X-IOTA

Une plateforme distribuée ouverte pour l'expérimentation en Recherche d'Information

Jean-Pierre Chevallet

Équipe Modélisation et Recherche d'Information Multimédia
Laboratoire CLIPS-IMAG, B.P. 53, 38041 Grenoble Cedex 9, France
E-mail : Jean-Pierre.Chevallet@imag.fr

RÉSUMÉ. *Réaliser des expérimentations en Recherche d'Information est une activité lourde car nécessitant à la fois des outils rapides pour traiter des collections de taille significative, mais également des outils flexibles pour laisser le plus de latitude possible au champ de l'expérimentation. Le système X-IOTA a été développé pour répondre tout particulièrement au critère de flexibilité et donc pour favoriser la mise en place rapide d'expérimentations variées introduisant des aspects traitement de la langue. L'architecture proposée permet également une distribution des calculs. Finalement une interface de commande est mise en place afin de faciliter le suivi et l'évolution des calculs sur les différents sites ainsi que la visualisation plus conviviale des résultats*

ABSTRACT. *Carrying out experiments in Information Retrieval is a heavy activity that require at the same time, fast tools to treat collections of significant size, and also flexible tools to leave the most possible freedom during the experimentation. System X-IOTA was developed to answer the criterion of flexibility and thus to support fast installation of varied experiments using automatic natural language treatments. Architecture proposed allows a distribution of computations among distributed servers. Finally a user interface is proposed that follows evolution of calculations on the various servers and that proposes a more convivial visualization of the results.*

MOTS-CLÉS : *plateforme d'expérimentation en Recherche d'Information, XML, architecture distribuée, interface de visualisation et de commande, architecture client/serveur.*

KEYWORDS: *platform of experimentation in Information Retrieval, XML, distributed architecture, interface of visualization and command, client/server structure.*

1. Introduction

Les nouvelles idées et les propositions de solutions dans le domaine de la Recherche d'Information (RI) doivent généralement être validées par des expérimentations d'indexation et de recherche, par exemple avec des mesures à l'aide de collections de test. Une collection de test rassemble une masse importante (ordre du giga octet) de documents et un nombre significatif de requête (ordre de la centaine). Ces requêtes sont résolues, c'est à dire que les documents qui leur sont pertinents sont connus. L'expérimentation consiste alors à laisser résoudre ces requêtes par le système à tester et à comparer les résultats qu'il propose avec les résultats attendus. Les campagnes TREC [VOO 02] sont les exemples canoniques de ce type d'évaluation.

Les expérimentations en Recherche d'Information ont de spécifique, l'unicité et l'originalité des tests, la grande quantité de paramètres, et la taille importante des corpus manipulés. Les développements réalisés pour ces expériences sont alors d'autant plus coûteux (temps et énergie dépensée) qu'ils ne sont parfois utilisés qu'une seule fois. Il faut alors savoir équilibrer les temps de programmation complémentaires, les temps de mise en place des expérimentations, et le temps passé au déroulement des expériences elles mêmes. En fonction de la complexité des traitements, et du nombre de paramètres que l'on désire faire varier, le temps de l'expérimentation peut se compter en semaines.

Même si le domaine de la RI est presque aussi ancien que celui du développement de l'informatique moderne, il n'existe paradoxalement que peu d'outils facilement disponibles et rapidement modifiables et adaptable pour réaliser à moindre frais, des expériences d'indexations en Recherche d'Information. Le plus célèbre est certainement le système SMART[1] de Gerald Salton [SAL 91, SAL 71], et développé par Chris Buckley [BUC 97]. Ce système est toujours utilisé [BUC 00], et même s'il est à l'origine assez mal documenté[2], il permet une bonne adaptation aux expérimentations lorsque l'on trouve l'énergie de se plonger dans le code pour ajouter ses propres modules. Il faut tout de même rester très près de la structure prédéfinie des traitements, et réaliser une intégration au niveau du code source lui même. Les modifications simples, comme changer le mode de pondération des requêtes, sont parfois compliquées à réaliser, car noyées dans les modules du code, et contraignent parfois à plutôt modifier directement les requêtes que le code du programme. Par exemple, il est plus simple d'augmenter le poids de certains termes de la requête en répétant ce terme que de chercher à modifier le code de SMART.

Le système INQUERY [CAL 92] développé à l'Université du Massachusetts Amherst's Center for Intelligent Information Retrieval, est devenu en 1998 un système commercialisé par la compagnie Sovereign Hill Software Inc. Il est donc assez peu disponible pour la recherche et non facilement modifiable. Le Système Okapi [ROB 97] réputé pour son modèle probabiliste, et connu pour ses bons résultats à l'évaluation

1. Pour télécharger ce système voir ftp ://ftp.cs.cornell.edu/pub/smart/
2. Pour une documentation simple voir http ://pi0959.kub.nl/Paai/Onderw/Smart/tutorial.html

dans des collections de tests, est disponible[3] avec une participation financière. C'est également un système intégré qu'il est préférable d'utiliser sans trop de modifications. HySpirit [LüB 99] est issu de la recherche universitaire, mais est devenu un produit commercial plus orienté applications RI qu'expérimentations.

Parmi les systèmes disponibles pour des expérimentations, nous pouvons citer MG[4] [WIT 99] qui est orienté indexation de larges quantités de textes, et utilise des techniques de compressionSalut gil pour stocker les index dans des fichiers inverses. Ce système a l'avantage de fonctionner sur une architecture par intégration sur les données (principalement la partie mise en forme des documents à indexer). Cela signifie que les modules d'indexation, d'interrogation, etc, sont des programmes (unix), indépendants connectable par flux (ex : pipe Unix), pour réaliser une expérimentation. Par contre, il ne permet pas d'accéder véritablement aux données (ex : pondération) sans intervenir sur le code. Les structures des fichiers sont par contre très clairement documentées.

