

UNIVERSITÉ DE NANTES
FACULTÉ DES SCIENCES

ÉCOLE DOCTORALE SCIENCES ET TECHNOLOGIES
DE L'INFORMATION ET DES MATÉRIAUX

Année 2004

N° attribué par la bibliothèque

--	--	--	--	--	--	--	--	--	--

Extraction d'informations à partir du Web

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE NANTES

Discipline : INFORMATIQUE

présentée et soutenue publiquement par

Benjamin HABEGGER

le 20 décembre 2004

à l'UFR Sciences et Techniques, Université de Nantes

devant le jury ci-dessous

Président	:	Pr. Mohand-Saïd HACID	Université Claude-Bernard Lyon 1
Rapporteurs	:	Georges GARDARIN, Professeur	Université de Versailles St-Quentin en Yvelines
		Amedeo NAPOLI, Directeur de Recherche	INRIA Lorraine
Examineurs	:	Khaldoun ZREIK, Professeur	Université de Caen
		Gilles NACHOUKI, Maître de Conférences	Université de Nantes
		Mohamed QUAFAFOU, Professeur	Université d'Avignon

Directeur de thèse : Pr. Mohamed QUAFAFOU
Laboratoire : Laboratoire d'Informatique de Nantes Atlantique

N° ED 366-192

EXTRACTION D'INFORMATIONS À PARTIR DU WEB

Information extraction from the Web

Benjamin HABEGGER



favet neptunus eunti

Université de Nantes

Benjamin HABEGGER

Extraction d'informations à partir du Web

xvii+210 p.

Ce document a été préparé avec $\text{\LaTeX}2_{\epsilon}$ et la classe `these-IRIN` version 0.92 de l'association de jeunes chercheurs en informatique LOGIN, Université de Nantes. La classe `these-IRIN` est disponible à l'adresse :

`http://login.irin.sciences.univ-nantes.fr/`

Impression : these.tex – 17/4/2007 – 16:41

Révision pour la classe : \$Id: these-IRIN.cls,v 1.5 2005/03/01 10:16:54 habegger Exp

The price of reliability is the pursuit of the utmost simplicity.

— C.A.R. HOARE, The Emperor's Old Clothes, Turing Award Lecture (27 October 1980).

Remerciements

A ma future épouse, Claire.

A mes parents Anne-Catherine et Nathan. A mes frères et soeurs Jeremy, Samuel, Saralina, Rachelle et Timothée.

A mes futurs beau-parents Patrick et Christiane. A mon futur beau-frère Camille.

Merci au Professeur Mohamed Quafafou, mon directeur de thèse, pour son temps, ses conseils et son amitié.

Je remercie profondément les Professeurs Georges Gardarin et Amedeo Napoli pour avoir pris le temps de lire ma thèse et d'en avoir fait le rapport. Merci pour vos commentaires et vos remarques intéressantes.

Je remercie les Professeurs Mohand Saïd Hacid et Khaldoun Zreik et au Docteur Gilles Nachouki d'avoir accepté de faire partie de mon jury de thèse.

Un grand merci à Magali Estienne et sa maman Mireille pour avoir eu le courage de relire ma thèse et d'en corriger les fautes d'orthographe. Je sais que c'était une tâche fastidieuse.

Merci à tous ceux qui, de prêt ou de loin, ont contribué à l'élaboration de cette thèse et à ma recherche. Je pense notamment à Vincent Dubois, Jimmy Leblet, Alban Mancheron, Gwen Salaün, Guillaume Raschia, Sami Naouali et les nombreuses autres personnes avec qui j'ai pu avoir des échanges d'ordre scientifique.

Finalement, merci à Login et à ses membres qui ont contribué et contribuent toujours à créer un cadre de travail agréable au sein du LINA.

Sommaire

Introduction	ix
I Extraction d'informations à partir du web	
Introduction.....	3
1 Extraction d'informations à partir du Web	5
2 Extraction par généralisation de contextes	53
3 Expérimentations du système IEREL	77
Conclusion.....	95
II Construction d'applications d'extraction d'informations à partir du Web	
Introduction	99
4 Applications d'extraction d'informations.....	101
5 Langage WetDL	115
6 Expérimentations du système WEBSOURCE.....	133
7 Web Services et extraction d'informations	141
8 Parcours intelligent du Web	149
Conclusion.....	157
Conclusion générale	159
Bibliographie	163
Liste des tableaux	171
Table des figures	173
Table des matières.....	177
A Notations.....	185
B DTD de WetDL	187
C Exemples de descriptions de tâches en WETDL	193

Introduction

1.1 Problème de l'extraction d'informations

Depuis la démocratisation du Web, de multiples sources de données en lignes, telles que les sites de commerce électronique, les annuaires électroniques, les moteurs de recherche, *etc.* sont apparues. C'est avec de tels sites qu'internet est devenu l'une des sources d'informations les plus riches au monde et un média de choix pour trouver la réponse à un besoin d'information. Cependant, bien que la réponse à un besoin donné se trouve potentiellement sur internet, le volume d'informations mise à disposition des utilisateurs peut rendre cette tâche complexe. Il est alors nécessaire d'interroger plusieurs sources de données différentes, par exemple pour trouver le prix le moins cher pour un produit. Ensuite, dans chaque source, il est nécessaire de parcourir une liste de résultats présentée sur plusieurs pages. Finalement, une fois les réponses trouvées, celles-ci devront être extraites et stockées.

L'automatisation de l'accès à de telles sources d'informations ouvre la porte à de nombreuses applications telles que la mise en place d'agents intelligents permettant d'assister l'utilisateur dans ses recherches d'informations sur internet, l'intégration de sources de données Web dans les systèmes de médiation d'informations, la mise en place de sites de comparaison de prix, *etc.*

Cependant, les sources de données Web ont été conçues pour un accès humain. En effet, leur interrogation se fait via des formulaires, le parcours des résultats se fait page par page en suivant des liens, les informations sont retournées dans des formats de présentation qui n'indiquent pas les types d'informations associés à chaque partie d'une page, *etc.* Ainsi, construire un accès machine à de telles sources est une tâche difficile. Deux problèmes majeurs se posent : d'une part celui de l'extraction des informations utiles d'un ensemble de pages résultats et d'autre part la mise en oeuvre d'une infrastructure permettant l'interrogation de la source, la récupération de l'ensemble des pages résultats et l'extraction même des résultats.

Bien que les pages retournées par une source de donnée Web sont destinées à être affichées dans un navigateur pour être visualisées par l'être humain, certaines régularités dans la présentation de ces données peuvent être détectées et exploitées dans l'objectif d'extraire ces informations et de les mettre sous une forme plus structurée et utilisable par une machine. Un premier problème que doit résoudre l'extraction d'informations est de définir comment exploiter les régularités qui peuvent se trouver dans les pages Web afin d'en extraire le contenu et de le mettre dans un format exploitable par la machine.

Nous avons remarqué que les sources de données disponibles via le Web semblent suivre un schéma de navigation récurrent consistant en un formulaire permettant d'obtenir un ensemble de pages liées entre elles par un lien "suivant". Il apparaît que pour certaines sources de données Web ce schéma ne s'applique pas. Toutefois, la mise en avant de certaines régularités dans les opérations effectuées peut faciliter la description de l'accès aux sources Web en définissant un ensemble d'opérations génériques.

1.2 Illustration du problème d'extraction d'informations

Nous illustrons ici, au travers d'un exemple, les différents problèmes rencontrés lors de l'extraction d'informations d'une source. Ici nous considérerons que les tâches sont effectuées manuellement.

Considérons, le domaine de la vente en ligne et, plus précisément, celui de la vente en ligne de DVD. Différents sites Web tel que `fnac.com`, `alapage.com` ou `amazon.fr` proposent la vente en ligne de DVD. Un utilisateur lambda cherchant à avoir des informations sur les tarifs de vente d'un ensemble de DVD se retrouvera face à différents problèmes.

Le premier problème auquel notre utilisateur aura à faire face est celui de la *recherche de sources* qui consiste à trouver les sites proposant la vente de DVD. En effet, il se peut que ce soit sa première recherche de DVD en ligne. Dans ce cas, il est nécessaire à l'utilisateur de trouver quels sites permettent de répondre à son besoin. Dans le cas de notre exemple, ces sites sont ceux de la FNAC, d'Amazon, *etc.* Un problème secondaire est de trouver le formulaire ou la suite de formulaires permettant d'interroger le site avec les critères de l'utilisateur.

Dans le cas des trois sources citées plus haut, les formulaires permettent à l'utilisateur d'avoir accès à une sélection de DVD selon des critères plus ou moins fins : par exemple il pourra choisir l'acteur, le réalisateur, le type de DVD, *etc.* Toutefois chaque site définit son propre formulaire avec son ensemble de champs spécifiques. Ainsi chaque site propose au travers de son formulaire un langage de requête qui lui est propre. L'utilisateur devra alors traduire son besoin dans ce langage en remplissant correctement le formulaire. Ce second problème auquel aura à faire face est celui d'*interrogation des sources*.

Une fois qu'il aura interrogé une source, l'utilisateur obtient une première page de résultats qu'il visualise dans son navigateur. Dans la plupart des cas celle-ci comportera un lien "suivant" permettant d'accéder à une nouvelle page de résultats. Ainsi l'utilisateur se retrouve à faire un *parcours des pages résultats*.

Selon les sources, les pages contiennent un certain nombre de réponses par page (souvent une ou deux dizaines). Ainsi, la plupart du temps, l'utilisateur n'a pas accès à chaque réponse de manière indépendante. S'il souhaite sauvegarder une réponse particulière il devra soit la copier à la main, soit sauvegarder l'ensemble de la page sur laquelle apparaît la réponse, soit imprimer la réponse (en annotant éventuellement la feuille pour marquer la réponse qu'il a sélectionnée). Toutefois, certains sites offrent des outils permettant de se créer un panier de réponses. Dans ce cas l'utilisateur peut sélectionner certaines réponses qu'il ajoute à son panier puis sauvegarder ou imprimer la page du panier. Par contre, il ne pourra pas conserver dans un seul et même fichier toutes les réponses sélectionnées sur des sites différents. De plus, les fichiers sauvegardés étant du HTML, aucun traitement ne pourra être effectué après la sauvegarde. Par exemple, il ne sera pas possible d'ajouter ou de supprimer des réponses, ni d'intégrer les réponses obtenues dans une base de données.

La principale cause du problème est que les pages n'ont pas été générées dans l'objectif d'être traitées par machine, mais pour être visualisées par l'utilisateur. Même si les pages ont été obtenues à partir d'une base de données, la structure de la base a été enfouie dans la présentation. Cependant certaines régularités de présentations existent : tout les valeurs d'un même type d'élément (*i.e.* le titre d'un DVD, le prix du DVD, la réponse décrivant le DVD lui-même, *etc.*) sont présentées (*e.g.* mise en gras, en italique, *etc.*) et formatées (*e.g.* le titre est placé au début de la description et le prix est placé à la fin, *etc.*) de manières similaires. Ainsi les informations de la structure qui lient les données entre elles sont en fait cachées dans la présentation. La transformation des données de la base en pages HTML peut alors être vue comme un codage de l'information dans un format particulier. Seulement, le format est inconnu de la machine.

Ainsi pour accéder à chacune des réponses de manière individuelle, il est nécessaire de procéder à une *extraction des résultats*. Celle-ci peut être automatisée à condition d'indiquer à la machine le format dans lequel les réponses ont été retournées. Ceci peut être fait en analysant le code HTML produit par les sources. Seulement, le format d'une page est spécifique à la source à laquelle elle appartient. Il faudra donc déterminer le format des pages de chacune des sources. Si on cherche à traiter des réponses d'une nouvelle source, il faudra déterminer le format de cette nouvelle source. De plus les sites Web sont

souvent mis à jour en changeant de présentation et donc la description du format d'une source mise à jour devra être modifiée en conséquence. A partir de cette description du format et d'un programme adéquat (sachant appliquer la description à une page) on obtient un nouveau programme appelé adaptateur permettant l'extraction des informations de la source.

Dans certains cas, le format des données extraites par un adaptateur est proche de celui de la source (*e.g.* l'ordre d'apparition des valeurs de la relation extraite est le même que celui de la source, les données ont la même structure logique). Dans ce cas une *harmonisation des formats* sera nécessaire pour avoir toutes les réponses des différentes sources dans le même format.

Pour résumé, la résolution du problème d'accès à des sources de vente de DVD en ligne, ou de manière plus générale le problème d'extraction d'informations à partir du Web, passe par la résolution de plusieurs sous-problèmes :

- la recherche des sources contenant l'information souhaitée ;
- l'interrogation des sources dans leur langage d'interrogation ;
- le parcours des pages résultats ;
- l'extraction des résultats ;
- transformer les données extraites dans un format commun.

1.3 Contributions de la thèse

Dans ce mémoire nous nous intéressons au problème de l'automatisation de tâches d'extraction d'information à partir du Web. La motivation principale derrière cette volonté d'automatisation, est issue des possibilités qu'elle permet d'envisager. Par exemple, elle permet l'intégration des sources de données Web dans des systèmes de médiations d'informations, la mise en oeuvre d'agents intelligents pouvant accéder à plusieurs sources Web, *etc.* Dans cette thèse nous nous intéressons principalement à deux problèmes : celui de la construction d'adaptateurs pour des sources de données Web et la mise en oeuvre de l'infrastructure pour la construction d'applications Web.

Pour répondre au premier problème, nous proposons une méthode de construction d'adaptateurs pour une source donnée. Cette méthode est basée sur la généralisation des contextes. En entrée, l'utilisateur donne un ensemble d'instances exemples d'une relation à extraire. Les contextes de ces instances sont alors recherchés dans un ensemble de pages issus de la source. La généralisation de ces contextes permet de construire des motifs permettant l'extraction de nouvelles instances. L'application de l'adaptateur sur les pages de la source permet alors d'obtenir les instances de la relation décrite par les exemples fournis par l'utilisateur. Cette méthode a plusieurs avantages par rapport aux méthodes existantes. Elle ne nécessite pas l'étiquetage d'un ensemble de pages exemples. Lors de la reconstruction d'un adaptateur suite à un changement de format de la source aucun étiquetage est nécessaire. En effet, les instances extraites avant le changement de format peuvent être utilisées pour construire un nouvel adaptateur. De plus la méthode de généralisation est efficace. Très souvent un adaptateur est construit en moins d'une seconde. Cette méthode a été implémentée par le système IEREL et évalué sur des données de différents domaines.

Pour répondre au second problème qui consiste à la mise en place de l'infrastructure nécessaire permettant un accès totalement automatisé à un site Web, nous proposons de décomposer une tâche d'extraction en un ensemble de sous-tâches. Les sous-tâches peuvent être décrites à l'aide d'un ensemble d'opérateurs génériques. Pour décrire la tâche d'extraction et les opérateurs qui la composent nous avons défini le langage XML WETDL. Il permet de définir un réseau d'opérateurs et leurs paramètres. L'exécution du réseaux permet de réaliser concrètement la tâche décrite. Cette méthode a été implémentée dans

le système WEBSOURCE et différentes tâches d'extractions ont été décrites en WETDL et exécutées à l'aide de WEBSOURCE.

1.4 Organisation du mémoire

Ce mémoire est organisé en trois parties. La première partie étudie le problème de la construction d'adaptateurs pour des sources de données Web. Elle est composée de trois chapitres. Le chapitre 1 donne un aperçu des approches et des méthodes d'extraction d'informations à partir du Web existantes. Dans le chapitre 2 nous présentons notre méthode de construction d'adaptateurs basée sur la généralisation de contextes. Le chapitre 3 présente les expérimentations que nous avons réalisées à l'aide du système IEREL. La seconde partie étudie le problème de la mise en oeuvre de l'infrastructure pour la construction d'applications Web. Cette partie est organisée en trois chapitres. Dans le chapitre 4 nous donnons différents exemples d'applications Web. Dans le chapitre 5 nous présentons le langage WETDL. Dans le chapitre 6 nous présentons nos expérimentations réalisées à l'aide du système WEBSOURCE. Dans la troisième partie nous proposons quelques sujets d'études liés au problème de l'extraction d'informations à partir du Web. Dans le chapitre 6 nous étudions le lien entre la construction d'adaptateurs et l'apprentissage de concepts. Dans le chapitre 7 nous abordons les liens entre l'extraction d'informations et les Web Services. Enfin, dans le chapitre 8 nous nous intéressons au problème du parcours intelligent du Web et de sa description comme une tâche d'extraction d'informations.

PARTIE I

Extraction d'informations à partir du web

Introduction

Dans cette partie nous nous intéressons au problème de l'extraction des informations contenus dans les pages Web issues de sources de données Web. L'objectif de cette extraction est de permettre un accès machine aux sources de données en ligne. La principale motivation vient du fait que depuis la démocratisation du Web de multiples sources d'informations en ligne sont apparues. Celle-ci sont nombreuses et riches en contenus. Donner un accès automatisé à de telles sources de données rend envisageable de multiples applications telles que l'utilisation de ces sources par des agents intelligents, leur intégration dans des systèmes de médiations de données, *etc.*

L'interrogation de ces sources se fait au travers d'un ou plusieurs formulaires qui permettent d'interroger la source d'information. Une fois rempli, le formulaire permet de construire une requête dont l'exécution engendre une réponse qui prend la forme d'une ou plusieurs pages Web. Ces pages sont destinées à être visualisées dans un navigateur par un utilisateur humain. Ainsi, bien qu'elles soient riches en informations, ces sources ne sont pas directement utilisables par les machines. Il est nécessaire que, d'une part, les données soient structurées et, d'autre part, que la sémantique des données soit connue à l'avance. Cependant, certaines régularités dans la mise en forme des pages apparaissent dans les données provenant du Web. En effet, les contenus des pages issues d'une source de données Web ont été obtenus à partir de données structurées appartenant à une base de données et mise en forme de telle sorte que les mêmes types d'informations soient présentées de la même manière. Ces régularités de mise en forme donnent aux documents une structure de présentation qui peut être exploitée afin d'extraire de ces pages les informations structurées dont l'utilisateur a besoin. L'objectif de l'extraction d'information à partir du Web est alors de construire des programmes capables, d'extraire d'une page Web une partie de son contenu et de le mettre dans un format structuré. Selon les cas, ce programme, souvent appelée *adaptateur*, peut être construit pour être appliqué à un ensemble de documents provenant d'une même source, à un ensemble de documents appartenant à un même domaine, ou à n'importe quel document du Web. Un tel adaptateur prend souvent la forme d'un programme générique appliquant un ensemble de motifs (ou règles d'extractions) à une page Web.

Le problème d'extraction d'informations à partir du Web se distingue de celui de l'extraction d'informations à partir de données en langage naturel. En effet, les méthodes d'extraction d'informations à partir du Web s'appuient sur la structure de présentation des documents et non la structure grammaticale des phrases ou une analyse lexicale du document. Une remarque importante est que le problème d'extraction d'informations à partir du Web s'intéresse aux parties des documents ayant une forte structuration (*e.g.* des tableaux, des énumérations, *etc.*) et non les parties purement textuelles en langage naturel. Ainsi nous avons deux formes d'extraction d'informations, l'extraction d'information en langage naturel et l'extraction d'information à partir du Web. Tout au long de cette thèse, et sauf en cas de précision contraire, "extraction d'information" signifiera extraction d'information à partir du Web.

Dans cette partie, nous nous intéressons au problème de la construction d'adaptateurs pour des données issues du Web. Elle est organisée en trois chapitres. Un premier chapitre présente les méthodes d'extraction d'informations existantes dans la littérature. Un second chapitre présente notre contribution dans la génération d'adaptateurs : une méthode basée sur la généralisation des contextes d'occurrences d'un ensemble d'instances exemples. Finalement, un troisième chapitre présente l'implémentation de notre méthode par le système IEREL et les expérimentations réalisées avec ce système.

CHAPITRE 1

Extraction d'informations à partir du Web

Différentes approches permettant d'obtenir un adaptateur existent dans la littérature. Celles-ci se distinguent par le type de données qu'elles prennent en entrée et le type d'adaptateur qu'elles donnent en sortie. Par exemple, certaines méthodes nécessitent d'avoir un ensemble de pages exemples avec un étiquetage des informations à extraire, certaines autres nécessitent une connaissance *a priori* du domaine, alors que d'autres permettent de construire un adaptateur simplement à partir des pages brutes. En sortie, ces méthodes, donnent un ensemble de motifs ou de règles d'extraction pouvant être appliqués soit aux pages d'une source donnée, soit aux pages d'un domaine donné, soit à n'importe quelle page Web.

Les premiers travaux sur la génération d'adaptateurs proposent des langages permettant de décrire des règles d'extractions. Dans ce cas les règles sont construites manuellement par le concepteur de l'adaptateur. Toutefois, certains outils permettant de faciliter l'écriture des règles, en proposant de guider interactivement l'utilisateur dans sa tâche de construction de motifs, existent dans la littérature.

