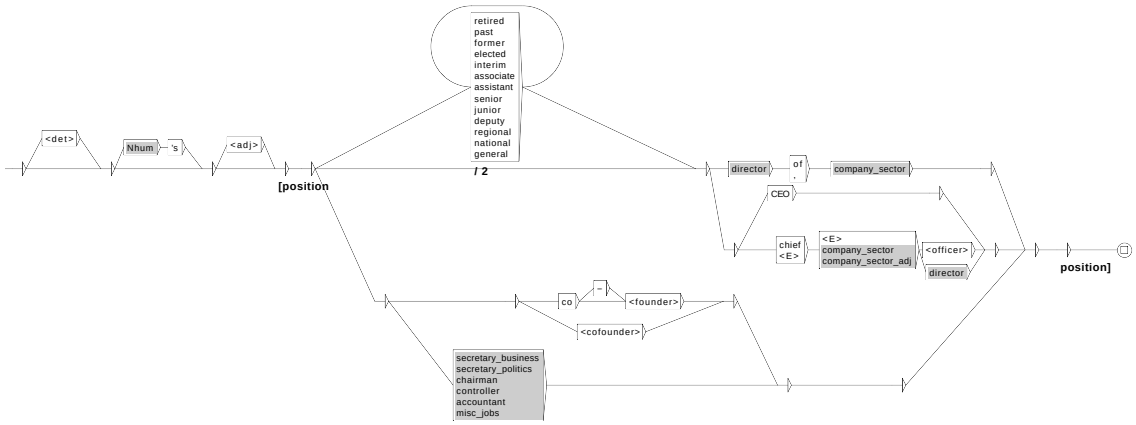

FIG. 13 – Sous-graphe `:position`