Le système ECLAIR [HAR 92] a choisi une optique différente : il a été une tentative de construction d'une architecture plus ouverte et plus moderne, car mettant en oeuvre le concept de programmation objet. Il est composé d'un ensemble de classes développées pour construire par assemblage des SRI adapté à chaque expérimentation. Malheureusement, ce système ne semble plus disponible et fonctionne sur une intégration forte au niveau du code source en C++. L'aide à la construction d'interfaces a également été étudié pour ces systèmes de RI, nous pouvons citer par exemple FireWorks [HEN 96] un complément d'ECLAIR.

Le système LEMUR [LAN 04] est également un système modulaire expérimental de recherche d'Information. Il a été conçu principalement pour l'expérimentation la RI basée sur un modèle de langage [ZHA 01]. Il permet également de travailler avec des modèles plus classiques. Tout comme ECLAIR, il est basé sur une intégration forte au niveau du code source.

Le système SIRE [SAN 96][5] est composé d'un ensemble de modules qui communiquent par un protocole commun en mode texte. Ce protocole est basé sur la notion de ligne : chaque entité est décrite sur une ligne composée d'un caractère indiquant le type de l'entité (début de document, mots, etc.), un entier décrivant sa taille, puis l'entité elle même. Dans l'idée, le système SIRE utilise les mêmes concepts que le système X-IOTA. La différence est principalement dans l'adoption de la norme XML pour la transmission des informations entre les modules augmentant alors la souplesse d'utilisation : c'est le langage de description qui est fixé, et non le format lui même qui est paramétrable contrairement à SIRE.

---

3. Okapi est disponible à l'adresse suivante http ://www.soi.city.ac.uk/ andym/OKAPI-PACK/
4. MG est disponible à cette adresse : http ://www.cs.mu.oz.au/mg/ sous licence GPL
5. le système et le rapport de recherche ne semblent plus disponibles, l'information trouvée se trouve dans [TOM 97] page 36.

Le projet IRTool[6] appelé aussi TeraScale[7] a pour objectif de fournir un SRI sous licence GPL pour l'expérimentation uniquement. Le projet est n'a pas l'air assez avancé pour être utilisable. De plus il n'est pas documenté.

Bow [MCC 96] est une bibliothèque logicielle pour faire des statistiques de textes et de la recherche d'information. Ce système en licence GPL, permet de construire un SRI par intégration au niveau du code écrit en C. Il est utilisé dans Rainbow, et CrossBow des classifieurs, ainsi que dans Arrow, un logiciel simple d'indexation et de recherche.

Le système Terrier [OUN 04] est un ensemble de classes en Java optimisé pour la recherche d'information sur de larges collections. Il n'est pas encore disponible en dehors de l'équipe de recherche de Glasgow.

Il existe encore une très grande quantité de système de RI effectivement disponibles[8] comme freeWAIS-sf [PFE 95] issu du système WAIS ou Xapian [LIM 03] du moteur de Recherche Muscat. Cependant, ces systèmes sont plus dédiés à des applications RI, principalement des indexations de sites Web, que des expérimentations pour lesquelles ils n'apportent aucune facilité.

En conclusion, il n'y a donc actuellement pas d'autre alternative pour l'expérimentation en RI que : soit utiliser un des systèmes existant et ayant fait ses preuves dans les expérimentations de recherche quasiment sans modifications, ou avec des modifications mineurs qui ne remettent pas en cause le modèle du SRI, soit construire un nouveau système complet pour chaque modèle de RI.

Nous pensons qu'il existe tout de même, une manière de faciliter les expérimentations par une autre architecture qui possède les caractéristiques suivantes :

Architecture ouverte : Une architecture ouverte signifie la possibilité de réorganiser les éléments du système de la manière la plus libre possible. Le système SMART, par exemple, permet une certaine souplesse dans la construction des expériences. Il offre la possibilité d'ajouter des variantes des modules de base (ex : lematisation), mais il ne permet pas de construire une chaîne de traitement différence de celle prévue par le modèle vectoriel.

Structure de donné ouverte : L'échange des données entre les modules ne doit pas être contrainte par des formats particuliers qui nécessitent un travail de codage pour l'adaptation de l'entrée et sortie des modules. Le langage XML est un bon candidat pour ce format d'échange.

Architecture distribuée : Une architecture distribuée signifie la possibilité de connecter librement des modules de traitements développé sur des sites différents. Il y a une différence de coût importante entre un logiciel utilisé sur son lieu de développement par l'équipe qui en a exprimée les besoins, et la diffusion de ce logiciel sur un autre site. Le coût concerne les adaptations (au système, au site,

6. http ://sourceforge.net/projects/irtools/
7. http ://ils.unc.edu/tera/
8. Voir le site http ://www.searchtools.com qui référence plusieurs centaines de SRI.

etc.) et l'effort pour la rédaction d'une documentation plus complète. De plus, la distribution d'un logiciel expérimental crée de fait deux versions : la version distribuée et la version développé qui est forcément plus en avance sur le plan des possibilités de fonctionnement ou sur le plan de la stabilité. Une architecture distribuée possède alors l'avantage de donner accès à une version toujours à jour. Elle réduit également les problèmes de propriété du code, puisque que seul l'utilisation à distance est permise. Cette architecture n'est néanmoins possible que si le site hôte accepte de fournir le service, le temps de calcul et éventuellement du stockage, et également si les temps de transmission des données ne sont pas prohibitifs par rapport aux temps de calcul sur le site.