Malgré cela, la volonté de pouvoir automatiser le processus de construction d'adaptateurs, ont rapidement suscité de nouvelles recherches. Celles-ci sont devenues d'autant plus nécessaires que l'évolution rapide du Web et le changement fréquent des formats de présentation obligent à recommencer souvent le travail d'écriture des adaptateurs. C'est ainsi que sont apparus les premières méthodes de construction, plus ou moins automatiques, d'adaptateurs. Les toutes premières méthodes ont posé le problème comme un problème d'apprentissage inductif. Les données en entrée sont alors des pages exemples dont le contenu à extraire est étiqueté (*i.e.* les parties à extraire ont été clairement identifiées par l'utilisateur). D'autres méthodes prennent une approche à base de connaissances. Dans ce cas, des connaissances sur les concepts du domaine et les formats dans lesquels ces concepts apparaissent sont utilisées pour extraire les informations. Finalement, certaines méthodes proposent de construire des motifs d'extractions en analysant la structure (*i.e.* les régularités de présentation : tables, listes, *etc.*) des documents. Dans ce cas aucune donnée en plus des documents n'est nécessaire.

Parallèlement, certains auteurs se sont intéressés au problème d'extraction d'instances d'une relation donnée (*e.g.* la relation (*auteur, titre*)) à partir d'un sous-ensemble conséquent du Web (*e.g.* la partie indexée du Web). Ce problème est similaire dans ses objectifs à celui de la construction d'un adaptateur. Cependant il se distingue par le fait qu'il ne considère pas qu'une seule source d'information mais le Web dans sa globalité (ou du moins sa partie indexée). Dans ce cas, l'objectif est d'extraire d'un sous-ensemble du Web, un ensemble significatif d'instances d'une relation donnée. Les données considérées en entrée sont un ensemble d'instances exemples d'une relation que l'utilisateur cherche à extraire. En sortie, un ensemble de motifs d'extractions sont générés. En appliquant ces motifs à une page Web de nouvelles instances de la relation donnée sont extraites.

Les méthodes d'extraction d'informations existantes peuvent donc se classer selon le type des données qu'elles prennent en entrée. Ainsi nous avons les approches manuelles, les approches inductives semi-automatiques nécessitant un ensemble de pages exemples étiquetées, les approches structurales ne nécessitant que les pages brutes, les approches à bases de connaissances, et les approches nécessitant uniquement un ensemble d'instances exemples d'une relation donnée. Nous présentons dans ce chapitre ces différentes approches ainsi que les principales méthodes associées trouvées dans la littérature.

1.1 Construction manuelle d'adaptateurs

L'approche naturelle pour construire une procédure d'extraction d'informations pour une source Web est de construire manuellement l'ensemble des motifs ou règles d'extraction. De nombreux systèmes proposent simplement des solutions pour extraire les informations à partir de règles prédéfinies sans préciser comment les règles sont obtenues. D'autres systèmes mettent en place des outils d'aide pour assister l'utilisateur dans la création de règles.

Hammer et al. [55] décrivent un système qui permet l'extraction de données semi-structurées à partir d'une page Web, mais, c'est l'utilisateur qui spécifie les règles d'extraction. Les règles d'extraction sont des triplets $\langle \text{variables}, \text{source}, \text{motif} \rangle$ où *source* décrit le fragment de texte considéré, *variables* est une liste de variables qui contiendront les résultats d'extraction, et *motif* est une expression permettant d'extraire les valeurs des variables dans le fragment donné. Par exemple le triplet $\langle [ville, loyer, surface], annonce, "<TD>\#</TD><TD>Loyer: \#</TD>\#EUR<TD>" \rangle$ signifie que les variables *ville*, *loyer* et *surface* seront extraites en appliquant au fragment correspondant à *annonce* le motif "`<TD>\#</TD><TD>Loyer: \#</TD>\#EUR<TD>`", les '#' correspondent aux parties extraites. Les données qui sont extraites ont un format intrinsèquement hiérarchique car l'extraction d'une variable se fait dans le fragment d'une autre variable. Ainsi le résultat obtenu aura comme format un arbre de données semi-structurées reflétant la structure logique du document. Dans l'exemple, la variable *annonce* correspond à un noeud et les variables *ville*, *loyer*, et *surface* à ses fils.

Différents systèmes, comme celui proposé par Adelberg [1] ou le système W4F proposé par Sahu-guet et Azavant [102, 103] définissent des langages permettant la description d'un ensemble de règles permettant l'extraction de types structurés (enregistrements, listes, ensembles). Dans les deux cas, un outil interactif d'aide à la construction des règles est proposé.

May et al. [78] proposent une architecture d'accès et d'extraction d'informations à partir du Web qui s'appuie sur la F-Logique¹. Dans cette méthode la médiation et la restructuration se font dans un même langage : celui de la F-Logique. Cependant les auteurs ne proposent qu'une architecture de médiation sans donner de solutions pour automatiser la construction des règles d'extractions.

Un autre outil plus récent est LIXTO proposé par Baumgartner et al. [4, 5]. Cette méthode est intéressante car elle propose un langage de description de règles et un langage de représentation des documents qui sont tout deux riches. Ces deux langages sont basés sur la logique du premier ordre. Pour la construction des règles, la méthode offre une interface simple et conviviale pour créer les règles avec une automatisation de certaines phases. Cependant dans une telle approche, même si l'utilisateur est assisté, c'est à lui de construire les règles d'extraction. D'autre part, bien que les résultats de la phase d'extraction permettent d'obtenir un document XML, l'arborescence XML obtenue est très proche de l'organisation logique du document.

En résumé, ces différents travaux proposent uniquement des langages permettant la description de règles d'extraction. Dans certains cas, un outil d'aide à leur construction est proposé à l'utilisateur. Par

¹La F-Logique est un langage logique intégrant des notions issues de la programmation objet

ailleurs, il arrive que le problème d'extraction d'informations ne soit pas explicitement séparé de celui de la médiation de données. Ainsi, certains systèmes proposent un outil et un langage commun pour les deux tâches. Finalement, les langages d'extractions proposés sont souvent très riches. Cependant, ces systèmes obligent l'utilisateur à construire ses règles d'extraction et ne permettent pas l'automatisation de cette tâche.

1.2 Induction d'adaptateurs à base de pages étiquetées

La construction manuelle d'adaptateurs est une tâche fastidieuse surtout face au nombre de sources pour lesquelles une telle tâche est nécessaire. C'est pour cette raison que l'idée d'automatiser cette tâche est apparue. Ce fut N. Kushmerick qui dans sa thèse [68] proposa une première méthode. Toutefois cette méthode ne permet de traiter que 70% des pages contenant des données structurées. Ce résultat est dû à un manque d'expressivité des motifs utilisés pour l'extraction. Pour combler cette lacune d'autres méthodes cherchant à construire un adaptateur à partir de pages exemples étiquetées sont apparues.

Nous nous intéresserons ci-après aux trois principales méthodes de construction d'adaptateurs utilisant de l'apprentissage à partir de pages exemples étiquetées. Dans l'ordre nous présentons le système WIEN proposé par Kushmerick initialement en 1997 et puis en 2000, puis le système STALKER proposé par Muslea et al. en 1998 et complété en 2001, et finalement le système SOFTMEALY proposé par Hsu et Dung en 1998.

1.2.1 WIEN

Dans ses travaux sur l'induction d'adaptateurs [68, 67, 65], Kushmerick a, d'une part, proposé une première formalisation du problème de construction d'un adaptateur pour une source d'information Web et, d'autre part, mis en oeuvre un système d'induction d'adaptateur : le système WIEN.

La formalisation s'appuie sur les notions de *source d'information*, de *page Web*, de *contenu* d'une page Web et d'*adaptateur*. Selon lui, une source d'information Web est "un système qui répond à une requête générant une réponse qui correspond *a priori* à la requête." Une implémentation classique est l'utilisation d'un script CGI appelé par un formulaire HTML, interrogeant une base de donnée et générant une réponse au format HTML. Cependant, dans le cadre de l'induction d'adaptateurs, cette source d'information peut être simplement vue comme une boîte noire générant un ensemble de documents Web.

1.2.1.1 Formalisation de la notion d'adaptateur

Une source d'information est une fonction permettant d'obtenir à partir d'une requête Q une réponse P . Il reste alors à définir les notions de requêtes et de réponses.

Définition 1.1 (Requête). Une requête Q est une expression d'un langage défini $\mathcal{Q}$ permettant de décrire l'information souhaitée.

Définition 1.2 (Réponse). Une réponse P est une chaîne de caractères d'un alphabet Σ qui contient la réponse à une requête.

Ainsi une source d'information peut se formaliser comme suit.

Définition 1.3 (Source d'information). *Une source d'information est une fonction*

$$\mathcal{S} : \mathcal{Q} \rightarrow \Sigma^* \quad (1.1)$$

où $\mathcal{Q}$ est un langage de requête et Σ est l'alphabet sur lequel sont construit les réponses.

L'un des principaux objectifs de la construction d'adaptateurs est qu'ils permettent de faire de la médiation de sources d'informations. La médiation de sources d'informations consiste à construire un système appelé *médiateur* permettant un accès homogène (*i.e.* utilisant un unique langage de requête et un même format pour les résultats des requêtes) à de multiples sources d'informations. Construire de tels systèmes pose plusieurs problèmes : celui de l'interrogation des sources ou comment traduire une requête posée au médiateur dans le langage de requête de la source et celui de l'extraction de l'information de la source. Dans le cas de l'induction d'adaptateurs les problèmes liés à l'interrogation d'une source sont mis à part. On s'intéresse uniquement à la partie extraction. Ainsi, on peut considérer qu'une source d'information est l'ensemble des réponses qu'elle peut produire. Ces réponses prennent la forme de documents HTML. Une source peut alors être considérée comme un ensemble de documents HTML. Un exemple de document d'une source d'information fictive est donné dans la figure 1.1. Celui-ci décrit les produits d'un distributeur automatique avec leurs tarifs en euros et en francs.

```

1 <html>
2 <body>
3   <h1>Les tarifs du distributeur</h1>
4   <table>
5     <tr><td>Café</td>
6       <td>0.40 EUR (2.96 F)</td></tr>
7     <tr><td>Soda</td>
8       <td>0.75 EUR (4.92 F)</td></tr>
9     <tr><td colspan="2"><hr></td></tr>
10    <tr><td>Gâteau</td>
11      <td>0.50 EUR (3.28 F)</td></tr>
12    </table>
13  </body>
14 </html>

```

Figure 1.1 – Un exemple de document d'une source fictive

En ce qui concerne les données à extraire, le modèle de données (ou format que vont prendre les données extraites) adopté est le modèle relationnel. Ainsi, un ensemble de K attributs distincts, représentant chacun une colonne du modèle relationnel est associé à chaque source d'information. Un tuple de la relation est un vecteur $\langle A_1, \dots, A_K \rangle$ de K chaînes où pour chaque i , $A_i \in \Sigma^*$. La chaîne A_i correspond à la valeur du $i^{\text{ème}}$ attribut.

Le contenu d'une réponse R est alors l'ensemble des tuples qu'elle contient. La représentation du contenu d'une page se fait au moyen d'un *label*. Un label est une matrice dont les éléments, appelés fragments, sont les sous-chaînes à extraire du document. Chaque fragment est représenté par un couple d'indices $(b_{i,j}, e_{i,j})$. Dans un label, chaque ligne correspond à un tuple et chaque colonne à un attribut. Ainsi la valeur du $j^{\text{ème}}$ attribut du $i^{\text{ème}}$ tuple sera représentée par un couple $(b_{i,j}, e_{i,j})$ où $b_{i,j}$ et $e_{i,j}$ représentent respectivement les indices de début et de fin de la valeur dans R . La figure 1.2 donne le label d'une page dont la relation à extraire contient M tuples .

$$L = \left\{ \begin{array}{cccc} (b_{1,1}, e_{1,1}) & \cdots & (b_{1,k}, e_{1,k}) & \cdots & (b_{1,K}, e_{1,K}) \\ & & \vdots & & \\ (b_{m,1}, e_{m,1}) & \cdots & (b_{m,k}, e_{m,k}) & \cdots & (b_{m,K}, e_{m,K}) \\ & & \vdots & & \\ (b_{M,1}, e_{M,1}) & \cdots & (b_{M,k}, e_{M,k}) & \cdots & (b_{M,K}, e_{M,K}) \end{array} \right\}$$

Figure 1.2 – Label d'une page contenant M tuples d'une relation à K attributs.

Par exemple, le contenu de la page de la figure 1.1 est celui décrit par le label de la figure 1.3 où les couples de nombres correspondent aux positions de début et de fin de la valeur correspondante dans le label. Dans ce cas les attributs à extraire sont le nom du produit, son prix en euros et son prix en francs.

$$L = \left\{ \begin{array}{ccc} (77, 81) & (103, 105) & (113, 115) \\ (143, 147) & (167, 171) & (178, 181) \\ (250, 256) & (276, 280) & (286, 290) \end{array} \right\}$$

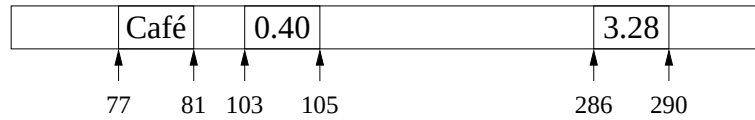


Figure 1.3 – Le label du document de la figure 1.1

Les cas de contenus auxquels s'intéresse Kushmerick sont ceux qui prennent une forme similaire à une table ou ayant un contenu *tabulaire*.

Définition 1.4 (Contenu tabulaire). *Un contenu est dit tabulaire lorsqu'il respect les contraintes suivantes :*

- Les fragments doivent être définis

$$\forall k, m, b_{m,k} \leq e_{m,k} \quad (1.2)$$

- Les fragments doivent être disjoints

$$\forall k, m, \neg \exists k', m', (b_{m,k} \leq b_{m',k'} \leq e_{m,k}) \vee (b_{m,k} \leq e_{m',k'} \leq e_{m,k}) \quad (1.3)$$

- Tous les attributs d'un tuple précèdent ceux du tuple suivant

$$\forall m < M, e_{m,K} < b_{m+1,1} \quad (1.4)$$

- Les attributs et leur occurrence dans le document se présentent dans le même ordre

$$\forall m, k < K, e_{m,k} < b_{m,k+1} \quad (1.5)$$

Finalement, étant donné ces définitions, un adaptateur peut se définir comme suit.

Définition 1.5 (Adaptateur). *Un adaptateur est une fonction $\mathcal{W} : \Sigma^* \rightarrow \mathcal{L}$ qui à une page $P \in \Sigma^*$ associe un label $L \in \mathcal{L}$ pour cette page.*

1.2.1.2 Délimiteurs de fragments

L'objectif d'un adaptateur est d'extraire les fragments d'un document contenant l'information qui intéresse l'utilisateur. Extraire un fragment revient à trouver sa position de début et sa position de fin. Kushmerick introduit la notion de *délimiteur*. Un délimiteur est une chaîne de caractères construite sur le même alphabet Σ que les documents desquels on souhaite extraire le contenu et permettant d'identifier une position de début ou de fin d'une valeur, d'un tuple, ou un d'ensemble de tuples. Reprenons l'exemple de la figure 1.1, le délimiteur '><td>' permet d'identifier le début des chaînes 'Café', 'Soda', et 'Gâteau'. Dans le cas où un délimiteur détermine une position de début, on dira qu'il s'agit d'un délimiteur gauche et lorsqu'il détermine une position de fin on dira qu'il s'agit d'un délimiteur droit.

1.2.1.3 Construction d'adaptateurs par apprentissage inductif

D'une manière générale, l'apprentissage inductif, consiste à caractériser un sous-ensemble d'éléments ou un *concept* par une description générale. Celle-ci est obtenue à partir de la description d'un sous-ensemble d'éléments du concept. Plus formellement, une tâche d'apprentissage s'appuie sur :

1. un ensemble $\mathcal{I}$ d'instances ;
2. un ensemble $\mathcal{L}$ de labels ;
3. un ensemble $\mathcal{H}$ d'hypothèses ;
4. un ensemble d'exemples étiquetés $\mathcal{E} = \{\dots, \langle I, \mathcal{T}(I) \rangle, \dots\}$ où $\mathcal{T}$ est un oracle permettant d'obtenir le label d'une instance ;
5. une fonction de généralisation $g_{\mathcal{H}}$.

Chaque hypothèse $H \in \mathcal{H}$ est une fonction de $\mathcal{I}$ à $\mathcal{L}$. Dans ce contexte l'objectif de l'induction consiste à construire une hypothèse $H \in \mathcal{H}$ permettant de décrire un ensemble d'instances exemples $\mathcal{E} \subset \mathcal{I} \times \mathcal{L}$. Pour ce faire l'algorithme d'induction général utilisé par Kushmerick fait appel à un ensemble d'exemples étiquetés, et une fonction de généralisation permettant de généraliser une hypothèse pour qu'elle couvre un ensemble donné d'exemples.

Dans le cadre de l'induction d'adaptateurs on a les correspondances suivantes :

1. une instance $I \in \mathcal{I}$ est une page réponse ;
2. les labels correspondent aux labels des réponses ;
3. les hypothèses correspondent à des adaptateurs candidats ;
4. les exemples sont des pages réponses associées à leurs labels ;
5. les fonctions de généralisation dépendant de la classe d'adaptateurs à laquelle doit appartenir l'adaptateur que l'on cherche à apprendre.

La classe d'adaptateurs la plus simple étudiée par Kushmerick est la classe LR (Left-Right). Cette classe correspond à la fonction générique de l'algorithme 1.1 ayant $2K$ paramètres $(l_1, r_1, \dots, l_K, r_K)$. Chaque paramètre est un délimiteur gauche (l_i) ou droit (r_i) associé à un attribut de la relation à extraire. L'ordre des paramètres détermine l'ordre dans lequel les valeurs des attributs d'un tuple devront apparaître dans le document. L'application de la procédure de l'algorithme 1.1 sur un document a pour résultat l'extraction d'un ensemble de tuples. L'extraction se fait tuple par tuple en recherchant successivement et dans l'ordre chacun des $2K$ paramètres. L'extraction d'un tuple commence par la recherche de la première occurrence du délimiteur l_1 à partir de la position courante (ie. le début du document si aucun tuple n'a encore été extrait ou la position suivant l'occurrence du délimiteur l_K qui a terminé l'extraction

du tuple précédant). Ensuite la première occurrence de r_1 est recherchée à partir de la position de fin de l'occurrence de l_1 . La sous-chaîne du document comprise entre les occurrences de l_1 et r_1 constitue la première valeur du tuple (*i.e.* celle correspondant à l'attribut A_1). Les valeurs du tuple correspondants aux autres attributs sont extraites de la même manière et dans l'ordre fixé par les paramètres. La procédure s'arrête lorsque, après l'extraction d'un tuple, aucune nouvelle occurrence de l_1 ne peut être trouvée.

Procédure $exec_{LR}$
Entrées

- Adaptateur $W = (l_1, r_1, \dots, l_K, r_K)$
- Page P

Sorties

- Label $L = (\dots, ((b_{m,1}, e_{m,1}), \dots, (b_{m,K}, e_{m,K})), \dots)$

$m \leftarrow 0$

tantque il reste des occurrences de l_1 **faire**

$m \leftarrow m + 1$

pour k **de** 1 **à** K **faire**

recherche de la prochaine occurrence de l_k dans P

sauver la position dans $b_{m,k}$

recherche de la prochaine occurrence de r_k dans P

sauver la position dans $e_{m,k}$

fin pour

fin tantque

Algorithme 1.1 – Algorithme $exec_{LR}$

Trouver une procédure de la classe LR , à partir des exemples donnés par l'utilisateur, revient à déterminer ses $2K$ paramètres $(l_1, r_1, \dots, l_K, r_K)$. Pour construire un algorithme permettant de trouver ces paramètres pour une source donnée, Kushmerick a mené une étude permettant de déterminer des contraintes que doivent vérifier ces paramètres. Ainsi, le problème de l'induction d'un adaptateur peut alors se poser comme un problème de satisfaction de contraintes. Les variables sont les paramètres de la fonction. Toutes les variables ont pour domaine l'ensemble Σ^* , ensemble de toutes les chaînes. Les contraintes sont que $W(P_n) = L_n$ pour chaque $(P_n, L_n) \in \mathcal{E}$ où W est un adaptateur LR avec les paramètres $(l_1, r_1, \dots, l_K, r_K)$, P_n est la $n^{\text{ième}}$ page exemple de la source et L_n est son label.

Afin de résoudre ce problème, la solution proposée par Kushmerick est une approche "générer et tester" : un ensemble de délimiteurs candidats est généré pour chacune des variables et leur validité est testée par rapport à un ensemble de contraintes. La génération des candidats pour un délimiteur gauche l_k (resp. droit r_k) se fait en analysant toutes les chaînes qui précèdent (resp. suivent) les valeurs de l'attribut k . Les candidats pour un délimiteur l_k sont choisis parmi les suffixes de la chaîne précédant une valeur de l'attribut k la plus courte. La construction des candidats pour l_k se fait en deux temps. D'abord on construit l'ensemble des sous-chaînes précédant toutes les occurrences de l'attribut k . Celui-ci est obtenu à partir du label associé à chaque document et correspond aux sous-chaînes se trouvant entre une occurrence de l'attribut $k - 1$ et une occurrence de l'attribut k . Lorsque $k = 1$, l'occurrence prise en compte est celle de l'attribut d'indice K (*i.e.* le dernier) du tuple précédant lorsqu'il existe. Dans le cas de la première occurrence du premier attribut, c'est toute la chaîne du début du document jusqu'au début

de l'occurrence qui est pris comme délimiteur candidat. Ainsi nous pouvons définir une fonction qui, pour un document P et un label L donnés, extrait la chaîne précédant la valeur de l'attribut k de la $i^{\text{ème}}$ instance. Cette fonction que nous noterons $prec_{P,L}(i, k)$ se définit comme suit :

$$prec_{P,L}(i, k) = \begin{cases} P[0, b_{1,1}^L] & \text{si } i = 1 \text{ et } k = 1 & (1) \\ P[e_{i-1,K}^L, b_{i,1}^L] & \text{si } i \neq 1 \text{ et } k = 1 & (2) \\ P[e_{i,k-1}^L, b_{i,k}^L] & \text{sinon} & (3) \end{cases} \quad (1.6)$$

Un candidat est valide lorsqu'il respecte les contraintes définies ci-dessous. Celles-ci peuvent être précisés en quatre contraintes sur les candidats u des chaînes l_k et r_k .

Deux contraintes s'appliquent sur les délimiteurs gauches. La contrainte $\mathcal{C}_l^A$ impose à chaque candidat u pour l_i d'être un suffixe propre² de l'ensemble des chaînes $\{prec(i, k)/i \in [1, N]\}$, c'est à dire que u doit être un suffixe de chaque $prec(i, k)$ et que ce suffixe doit être l'unique occurrence de u dans $prec(i, k)$. La contrainte $\mathcal{C}_l^B$ exprime le fait qu'on ne doit pas trouver d'occurrences de la chaîne l_1 après avoir extrait la dernière valeur du dernier tuple dans la page. En effet, chaque nouvelle occurrence de l_1 détecte la présence d'un nouveau tuple.

Contrainte $\mathcal{C}_l^A$: u doit être un suffixe propre du texte précédant chaque instance de l'attribut k et ce dans chaque page.

Contrainte $\mathcal{C}_l^B$: pour l_1 , u ne doit pas être une sous-chaîne suivant la dernière instances du dernier attribut d'une page.

Deux autre, contraintes s'appliquent aux délimiteurs droits. La contrainte $\mathcal{C}_r^A$ impose aux candidats u permettant de délimiter la fin d'une valeur d'un attribut d'être un préfixe de toutes les chaînes suivant les valeurs de cet attribut. La contrainte $\mathcal{C}_r^B$ exprime le fait qu'un délimiteur droit r_i ne doit pas être une sous-chaîne d'une des valeurs de l'attribut k . En effet, si ce n'était pas le cas, la valeur contenant la sous-chaîne r_i ne serait pas extraite entièrement (du à la détection prématurée de r_i).

Contrainte $\mathcal{C}_r^A$: u doit être un préfixe du texte suivant immédiatement chaque instance de l'attribut k et ce dans chaque page.

Contrainte $\mathcal{C}_r^B$: u ne doit pas être une sous-chaîne d'une instance de l'attribut k dans aucune page.

L'algorithme 1.2 décrit comment est construit le délimiteur gauche l_i d'un adaptateur de la classe LR étant donné un ensemble de pages et leur labels associés. La construction des délimiteurs droits r_i se fait de manière analogue. Pour construire r_i un ensemble de candidats est généré à partir des préfixes de la chaîne se trouvant dans la première page entre les premières occurrences des attributs k et $k + 1$. Les contraintes sont appliquées à chacun des candidats, jusqu'à ce qu'il y en ait un qui les vérifient toutes. Ce candidat est alors donné comme solution.

En plus de la classe LR , Kushmerick propose diverses classes d'adaptateurs et une méthode de généralisation pour chacune d'entre elles. Par exemple il propose la classe $HLRT$ qui ajoute deux paramètres h et t aux paramètres de la classe LR . Ces deux paramètres permettent de délimiter une zone à l'intérieur du document dans laquelle l'extraction de tuples aura lieu. Ceci permet notamment d'avoir des occurrences de la chaîne correspondant au délimiteur l_1 entre le début de la page et h et entre t et la fin de la page, sans que des tuples non souhaités soit extraits. Le fonctionnement d'un adaptateur de la classe $HLRT$ consiste à commencer par chercher la première occurrence du délimiteur h puis itérer par l'extraction de chaque tuple de la même manière qu'un adaptateur LR , et finalement terminer par

²Soit u et s deux chaînes. s est un suffixe de u s'il existe une chaîne v telle que $u = v \cdot s$. De plus, s est un suffixe propre de u s'il s'agit de l'unique sous-chaîne de u

Procédure $induce_{LR}^l$ **Entrées**

- $E = \{\langle P_1, L_1 \rangle, \dots, \langle P_N, L_N \rangle\}$ les N pages exemples et leurs labels associés
- k l'indice de attribut pour lequel construire un délimiteur gauche
- K le nombre d'attributs de la relation à extraire

Sorties

- Un délimiteur gauche l pour l'attribut k

```

 $cand_l \leftarrow \emptyset$ 
pour  $i \leftarrow 1 \dots n$  faire
  Soit  $n_i$  le nombre de tuples dans  $L_i$ 
  pour  $j \leftarrow 1 \dots n_i$  faire
     $cand_l \leftarrow cand_l \cup prec_{P_i, L_i}(j, k)$ 
  fin pour
fin pour
Soit  $l$  la chaîne la plus courte de  $cand_l$ 
tantque  $|u| \neq 0$  et  $u$  ne satisfait pas  $\mathcal{C}_1^A$  ou  $\mathcal{C}_l^B$  faire
   $u \leftarrow substring(u, 1, |u| - 1)$ 
fin tantque
retourne  $u$ 

```

Algorithme 1.2 – Induction des délimiteurs gauches pour un adaptateur de la classe LR

une occurrence de t . D'autres classes telles que les classes $OCLR$, $HOCLRT$, etc. sont proposées. À chaque classe est associé un ensemble de contraintes que doivent vérifier les délimiteurs.

L'apprentissage d'un adaptateur revient alors à apprendre les délimiteurs paramètres de la classe. L'apprentissage d'un adaptateur $\mathcal{W}(\langle p_1, \dots, p_K \rangle)$ consiste à déterminer les candidats pour chaque p_i et évaluer pour chacun d'entre eux si les contraintes de la classe sont vérifiées. Dans le cas de la classe LR (et de manière similaire dans les autres classes) les candidats des délimiteurs gauche l_i (resp. délimiteurs droits r_i) sont choisis parmi les préfixes (resp. suffixes) des chaînes précédents (resp. suivants) la $i^{\text{ème}}$ valeur du premier tuple de la première page exemple.

1.2.1.4 Applications

Kushmerick a évalué sa méthode sur différentes sources de données. La table 1.1 récapitule les expérimentations qu'il a publiées. Dans cette table la colonne *Source* donne le nom de la source de données, la colonne *URL* donne l'URL de la source, la colonne *K* donne le nombre d'attributs de la relation à extraire et la colonne *Résultat* indique s'il a été possible de construire un adaptateur dans l'une des classes proposées par Kushmerick.

1.2.1.5 Discussion

La formalisation de Kushmerick a donné lieu à une première méthode d'induction d'adaptateurs à partir de pages exemples étiquetées. Cependant les performances des algorithmes peuvent être améliorées (voir grandement améliorées) en réduisant l'espace des candidats avec une analyse plus fine des

<i>Source</i>	<i>URL</i>	<i>K</i>	<i>Résultat</i>
Computer ESP	http://www.computeresp.com	4	succès
CNN/Time AllPolitics	http://allpolitics.com	4	échec
Film.com	http://www.film.com	6	succès
Yahoo People Search	http://www.yahoo.com/search/people/	4	succès
Cinemachine	http://www.cinemachine.com/	2	succès
PharmWeb	http://www.pharmweb.net/	13	échec
TravelData	http://www.ultranet.com/biz/inns/search-form.html	4	succès
NEWS.COM	http://www.news.com/	3	succès
Internet Travel Network	http://www.itn.net/	13	échec
Time World Wide	http://pathfinder.com/time/	4	succès
Internet Address Finder	http://www.iaf.net/	6	échec
Expedia World Guide	http://www.expedia.com/pub/genfts.dll	2	succès
thrivepathfinder	http://pathfinder.com/thrive/index.html	4	succès
Monster Job Newsgroups	http://www.monster.com/	3	succès
NewJour	http://gort.ucsd.edu/newjour/	2	succès
Zipper	http://www.voxpop.org/zipper/	11	échec
Coolware Classifieds Electronic Job Guide	http://www.jobsjobsjobs.com	2	succès
Ultimate Band List	http://ubl.com	2	succès
Shops.Net	http://shops.net/	5	succès
Democratic Party Online	http://www.democrats.org/	6	succès
Complete Works of William Shakespeare	http://the-tech.mit.edu/Shakespeare/works.html	5	échec
Bible	http://etext.virginia.edu/rsv.browse.html	3	succès
Virtual Garden	http://pathfinder.com/vg/	3	succès
Foreign Languages for Travelers	http://www.travlang.com/	4	échec
U.S. Tax Code On-Line	http://www.fourmilab.ch/ustax/ustax.html	2	succès
CD Club Web Server	http://www.cd-clubs.com/	5	échec
Expedia Currency Converter	http://www.expedia.com/pub/curcnvt.dll	6	succès
Cyberider Cycling WWW-Site	http://blueridge.infomkt.ibm.com/bikes/	3	succès
Security APL Quote Server	http://qs.secapl.com/	18	échec
Congressional Quarterly	http://voter96.cqalert.com/cq_job.htm	8	succès

Table 1.1 – Résultats publiés pour le système WIEN

contraintes. En effet, on peut constater, par exemple, que pour un délimiteur gauche donné l_i l'ensemble des délimiteurs candidats valides est l'ensemble des suffixes communs aux $prec_{P,L}(i, k)$ pour tout k , tout P et tout L . De plus, les classes d'adaptateurs proposées ne permettent pas de décrire des adaptateurs pour toutes les sources ayant un contenu structuré. La méthode de Kushmerick, selon sa propre étude, ne permet de construire un adaptateur que pour 70% des sources ayant un contenu structuré. Ce manque d'expressivité est dû à l'utilisation de délimiteurs, à la contrainte sur un ordre identique dans les données, et, au fait que le contenu des documents des sources est souvent organisé de manière hiérarchique, ce qui n'est pas pris en compte.

1.2.2 STALKER

Une des premières méthodes proposant de répondre au manque d'expressivité du système WIEN de Kushmerick fut le système STALKER proposé par Muslea et al. [88]. Les principaux apports de cette méthode sont l'introduction du formalisme des arbres $\mathcal{EC}$, l'utilisation de l'abstraction en séquence de symboles des documents, la représentation d'une règle d'extraction par un automate, la proposition d'une méthode d'induction par raffinements successifs, et l'élargissement de la notion de délimiteur. Le formalisme d'arbre- $\mathcal{EC}$ permet de décrire la structure logique du document. L'association à chaque noeud de règles d'extractions permet la prise en compte de contenus hiérarchiques. Un contenu est hiérarchique lorsqu'il peut se décomposer en parties qui elles-mêmes peuvent se décomposer en sous-parties. Par exemple, la figure 1.4, décrit la décomposition hiérarchique des contenus d'un ensemble de pages d'une même source. Dans cette source, chaque page contient une liste de descriptions de restaurants, chaque description de restaurant est un tuple formé d'un nom, d'une adresse, d'un numéro de téléphone, d'un commentaire, et d'une liste de cartes bancaires acceptées.

1.2.2.1 Représentation d'un document

Muslea est parmi les premiers, dans le cadre de l'extraction d'informations à partir du Web, à proposer d'abstraire les documents à traiter en effectuant une analyse lexicale des documents. Cette abstraction introduit des symboles de trois types : les mots (chaîne de caractères alphanumériques), les symboles (caractères non alphanumérique) et les balises HTML. Ainsi l'alphabet des symboles est alors $\Sigma_d = words(P) \cup symbols(P) \cup html(P)$ où $words(P)$ renvoie les mots contenus dans P , $symbols(P)$ renvoie les symboles (e.g. " ", ";", ":", ".") contenus dans P , $html(P)$ renvoie les balises HTML (e.g. '', '<td colspan="2">', '</i>') contenues dans P . Une phase d'analyse lexicale est alors nécessaire pour pouvoir traiter le document.

Afin de simplifier le problème d'extraction d'informations dans des documents dont le contenu n'est pas tabulaire mais hiérarchique, Muslea introduit le formalisme d'arbre- $\mathcal{EC}$ (embedded catalog). Un tel arbre décrit la décomposition logique d'un document. Par exemple une page de restaurant pourrait prendre la décomposition donnée par la figure 1.4.

Définition 1.6 (Arbre- $\mathcal{EC}$). *Un arbre- $\mathcal{EC}$ d , est un arbre où chaque noeud est*

- *une feuille (élément primitif) e*
- *un noeud liste ayant un unique fils d'*
- *un noeud tuple ayant k fils $d_1, \dots, d_k$*

Par exemple, dans la figure 1.4 les noeuds “Nom”, “Adresse”, “Téléphone”, “Commentaire” et “CarteBancaire” sont des feuilles. Le noeud “TUPLE(Restaurant)” est un noeud tuple à 5 fils, et les noeuds “LISTE(Restaurants)” et “LISTE(CarteBancaires)” sont des noeuds liste.

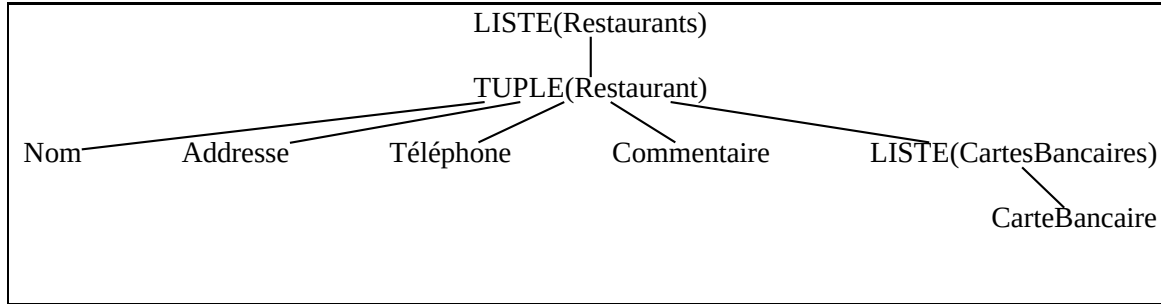


Figure 1.4 – Exemple de décomposition d'une page

Dans un document, à chaque noeud n de l'arbre- $\mathcal{EC}$ correspond une ou plusieurs sous-chaînes appelées occurrences. Lorsque le noeud n est une feuille, les occurrences de n correspondent à une valeur à extraire. Dans le cas où n est un noeud tuple chacune de ses occurrences doivent contenir (comme sous-chaînes) une occurrence de chaque noeud fils. Finalement, si le noeud n est un noeud liste alors il devra contenir zéro ou plusieurs occurrences de son noeud fils, formant une liste.

1.2.2.2 Représentation d'un adaptateur

Dans ce cadre un adaptateur est une procédure qui permet d'extraire les occurrences de chacun des noeuds de l'arbre- $\mathcal{EC}$ associé au document. L'algorithme d'extraction cherche à reconnaître les occurrences des noeuds de l'arbre- $\mathcal{EC}$ en parcourant l'arbre de haut en bas. Ainsi si N est un noeud de parent M , les occurrences de N seront cherchées dans les chaînes des occurrences du noeud M .

Étant donnée, une occurrence m d'un noeud M , chercher une occurrence n d'un noeud N dont le parent est M revient à chercher dans la chaîne m les positions de début et de fin pour n . Ces positions peuvent être exprimées à l'aide de règles d'extraction. Une règle d'extraction est composée d'une disjonction de règles simples. Une règle simple est de la forme $R = \text{SkipTo}(w_1) \dots \text{SkipTo}(w_k)$ où chaque w_i est un délimiteur au sens de Kushmerick. $\text{SkipTo}(x)$ indique qu'il faut rechercher la première occurrence de x dans le document à partir de la position courante. Reconnaître une telle règle revient à chercher la première occurrence de w_1 , puis à partir de celle-ci, chercher la première occurrence de w_2 , etc. jusqu'à w_k . La position de début ou de fin de l'occurrence n , reconnue par une telle règle est celle se trouvant juste après w_k . Ainsi une règle n'est plus constituée que d'un seul délimiteur à la Kushmerick mais peut être composée d'une succession de délimiteurs.

De plus, l'alphabet des symboles sur lequel sont construits les délimiteurs est Σ_d étendu avec des symboles jokers pour chaque classe. L'introduction de jokers permet de construire des règles dans lesquelles certains symboles sont abstraits. Un joker est un nouveau symbole qui reconnaît n'importe quel symbole de la classe du joker. Ainsi nous avons l'alphabet des délimiteurs $\Sigma_p = \Sigma_d \cup \{\star_s, \star_h, \star_w\}$ avec $\star_s$ comme joker des caractères symboliques (i.e. ":", ";", etc.), $\star_h$ comme joker des balises HTML, et $\star_w$ comme joker des mots.

Dans leur travaux, Muslea et al. ont choisi de représenter une règle d'extraction à l'aide d'un automate à délimiteurs. Dans un tel automate les transitions ne sont plus composées d'un unique symbole mais d'une chaîne de symboles. Étant donné une chaîne et une position courante dans cette chaîne, le passage d'une transition peut se faire lorsque, à partir de la position courante, une occurrence de la chaîne associée à la transition peut être trouvée. Une fois la transition passée la position courante est celle se trouvant juste après l'occurrence de la chaîne de transition. Toutefois les automates construits par le système STALKER ne prennent que la forme générale décrite par la figure 1.5. En somme, il s'agit de

représenter une disjonction de règles simples. Chaque règle simple correspond à une succession d'états ayant chacun une unique transition sortante, sauf pour le premier qui est un état partagé par chacune des règles simples.

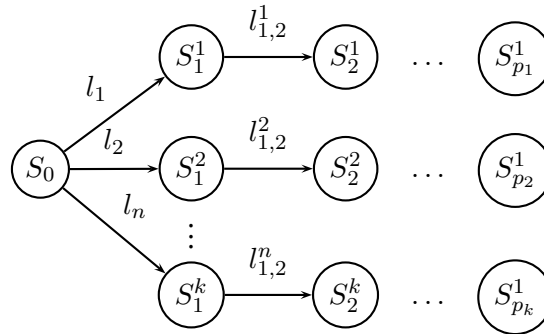


Figure 1.5 – Forme générale d'un automate à délimiteur simple

À chaque noeud est associé deux règles : l'une permettant de référencer la position de début d'une valeur (règle gauche) et l'autre permettant de référencer la position de fin (règle droite). La reconnaissance d'un élément d'une décomposition se fait à l'intérieur de son parent. Par exemple, dans la décomposition de la figure 1.4, la reconnaissance de l'élément "Nom" se fait à l'intérieur du texte correspondant à "TUPLE(Restaurant)". L'exécution de la règle gauche s'effectuant de gauche à droite et celle de la règle droite de droite à gauche. La figure 1.7 donne une vue schématique de l'apparition d'une occurrence d'un noeud "Nom" à l'intérieur d'une occurrence d'un noeud "TUPLE(Restaurant)".

1.2.2.3 Apprentissage par raffinement

Dans le cas du système STALKER, l'apprentissage d'un adaptateur pour une source, étant donné son arbre- $\mathcal{EC}$, peut se faire en apprenant séparément une règle gauche et une règle droite pour chacun des noeuds de l'arbre. Seulement, pour ce faire, l'utilisateur doit avoir au préalable étiqueté le début et la fin des occurrences de chacun des noeuds. Ensuite pour apprendre une règle (gauche ou droite) pour un noeud de l'arbre donné la méthode proposée par Muslea consiste à procéder par raffinements successifs.