Indépendance au langage de programmation : Dans le milieu expérimental universitaire, il est à notre avis illusoire de vouloir imposer un langage de programmation unique pour des logiciels très différents et non intégrés. Au contraire, préférer une indépendance au langage de programmation assure un plus grand partage des ressources logicielles mais implique par contre une liaison des modules de traitements par le système d'exploitation et donc sur les données échangées (ex : fichiers, pipe, réseau).

Indépendance au système d'exploitation : Pour les mêmes raison qu'au dessus, il est préférable de ne pas devoir imposer de système d'exploitation particulier pour la connexion des modules : il suffit simplement d'un mode de communication réseau commun (ex :la norme TCP/IP) et d'un protocole commun (ex : XML plus http) pour que les modules coopèrent.

En résumé, nous présentons dans le tableau 1 quelques caractéristiques des systèmes actuellement disponibles à des fin de recherche et en accès libre sur le web. La colonne "Integ" indique le type d'intégration des modules entre eux : "code" signifie qu'elle se fait par le code source, alors que "data" indique que l'intégration ne se fait que par échange de données des modules. La colonne "Lang." représente le langage de programmation majoritairement utilisé. Le type d'architecture, c'est à dire les schémas de traitements des documents et requêtes, peut être prédéfini ("prédef") et imposé par le logiciel, ou bien programmable ("prog") lorsqu'il s'agit d'une librairie, ou plus libre ("ouverte") si aucun enchaînement n'est imposé hormis les contraintes de compatibilités de modules. La structure des données produites ("data str") peut être en format binaire non clairement documenté ("bin"), format binaire documenté ("bin+doc"), ou en "XML" et documentée. La colonne "interface" indique si le système possède une interface utilisateur dédiée autre que la ligne de commande. Une interface est soit en mode texte soit sous la forme d'un site WEB (norme "HTTP").

Les contraintes précisées précédemment nous amènent à définir une architecture en modules de traitement indépendants, connectés entre eux soit par le système d'exportation, soit par une connexion réseau de type socket TCP/IP avec des échanges d'informations en XML. Nous détaillons cette architecture dans la partie suivante. Nous donnons ensuite quelques exemples simples d'enchaînement de modules. La dernière partie concerne l'interface de commande et de visualisation des résultats qui complète cette structure de plateforme distribuée d'expérimentation en RI.

Système	Integ.	Lang.	Archi.	Data str.	Interface
SMART	code	C	prédef	bin	texte
BOW	code	C++	prog	bin	non
IRTool	code	C++	prog	bin	non
LEMUR	code	C++	prog	bin	non
MG	data	C	prédef	bin+doc	non
X-IOTA	data	C++	ouverte	XML	HTTP

Tableau 1. *Résumé des caractéristiques des SRI de recherche disponibles*

2. Proposition d'architecture ouverte

L'architecture de la plateforme complète se compose de deux parties : une partie modules de traitements à enchaîner pour réaliser une expérimentation, et une partie pilotage du noyau fonctionnel et présentation des résultats sous la forme d'une interface avec une technologie Web. Nous présentons d'abord la partie modules de traitements élémentaires.

L'architecture proposée pour ce système repose sur un enchaînement de modules élémentaires s'échangeant des informations au format XML pour réaliser une expérimentation particulière. Par exemple, l'indexation et l'interrogation se réalisent par deux chaînes de traitements XML. Cette architecture permet d'accéder aux données intermédiaires à tout moment du traitement. Ce point est important pour la flexibilité de ce système qui est orienté expérimentations. En particulier le langage XML assure une lisibilité sans outil de décodage particulier, et une présentation simple dans un navigateur. Le codage XML apporte également l'extensibilité et la flexibilité dans l'utilisation, en permettant de paramétrer les balises à traiter et en conservant dans le flux, les informations supplémentaires non comprises par le module. De manière pratique, traiter les données par flux et enchaînement de modules limite les risques de dépendances au niveau des codes sources et donc rend plus aisé la maintenance et correction du code. Cette architecture est également favorisée par un environnement multiprocesseurs[9] puisque les modules d'une chaîne de traitement vont être automatiquement répartis par le système sur les processeurs disponibles. Les limitations de ce choix portent sur l'augmentation de la taille des données manipulées et la nécessité du codage et décodage du flux XML pour toutes les opérations de traitement, qui engendre un surcoût de traitement.

La figure 1 présente l'architecture globale de cette plateforme : une expérience se compose de scripts locaux au site, pouvant s'interconnecter avec des modules d'autres sites. Une télécommande globale permet de construire et de suivre les expérimentations en cours, de même que de visualiser les résultats intermédiaires ou finaux. Certaines fonctions de stockages et d'accès particulières peuvent être confiées à des outils dédiées (ex : SGBD), mais dans la mesure du possible, on tentera de conserver au