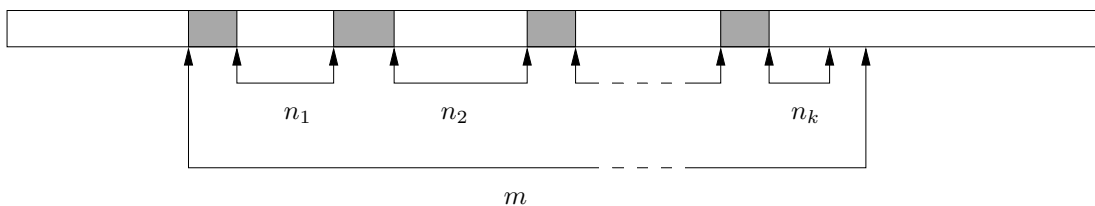


Figure 1.6 – Exemples pour l'apprentissage d'une règle gauche pour les occurrence n_i d'un noeud N

Lorsque l'on cherche à construire une règle permettant d'extraire les valeurs d'une occurrence m_i d'un noeud M de type tuple (i.e. $M = TUPLE(N_1, \dots, N_k)$), on procède par l'extraction séparée de chacune des occurrences des N_j . À chacun des N_j est associée une règle R_{N_j} . L'objectif de R_{N_j} est de permettre d'extraire pour toute occurrence m_i de M l'occurrence de N_j qu'elle contient. Par exemple,

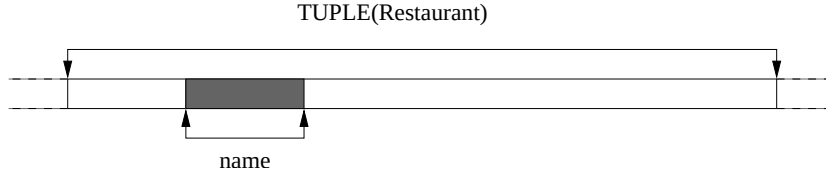


Figure 1.7 – Exemple d’occurrence d’un noeud “Nom” à l’intérieur d’une occurrence d’un noeud “TUPLE(Restaurant)”

la figure 1.7 montre une occurrence d’un noeud “Nom” dans un noeud “TUPLE(Restaurant)”. Dans ce cas, l’objectif est de construire une règle permettant d’extraire l’occurrence du noeud “Nom” qu’elle contient. L’extraction d’une occurrence se fait en deux temps : par la détection de son début et puis par la détection de sa fin. Ainsi, il y a deux règles à construire. Les deux se construisant de manière analogue, nous nous concentrerons sur la construction d’une règle de détection du début des n_{ji} . Pour construire une telle règle il suffit de détecter à partir du début de m_i la fin de la sous-chaîne comprise entre le début de m_i et le début de n_{ji} .

Dans le cas où M est de type liste de N (i.e. $M = LISTE(N)$) une occurrence m_i de M contient plusieurs occurrences de N . La chaîne m_i , occurrence de M contient k_i occurrences de N . Elle peut donc s’écrire $m_i = w_0 \cdot n_{i1} \cdot w_{i1} \cdot \dots \cdot w_{ik_i-1} \cdot n_{ik_i} \cdot w_{ik_i}$, où les n_{ij} sont les occurrences de N , w_0 est la chaîne comprise entre le début de m_i et celui de n_{i1} , les w_{ij} ($0 < j < k_i$) sont les chaînes comprises entre deux n_{ij} , et w_{ik_i} est la chaîne comprise entre la fin de n_{ik_i} et le fin de m_i .

Dans tous les cas on cherche à caractériser le début de l’ensemble des occurrences d’un noeud N à partir des chaînes qui précèdent directement les occurrences du noeud. Ces chaînes sont les exemples permettant l’induction d’une règle gauche pour N . Dans la figure 1.6, les chaînes en gris correspondent aux exemples utilisés pour déterminer un délimiteur gauche pour le noeud N . La construction des exemples pour l’apprentissage d’une règle droite, permettant de détecter la fin des occurrences, se fait de manière symétrique en partant de la fin de l’occurrence parente.

Avant la formation d’une règle, on construit l’ensemble des symboles terminaux T qui correspond à l’ensemble des symboles apparaissant dans les exemples. Cet ensemble est ensuite augmenté avec l’ensemble des généralisations de chaque symbole de T . La généralisation d’un symbole étant l’ensemble des symboles jokers correspondant à sa classe. Ainsi, $T' = T \cup \{t'/t' \in g(t), t \in T\}$ où $g(t)$ est une fonction qui renvoie les généralisations de t , T' est l’ensemble T auquel on a ajouté tous les jokers qui ont au moins un symbole de la même classe dans T .

L’algorithme 1.3 décrit la méthode permettant de construire une règle disjonctive dans STALKER. La fonction $Terminaux(E)$ renvoie l’ensemble des symboles contenues dans E . Cet ensemble est alors complété en appliquant la fonction $Generalisations(T)$ qui renvoie les généralisations des symboles se trouvant dans T . La fonction $ApprendreDelimiteur(E, T')$, que nous décrivons plus loin, construit le délimiteur permettant d’extraire le maximum d’exemples. Le délimiteur obtenu est ajouté à la disjonction de délimiteurs R , et l’ensemble des exemples est réduit pour ne contenir que les exemples non encore couverts en appliquant la fonction $NonCouverts$. L’algorithme s’arrête lorsque tous les exemples sont couverts.

La construction d’un délimiteur effectuée par la fonction $ApprendreDelimiteur(E, T)$ à partir d’un ensemble d’exemples et de terminaux est décrit dans l’algorithme 1.4. Dans ce dernier, la fonction $GenereCandidats(E)$ génère un ensemble de délimiteurs candidats en prenant comme délimiteur le dernier symbole de chaque exemple. Ensuite, la fonction $MeilleurCandidat(C, E)$ sélectionne parmi les délimiteurs candidats C celui qui couvre le plus d’exemples de E . Si ce candidat D est parfait (i.e. ne

Procédure Stalker**Entrées**

- un ensemble E de chaînes précédant les occurrences des exemples d'un nœud N de l'arbre $\mathcal{EC}$

Sorties

- une disjonction de délimiteurs R formant la règle permettant de détecter le début des occurrences de N .

 $R \leftarrow \emptyset$

{La position à déterminer est la fin des chaînes des exemples}

tantque $E \neq \emptyset$ **faire** $T \leftarrow \text{Terminaux}(E)$ $T' \leftarrow T \cup \text{Generalisations}(T)$ $D \leftarrow \text{ApprendreDelimiteur}(E, T')$ $R \leftarrow R \cup \{D\}$ $E \leftarrow \text{NonCouverts}(E, D)$ **fin tantque**

Algorithme 1.3 – Induction d'une disjonction dans STALKER

génère aucun exemple négatif) il est alors retourné tel quel. Sinon D est remplacé par ses raffinements dans l'ensemble des délimiteurs candidats.

STALKER introduit deux types de raffinements de motifs étant donné un ensemble de symboles terminaux T : le raffinement de délimiteur et le raffinement topologique. Le raffinement de délimiteur consiste à ajouter un symbole de T au délimiteur en tête d'un motif. Le raffinement topologique consiste à ajouter en tête de motif un nouveau délimiteur composé d'un unique symbole de T . La figure 1.8 donne l'ensemble des raffinements du motif $\text{SkipTo}(\cdot)$ étant donné l'ensemble des terminaux $T = \{;, \text{reservation}, \star_s, \star_w\}$.

Raffinements de délimiteur	Raffinements topologique
$(\text{SkipTo}(\cdot))$	$(\text{SkipTo}(\cdot)\text{SkipTo}(\cdot))$
$(\text{SkipTo}(\text{reservation}))$	$(\text{SkipTo}(\text{reservation})\text{SkipTo}(\cdot))$
$(\text{SkipTo}(\star_s))$	$(\text{SkipTo}(\star_s)\text{SkipTo}(\cdot))$
$(\text{SkipTo}(\star_w))$	$(\text{SkipTo}(\star_w)\text{SkipTo}(\cdot))$

Figure 1.8 – Raffinements de la règle $(\text{SkipTo}(\cdot))$

Dans le cas de STALKER, la construction d'un adaptateur nécessite de fournir l'arbre- $\mathcal{EC}$ de la source, ainsi que les exemples pour chaque nœud de l'arbre- $\mathcal{EC}$. Dans ses travaux, Muslea ne précise pas comment obtenir ces exemples, mais comme pour la méthode de Kushmerick, un étiquetage peut être effectué. Toutefois, cette méthode nécessite un étiquetage de chaque instance d'un nœud de l'arbre- $\mathcal{EC}$. Ceci implique que, pour constituer un ensemble de documents exemples, l'utilisateur doit non seulement étiqueter les valeurs élémentaires qu'il souhaite extraire mais aussi chaque tuple ou liste des valeurs.

Procédure ApprendreDelimiteur**Entrées**

- E un ensemble d'exemples
- T un ensemble de terminaux

Sorties

- un délimiteur D

```

 $C \leftarrow \text{GenereCandidats}(E)$ 
tantque  $C \neq \emptyset$  faire
   $D \leftarrow \text{MeilleurCandidat}(C, E)$ 
  si  $\text{estParfait}(D, E)$  alors
    retourne  $D$ 
  sinon
     $C = (C \setminus D) \cup \bigcup_{t \in T} \text{Raffiner}(D, t)$ 
  fin si
fin tantque

```

Algorithme 1.4 – Construction d'un délimiteur dans STALKER

1.2.2.4 Applications

Le système STALKER a été évalué sur cinq sources de données différentes. La table 1.2 donne ces résultats. Dans cette table les colonnes donnent respectivement le nom de la source (*Source*), l'URL de la source (*URL*), le nombre d'attributs de la relation à extraire (K), le nombre de pages étiquetées données en exemple (*Ex.*) et le temps d'exécution nécessaire pour la construction de l'adaptateur (*Tmps.*).

<i>Source</i>	<i>URL</i>	K	<i>Ex.</i>	<i>Tmps.</i>
Okra	okra.ucr.edu	4	2 pages	30s
BigBook	www.bigbook.com	6	4 pages	2m 25s
Address Finder	www.iaf.net	6	5 pages	9m 02s
Quote Server	qs.secalp.com	18	4 pages	39m 10s
LA Weekly	www.laweekly.com	5	4 pages	1m 08s

Table 1.2 – Résultats publiés pour le système STALKER

1.2.2.5 Conclusion

La méthode STALKER est une avancée intéressante dans l'induction d'adaptateurs. En effet, un premier pas vers le traitement de documents ayant une structure hiérarchique est proposé. Cependant le format des données extraites reste celui du document et non le format que souhaiterait l'utilisateur. Ceci impose un post-traitement consistant à transformer les données extraites. De plus la phase d'étiquetage est alourdie : en plus d'avoir à étiqueter complètement les pages, l'utilisateur doit fournir les positions de début et de fin de chaque occurrence d'un noeud interne ou non de l'arbre- $\mathcal{EC}$. Par exemple, en reprenant la décomposition de l'exemple de la figure 1.4, pour chaque page exemple contenant une liste de restaurants, il faudra donner la position de début de chaque partie contenant une description de restaurants, puis

dans chaque partie donner les débuts et fins des sous-parties “Nom”, “Adresse”, “Téléphone”, “Commentaire”, et “LISTE(CartesBancaires)”. Finalement, dans chaque sous-partie “LISTE(CartesBancaires)”, il faudra donner le début et fin de chaque occurrence d'une partie “CarteBancaire”. Cette tâche en plus d'être fastidieuse n'est pas déterministe, car plusieurs possibilités de début et de fin d'un noeud interne existent. En effet, il est uniquement nécessaire que les occurrences de noeuds internes soit des chaînes contenant les occurrences de leur fils. On pourrait, par exemple, considérer que l'occurrence d'un noeud interne commence à la même position que la première occurrence d'un de ses fils et qu'elle termine à la même position que la dernière occurrence d'un de ses fils. Cependant, toute position entre le début (resp. la fin) de l'occurrence parent et le début (resp. la fin) de la première (resp. dernière) occurrence aurait tout aussi bien convenu.

1.2.3 SOFTMEALY

Le système d'induction d'adaptateur SOFTMEALY proposé dans [57] par Hsu et Dung est apparu en 1998. Les auteurs cherchent principalement à proposer une solution aux problèmes de l'ordre des attributs et des attributs manquants. Les règles d'extractions qu'ils cherchent à construire doivent tenir compte des différentes permutations des attributs qui apparaissent dans les occurrences de la relation à extraire. De plus, ils apportent une approche qui n'est plus basée sur des délimiteurs mais sur des séparateurs. Un séparateur permet de caractériser une position à la fois à partir du texte se trouvant juste avant cette position et du texte se trouvant juste après. Un séparateur prend alors en compte à la fois ce qui se trouve à gauche et à droite de la position qu'il détermine. Cette position correspond soit au début ou à la fin d'une valeur. Ainsi, même le format du contenu de cette valeur est pris en compte par le séparateur. Par exemple, un séparateur permettant de déterminer la position de début d'une valeur prendra en compte à la fois ce qui est avant et après cette position. La valeur déterminé étant elle-même juste après la position, son contenu est pris en compte par le séparateur.

1.2.3.1 Représentation d'un adaptateur

Dans une chaîne de symboles, on peut considérer qu'entre chaque paire de symboles successifs il existe un point de séparation formant une limite invisible entre deux caractères. Ainsi, toute sous-chaîne de caractères se trouve entre deux telles limites. Dans le cas de SOFTMEALY on cherche à caractériser cette limite invisible ou point de séparation. Ceci peut être fait à l'aide d'un séparateur composé de deux sous-chaînes de symboles se trouvant, respectivement avant et après la limite.

D'autre part, dans leur système, Hsu et Dung, introduisent une classification hiérarchique des symboles. Les classes de symboles sont données par la table 1.3. Celle-ci permet d'introduire des symboles jokers à plusieurs niveaux d'abstraction. Par exemple, le symbole *PROFESSOR* est à la fois un symbole des classes *CAlph*, *C1Alph* et *OAlph*. Ainsi ce symbole sera reconnu par les symboles jokers *CAlph*($_$), *C1Alph*($_$) et *OAlph*($_$).

Dans cette méthode un adaptateur est représenté par un transducteur. Un transducteur est une machine abstraite définie par un ensemble d'états, un alphabet en entrée, un alphabet en sortie, et une fonction de transition. Un transducteur permet de transformer une chaîne de symboles en entrée en une nouvelle chaîne de symboles.

Dans le cas de SOFTMEALY, un transducteur permettant l'extraction d'une relation à n attributs aura $2n + 1$ états : 1 état par attribut, 1 état entre chaque paire d'attributs successifs, un état de début et un état de fin. A chaque attribut A est associé un attribut factice $\overline{A}$, correspondant aux sous-chaînes se trouvant entre l'attribut A et l'attribut suivant. L'attribut factice $\overline{A}$ correspond à la sous-chaîne entre une valeur

$CAlph(x)$	Chaîne de caractère x est toute en majuscule
$C1Alph(x)$	Chaîne de caractère x commence par une majuscule
$Num(x)$	Chaîne de caractère x est une suite de chiffre
$Htm1(x)$	Chaîne de caractère x est une balise HTML
$OAlph(x)$	Chaîne de caractère x composé de caractères alphanumériques
$Punc(x)$	Le symbole x est un caractère de ponctuation
$NL(n)$	Une suite de n retour à la ligne
$Tab(n)$	Une suite de n tabulations
$Spc(n)$	Une suite de n espaces

Table 1.3 – Classification des symboles utilisé dans SOFTMEALY

de l'attribut A et l'attribut suivant A . Formellement, un transducteur du système SOFTMEALY est un 5-uplet $(\Sigma_1, \Sigma_2, Q, R, E)$ où :

- Σ_1 est l'ensemble des séparateurs constituant l'alphabet en entrée ;
- Σ_2 est l'ensemble des caractères de l'alphabet de sortie ;
- Q est un ensemble fini d'états contenant :
 - un état initial b et un état final e représentant les attributs fictifs de début et de fin d'un tuple ;
 - un état k associé à chaque attribut k de la relation à extraire ;
 - un état $\bar{k}$ pour chaque attribut k représentant un *attribut fictif* : l'ensemble des symboles qui doivent être ignorés entre une valeur de l'attribut k et la valeur de l'attribut suivant.
- R est l'ensemble des règles contextuelles. Chaque règle contextuelle notée $s\langle i, j \rangle$ décrit une classe de séparateurs entre deux attributs (fictifs ou non) ;
- $E \subset Q \times R \times \Sigma_2^* \times Q$ est l'ensemble des transitions.

L'utilisation d'un transducteur permet de sauter certains attributs et ainsi permettre de prendre en compte d'éventuelles valeurs manquantes. En effet, certaines instances d'une relation peuvent avoir des attributs non définis. Ceci se traduit dans les documents par l'absence d'une valeur dans l'occurrence.

La figure 1.9 donne un exemple de transducteur qu'il est possible d'obtenir avec SOFTMEALY pour la source d'adresses emails Address Finder. La relation que contient Address Finder est la relation $address(nom, email, organisation, societe)$. Cependant dans certaines instances les attributs *organisation* et *societe* peuvent être manquant. Dans ce transducteur, les états N, E, O, S , correspondent respectivement aux attributs *nom, email, organisation, et societe*, les états $\bar{N}, \bar{E}, \bar{O}$ associés aux attributs fictifs, les états b et e à l'état initial et l'état final. Deux fonctions $suiv()$ et $sep(x, y)$ apparaissent dans les transitions. La fonction $suiv()$ retourne la valeur du dernier symbole lu et $sep(x, y)$ renvoie le séparateur associé au couple d'attributs (x, y) .

Transiter d'un état i à un état j est possible s'il existe une transition $\langle i, r, o, j \rangle \in E$ telle que le séparateur courant satisfait r .

1.2.3.2 Algorithme d'induction

Dans ce cadre, la construction d'un transducteur revient à apprendre les règles contextuelles permettant de transiter d'un état à l'état suivant, autrement dit, transiter d'un attribut à un autre. Pour une relation à n attributs le transducteur est initialisé avec $2n + 1$ états correspondant à n attributs k , $n - 1$ attributs fictifs $\bar{k}$, un attribut de début b et un attribut de fin e . Pour chaque paire d'états (ou d'attributs, fictifs ou non) un ensemble de séparateurs initiaux est construit à partir des exemples. Les exemples correspondent à des occurrences des instances de la relation à extraire. Chaque occurrence est représentée par les posi-

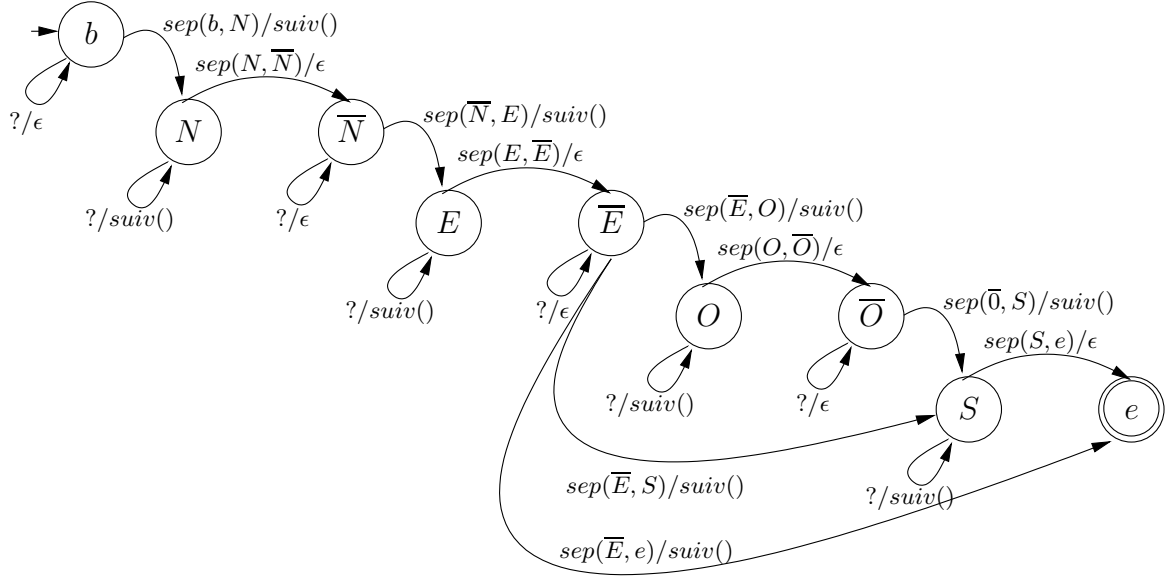


Figure 1.9 – Exemple de transducteur dans SOFTMEALY

tions de débuts et de fin des valeurs de chacun des attributs. Étant donné que les occurrences des valeurs peuvent apparaître dans des ordres différents pour chaque instance, il est nécessaire d'associer un attribut à chaque valeur. Ainsi on peut représenter une occurrence par un ensemble de triplets $\langle a, b, e \rangle$ où a , b et e correspondent respectivement au nom de l'attribut auquel appartient l'occurrence, à sa position de début et sa position de fin dans le document exemple. L'occurrence d'une instance est alors un ensemble d'occurrences de valeurs $\langle a_i, b_i, v_i \rangle$.

Un ordre entre les occurrences des valeurs d'une même instance peut être établi en comparant leur positions de début (i.e. $\langle a, b, e \rangle < \langle a', b', e' \rangle$ ssi $b < b'$). Dans la suite nous choisirons les indices de telle sorte qu'ils reflètent cet ordre. Ainsi $\langle a_i, b_i, e_i \rangle < \langle a_j, b_j, e_j \rangle$ ssi $i < j$. A partir d'une occurrence $\langle a_i, b_i, e_i \rangle$ un séparateur est construit pour chaque b_i et pour chaque e_i et ajouté comme exemple associé à une transition du transducteur. Pour la première occurrence, le séparateur issue de b_1 est ajouté comme exemple pour la transition $b \rightarrow a_1$. Les séparateurs des occurrences des autres b_i sont respectivement ajoutés comme exemples pour les transitions $\overline{a_{i-1}} \rightarrow a_i$. De manière analogue, la valeur de fin e_n de la dernière valeur de l'occurrence donne lieu à un séparateur qui est ajouté comme exemple pour la transition $a_n \rightarrow e$. Les séparateurs issus des autres valeurs e_i sont des exemples respectivement pour les transitions $a_i \rightarrow \overline{a_i}$.

Ainsi, un ensemble de séparateur est associé à chaque transition d'un état deux états. L'apprentissage d'une règle contextuelle permettant la transition entre un état i et un état j se fait en généralisant cet ensemble d'exemples. Une règle contextuelle est un ensemble de séparateurs. Cet ensemble est obtenu en généralisant les séparateurs exemples initiaux.

Chaque séparateur est constitué de deux parties : l'une représente la sous-chaîne précédant la limite invisible que représente le séparateur et l'autre représente la sous-chaîne suivant cette limite. Les généralisations sont obtenues en généralisant séparément les parties gauches et les parties droites. Les chaînes sont d'abord alignées à droite pour les parties gauches et à gauche pour les parties droites. En positionnant l'ensemble des chaînes alignées les une au dessus des autres nous avons une matrice $N \times M$ de symboles où N est le nombre de séparateurs et M la longueur de la partie la plus longue. Chaque ligne

correspond à la partie gauche ou droite d'un séparateur. L'ordre des lignes n'ayant pas d'importance nous parlerons d'un ensemble aligné de chaînes. Une fois les chaînes alignées, les colonnes sont généralisées une par une en remontant la hiérarchie des symboles. L'algorithme 1.5 décrit comment sont généralisés les parties (gauches ou droites) d'une règle contextuelle à partir de l'ensemble aligné des parties des exemples. Les colonnes sont parcourues une à une et lorsqu'un symbole d'une ligne peut être généralisé avec un symbole d'une autre ligne il est remplacé par cette généralisation. A la fin l'ensemble des séparateurs contient les généralisations des séparateurs exemples. Certains séparateurs pouvant apparaître plusieurs fois, l'ensemble des généralisations est réduit en supprimant les doublons.

Procédure SoftMealy

Entrées

- D un ensemble de parties (gauches ou droites) alignées des séparateurs exemples

Sorties

- l'ensemble D des parties généralisées

Soit M la longueur de la plus longue chaîne de D

pour i de 1 à M **faire**

pour j de 1 à $|D|$ **faire**

si $\exists j', 1 \leq j' \leq |D|, D[j', i]$ et $D[j, i]$ sont dans le même sous-arbre de ma hiérarchie de symboles **alors**

 remplacer $D[j, i]$ par le plus petite ancêtre commun de $D[j, i]$ et $D[j', i]$.

fin si

fin pour

fin pour

Supprimer les parties en double dans D

Algorithme 1.5 – Algorithme de généralisation de SOFTMEALY

1.2.3.3 Applications

Les auteurs ont appliqué leur méthode sur des pages Web de départements informatiques afin d'en extraire la liste des membres et les informations associées. Les résultats sont repris dans la table 1.4.

1.2.3.4 Conclusion

L'un des apports intéressants de la méthode SOFTMEALY est l'introduction de la notion de séparateur. Ceux-ci permettent de prendre en compte, dans les motifs d'extractions, le type des valeurs à extraire. En effet, les méthodes à base de délimiteurs ne prennent en compte que le contexte précédant (ou suivant) les valeurs à extraire alors que le format des données à extraire est une information exploitable et qui enrichi l'expressivité des motifs. Un autre aspect intéressant est la construction des règles de transitions en utilisant la généralisation des séparateurs. Néanmoins, cette méthode nécessite toujours un étiquetage des données. Il est nécessaire d'avoir dans les exemples au minimum une occurrence de chaque ordre possible des attributs.

URL	Tuples	Exemples
www.cs.nmt.edu/Faculty.html	6	6
www.cs.olemiss.edu/faculty/	8	5
www.cs.fit.edu/people/faculty.html	11	5
www.cs.brandeis.edu/faculty-description/	11	6
www.cs.caltech.edu/csstuff/faculty.html	15	5
www.cs.colostate.edu/people.html	18	4
www.cd.dartmouth.edu/faculty/	18	6
www.cs.msstate.edu/FACULTY_AND_STAFF/	18	9
www.cse.fau.edu/faculty.html	21	5
www.cs.gmu.edu/faculty/	21	2
www.cs.colombia.edu/home/people/people.html	22	3
cyclone.cs.clemson.edu/html/whoswho/facultyindex.html	22	5
www.cs.gmu.edu/ofchours.html	23	9
www.cs.jhu.edu/faculty.html	24	4
www.cs.wm.edu/cspages/people/faculty.html	25	9
www.csmsu.edu/fac/index.html	29	4
www.cs.iastate.edu/faculty/index.html	29	4
www.cs.nps.navy.mil/people/faculty/	29	5
www.cs.concordia.ca/People/Faculty.html	31	7
www.cs.duke.edu/cgi-bin/factable?text	32	5
www.cs.washington.edu/people/faculty/	34	2
www.eas.asu.edu/csedep/people/faculty.html	35	5
gauss.nmsu.edu:8000/faculty/faculty.html	35	5
www.cs.nyu.edu/cs/new_faculty.html?	36	6
simon.cs.cornell.edu/Info/Faculty/faculty-list.html	39	5
www.seas.gwu.edu/seas/eecs/faculty.html	41	6
cbis.ece.drexel.edu/ECE/ece_bios.html	46	25
class.ee.iastate.edu/fac_staff.index.html	48	6
www.ri.cmu.edu/ri-home/people.html	51	5
www-eecs.mit.edu/faculty/index.html	157	6

Table 1.4 – Résultats de SOFTMEALY publiés par les auteurs

1.2.4 Synthèse

Dans cette section nous avons présentés trois systèmes d'induction d'adaptateurs utilisant des pages d'exemples étiquetés. Cette approche de la construction d'adaptateurs permet de faciliter la conception d'adaptateurs pour le Web. Les sources que ces méthodes permettent de traiter sont diverses : le système WIEN permet de traiter des contenus tabulaires, la méthode utilisée dans STALKER permet de traiter des données hiérarchiques, et finalement SOFTMEALY s'attaque au problème des valeurs manquantes et de l'ordre des attributs. Des trois méthodes STALKER est la plus adaptée car elle permet implicitement le traitement des deux autres formats de données. Toutefois la méthode SOFTMEALY qui fonctionne par généralisation nécessite moins d'exemples et permet d'obtenir un adaptateur de manière plus efficace car l'ensemble des candidats considérés permet toujours de reconnaître au moins une occurrence.

Le problème majeur de ces méthodes inductives est la tâche d'étiquetage des pages. En effet, toutes ces méthodes construisent l'adaptateur à partir d'un ensemble de pages étiquetées, c'est à dire que dans chaque page exemple toute occurrence d'une instance doit être identifiée par l'utilisateur. De plus, dans le cas de données hiérarchiques, chaque occurrence d'un noeud (indépendamment de son niveau dans la hiérarchie) doit être étiquetée. Cette tâche d'étiquetage peut alors s'avérer fastidieuse, et sujette à erreurs.

De plus, dans le cas de données hiérarchiques plusieurs étiquetages sont possibles. Ces étiquetages sont fastidieux car chaque occurrence de chaque valeur doit être sélectionnée et labelisée par l'utilisateur. L'utilisateur peut facilement oublier d'étiqueter une valeur, ce qui peut résulter en un adaptateur erroné. Finalement, la tâche est non-déterministe dans le cas de données hiérarchiques car plusieurs étiquetages d'un nœud sont possibles. C'est ainsi que sont apparues des méthodes d'apprentissage de motifs non-supervisé basées sur l'analyse de la structure des documents d'une source.

1.3 Extraction de motifs via l'analyse de la structure des documents

Une autre approche de l'extraction d'informations à partir du Web consiste à analyser la structure HTML des documents. Cette analyse permet la génération d'expressions décrivant le format général dans lequel se trouvent les données à extraire. De tels expressions sont appelés des motifs. L'extraction de motifs via l'analyse de la structure HTML des documents est apparue avec le constat que les pages Web ou documents semi-structurés sont présentés de manière régulière. Ceci se traduit dans la structure HTML par des séquences de balises suivant le même format. L'objectif dans ce cas est d'exploiter cette régularité pour la construction de motifs. A la différence de la construction d'adaptateurs à partir d'exemples, dans cette approche, la structure HTML des documents est prise en compte. La première méthode, utilisée par le système IEPAD, consiste à analyser la chaîne formée par un unique document et d'en extraire les régularités. La seconde méthode, mise en oeuvre par ROADRUNNER cherche les correspondances entre deux documents d'une même source pour construire les motifs. Le système IEPAD a été proposé dans [15] par Chang et Lui et ROADRUNNER dans [22] par Crescenzi et al. A la différence des méthodes inductives, ces deux méthodes ne nécessitent pas d'étiquetage des pages exemples. Le système IEPAD nécessite une page exemple pour construire des motifs et le système ROADRUNNER en nécessite au minimum deux.

1.3.1 IEPAD : Chang

La méthode proposée par Chang et Lui, consiste à construire des motifs en analysant la structure d'un document HTML. La découverte des motifs s'effectue en plusieurs étapes. La première étape consiste à *encoder* le document analysé en une séquence de symboles en faisant abstraction de certaines parties du document notamment les parties textuelles. Une fois cette séquence obtenue un *arbre PAT*³ est construit afin de procéder à une *recherche des répétitions maximales* dans la séquence qui forment alors un ensemble de motifs candidats. Ensuite une *sélection des motifs* est effectuée selon divers critères comme la taille du motif, sa régularité, etc. Un *partitionnement* des occurrences est ensuite effectué pour les motifs ayant une mesure de variance dépassant le seuil donné. Finalement les motifs ayant une faible densité sont alors *étendus* via l'utilisation de l'alignement multiple de séquences.

1.3.1.1 Encodage du document

La première étape de la méthode IEPAD consiste à construire la séquence de symboles à analyser. Pour construire cette séquence, les auteurs s'appuient sur l'hypothèse que l'information structurelle d'un

³Un arbre PAT d'une chaîne s est un arbre Patricia (Practical Algorithm to Retrieve Information Coded in Alphanumeric) construit sur l'ensemble suffixes de s

document n'est contenue que dans les balises HTML. Dans ce cas les informations utiles à la construction de motifs d'extraction sont les informations structurales. La version encodée d'un document est alors construite à partir d'un alphabet contenant deux types de symboles : les symboles $HTML(x)$ et les symboles $TEXT$. Le paramètre des symboles HTML est le nom de la balise correspondante et la liste de ses attributs. On ne s'intéresse pas au contenu des parties textuelles pour les raisons citées ci-dessus. Ainsi la version encodée du document de la figure 1.1 est donnée dans la figure 1.10.

```
HTML(<html>) HTML(<body>) HTML(<h1>) TEXT HTML(</h1>) HTML(<table>)
HTML(<tr>) HTML(<td>) TEXT HTML(</td>) HTML(<td>) TEXT HTML(</td>)
HTML(</tr>) HTML(<tr>) HTML(<td>) TEXT HTML(</td>) HTML(<td>) TEXT
HTML(</td>) HTML(</tr>) HTML(<tr>) HTML(<td colspan="2">) HTML(<hr>)
HTML(</td>) HTML(</tr>) HTML(<tr>) HTML(<td>) TEXT HTML(</td>)
HTML(<td>) TEXT HTML(</td>) HTML(</tr>) HTML(</table>) HTML(</body>)
HTML(</html>)
```

Figure 1.10 – Résultat de l'encodage de la source donnée par la figure 1.1

1.3.1.2 Génération des motifs initiaux

L'étape suivante consiste à extraire du document encodé un ensemble de motifs candidats. Ceci est fait en remplaçant chaque symbole apparaissant dans la séquence par un code composé d'une succession de bits. Chaque symbole est représenté par une suite de n bits uniques, n étant choisi en fonction du nombre de symboles initiaux. La figure 1.11 donne les codes associés aux symboles qui apparaissent dans le document encodé de la figure 1.10. Dans ce cas il y a 15 symboles. Pour coder la source en binaire, 4 bits sont alors nécessaires.

Symbole	Code	Symbole	Code
HTML(<html>)	0000	HTML(<body>)	0001
HTML(<h1>)	0010	TEXT	0011
HTML(</h1>)	0100	HTML(<table>)	0101
HTML(<tr>)	0110	HTML(<td>)	0111
HTML(</td>)	1000	HTML(</tr>)	1001
HTML(<td colspan="2">)	HTML(<hr>)	1011	
HTML(</table>)	1100	HTML(</body>)	1101
HTML(</html>)	1110		

Figure 1.11 – Codes binaires associés aux symboles de la source encodée de la figure 1.10.

```
0000 0001 0010 0011 0100 0101 0110 0111 0011 1000 0111 0011 1000 1001
0110 0111 0011 1000 0111 0011 1000 1001 0110 1010 1011 1000 1001 0110
0111 0011 0111 0111 0011 1000 1001 1100 1101 1110
```

Figure 1.12 – Codage en binaire de la source encodée de la figure 1.10

A partir du codage en binaire de la source, un arbre PAT est construit. Chaque feuille d'un tel arbre représente un suffixe de la chaîne initiale. Les propriétés de ce type d'arbres sont intéressantes pour la découverte de répétitions dans une chaîne. En effet, toutes les chaînes représentées par une feuille d'un

même sous-arbre ont un préfixe commun qui correspond à l'étiquette du chemin de la racine de l'arbre à celle du sous-arbre. De plus, chaque noeud interne représente à la fois l'ensemble des occurrences du mot correspondant au chemin de la racine au noeud. Dans la méthode IEPAD les motifs initiaux sont construits à partir des *répétitions maximales* dans la séquence codée. Un mot m est une répétition maximale dans une chaîne s si m a au moins deux occurrences dans s et qu'elles ne peuvent pas être "élargies" à gauche ou à droite : c'est à dire que m est divergente droite et gauche dans s . Une répétition m est divergente droite (resp. gauche) s'il existe au moins deux occurrences m_i et m_j de m dans s telles que le caractère suivant (resp. précédant) m_i dans s est différent du caractère suivant (resp. précédant) m_j . Par exemple, dans la chaîne **cbababbaba**, la sous-chaîne **ab** est divergente droite car deux de ses occurrences (les deux premières) sont suivies d'un symbole différent (a et b). Par contre elle n'est pas divergente gauche car toutes les occurrences sont précédées du même symbole (b). En ajoutant en tête ce symbole on obtient la sous-chaîne **bab** qui elle est divergente droite et gauche donc est une répétition maximale. Plus formellement une répétition m est maximale si elle est à la fois divergente gauche et divergente droite dans s . L'une des propriétés intéressante d'un arbre PAT est que chaque noeud interne représente une répétition divergente droite. En effet, un tel noeud a toujours deux fils. Ainsi, pour trouver les répétitions maximales il suffit de chercher les noeuds internes de l'arbre PAT qui représentent des mots qui sont aussi divergents gauche. Pour vérifier si un noeud interne représente un mot divergeant gauche, il faut chercher deux occurrences précédées chacune d'un symbole différent. Dans un arbre PAT, pour un noeud interne donnée n , chaque feuille du sous-arbre correspond à une occurrence. En stockant dans les feuilles les informations relatives aux positions de l'occurrence dans l'arbre il est aisé de connaître la position de début de chacune de ces occurrences de n . Ainsi pour savoir si le mot représenté par n est divergeant gauche il suffit de trouver deux occurrences o_1 et o_2 , de positions respectivement p_1 et p_2 , telles que les symboles dans la chaîne initiale de positions $p_1 - 1$ et $p_2 - 1$ diffèrent. Ainsi la génération des motifs initiaux se fait en parcourant l'arbre PAT à la recherche des noeuds qui sont divergents gauche.

1.3.1.3 Sélection des motifs

Dans l'ensemble initial, tous les motifs candidats (répétitions maximales) ne sont pas intéressants comme motifs d'extraction. Afin de sélectionner les meilleurs candidats, une évaluation des candidats est effectuée selon trois mesures : la régularité, la compacité et la couverture. Considérons un motif α dont les occurrences n dans la séquence initiale s ont pour positions $p_1, \dots, p_n$ tels que $p_1 < p_2 < \dots < p_n$.

- La *régularité* de α est calculé à partir de la déviation standard⁴ des intervalles entre deux occurrences successives de α . Son calcul se fait en divisant la déviation standard par la moyenne des longueurs. Soit $d_i = p_{i+1} - p_i$ pour $1 \leq i \leq n - 1$, et $\bar{d}$ la moyenne des d_i . Alors

$$reg(\alpha) = \frac{\sqrt{\frac{1}{n-1} \sum_{k=1}^n (d_k - \bar{d})^2}}{\bar{d}} \quad (1.7)$$

- La *compacité* de α est une mesure de densité du motif. Son calcul se fait de la manière suivante, où $|\alpha|$ est la longueur de α :

$$comp(\alpha) = \frac{|\alpha|(n-1)}{(p_n - p_1)} \quad (1.8)$$

- La *couverture* mesure le rapport entre la longueur de la sous-séquence couverte par l'ensemble des

⁴La déviation standard est la racine carré de la variance

motifs et la taille de la séquence entière. Elle se calcule comme suit :

$$cov(\alpha) = \frac{(p_n + |\alpha| - p_1)}{|s|} \quad (1.9)$$

Ces trois mesures servent à associer à chaque motif un rang permettant de les classer. D'autre part, une sélection des motifs se fait en fixant un seuil pour les mesures de compacité et de régularité. Selon le cas, les motifs ayant une mesure en dessous ou au dessus du seuil sont conservés. Toutefois, dans certains cas, une régularité élevée est due au fait que les occurrences du motif sont regroupées par “paquets”. Pour pallier à ce problème les auteurs proposent de faire un partitionnement des occurrences et d'évaluer la régularité de chaque partition. Ainsi, pour construire une règle à partir d'un motif les auteurs utilisent, dans leurs expérimentations, la procédure de décision suivante : pour un motif α si $reg(\alpha) \geq \sigma_{reg}$ il est envoyé au module de partitionnement, sinon si $comp(\alpha) < \sigma_{comp}$ alors α est rejeté, sinon il est envoyé au module de construction de règle.

1.3.1.4 Gestion des motifs à régularité trop élevée

Dans le cas de motifs ayant une régularité au dessus du seuil donné, les auteurs proposent de partitionner les occurrences du motif, pour ensuite calculer la régularité pour chaque partition. Ce partitionnement permettra alors de recalculer la régularité pour chacune des partitions, et, éventuellement, de détecter des occurrences de motifs groupés par “paquets”.

Les partitions sont composées en recherchant dans le document initial les occurrences successives. Elles sont construites en commençant par une seule partition contenant la première occurrence du motif dans l'ordre du texte. Puis de nouvelles occurrences du motifs sont recherchées. La dernière partition créée est la partition courante. Lorsqu'une nouvelle occurrence est rencontrée deux choix sont possibles : soit l'occurrence est rajoutée à la partition courante, soit une nouvelle partition contenant uniquement l'occurrence est créée. Lorsqu'une partition donnée a une régularité satisfaisante (*i.e.* ayant une régularité supérieur au seuil σ_{reg}) le motif est conservé. Lors de l'extraction des occurrences, seules les occurrences issues d'une partition ayant une régularité satisfaisante seront conservées.

1.3.1.5 Gestion des motifs à trop faible densité

Les motifs construits à partir de répétitions maximales ne permettent pas d'obtenir des motifs tolérants un certain nombre d'erreurs. Ainsi, bien que des morceaux de motifs sont découverts avec une densité faible, ils ne sont pas satisfaisants car ils ne couvrent pas des occurrences complètes (*i.e.* certaines valeurs manquent). Par exemple, si un document contient des tuples à quatre valeurs, il peut arriver de ne trouver qu'un seul motif permettant d'extraire deux ou trois valeurs de chaque tuple. Ce genre de motif à une densité faible car entre deux occurrences du motif se trouvent les chaînes contenant les valeurs manquantes. Afin d'étendre de tels motifs, les auteurs proposent d'étendre les occurrences avec le texte les suivant immédiatement et de faire un alignement multiple des occurrences étendus. L'alignement multiple de séquences [36] est un problème rencontré en bio-informatique. Il s'agit de trouver un alignement qui minimise une distance entre un ensemble de séquences. Un tel alignement permet de regrouper de manière plus ou moins optimale les régularités se trouvant dans l'ensemble de séquences. Dans le cas de IEPAD, lorsqu'un motif α , ayant k occurrences de positions respectives $p_1, \dots, p_k$ dans la séquence initiale s , est de densité trop faible, un alignement approximatif des sous-chaînes $s[p_i, p_{i+1} - 1]$, $1 \leq i < k$, est calculé avec l'algorithme *center star*[47, pages 348-350]. Cet algorithme, construit un alignement multiple approximatif (non optimale) d'un ensemble de n séquences. Il commence par ordonner les

séquences par la mesure de leur proximité avec les autres séquences. Ensuite, chaque séquence s_i est alignée de manière optimale avec une séquence s^* construite à partir de l'alignement des $i - 1$ séquences précédentes. Cette séquence est calculée en prenant dans chaque colonne de l'alignement le caractère le plus fréquent. L'algorithme 1.6 décrit cette méthode. L'alignement obtenu permet de construire une expression généralisée. Ceci permet la construction de motifs plus robustes car l'alignement ne nécessite pas que les chaînes soit reconnues de manière exacte.