9. Avec le système Linux.

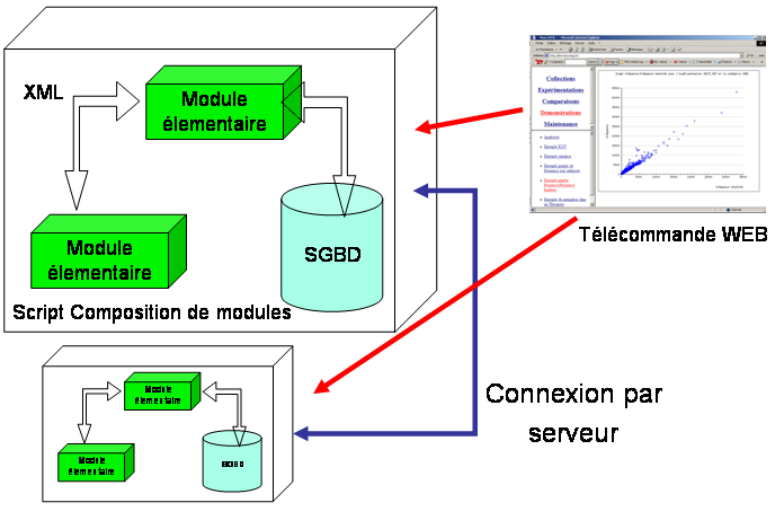

Figure 1. *Architecture de la plateforme*

maximum les informations au format XML. Les modules de traitement se classent selon les caractéristiques suivantes :

généricité/spécificité en entrée : les modules génériques en entrée, fonctionnent sur tout type de flot XML d'entrée. Ils nécessitent bien sûr une paramétrisation sur le type de balise à traiter. A l'inverse, les modules spécifiques en entrée ne comprennent qu'un type XML bien particulier de donnée, (ex : des vecteurs) même si les noms de balises eux mêmes restent paramétrables.

généricité/spécificité en sortie : C'est la caractéristique inverse de la précédente. Un module ayant une sortie spécifique, produira un type de donné particulier, alors qu'un module générique en sortie produit une structure de donnée uniquement dépendante de son entrée. Bien évidemment, seul les modules génériques en entrée peuvent l'être en sortie.

opacité/transparence : cet attribut qualifie la faculté d'un module à laisser intact ou au contraire à supprimer les informations qu'il ne reconnaît pas explicitement. Dans la mesure du possible, il est souhaitable que les modules restent transparents pour permettre le maximum de flexibilité. Nous parlerons de traitement transparent pour désigner les modules qui ne font que modifier la structure XML en entrée, et de traitement opaque pour les modules qui produisent une structure fixe à partir de leur entrée.

traitement par flot/par accès direct : le mode de traitement par flot concerne les modules qui n'ont pas besoin d'accéder de manière directe à un élément de la structure XML, mais produisent leur sortie au fur et à mesure que leur entrée

est alimentée. A l'inverse, l'accès direct nécessite le recours à un fichier pour réaliser le traitement.

Le type de module le plus général et donc le plus réutilisable devra être générique en entrée et sortie, transparent et par traitement par flot. Il convient également de conserver au maximum la propriété de transparence du module, même dans le cas ou les entrées/sorties sont spécifiques. Cette transparence permet d'accepter de la variation dans le type spécifique. Par exemple, dans les modules actuels de X-IOTA, le module `xml2vector` produit un vecteur, qui est un type spécifie `vector`, composé d'un ensemble de coordonnées avec une pondération :

```
<vector id="DOC0001">
<c id="1968" w="1"/>
<c id="accompagn" w="1"/>
</vector>
```

La définition de cette structure doit être comprise comme une définition nécessaire, et non comme une définition suffisante. Cela signifie que les modules qui sont spécifiques en entrée à ce type de structure, doivent néanmoins tolérer plus d'informations, comme des attributs supplémentaires qui seront tous simplement retransmis sans modifications.

```
<vector id="DOC0001">
<c id="1968" w="1" pos="NUMBER"/>
<c id="accompagn" w="1" pos="VERB"/>
</vector>
```

Pour illustrer le fonctionnement d'un système réel utilisant cette architecture, nous développons dans la partie suivante, l'exemple canonique d'une indexation et interrogation par un système basé sur un modèle vectoriel.

3. Exemple d'enchaînements

Cette partie illustre le fonctionnement possible de cette plateforme en présentant des enchaînements de base pour une indexation et une interrogation classique de type modèle vectoriel.

3.1. *Indexation*

La première partie de cette indexation consiste à produire un fichier direct après les traitements minimum des documents en entrée. La chaîne des traitements comprend la suppression des caractères diacritiques, la mise en minuscules, le passage à un antidictionnaire contenu dans le fichier `common_words.fr`, une racinisation dans le style

français, puis la mise sous la forme d'un vecteur document. Chaque entité identifiée par le tag en paramètre de `-docTag` sera considéré comme un document. Le tout est décrit comme une commande unique interprétée par le script shell du système. Nous mettons à profit ici, le mécanise de pipe (symbole `|`) du système unix. N'importe quel autre langage de script peut être utilisé (make, PHP, perl, etc.), du moment qu'il permet la connexion des entrées et sorties des modules exécutables. Egalement, n'importe quel type de traitement transparent peut s'insérer dans cette chaîne pour la compléter, et l'adapter à une expérience particulière.

```
cat OD1 | xmldeldia | xmlcase | xmlAntiDico -dico common_words.fr
        | xmlStemFr | xml2vector -docTag div -id id > OD1.vector
```

Dans cette séquence, le module `xml2vector` construit le vecteur document par extraction du texte contenu entre la balise `div`. Dans cet exemple, chaque document possède une identification dans le paramètre `id` de la balise qui est conservée. Tous les modules sont génériques en entrée et en sortie et totalement transparents. Seul le module `xml2vector` produit un type de données particulier : un vecteur pondéré de termes. Il est alors générique en entrée et spécifique en sortie. Il peut conserver un certain degré de transparence[10]. Cette chaîne de traitement produit par exemple le vecteur suivant pour le premier document de la collection test l'OFIL de la partie Amaryllis de la campagne de test CLEF :

```
<vector id="2271448" size="188">
<c id="1968" w="1"/>
<c id="accompagn" w="1"/>
<c id="achev" w="1"/>
<c id="an" w="2"/>
<c id="analys" w="1"/>
<c id="ang" w="2"/>
<c id="appel" w="1"/>
<c id="asson" w="1"/>
<c id="aut" w="2"/>
<c id="avait" w="1"/>
<c id="avec" w="5"/>
<c id="baudoin" w="3"/>
...
</vector>
```

La structure de donnée `vector` produite par le traitement opaque `xml2vector` est une simple liste de coordonnées identifiée et pondérées.