Entrées

- n séquences à aligner

Sortie

- Un alignement A des n séquences

Procédure

ordonner les n séquences en $s_1, \dots, s_n$

soit A un alignement ne contenant que s_1

$s^* \leftarrow s_1$

pour i de 2 à n **faire**

aligner s^* et s_i en ajoutant dans A les mêmes colonnes que celles ajoutés à s^*

calculer une séquence compromis pour A en choisissant pour chaque colonne le symbole le plus fréquent

fin pour

Algorithme 1.6 – Alignement multiple de n séquence avec l'algorithme *center star*

1.3.1.6 Applications

Dans [15] a été publié un ensemble de résultats de l'application de IEPAD sur des pages issues de moteurs de recherche. La table 1.5 récapitule ces résultats. Les quatre colonnes représentent dans l'ordre, le nom du moteur de recherche, le taux de rappel de la méthode, le taux de précision de la méthode, et la proportion des résultats qui est extraite. Le taux de rappel est le nombre de résultats corrects extraits par rapport au nombre total de résultats figurant sur la page. Le taux de précision et le nombre de résultats corrects par rapport au nombre extraits. Pour calculer ces taux, un résultat extrait est considéré comme correct lorsque ce qui est extrait correspond à au moins une partie du résultat réel. La dernière colonne du tableau permet de connaître quelle proportion de résultats a été extrait. Cette valeur se calcule en faisant la moyenne des rapports entre la longueur extraite d'un résultat et la longueur réelle du résultat. Par ailleurs, les auteurs considèrent qu'une valeur est correctement extraite lorsque la partie textuelle à laquelle appartient la valeur recherchée est extraite. Par exemple, si on cherche à extraire l'URL "http://www.sciences.univ-nantes.fr/" et que la partie textuelle "http://www.sciences.univ-nantes.fr/ - 5k - 28/10/2001" a été extraite, alors les auteurs considèrent ce résultat comme correct. Ceci se justifie par le fait que, dans le cas d'une méthode qui abstrait les parties textuelles en un seul symbole, il n'est pas possible de faire mieux. Toutefois ceci implique qu'à l'issue de l'extraction un post-traitement devra être appliqué pour extraire correctement l'information souhaitée.

En observant ces résultats on constate que, bien que les taux soit élevés, dans aucun des cas toutes

Moteur de recherche	Rappel	Précision	Portion extraite
Altavista	1.00	1.00	0.91
Cora	1.00	1.00	0.97
Excite	1.00	0.97	1.00
Galaxy	1.00	0.95	0.99
Hotbot	0.97	0.86	0.88
Infoseek	0.98	0.94	0.87
Lycos	0.94	0.63	0.94
Magellan	1.00	1.00	0.76
Metacrawler	0.90	0.96	0.78
NorthernLight	0.95	0.96	0.90
Openfind	0.83	0.90	0.66
SavvySearch	1.00	0.95	0.97
Stpt.com	0.99	1.00	0.95
Webcrawler	0.98	0.98	0.98

Table 1.5 – Résultats publiés pour le système IEPAD

les valeurs sont à 1.00 en même temps. Ceci signifie que pour tous les moteurs : soit des valeurs sont extraites en trop (précision inférieure à 1.00), soit des résultats ne sont pas extraits (rappel inférieur à 1.00), soit les résultats ne sont pas extraits complètement (portion extraite inférieure à 1.00).

1.3.1.7 Discussion

Cette méthode propose une nouvelle approche à l'extraction d'informations. Elle se veut non-supervisée dans le sens où pour la génération de motifs elle ne nécessite aucune intervention humaine. Toutefois, pour être viable il est nécessaire de faire appel à l'utilisateur pour choisir parmi les motifs générés et corriger les motifs pour qu'ils puissent extraire entièrement toutes les valeurs sans toutefois en extraire de trop. Le choix des motifs n'est pas toujours facile car l'utilisateur n'a pas forcément la connaissance nécessaire pour comprendre les motifs. De plus, il n'est pas forcément évident pour l'utilisateur de comprendre les effets des différents paramètres (*i.e.* les seuils pour chacune des mesures) sur l'extraction.

Par ailleurs, l'hypothèse implicite que les parties textuelles sont indivisibles, pose problème. En effet, il arrive que plusieurs informations de natures différentes nécessitant d'être séparées se trouvent dans une même partie textuelle. Par exemple, la chaîne "<http://www.sciences.univ-nantes.fr/> - 5k - 28/10/2001" ne contient pas une information mais trois : une URL, une taille en kilo-octet et une date, or elle n'est considérée par le système IEPAD que comme un unique bloc de texte.

Finalement, bien que les motifs générés permettent d'extraire des tuples avec des valeurs manquantes, ceux-ci ne permettent pas, dans l'état, de traiter des documents ayant une structure logique hiérarchique telles que ceux traités par STALKER. En effet les motifs recherchés ne peuvent pas se recouvrir. Hors dans des documents ayant une structure logique hiérarchique certaines valeurs sont partagées par plusieurs résultats. Ce dernier problème, peut être résolu par le système ROADRUNNER présenté ci-après.

1.3.2 ROADRUNNER

Dans [22], Crescenzi et al. proposent une méthode de recherche de la structure commune d'un ensemble de documents en analysant leurs disparités. La méthode mise en oeuvre ne nécessite aucune

intervention humaine *a priori*.

Les auteurs posent le problème de la construction d'un adaptateur comme un problème d'inférence d'une grammaire régulière sans union⁵. Dans leur formulation du problème, les auteurs s'appuient sur le fait que les documents sont générés par des scripts à partir des résultats de l'exécution d'une ou plusieurs requêtes vers un SGBD contenant des données relationnelles à extraire du document. Ils interprètent la transformation des résultats de ces requêtes en un code HTML comme un encodage des données initiales. L'objectif de cette approche est alors de redécouvrir la structure éventuellement imbriquée des données ayant permis la génération des pages analysées. Il s'agit donc de retrouver le schéma des données de la source et d'en extraire les données. Les schémas qu'ils considèrent sont des schémas imbriqués constitués de listes ou de tuples imbriqués et pouvant avoir des valeurs optionnelles. L'expression permettant de décrire la structure est un motif permettant d'extraire à partir d'un ensemble de documents résultants d'un même script les informations qu'ils contiennent.

1.3.2.1 Formalisation

Pour redécouvrir le schéma des données les auteurs utilisent une correspondance entre les schémas imbriqués et les expressions régulières sans union. Étant donné un symbole $\#PCDATA$, et un alphabet Σ ne contenant pas $\#PCDATA$ une expression régulière sans union (ERSU) est une chaîne de caractères construite sur $\Sigma \cup \{\#PCDATA, \cdot, +, ?, (,)\}$ et défini ci-après. Intuitivement un ERSU est une expression régulière dans laquelle n'apparaît pas le symbole d'union $' \mid '$.

Définition 1.7 (Expression régulière sans union). *Étant donné, un alphabet Σ et un symbole $\#PCDATA$ tel que $\#PCDATA \notin \Sigma$, on peut définir une ERSU récursivement. Nous avons $\forall x, x \in \Sigma \cup \{\#PCDATA\}$, x est une ERSU.*

Soit a et b deux ERSU.

- $a \cdot b$ est une ERSU ;
- $(a)^+$ est une ERSU ;
- $(a)^?$ est une ERSU.

La sémantique associée à ces expressions (ie. l'ensemble des chaînes qu'elles reconnaissent) est la sémantique habituellement associée aux expressions régulières. Soit a et b sont deux expressions régulières. Pour une expression régulière donnée x , nous notons $\mathcal{L}(x)$ l'ensemble de chaînes reconnues par x . Cette ensemble est appelé le *langage* reconnu par x . Notons ϵ la chaîne vide. Nous avons :

$$a|b = \mathcal{L}(a) \cup \mathcal{L}(b) \quad (1.10)$$

$$a^+ = \mathcal{L}(a)^* / \{\epsilon\} \quad (1.11)$$

$$a^? = \mathcal{L}(a) \cup \{\epsilon\} \quad (1.12)$$

$$a \cdot b = \mathcal{L}(a) \times \mathcal{L}(b) \quad (1.13)$$

Il existe une correspondance évidente entre une ERSU et un schéma imbriqué. En effet, $\#PCDATA$ correspond à un champ, $+$ correspond à une liste (éventuellement imbriquée), $?$ correspond à un champ optionnel.

Le principe sur lequel repose la méthode ROADRUNNER est que chaque page est l'encodage d'une instance d'un type construit τ . Ainsi pour un ensemble de pages HTML, $s_1, \dots, s_n$ il correspond un

⁵Une grammaire régulière sans union est une grammaire qui peut être décrite à l'aide d'une expression régulière avec les opérateurs habituels (i.e. $\cdot, +, ?, (,), |$) sauf $|$. Une telle expression s'appelle une expression régulière sans union ou ERSU.

ensemble d'instances $i_1, \dots, i_n$ dont les s_i sont les encodages. Extraire des s_k les instances correspondantes i_k revient à construire une expression σ telle que les s_k appartiennent au langage de $\mathcal{L}(\sigma)$. Le type des instances peut être découvert en calculant le type de σ , autrement dit $\tau = \text{type}(\sigma)$. L'application de σ à chacun des s_k permet alors d'en extraire le i_k correspondant. L'objectif est alors de construire l'expression régulière sans union minimale dont le langage contient toutes les pages exemples s_k .

La méthode de construction d'une telle expression considère un treillis d'ERSU, tel que $\sigma_1 \leq \sigma_2$ si et seulement si $\mathcal{L}(\sigma_1) \subset \mathcal{L}(\sigma_2)$. L'ERSU recherchée est, alors, la plus petite borne supérieure des chaînes s_i données en entrée. Autrement dit $\sigma = \sup(s_1, \dots, s_k)$. Étant donné que l'opérateur $\sup$ est associatif, il suffit de faire des calculs de $\sup$ deux à deux pour le calcul de σ , i.e. $\sigma = \sup(s_1, \dots, s_k) = \sup(\sup(\dots \sup(s_1, s_2), \dots), s_k)$. Dans ce contexte, un adaptateur est une expression régulière sans union.

1.3.2.2 Algorithmes

Pour le calcul de l'ERSU minimale dont le langage contient toutes les pages données en exemples les auteurs proposent l'algorithme *match*. Cet algorithme est basé sur une technique que les auteurs ont nommé ACME (Align, Collapse under Mismatch, and Extract). Cette technique consiste à analyser un nouveau document d avec une expression w et de modifier cette expression lorsqu'une disparité apparaît. Deux types de disparités peuvent apparaître : les disparités de balise et les disparités de texte. Il y a une disparité de texte lorsque, des deux côtés, un texte différent apparaît à la position courante. Ceci indique qu'une valeur d'un champ a été détectée. Dans les autres cas, il y a disparité de balise. Dans ce cas, une option ou une répétition a été détectée.

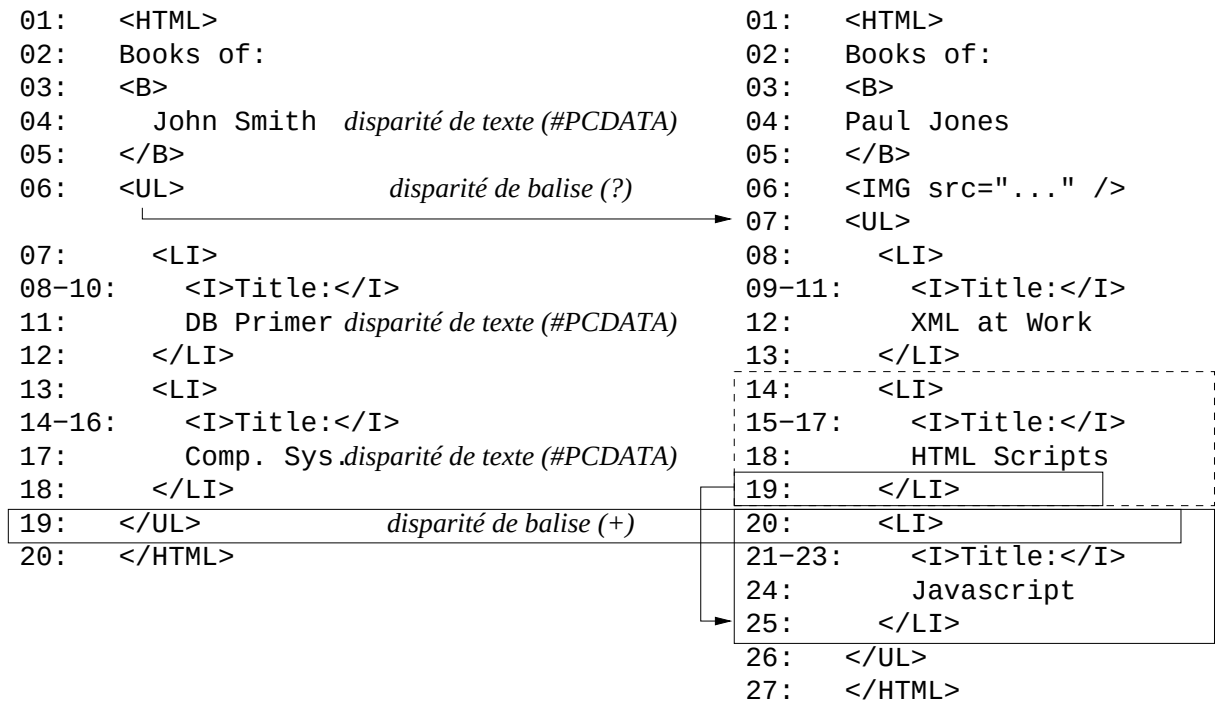


Figure 1.13 – Exemple d'exécution de la fonction *match* (figure issue de [22])

L'algorithme qui est mis en oeuvre dans ROADRUNNER pour résoudre les disparités est l'algorithme

$match(w, d)$, s'appliquant à un adaptateur w et un document d . Lors du premier appel de $match$, w , d sont deux des documents exemples. À la fin de cet appel un nouvel adaptateur est généré. $match$ est alors à nouveau appelé avec w comme nouvel adaptateur et d étant un autre document exemple. Lors d'un appel de $match(w, d)$, w et d sont parcourus simultanément. Lorsque les symboles correspondant ne sont pas les mêmes alors une disparité est détectée. Celle-ci engendre une phase de réconciliation modifiant l'adaptateur. Lorsqu'une disparité de texte est détectée, le symbole textuel à la position courante dans l'adaptateur est remplacé par le symbole #PCDATA. Dans le cas d'une disparité de balise, deux cas peuvent se produire : soit il y a détection d'une option, soit il y a détection de la fin d'une itération. La stratégie mise en oeuvre pour résoudre ce type de disparité est de tenter en premier une résolution par une itération puis, si celle-ci échoue, de résoudre la disparité par l'ajout d'une option. Ce choix est imposé par le fait que lorsqu'une résolution par itération existe, il existe aussi une résolution par option. En effet lorsqu'une disparité de balise peut être résolue par une itération on vient de reconnaître chaque côté (celui de l'adaptateur et celui du document) une même séquence d'éléments. La disparité détecte la fin de la séquence d'un des deux côtés. Cette disparité peut être résolue en rendant optionnel le nouvel élément de la séquence de l'autre côté.

Détection des champs Étant donné deux instances d'un même schéma imbriqué, il est aisé de comprendre que deux valeurs différentes se trouvant à une même position sont des valeurs d'un même champ dans le type imbriqué. Ainsi lorsqu'une disparité de texte apparaît, un symbole #PCDATA est placé à la position correspondante dans l'adaptateur généré. Dans l'exemple de la figure 1.13, les symboles 1 à 3 du côté adaptateur (gauche) correspondent aux symboles 1 à 3 respectivement du côté document les symboles 4 à gauche et 4 à droite sont disparates et de type texte. Ainsi dans l'adaptateur qui sera généré cette position comportera le symbole #PCDATA.

Détection des itérations La détection d'une itération est plus complexe. Une itération est détectée lorsqu'une disparité de balise a eu lieu. Soit w_i et d_j les symboles disparates. S'il s'agit d'une itération, alors l'un des deux est le symbole de fin de liste et l'autre le symbole correspond au début d'un nouvel élément de liste. L'objectif dans la recherche d'une itération est de trouver un motif qui se répète. La méthode proposée consiste à rechercher un bloc se répétant. Supposons que w_i correspond au symbole de fin de la liste et donc que d_j correspond au symbole de début d'un élément. Alors, dans d il existe une occurrence d'un bloque se terminant juste avant d_j et une autre débutant par d_j . La détection de la fin du bloc itérant commençant par d_j se fait en regardant le symbole d_{j-1} précédant d_j . Une autre occurrence d_k de ce symbole est recherchée dans d en commençant au symbole juste après d_j . Ainsi nous avons deux blocs dans d se terminant respectivement en d_{j-1} et d_k . Afin de construire la partie itérée du motif (i.e. celle qui se répète), $match$ est appliqué sur ces deux blocs. Si M est le motif généré alors $M+$ est ajouté à l'adaptateur. Dans l'exemple de la figure 1.13, les symboles 19 à gauche et 20 à droite sont disparates. Dans un premier temps, on teste l'hypothèse que le symbole 19 à gauche est un symbole de fin, ce qui est le cas. En cherchant une occurrence à droite à partir du symbole 21 on trouve le symbole 26. On en conclue qu'il s'agit d'une itération et il faut alors chercher le motif d'itération. Ceci se fait en appliquant sur la partie droite l'algorithme $match$ en arrière en commençant aux positions 21 et 25. La position 25 correspond à l'occurrence du symbole 19.

Détection des options Lorsqu'une résolution par option est tentée, nous sommes dans le cas où une disparité de balise a été détectée et où une résolution par itération a échoué. Ainsi pour résoudre la disparité les auteurs supposent que l'un des symboles disparates est alors optionnel. Afin de détecter

lequel, une recherche croisée est opérée. Si w_i et d_j sont les symboles disparates respectivement dans w et dans d alors soit w_i apparaît dans d après d_j (cas 1) soit d_j apparaît dans w après w_i (cas 2). En supposant que c'est w_i qui soit optionnel, il existe une occurrence de d_j dans w se trouvant après w . Soit w_k cette occurrence, alors toute la chaîne débutant avec w_i (inclus) jusqu'à w_k (exclus) est considérée comme optionnel et l'algorithme *match* continue avec w_k et d_j comme positions courantes. Lorsque la recherche échoue pour le cas 1, on est dans le cas 2 qui est traité de manière similaire. Dans l'exemple de la figure 1.13, les symboles 6 à gauche et 6 à droite sont disparates. Dans ce cas le symbole 6 à gauche est recherché dans la partie droite à partir du symbole 7 et est trouvé à cette position. Ainsi, le symbole 6 à droite est un symbole optionnel et donne lieu à la génération d'une option dans l'adaptateur : *disparité de balise (?)*.

1.3.2.3 Applications

Le système ROADRUNNER a été appliqué sur différentes source de données. Les résultats obtenus sont donnés dans la table 1.6. Les cinq dernières lignes correspondent à des données issues de dépôt RISE. Dans le cas de ce système, la méthode peut soit aboutir à un succès soit à un échec. Le succès de la méthode correspond à la réussite de la construction d'une expression. L'évaluation ne tient pas compte du fait que cette expression n'est peut-être pas celle que l'utilisateur souhaite extraire.

Site Web	Description	Ex.	Résultat	Tmps
amazon.com	voitures par marque	21	oui	266ms
amazon.com	musique par style	20	non	—
buy.com	sous-catégories de produits	20	oui	1s 107ms
buy.com	information produit	10	oui	735ms
rpmfind.net	liste des paquets par nom	30	oui	4s 827ms
rpmfind.net	liste des paquets par distribution	20	oui	1s 963ms
rpmfind.net	description d'un paquet	18	oui	299ms
rpmfind.net	contenue d'un paquet	20	non	—
uefa.com	clubs par pays	20	oui	434ms
uefa.com	joueurs dans l'équipe nationale	20	oui	260ms
(n'existe plus)	Okra	20	oui	700ms
bigbook.com	BigBook	20	oui	781ms
laweekly.com	LaWeekly	28	oui	391ms
iaf.net	Address Finder	10	non	—
pharmweb.net	Pharmweb	10	oui	350ms

Table 1.6 – Résultats publiés pour le système ROADRUNNER

1.3.2.4 Conclusion

La caractéristique qui rend la méthode ROADRUNNER intéressante est le fait que la partie génération de motif est totalement non-supervisée en donnant deux ou plusieurs pages exemples. De plus elle permet de détecter des structures plus complexes qu'une simple relation. Par ailleurs, les auteurs montrent que la solution est viable par des expérimentations sur des données réelles. Cependant quelques problèmes, subsistent. L'utilisateur se retrouve encore face à un choix *a posteriori* d'un motif parmi plusieurs possibles. En effet l'algorithme *match* peut générer plusieurs solutions selon les choix qui sont fait. D'autre

part, bien que *match* découvre la structure, il n'a aucun moyen d'associer une étiquette sémantique aux différentes parties de la structure découverte. Par exemple, l'algorithme pourra trouver que la structure des documents est de la forme $A, (B, C, D^?)+, E$ mais sans savoir que A est un nom d'auteur, que B est le titre d'un des livres qu'il a écrit, que C est le nom de l'événement du livre, etc. Enfin, cette méthode travaille uniquement sur la structure HTML du document en faisant, dès qu'il y a conflit, abstraction des parties textuelles. Ainsi, même si plusieurs informations sont contenues dans une même partie textuelle elles seront considérées comme un unique bloc. Il faudra alors écrire à la main un adaptateur pour ces parties.

1.3.3 Synthèse

Les méthodes structurelles ont comme principal avantage qu'elles sont totalement automatiques dans la construction d'un adaptateur pour une source. Par contre, elles nécessitent toutes une abstraction des parties textuelles du document. Ainsi lorsque qu'une partie textuelle contient plusieurs informations celles-ci seront retournées par ces méthodes en un seul bloc. D'autre part, le format final dans lequel les données extraites se retrouvent n'est pas défini par l'utilisateur. Ainsi, pour ces deux raisons, des post-traitements seront nécessaires pour d'une part séparer complètement les informations et, d'autre part, mettre les données dans un format exploitable par l'utilisateur.

1.4 Extraction de relations

À la différence des approches précédentes, le problème d'extraction de relations à partir du Web, n'a pas en soit l'objectif de générer un adaptateur pour une source ou un domaine donné. Dans ce cas, l'objectif est de construire un ensemble de motifs permettant d'extraire du Web dans sa globalité un sous-ensemble des instances d'une relation donnée. Les motifs sont alors applicables sur l'ensemble des pages du Web. Cette approche a été proposée par Brin dans [10] et a donné lieu à l'application DIPRE. Nous nous intéressons à cette approche car c'est la seule qui permet à l'utilisateur d'exprimer *a priori* son besoin d'informations.

1.4.1 DIPRE : Brin

Le système DIPRE proposé par S. Brin [10] fait l'hypothèse de l'existence d'une dualité entre les motifs et les relations. C'est à dire que, pour une relation donnée, il existe un ensemble de motifs permettant de retrouver une partie des occurrences de la relation. Une occurrence est constituée de tous les champs d'un tuple de la relation sous forme de chaînes dans la base de documents.

Le problème que Brin propose de résoudre est l'extraction itérative d'une relation. Pour ce faire, il utilise la dualité dans les deux sens. À partir d'un ensemble d'exemples d'une relation, on cherche leurs occurrences dans la base de documents. Ensuite, ces occurrences sont utilisées pour construire des motifs. La pertinence de ces motifs est ensuite testée avec un calcul des taux de couverture et d'erreur. À partir des motifs ainsi générés, on recherche de nouvelles occurrences dans les textes pour compléter la relation. Le processus (représenté dans la figure 1.14) est itéré à plusieurs reprises pour aboutir à un point fixe, s'il existe, pour lequel aucune nouvelle instance de la relation n'est générée ou jusqu'à ce que l'utilisateur soit satisfait du nombre d'instances extraites.

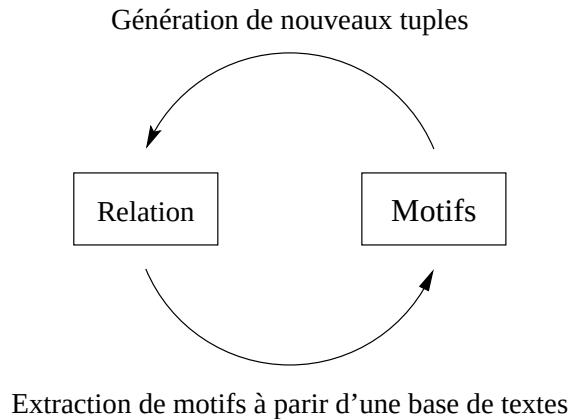


Figure 1.14 – Principe de la dualité motif-relation dans DIPRE

1.4.1.1 Occurrences

La première étape de l'algorithme est la recherche des occurrences des exemples donnés de la relation. Une occurrence est composée des valeurs des champs du tuple correspondant, ainsi que des sous-chaînes du document se trouvant avant la première valeur, après la dernière valeur et entre les valeurs. Ces différentes sous-chaînes forment le contexte de l'occurrence. Dans son exemple sur la recherche de paires (Auteur,Titre), Brin définit une occurrence comme un 7-tuple :

(*auteur, titre, ordre, url, prefixe, milieu, suffixe*)

Les parties *auteur* et *titre* ont la signification qu'elles indiquent d'elles-mêmes, c'est à dire elles correspondent respectivement à un nom d'auteur et un ouvrage. La valeur de *ordre* correspond à une valeur booléenne qui représente l'ordre dans lequel sont apparus l'*auteur* et le *titre* correspondant dans le texte. Par simple convention cette valeur est à vrai si l'auteur apparaît avant le titre et à faux si l'auteur apparaît après. Les parties *url, prefixe, milieu* et *suffixe* correspondent respectivement à l'URL du fichier dans lequel le tuple a été trouvé, la partie précédant le tuple, celle séparant les deux éléments, et celle suivant le tuple. La recherche d'occurrences consiste simplement à trouver les éléments composant un tuple et d'extraire le contexte (URL, ordre, préfixe et suffixe), correspondant.

1.4.1.2 Motifs

Un motif, construit à partir d'un ensemble d'occurrences, peut, *a priori*, prendre n'importe quelle forme. Il est seulement nécessaire qu'il représente un contexte dans lequel les tuples d'une relation peuvent se trouver et qu'il permet d'extraire ces tuples. Ceci peut être obtenu, par exemple, en construisant des motifs représentant les chaînes de caractères autour des valeurs à extraire. Brin spécifie que les contextes doivent permettre l'extraction d'un "format" d'occurrence et que idéalement ne doivent générer que des tuples appartenant à la relation recherchée. Dans cet objectif, pour un motif donné on peut définir deux mesures. La première correspond à la couverture, c'est à dire l'ensemble des éléments de la relation recherchée qui sont générés par le motif. La seconde correspond au taux d'erreur qui est le nombre d'occurrences vérifiant le motif mais dont les tuples correspondants n'appartiennent pas à la relation.

Définition 1.8 (Taux de Couverture). La couverture d'un motif p pour une relation R dans un ensemble de documents D est égale à :

$$C_D(p, R) = \frac{|M_D(p) \cap R|}{|R|}$$

où $M_D(p)$ est l'ensemble des tuples extraits à partir de D en utilisant le motif p .

Définition 1.9 (Taux d'erreur). Le taux d'erreur d'un motif p pour une relation R dans un ensemble de documents D est égale à :

$$E_D(p, R) = \frac{|M_D(p) - R|}{|M_D(p)|}$$

1.4.1.3 Génération de motifs et de tuples

La génération de motifs est la partie la moins évidente du problème. Si un motif est trop général il risque de générer des tuples n'appartenant pas à la relation cherchée. En effet lors des itérations si de tels tuples sont retrouvés dans des contextes différents, des motifs ne correspondant pas à la relation seront générés, et tout le processus d'extraction serait perturbé.

Le principe sur lequel se base le système DIPRE est le suivant : étant donné que l'ensemble de textes desquels sont extraits les motifs est très volumineux, il n'est pas nécessaire d'avoir une grande couverture par motif mais il faut minimiser le taux d'erreur. En suivant ce principe on préférera garder des motifs trop spécifiques plutôt que des motifs généraux ayant une meilleure couverture, et par la même, un taux d'erreur plus élevé.

Une méthode simple de construction, est de regrouper les occurrences partageant le même ordre et le même milieu pour construire un motif. Chacune des parties, *url*, *prefixe*, et *suffixe*, des contextes obtenues à partir de ces occurrences sont généralisés. La généralisation consiste à prendre le préfixe commun pour les parties *url* et *suffixe*, et le suffixe commun pour la partie *prefixe*. Pour éviter d'avoir des motifs trop généraux une mesure de spécificité est calculée. Celle qu'utilise Brin est, où $|x|$ est la longueur de x :

$$specificite(p) = |p.milieu| \times |p.url| \times |p.prefixe| \times |p.suffixe|$$

Étant donné un motif p ayant n occurrences et un seuil t , si la spécificité du motif multiplié par le nombre de motifs est en dessous de t (i.e. $specificite(p) \times n < t$) alors le motif est éliminé. On élimine ainsi des motifs trop généraux.

Les motifs ainsi générés peuvent maintenant être recherchés dans la base de textes pour permettre de retrouver de nouveau tuples. Pour se faire, il suffit d'effectuer l'opération inverse réalisée auparavant : rechercher les parties du motif, puis extraire les éléments du tuple. La figure 1.15 montre les étapes de la construction d'un nouveau tuple.

1.4.1.4 Applications

Cette méthode a été appliquée par Brin pour rechercher des couples (auteur, titre). Ses premiers résultats sont intéressants. La base de documents sur laquelle a travaillé Brin était composée de 24 millions de pages Web d'une taille de 147 gigaoctets. Toutefois uniquement des sous-ensembles de cette base ont été utilisés à chaque itération. Les exemples de départs étaient au nombre de cinq. Après une première itération 199 occurrences ont été trouvées et trois motifs ont été générés. A la seconde itération 3972 occurrences ont été trouvées pour générer 105 motifs. Finalement la dernière itération a permis d'extraire 15257.

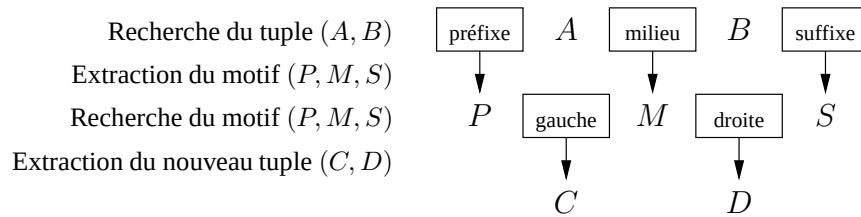


Figure 1.15 – Construction d'un nouveau tuple (C, D) d'une relation à partir d'un exemple (A, B) en utilisant comme motif intermédiaire (P, M, S)

Nous avons alors implanté une version de DIPRE que nous avons testé sur deux autres relations : l'une sur la correspondance entre le nom d'une personne et son e-mail, et la seconde sur la relation (réalisateur, film). Dans la première application, nous avons trouvé des résultats similaires bien que notre base de texte (une boîte aux lettres électronique) était bien plus petite. Par contre la seconde application à mise en avant le caractère trop spécifique des contextes trouvés dans les parties non-structurées des documents. En effet, il est apparu des occurrences pour lesquelles la partie entre deux valeurs était composé d'une longue chaîne contenant du texte. Celle-ci étant spécifique à l'occurrence, le contexte obtenu n'a pu être généralisé avec ceux des autres instances. Ceci est dû au fait que la méthode de Brin impose de conserver intactes les parties milieu des contextes. A cause de cette contrainte les motifs obtenus sont souvent trop spécifique pour faire une extraction exhaustive des données d'une page ayant un grand nombre d'instances de la relation souhaitée.

1.4.1.5 Conclusion

Cette méthode est intéressante car son principe est simple mais efficace. De plus, l'utilisateur peut exprimer très simplement son besoin d'information en donnant quelques exemples de la relation qu'il recherche à construire. Cependant, bien qu'elle permet de répondre à l'objectif d'extraire un grand nombre d'instances à partir d'un ensemble volumineux de pages Web, les motifs obtenus sont très spécifiques. Dans le cas d'extractions de gros volumes, ceci permet d'avoir des motifs qui n'extraient pas d'instances erronées. Par contre, cette trop grande spécificité des motifs rends difficilement d'envisageable l'utilisation de cette méthode pour la construction d'adaptateurs. En effet, des motifs trop spécifiques ne s'appliquent uniquement dans des cas où les données contiennent peu de texte entre deux valeurs successives, ce qui n'est pas le cas de nombreuses sources de données Web.

1.5 Adaptateurs à base de connaissances

Une dernière approche de l'extraction d'informations à partir du Web est celle utilisant des connaissances du domaine. Dans ce cas, l'objectif n'est pas d'obtenir un adaptateur spécifique à une source donnée mais de construire un adaptateur générique pouvant s'appliquer à des pages appartenant à un domaine donné. Dans la littérature, deux méthodes basées sur ce principe ressortent. La première est implantée par le système de Gao et Sterling qui procède par l'extraction d'unités de connaissances (par exemple un prix, un nombre de chambres, un type de logement *etc.*) puis les regroupent en concepts plus complexes tels que une chambre d'hôtel, une location, *etc.* La seconde est implantée par le système XTROS qui découpe un document à analyser en lignes logiques, et utilise la connaissance du domaine pour abstraire les valeurs reconnues dans la ligne en la séquence des types de ces valeurs. Ensuite une

recherche du motif le plus fréquent dans la séquence des lignes ainsi abstraites est effectuée. Dans les deux cas, la connaissance du domaine prend la forme de motifs généraux permettant de détecter des valeurs d'un type donnée. Par exemple, la connaissance peut définir qu'un nombre précédé de la chaîne "Price : \$" est de type prix. Dans les deux cas seule la connaissance est nécessaire pour faire l'extraction. Toutefois il est nécessaire de constituer au préalable l'ensemble des connaissances. Dans ce contexte, un adaptateur est composé de la connaissance et du programme permettant d'appliquer cette connaissance à un document. La connaissance étant générée pour un domaine donné, l'adaptateur ainsi obtenu sera le même pour toute page du domaine indépendamment de la source. Ci-après nous décrivons plus en détails ces deux méthodes.

1.5.1 Gao et Sterling

Dans leur article [41], Gao et Sterling proposent de construire un adaptateur à base de connaissances pour le domaine des annonces immobilières. La méthode qu'ils mettent en oeuvre consiste à extraire des unités élémentaires de connaissances puis de regrouper ces unités en concepts plus complexes. Par exemple, un prix est une unité élémentaire et une annonce est un concept complexe.

1.5.1.1 Connaissances du domaine

Les connaissances du domaine qui sont fournis à l'adaptateur générique sont de deux types : une hiérarchie de décomposition de concepts et une description des concepts élémentaires. La figure 1.16 donne la hiérarchie de concepts pour le domaine des annonces immobilières.

La déclaration des liens entre les concepts peut se faire au travers de trois prédicats *sets_of*, *a_set_of* et *a_sequence_of*. Soit C_1 et C_2 deux concepts, et soit L une liste de concepts. Les trois prédicats se définissent alors comme suit :

- $sets_of(C_1, C_2)$: une instance de C_1 se décompose en un ensemble d'instances de C_2 ;
- $a_set_of(C_1, L)$: une instance de C_1 se décompose en un ensemble contenant des instances des concepts de L ;
- $a_sequence_of(C_1, L)$: une instance de C_1 se décompose en une liste contenant des instances des concepts de L .

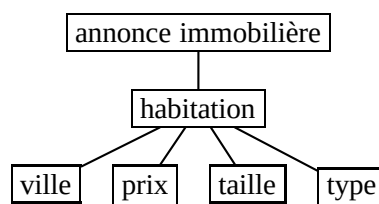


Figure 1.16 – Exemple de hiérarchie de concepts dans la méthode de Gao et Sterling

La description des concepts élémentaires comprend :

- une *fonction d'extraction* permettant à partir d'une représentation textuelle d'un concept d'en extraire la valeur et de la mettre dans une unité standard : par exemple, dans le cas où un loyer est exprimé euros par mois, la chaîne "200 EUR/semaine" sera transformée en la valeur réelle de 800 ;
- une *valeur de certitude* comprise entre 0 et 1 permettant d'associer un taux de confiance à la description (une description très fiable aura un taux proche de 1 alors qu'une description peu fiable aura un taux proche de 0) ;

- une séquence *d'entrées lexicales* qui doivent apparaître pour détecter la présence du concept.

Chaque entrée lexicale décrit un ensemble de sous-chaînes du document par l'intermédiaire d'un ensemble de contraintes. Lorsqu'une chaîne vérifie l'ensemble de ces contraintes on dit qu'elle est reconnu par l'entrée lexicale. Les types de contraintes qu'il est possible de définir sont les suivants :

- un *style* que doit vérifier la chaîne parmi l'ensemble suivant : balise (la chaîne est une balise HTML), base de données (la chaîne appartient à un ensemble de mots prédéfinis), caractère (la chaîne est un unique caractère), mot (la chaîne est composée uniquement de caractères alphanumériques), phrase (la chaîne forme une phrase) ou chaîne (la chaîne est quelconque) ;
- une liste de mots clés positifs à laquelle la chaîne doit appartenir ;
- une liste de mots clés négatifs à laquelle la chaîne ne doit pas appartenir ;
- un motif que doit vérifier la chaîne, il peut s'agir d'un prédicat tel que *any_number(X)* spécifiant que la chaîne doit être uniquement composé de chiffres ;
- des contraintes de tailles minimale ou maximale que peut prendre la chaîne ;
- des contraintes de domaines (valeurs minimales ou maximales) pour les données numériques.

Construire ces connaissances n'est pas une tâche évidente. Elle nécessite d'une part une connaissance profonde du domaine en questions : quels sont les concepts, comment ils s'organisent, *etc.* D'autre part, elle nécessite aussi une connaissance des différentes manières de présenter une même information. Par exemple, un prix peut être précédé d'un préfixe "Prix :", être suivi d'une monnaie "EUR", *etc.* La construction d'un adaptateur permettant d'extraire correctement toutes les valeurs d'un même domaine nécessite de faire l'inventaire de toutes les formes de présentations possibles. Dans certains cas, il peut même arriver qu'il soit impossible de détecter qu'une valeur donnée est d'un certain type. Par exemple, dans le cas du site d'annonces immobilières *bonjour.fr*, les prix se retrouvent isolés de tout contexte immédiat qui permettrait de construire une description pouvant détecter qu'il s'agit d'un prix. D'autre part, il est facile de construire une expression permettant de détecter et de délimiter très exactement une valeur numérique comme un prix. Par contre, dans le cas d'un titre, il est plus difficile d'avoir une telle expression. Ainsi, la construction d'une connaissance d'un domaine, pour l'extraction d'informations est loin d'être évidente.

1.5.1.2 Algorithme d'extraction

Les hypothèses sur lesquelles se base l'algorithme d'extraction sont les suivantes :

- les instances de concepts élémentaires peuvent être identifiées par leur description ;
- les instances de concepts de plus haut niveau peuvent être identifiées par leur composition donnée par la hiérarchie des concepts ;
- si l'ordre d'apparition des concepts n'a pas d'importance, la proximité des concepts unitaires peut aider à déterminer leur appartenance à un concept de plus haut niveau ;
- les instances de concepts unitaires appartenant à un titre (*i.e.* dans une balise `<title>`, `<h1>`, *etc.*) peuvent faire partie des instances de concepts des parties se trouvant sous le titre.

La méthode proposée s'appuie sur la présence de balises de trois niveaux. Du plus bas au plus haut nous avons : le niveau mot, le niveau ligne et le niveau paragraphe. Le principe générale de la méthode est d'abstraire de plus en plus le document, en supprimant au fur et à mesure les balises de niveau de plus en plus élevés. La première phase consiste à appliquer l'ensemble des descriptions des concepts élémentaires au document. Une fois ces concepts initiaux détectés les parties textuelles restantes sont supprimées ainsi que les balises du niveau mot. Cela donne une abstraction du document dans laquelle certaines occurrences de concepts se retrouvent côte à côte. La hiérarchie des concepts est alors utilisée pour regrouper les occurrences de concepts ayant un concept parent commun. Ensuite, une nouvelle

phase d'abstraction a lieu en supprimant les balises de niveau ligne et en regroupant à nouveau les concepts se retrouvant côte à côte. Finalement, une dernière phase d'abstraction supprime les balises de niveau paragraphe et effectue les regroupements possibles. À la fin il ne reste plus qu'une séquence de concepts.