10. Dans la version actuelle, seuls les commentaires traversent librement ce module. Chaque module laisse une trace de son traitement sous la forme d'un commentaire : le fichier final contient alors la trace de tous les traitements effectués dans la chaîne depuis l'information source.

La valeur `size` de chaque vecteur indique le nombre de coordonnées qui suivent. L'identification du vecteur est celle qui a été prise dans le document dans la balise indiquée dans le paramètre `-docTag` et `-id`. Les pondérations associées à l'attribut `w` sont à ce stade uniquement des nombres d'occurence des termes dans le document calculé par `xml2vector`.

Une matrice est la simple concaténation de vecteurs identifiés. L'identification correspond ici directement à l'identification du document dans la collection de test. L'étape finale de l'indexation consiste à inverser la matrice direct en calculant les pondérations. La version actuelle de X-IOTA, propose un module d'inversion travaillant uniquement en mémoire vive. Cela permet d'obtenir une grande vitesse d'inversion, mais limite la taille des matrices manipulées. Par contre, tout est en mémoire, le décompte du nombre de vecteurs de la matrice et les pondérations sont établis à cette occasion. Pour l'OFIL qui fait 34Mo de source, environ 50Mo de mémoire sont nécessaires. Le type de pondération est codé dans ce module d'inversion. Il est possible de réaliser une inversion sans pondération et de construire à part un module qui calcule toutes les pondérations de la matrice.

```
cat OD1.vector | xmlInverseMatrix -w ltc > OD1.inverse.ltc
```

La matrice inverse possède exactement le même format XML que la matrice direct, à ceci près que les identifications des vecteurs sont celles des termes et celles des coordonnées correspondent aux documents. Deux modules génériques à traitement en accès direct permettent de sélectionner et d'extraire un arbre XML connaissant la valeur des identificateurs uniques de ses attributs. Cette fonctionnalité est suffisante pour accéder à la matrice pour une interrogation. Il s'agit du module `xmlIndex` qui construit un index d'accès direct à un fichier XML connaissant la balise à indexer et le nom de l'attribut identifiant unique.

```
xmlIndex OD1.inverse.ltc vector id
```

On indique par cette commande que l'on doit indexer chaque vecteur `vector` par son attribut `id`. La matrice est alors prête pour une interrogation. A ce stade il est possible d'extraire un vecteur par le terme qui l'indexe à l'aide du module d'accès direct :

```
xmlIdxSelect OD1.inverse.ltc baudoin
```

Cette commande extrait le vecteur d'identificateurs de documents indexés par le terme `baudoin`. Cet exemple illustre d'une part la simplicité de la mise en oeuvre d'une chaîne d'indexation avec cette architecture, et prouve également qu'il est possible d'intervenir a tout moment dans le processus d'indexation car toutes les données restent en XML. Nous allons maintenant examiner le cas de l'interrogation.

3.2. *Interrogation*

Une interrogation dans le cadre d'une collection de test consiste à lancer toutes les requêtes prévues sur la collection indexée. Les requêtes doivent subir le même type de traitement pour les termes, pour que les dimensions des vecteurs soient compatibles. Voici un exemple de requête : dans cet exemple, tous les termes sont utilisés pour construire le vecteur requête.

```
<record>
<num>1</num>
<dom>International</dom>
<suj>La séparation de la Tchécoslovaquie</suj>
<que>Pourquoi et comment avoir divisé la Tchécoslovaquie et
quelles ont été les répercussions économiques et sociales ?
</que>
<cinf>Prendre en compte les différentes versions présentées</cinf>
<ccept>
<c>Partition de la Tchécoslovaquie</c>
<c>Causes et modalités de la partition</c>
<c>Création de la Slovaquie et de la République Tchèque</c>
<c>Points de vue</c>
<c>Economie</c>
</ccept>
</record>
```

Nous pouvons utiliser la chaîne de traitements suivante :

```
cat OT1 | xmldeldia | xmlcase | xmlAntiDico -dico common_words.fr
        | xmlStemFr | xml2vector -docTag record -idTag num > OT1.vector
```

Il est important de noter que cette chaîne est quasiment identique à la chaîne de l'indexation. Seuls les noms de balise ont été modifiés. En effet, les requêtes de la collection OFIL sont dans une structure XML nommée `record`. L'identificateur d'une requête n'est pas dans un attribut de la balise `record`, mais à l'interieur de la balise `<num>`. C'est pour cette raison que l'option `-idTag` a été utilisée ici à la place de l'option `-id` pour le fichier des documents. On peut noter que par défaut tous les champs de la requête ont été pris en compte pour construire le vecteur. La pondération est simplement le comptage des termes. Par exemple, on obtient à ce stade le vecteur :

```
<vector id="1" size="25">
<c id="avo" w="1"/>
<c id="caus" w="1"/>
<c id="compt" w="1"/>
<c id="cré" w="1"/>
```

```
<c id="divis" w="1"/>
<c id="econom" w="1"/>
<c id="internation" w="1"/>
<c id="modal" w="1"/>
<c id="ont" w="1"/>
...
</vector>
```

L'interrogation proprement dite consiste à réaliser un produit (xmlVectorQuery) de la matrice des requêtes par la matrice inverse des documents indexés :

```
cat OT1.vector | xmlVectorQuery OD1.inverse.ltc
               | xmlVector2trec > trec_top_file
```

Le module `xmlVector2trec` est chargé de convertir la matrice requete/documents en un format compatible avec l'utilitaire `trec_eval` qui produit la courbe normalisé de rappel et précision. Nous constatons que la mise en place d'une chaîne d'indexation, interrogation se décompose en peu de modules. Faire varier les paramètres ou introduire un traitement particulier devient une opération d'insertion d'un traitement du flux XML dans la chaîne originale. Par exemple, il suffit d'insérer un module générique de suppression de sous arbre XML (`xmldelsub`) pour choisir les champs des requêtes à utiliser :

```
cat OT1 | xmldelsub ccept | xmlAntiDico -dico antiDico.txt
        | xml2vector -docTag record -idTag num > OT1.vector
```

Dans ce cas, nous avons simplement supprimé du flux des requêtes, les champs concept avant de les transformer en vecteurs. De même, modifier la pondération des termes de la requête peut se réaliser soit de manière intégrée en modifiant le code de construction des vecteurs, soit par la construction d'une module supplémentaire du traitement du flux. Le code du module de construction des vecteurs étant de petite taille (comme la plupart des modules), la modification est très localisé et donc comporte moins de risques que la modification de code d'un système intégré comme SMART. En effet, une erreur de programmation peut affecter l'importe quel partie du fonctionnent du SRI si l'intégration est réalisée par le code et non par le système d'exploitation. Un autre module peut traiter la structure vectorielle de la requête pour réaliser une normalisation particulière.

3.3. *Fonctionnement distribué*

La nature modulaire au niveau du processus système permet de mettre en place rapidement une distribution des modules sur des sites et systèmes différents. Les modules deviennent des serveurs de traitement qui reçoivent et réémettent des données au

format XML sur un port particulier d'un machine identifiée sur le réseau Internet. La communication se fait alors par Internet. N'importe quel opérateur de redirection des entrées/sorties vers un serveur et un port de communication, permet de réaliser cette connexion. La complexité de la mise en fonctionnement distribuée dépend du système d'exploitation. Par exemple, les systèmes Unix proposent une structure d'accueil très simple d'utilisation[11] qui permet à tous modules de devenir immédiatement un serveur associé à un port de la machine. La seule adaptation à réaliser pour la mise en réseau, concerne le passage des paramètres de calculs non plus en ligne de commande mais directement dans le flux XML. L'analyseur de surface du Français du système IOTA a été installé de cette manière sur une des machines de l'équipe recherche d'information de Grenoble[12].

Le problème des droits et de la sécurité d'une telle architecture distribuée peut être résolue avec les technologies de cryptage du protocole SSH couplé au mécanisme de tunneling pour une mise en communication de modules aux travers de pare-feux.

4. Architecture de l'interface de commande

Pour faciliter la mise en place et le suivit des expérimentations, qui peuvent se dérouler sur plusieurs sites, nous proposons la mise en place d'une interface de commande et de visualisation compatible avec la technologie WEB.

L'architecture proposée comprends trois nouvelles couches supplémentaires au noyau fonctionnel (cf figure 4). La première couche est un serveur de calcul. Son rôle est de fournir une entrée unique pour la description des services offerts par la machine ou il est lancé. Il permet donc de lancer et suivre l'exécution de modules, et de transmettre les résultats des calculs.

La seconde couche est l'interface proprement dite qui se connecte à l'ensemble des serveurs de calculs dont elle a connaissance. Cette interface est concrètement un serveur WEB. Elle peut donc être localisés sur une autre machine. Elle possède également ses propres informations qu'elle peut conserver dans une base de donnée. Finalement, la dernière couche est en fait le navigateur sur la machine de l'expérimentateur. C'est le client de l'interface.

L'interface de commande doit permettre de construire et paramétrer de façon conviviale les chaînes de traitements qu'il faut mettre en place. Elle est également un moyen de rassembler et décrire les divers modules disponibles à travers le réseau de serveurs de la plateforme. Il est alors nécessaire de produire une descriptions des ressources disponibles, tant les ressources statiques (documents bruts,ou pré-traités), que les mo-

11. il s'agit de xinet, un outil système centralisé de tous les processus serveurs d'une machine. L'utilisateur doit seulement fournir le programme client qui interprète les commandes en entrée standard et produit les résultats en sortie standard. Toute la gestion logicielle de la communication réseau est masquée, et n'importe quel programme peut alors instantanément devenir un serveur.
12. Actuellement en accès libre sur le port 2000 de la machine siota.imag.fr

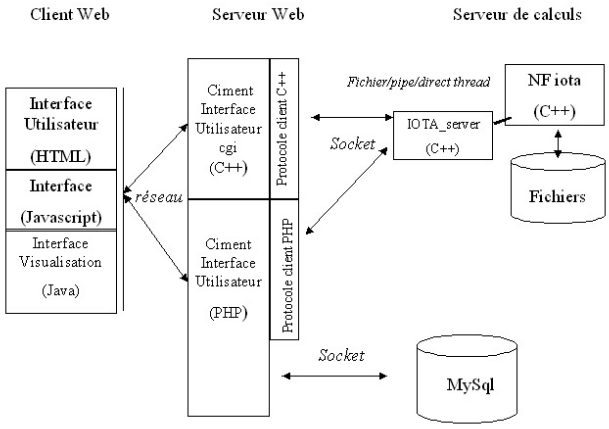

Figure 2. *Architecture de l'interface*

dules de traitement. Par exemple la description XML du module de filtrage par anti-dictionnaire peut être :

```
<module cmd="xmlAntiDico" name="filtre par anti dictionnaire" icone=""
  path="/home/maisonlo/IOTA/src/IOTA_index/" categ="Filtre">
  <comment>
    Filtre l'entrée standard par un anti dictionnaire fournit
    en paramètre -dico. Si l'option -t est présente,
    ne filtre que les mots présents entre la liste des tags.
  </comment>
  <output type="xml"/>
  <input type="xml"/>
  <param opt="-dico" name="antidico utilisé" type="string"
         default="/home/maisonlo/IOTA/data/common_words.fr" >
    <help> chemin de l'anti dico utilisé </help>
  </param>
  <param opt="-t" name="balises traitées" type="string">
    <help> de la forme : -t tag1 .. tagn </help>
    <comment> liste des balises devant être traitées
    si vide traite toutes les balises</comment>
  </param>
</module>
```

L'objectif d'une telle description, est qu'elle contienne toutes les informations pratiques et techniques pour d'une part comprendre l'utilisation du module dans la chaîne de traitement, et d'autre part, pouvoir produire de manière automatique une interface de paramétrisation du module (cf. figure 3).

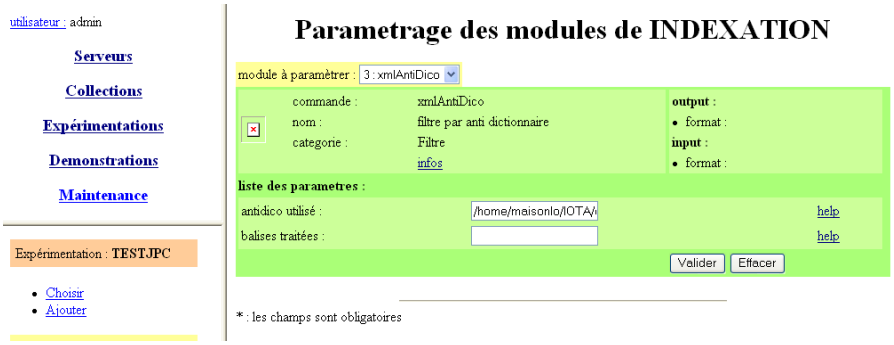

Figure 3. *Exemple d'écran de paramétrisation*

Dans l'état actuel, l'interface de X-IOTA permet de construire paramétrer et de lancer des chaînes de traitement linéaires simples.

5. Conclusion

Le but de cet article est de signaler les difficultés qu'ont les chercheurs du domaine de la RI pour mener à bien des expérimentations, par le manque d'outils flexibles, malgré l'apparence pléthorique de SRI disponibles. Nous proposons alors une architecture simple et ouverte, basée sur des technologies récentes. Cette architecture doit permettre l'intégration rapide de modules de traitements de la langue pour des expérimentations en RI mais aussi pour les domaines connexes comme les systèmes question/réponse ou les recherches de passages (passage retrieval).

Nous pensons que sacrifier un peu d'efficacité de traitement pour gagner en souplesse de construction des expériences, est un pari qui vaut la peine d'être tenter. Il est vrai que l'on peut contester le choix du codage XML pour son manque de compacité et aussi le surcoût engendré par le codage et décodage des informations. La question de pose également dans la communauté XML sur l'utilité de l'introduction d'un "XML Binary" format qui représente de manière plus compacte les données. Une solution consiste alors en un recodage temporaire du flux XML, en respectant la linéarité du flux, c'est à dire que toutes les informations utiles au décodage se trouvent en amont du flux. Nous avons en cours de développement un module de compression/décompression (xml2xbin, xbin2xml), du flux XML pour réduire la taille des données transmises entre les modules. Ce codage permettra également de réduire la

bande passante utilisée lorsque les modules s'échangent des données sur des sites distants.

Le système X-IOTA a déjà été utilisé avec succès pour la campagne CLEF 2003 [SéR 03], également pour l'expérimentation d'extension de requêtes par des règles floues [LAT 03]. Il a montré que son efficacité est suffisante pour ce type de collections de test. Le temps d'indexation et de recherche offre des performances du même ordre que le système intégré SMART. La version actuelle du système avec un nombre suffisant de modules pour réaliser des expériences de bases en RI est disponible à l'adresse xiota.imag.fr. Par la simplicité de mise en oeuvre, ce système devait être adapté à expérimentations pour l'enseignement de la Recherche d'Information.

L'architecture proposée doit encore faire ses preuves au cours d'une expérience de plus grande envergure, impliquant des parties de systèmes réellement différents. Ce sera probablement au cours de la construction d'une plateforme nationale (PLEXIR http ://www.irit.fr/PLEXIR/) en cours de définition. Le système X-IOTA participera au maquettage de cette plateforme de recherche d'information. Elle est financée par le CNRS entre autre par l'intermédiaire du Réseau Thématique Prioritaire "RTP 33 "Documents et contenu : création, indexation, navigation ", en coopération avec les laboratoires suivants :

– IRIT Institut de Recherche en Informatique de Toulouse

– ERSS Equipe de Recherche en Syntaxe et Sémantique

– CLIPS Laboratoire Communication Langagière et Interaction Personne-Système

– IRIN Institut de Recherche en Informatique de Nantes

– LIRIS Laboratoire d'InfoRmatique en Images et Systèmes d'information de Lyon

– LIP6 Laboratoire d'Informatique de Paris 6

– LIMSI Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur

Je tiens à remercier tous ceux qui ont participé au codage du système X-IOTA, ainsi que François Paradis qui par l'expérience de son système expérimental PIF, a inspiré l'orientation du système initial IOTA [CHI 86] vers le système X-IOTA actuel.

6. Bibliographie

[BUC 97] BUCKLEY C., « The SMART Lab Report : The Modern SMART Years (1980-1996). », *SIGIR Forum*, vol. 31, n° 1, 1997.

[BUC 00] BUCKLEY C., MITRA M., WALZ J. A., CARDIE C., « Using clustering and SuperConcepts within SMART : TREC 6 », *Information Processing and Management*, vol. 36, n° 1, 2000.

[CAL 92] CALLAN J. P., CROFT W. B., HARDING S. M., « The INQUERY Retrieval System », *Proceedings of DEXA-92, 3rd International Conference on Database and Expert Systems Applications*, 1992, p. 78–83.

[CHI 86] CHIARAMELLA Y., DEFUDE B., BRUANDET M., KERKOUBA D., « IOTA : a full text information retrieval system », *ACM conference on research and development in information retrieval., Pisa, Italy*, september 8 1986, p. 207–213.

[HAR 92] HARPER D. J., WALKER A. D. M., « ECLAIR : an extensible class library for information retrieval », *The Computer Journal : Special issue on information retrieval*, vol. 35, n° 3, 1992, p. 256 - 267.

[HEN 96] HENDRY D. G., HARPER D. J., « An architecture for implementing extensible information-seeking environments », *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, Zurich, Switzerland, 1996, p. 94-100.

[LAN 04] LANGUATE TECHNOLOGIES INSTITUTE C. M. U., FOR INTELLIGENT INFORMATION RETRIEVAL UNIVERSITY OF MASSACHUSETTS C., « The Lemur Toolkit for Language Modeling and Information Retrieval », www.cs.cmu.edu/ lemur, 2004.

[LAT 03] LATIRI C. C., YAHIA S. B., CHEVALLET J., 3 A. J., « Query expansion using fuzzy association rules between terms », *JIM'2003 conference Journées de l'Informatique Messine, Metz , France*, September 3–6 2003.

[LüB 99] LÜBECK R., RÖLLEKE T., « HySpirit (HYpermedia System with Probabilistic Inference for the Retrieval of InformaTion) a probabilistic deductive system », http ://www.hyspirit.com/go.html, 1999.

[LIM 03] LIMITED O., « Xapian - the open source search engine », Xapian is an Open Source Probabilistic Information Retrieval library, released under the GPL http ://www.xapian.org/, 2003.

[MCC 96] MCCALLUM A. K., « Bow : A toolkit for statistical language modeling, text retrieval, classification and clustering », http ://www.cs.cmu.edu/ mccallum/bow, 1996.

[OUN 04] OUNIS I., AMATI G., VAN RIJSBERGEN C. K., PLACHOURAS V., HE B., CACHEDA F., MACDONALD C., JOHNSON D., LO R. T.-W., CHAN S. K., « Terrier toolkit in Java for the rapid development of Information Retrieval applications », http ://ir.dcs.gla.ac.uk/terrier/index.html, 2004.

[PFE 95] PFEIFER U., GÖVERT N., « FreeWAIS-sf is an extension of the freeWAIS software », http ://ls6-www.cs.uni-dortmund.de/ir/projects/freeWAIS-sf Information Retrieval Group in the Computer Science Department of University of Dortmund, Germany, 1995.

[ROB 97] ROBERTSON S. E., « Overview of the Okapi projects », *Journal of Documentation*, vol. 53, n° 1, 1997, p. 3-7.

[SAL 71] SALTON G., *The SMART retrieval system*, Prentice Hall, Englewood Cliffs, 1971.

[SAL 91] SALTON G., « The Smart Project in Automatic Document Retrieval », *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, Chicago, Illinois, United States, 1991, p. 356 - 358.

[SAN 96] SANDERSON M., « System for Information Retrieval Experiments (SIRE) », rapport, Nov 1996, Technical report, Department of Computing Science, Glasgow University, Glasgow G12 8QQ, Scotland, UK.

[SéR 03] SÉRASSET G., CHEVALLET J.-P., « Simple Translations of Monolingual Queries Expanded through an Association Thesaurus. X-IOTA IR system used for CLIPS Bilingual Experiments », *CLEF : Cross-Language Evaluation Forum*, 2003.

[TOM 97] TOMBROS A., « Reflecting user information needs through query biased summaries », rapport, Nov 1997, MSc Thesis, Technical Report (TR-1997-35) of the Department

of Computing Science at the University of Glasgow, Glasgow G12 8QQ, UK.

[VOO 02] VOORHEES E., « Overview of TREC 2002 », *The Eleventh Text Retrieval Conference (TREC 2002)*, Gaithersburg, Maryland, http ://trec.nist.gov, novembre 2002.

[WIT 99] WITTEN I. H., MOFFAT A., BELL T. C., *Managing Gigabytes : Compressing and Indexing Documents and Images*, Morgan Kaufmann Publishing, San Francisco, mai 1999.

[ZHA 01] ZHAI C., LAFFERTY J., « A study of smoothing methods for language models applied to Ad Hoc information retrieval », *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM Press, 2001, p. 334–342.