La figure 1.17 donne une vision schématique de l'abstraction, et la figure 1.18 donne un exemple d'application de cette méthode. Les deux figures représentent l'application de la méthode sur le même exemple. Initialement le document est celui du début de la figure 1.18. La première étape consiste à séparer le document en symboles textes et balises : les blocs de textes donnent lieu à un symbole `text(X)` et les balises à un symbole `tag(X)`. Ensuite toutes les balises sont remplacées par leur type (mot, ligne, paragraphe). Dans l'exemple, `tag(<BR>)` est remplacé par `b(line)` car la balise `<BR>` est une balise de type ligne. De manière similaire `<B>` est remplacé par `b(word)` car `<B>` est de niveau mot. L'étape suivante consiste à appliquer les règles de détection des concepts élémentaires sur les parties textuelles en les divisant si nécessaire. C'est ainsi que, par exemple, `text(1 BR)` est transformé en `g("taille", ku, ["taille"], [4])`. Lorsque plusieurs règles détectent des chaînes se chevauchant comme occurrences de concepts élémentaires, la règle avec la plus grande fiabilité est choisie. Après l'extraction des concepts élémentaires, trois étapes d'abstractions et de regroupement ont lieu. Ces phases sont schématiquement représentées dans la figure 1.17. Dans cette figure nous avons initialement le document de la figure 1.18 après l'extraction des concepts élémentaires. La deuxième ligne correspond à ce même document après une phase d'abstraction consistant à supprimer les balises de type mot. Deux groupes de concepts apparaissent alors : le groupe 4, 5 et 6, et le groupe 7, 8 et 9 qui ne forment alors plus qu'un concept à la phase suivante (troisième ligne de la figure 1.17 et à la cinquième étape de la figure 1.18). La phase suivante consiste à supprimer les balises de niveau ligne, puis, à effectuer à nouveau une phase regroupement de concepts. Dans l'exemple, les groupes 4, 5, 6 et 7, 8, 9 ayant la même structure et se trouvant alors côte à côte sont regroupés en un ensemble de parties ("`part_sets`"). Ensuite étant donné que le concept élémentaire 3 n'est pas séparé de cette liste par une balise il vient compléter chacun des éléments de la liste. Cette phase terminée, une nouvelle phase d'abstraction, consistant à abstraire les balises de niveau paragraphe, permet d'éliminer les balises restantes. Ensuite, une dernière phase de regroupement de concepts a lieu. Dans cette phase, les éléments de la liste de parties `[ [3, 4, 5, 6], [3, 7, 8, 9] ]` sont transformés en deux concepts formant une nouvelle liste de concepts "`habitation`" `[ [9], [10] ]`. Finalement, cette liste de concepts est transformée en un unique concept de type "`annonces immobilières`" `[12]`.

L'algorithme d'extraction qui est mise en oeuvre travaille en trois temps : (1) abstraction de la page, (2) recherche des concepts élémentaires et (3) regroupements. Les auteurs indiquent que l'algorithme de regroupement se compose de 23 règles sans toutefois préciser lesquelles.

L'abstraction de la page consiste d'abord à lire caractère par caractère le document duquel il faut extraire. Ensuite, ceux-ci sont regroupés en symboles balise ou texte. Les balises sont classifiées selon leur niveau (mot, ligne, paragraphe ou titre) et remplacées par leur type (resp. `b(word)`, `b(line)`, `b(para)`, `b(title)`). Les symboles textes `text(_)` et symboles `b(word)` sont regroupés en lignes logiques. L'extraction des concepts élémentaires consiste à appliquer les descriptions sur les lignes logiques en remplaçant toute sous-chaîne vérifiant la description par une expression `g(C, "ku", [C], [V])` où `C` est le nom du concept extrait, "`ku`" indique qu'il s'agit d'un concept élémentaire ("knowledge unit"), et `V` est l'index du concept dans une liste des concepts extraits.

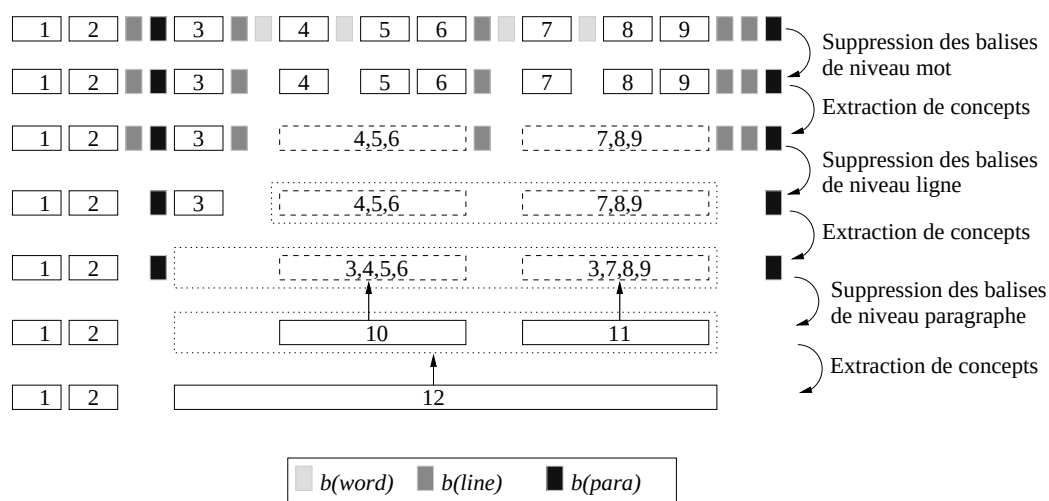


Figure 1.17 – Abstraction d'un document avec la méthode de Gao et Sterling

1.5.1.3 Applications

Gao et Sterling ont appliqué leur méthode dans le domaine des petites annonces en ligne. Ils ont testé leur méthode sur divers sites proposant des annonces de location immobilières ou des annonces de ventes d'automobiles. La table 1.7 donne les résultats obtenus par les auteurs sur deux sites. Cette table est divisée en trois parties : la première décrit les résultats de la détection des concepts élémentaires, la seconde donne les résultats du regroupement des concepts, et la dernière donne les performances globales. Les colonnes NbI donnent le nombre total de concepts, P donne la précision obtenues (*i.e.* le nombre de concepts extraits et corrects sur le nombre de concepts extraits) et R donne le rappel obtenu (*i.e.* le nombre de concepts extraits et corrects sur le nombre de concepts réelles). Dans la partie regroupement de concepts, l'évaluation ne prends pas en compte les erreurs due à la détection de faux concepts élémentaires (faux positifs) ou la non détection de concepts élémentaires (faux négatifs).

Site Web	Extraction des concepts unitaires			Regroupements			Performances globales		
	Nb I	P	R	Nb I	P	R	Nb I	P	R
www.newsclassifieds.com	1082	95	95	312	75	76	312	69	70
market.fairfax.com.au	353	93	93	94	79	78	94	70	70

Table 1.7 – Résultats de la méthode de Gao et Sterling

1.5.1.4 Conclusion

Le système proposé par Gao et Sterling, se base sur une connaissance du domaine constituée de descriptions des concepts élémentaires, d'une hiérarchie de décomposition des concepts et d'une classification des balises HTML en trois classes : les balises de niveau paragraphe, ligne et mot. La méthode proposée fonctionne en plusieurs phases successives d'abstractions et d'extraction de concepts. L'extraction se fait en appliquant un ensemble de règles sur le document. Cependant ni les règles qui ont été utilisées par les auteurs, ni une évaluation de celles-ci n'ont été publiés à notre connaissance. Ainsi

Document initial	
1	HOUSES & UNITS
2	<P>RESERVOIR
3	1 BR SC flat, fully furnished, \$110 pw
4	2 BR unit, BIR's, Inge, gas appl, \$135pw
5	Ph 9478 2222 </P>
Séparation en texte/balise	
1	text(HOUSES & UNITS)tag()
2	tag(<P>)text(RESERVOIR)tag()
3	tag()text(1 BR)tag()text(SC flat, fully furnished, \$110 pw)tag()
4	tag()text(2 BR)tag()text(unit, BIR's, Inge, gas appl, \$135pw)tag()
5	text(Ph 9478 2222)tag()tag(</P>)
Remplacement des balises par leur type	
1	text(HOUSES & UNITS)b(line)
2	b(para)text(RESERVOIR)b(line)
3	b(word)text(1 BR)b(word)text(SC flat, fully furnished, \$110 pw)b(line)
4	b(word)text(2 BR)b(word)text(unit, BIR's, Inge, gas appl, \$135pw)b(line)
5	text(Ph 9478 2222)b(line)b(para)b(para)
Extraction des concepts élémentaires	
1	g("type",ku,["type"],[1])g("type",ku,["type"],[2])b(line)
2	b(para)g("ville",ku,["ville"],[3])b(line)
3	b(word)g("taille",ku,["taille"],[4])b(word)g("type",ku,["type"],[5])g("prix",ku,["prix"],[6])b(line)
4	b(word)g("taille",ku,["taille"],[7])b(word)g("type",ku,["type"],[8])g("prix",ku,["prix"],[9])b(line)
5	b(line)b(para)
Suppression des balises de niveau mot et extraction de concepts	
1	g("type",ku,["type"],[1])g("type",ku,["type"],[2])b(line)
2	b(para)g("ville",ku,["ville"],[3])b(line)
3	g("habitation",part,["taille","type","prix"],[4,5,6])b(line)
4	g("habitation",part,["taille","type","prix"],[7,8,9])b(line)
5	b(line)b(para)
Suppression des balises de niveau ligne et extraction de concepts	
1	g("type",ku,["type"],[1])g("type",ku,["type"],[2])
2	b(para)g("ville",ku,["ville"],[3])
3	g("habitation",part_sets,["taille","type","prix"],[[4,5,6],[7,8,9]])
4	b(para)
1	g("type",ku,["type"],[1])g("type",ku,["type"],[2])
2	b(para)
3	g("habitation",part_sets,["ville","taille","type","prix"],[[3,4,5,6],[3,7,8,9]])
4	b(para)
Suppression des autres balises et extraction de concepts	
1	g("type",ku,["type"],[1])g("type",ku,["type"],[2])
2	g("habitation",concept_set,["ville","taille","type","prix"],[10,11])
1	g("type",ku,["type"],[1])g("type",ku,["type"],[2])
2	g("annonce immobilière",concept,["habitation","habitation"],[12])

Figure 1.18 – Exemple d'application de la méthode Gao et Sterling

il est impossible de connaître la réelle généralité de ces règles. De plus, ces règles ont probablement été écrites de manière ad-hoc, laissant un doute sur leur complétude. Par ailleurs, la construction des règles d'extraction des concepts élémentaires ne semble pas évidente. De plus, la construction de telles règles nécessite une étude profonde des documents du domaine pour lequel on souhaite construire un adaptateur.

1.5.2 XTROS

Une autre méthode à base de connaissances plus récente est adoptée par le système XTROS [105]. Ce système, à la manière de celui proposé par Gao et Sterling, cherche à détecter les concepts élémentaires en s'appuyant sur une connaissance du domaine. Il s'appuie aussi sur les balises HTML pour décomposer les documents. La différence entre les deux systèmes est que celui de Gao et Sterling cherche à abstraire successivement le document en utilisant une hiérarchie de concepts, alors que le système XTROS fait une recherche de séquences de concepts fréquents pour construire les concepts de plus haut niveau. Cette recherche s'appuie sur une décomposition du document en une séquence de lignes et de la reconnaissance des concepts élémentaires dans chacune des lignes. Chaque ligne forme alors une séquence de concepts ou motif. Les motifs les plus fréquents sont alors utilisés pour construire des règles d'extraction.

1.5.2.1 Types de documents considérés

Le type de documents sur lesquels la méthode s'applique sont des documents dit à *labels*. Un *label* est une chaîne permettant de déterminer la sémantique d'une valeur. Par exemple, dans la chaîne "Chambres : 3" la sous-chaîne "Chambres :" est un label permettant d'indiquer que "3" est un nombre de chambres. Dans un document à label toute valeur pouvant être extraite se trouve à proximité d'un *label*. Dans le cas de ce système l'utilisateur donne un ensemble de connaissances sur les concepts élémentaires qu'il souhaite extraire. Par exemple, dans le cas des annonces immobilières de tels concepts peuvent être le prix, le nombre de chambre, la ville, etc. A chaque concept est associé une description. Par ailleurs, le format des pages considérées dans XTROS sont des pages de la forme $P = \langle H, L, T \rangle$ où H (resp. T) est un préfixe (resp. suffixe) de la page à ignorer et où $L = \langle L^1, \dots, L^m \rangle$ se décompose en plusieurs tuples $L^i = \langle V_1^i, \dots, V_n^i \rangle$. Dans la terminologie de Kushmerick les pages considérées sont des pages tabulaires. De plus, il est supposé que chaque occurrence d'un concept élémentaire est associé à un label. Ceci implique que cette méthode ne pourra pas s'appliquer lors de l'extraction de valeurs n'ayant pas de label associé.

1.5.2.2 Format des connaissances

Dans le système XTROS les connaissances du domaine sont décrits à l'aide d'un document XML. Celui-ci se décompose en deux parties : une liste des noms de concepts élémentaires et une suite de descriptions associées à chaque concept élémentaire. Une description contient une liste de termes pouvant jouer le rôle de label pour le concept et un ensemble de formats permettant de définir la position de la valeur associée aux concept (par exemple une valeur numérique dans le cas du concept prix) relativement à son label. La figure 1.19 donne un extrait de la base de connaissance correspondant à la description du concept prix (PRICE). Elle se décompose en deux parties : une partie décrite par l'élément **ONTOLOGY** et une autre décrite par l'élément **FORMAT**. La première contient un ensemble d'éléments **TERM** décrivant chacun une chaîne pouvant apparaître comme label. La seconde partie est composée d'un ensemble d'éléments **FORM** décrivant chacun un format dans lequel peuvent apparaître les concepts. Par exemple,

le premier élément **FORM** décrit qu'un prix peut prendre la forme d'un label issu de l'ensemble des termes suivi de chiffres.

```

1 <PRICE>
2 <ONTOLOGY>
3   <TERM>PRICE</TERM>
4   <TERM>$</TERM>
5   <TERM>PRICE $</TERM>
6 </ONTOLOGY>
7 <FORMAT>
8   <FORM>[ONTOLOGY] DIGITS</FORM>
9   <FORM>DIGITS [ONTOLOGY]</FORM>
10 </FORMAT>
11 </PRICE>

```

Figure 1.19 – Exemple de description d'un prix dans XTROS

1.5.2.3 Algorithme d'extraction

L'algorithme mis en oeuvre par XTROS consiste à construire un adaptateur en exploitant la connaissance du domaine. L'algorithme procède en plusieurs étapes : découpage du document en lignes logiques, interprétation des lignes en utilisant la connaissance, et recherche des motifs les plus fréquents.

Le première étape consiste à découper le document en lignes logiques (*i.e.* les lignes telles qu'elles s'affichent à l'écran). Elles sont construites en interprétant certaines balises HTML telles que **
, **<P>, *etc.* Les balises ne servant pas au découpage sont simplement supprimées. Par rapport à la méthode de Gao et Sterling ceci correspond à la suppression des balises de niveau mot et au découpage du document à chaque balise de niveau ligne.

L'étape suivante consiste à interpréter les lignes logiques en recherchant les concepts qu'elles contiennent. Pour ce faire les termes associés à chaque concept sont recherchés dans chaque ligne. Lorsqu'une ligne contient un terme d'un concept et qu'un des formats du concept est vérifié alors le sens associé à la ligne est le concept élémentaire reconnu. Par exemple, en reprenant la définition du concept "prix" de la figure 1.19, une ligne contenant "PRICE \$ 234" se verra associée aux concept "prix". Lorsque plusieurs concepts se retrouvent dans une même ligne, celle-ci est décomposée en autant de nouvelles parties. Ensuite un symbole unique est associé à chaque concept. On construit alors une séquence de symboles correspondant à la séquence de parties. Chaque symbole de la séquence correspond au concept de même position dans la séquence des parties.

La dernière étape consiste à rechercher les suites de symboles les plus fréquentes dans la séquence des symboles. Ceci commence par la construction d'un ensemble de séquences candidates composé des sous-séquences ne contenant pas deux fois le même symbole. Ensuite, l'algorithme 1.7 est utilisé pour faire la recherche de la séquence candidate la plus fréquente. Dans l'algorithme la fonction *frequence* calcule la fréquence d'une séquence candidate *s* dans la séquence de symboles correspondant au document *D*. La fonction *nb_concepts* calcule, pour une séquence *s*, le nombre de concepts qu'elle contient. On remarquera qu'étant donné que chaque séquence ne contient qu'une occurrence au plus d'un même symbole, que le nombre de concepts contenu dans une séquence correspond à sa longueur. Dans l'algorithme, pour chaque candidat *p* plusieurs valeurs sont calculées : sa fréquence f_p , le nombre de concepts

qu'il contient A_p , le nombre total de valeurs $V_p = f_p \times A_p$ contenues dans l'ensemble des occurrences de p . Le motif choisi est celui qui maximise V_p . S'il y en a plusieurs, le plus long est choisi.

```

Soit  $D$  la séquence de symboles issue de l'interprétation des lignes
Soit  $cands$  l'ensemble des sous-séquences de  $D$  ne contenant pas deux fois le même symbole
Soit  $V_{max} = 0$ 
Soit  $len_{max} = 0$ 
pour tout  $x \in cands$  faire
     $f(x) \leftarrow frequency(x, D)$ 
     $A(x) \leftarrow nb\_concepts(x)$ 
     $V(x) \leftarrow f(x) * A(x)$ 
    si  $V(x) > V_{max} \vee (V(x) = V_{max} \wedge nb\_concept(x) > len_{max})$  alors
         $max \leftarrow x$ 
         $V_{max} \leftarrow V(x)$ 
         $len_{max} \leftarrow nb\_concept(x)$ 
    fin si
fin pour
retourne  $max$ ;

```

Algorithme 1.7 – Algorithme de recherche du motif le plus fréquent dans XTROS

1.5.2.4 Applications

Source	URL	Résultat
Homes	http://www.homes.com	succès
Realtor	http://www.realtor.com	succès
Yahoo Real Estate	http://realestate.yahoo.com	échec
iOwn	http://www.iown.com	succès
CyberHomes	http://www.cyberhomes.com	succès
HomeScape	http://www.listinglink.com	échec
Move	http://www.move.com	succès
ERA Real Estate	http://www.era.com	succès
Coldwell Banker	http://www.coldwellbanker.com	succès
HomeSekkers	http://www.homesekkers.com	succès

Table 1.8 – Résultats publié pour le système Xtros

Les auteurs ont rapporté que leur système a été évalué sur les 10 sites d'annonces immobilières de la table 1.8. Sur les 10, un adaptateur n'a pu être généré que pour deux d'entre eux. Les auteurs expliquent cette échec par le fait que les documents issus de ces deux sites ne sont pas des documents à labels. En effet, dans les deux cas il s'agit de sites qui présentent leurs données dans des tables dans lesquels seules les valeurs apparaissent (*i.e.* sans label). Par ailleurs, les auteurs estiment le rappel de leur méthode à 80% et la précision à 100%, sans toutefois préciser comment ces valeurs ont été calculées.

1.5.2.5 Conclusion

Le système XTROS est un adaptateur à base de connaissances permettant d'extraire des données d'un document d'un domaine donné. La méthode utilisée consiste à découper le document en lignes logiques, à rechercher les concepts élémentaires, décrit par la connaissance, dans ces lignes et à rechercher la séquence de concepts la plus fréquente dans le document. Toutefois, les connaissances fournies au système sont des descriptions de formats dans lesquelles les concepts à extraire peuvent apparaître. En effet, les termes qui doivent être fournis par l'utilisateur sont des labels qui permettent la détection de la présence d'une valeur d'un certain type et non un terme du domaine. Par exemple, si l'utilisateur souhaite extraire des prix la terminologie qu'il devra fournir ne sera pas "prix" mais "\$", "PRICE", "PRICE \$", etc. Pour obtenir de tels descriptions il est nécessaire de faire une analyse préalable des documents des différents sites du domaine sur lesquelles pourront s'appliquer l'adaptateur. Il n'est pas suffisant de construire une conceptualisation du domaine. On peut considérer que les labels sont des informations du domaine, mais leur calcul n'est *a priori* pas évident et nécessite une analyse fine de l'ensemble des documents desquels nous souhaitons extraire. Hors l'intérêt même des adaptateurs à base de connaissances est d'être indépendant des sources et de permettre à l'utilisateur de construire un adaptateur sans connaître à l'avance les sites sur lesquels il pourra s'appliquer.

1.5.3 Synthèse

Dans leur objectif, les adaptateurs à base de connaissances permettent la construction d'un adaptateur pour un domaine donné en spécifiant certaines caractéristiques de présentations du domaine. Cependant, recenser l'ensemble des formes de présentations possibles nécessite une analyse de l'ensemble des sites desquelles l'utilisateur souhaite extraire afin de les intégrer dans la connaissance du domaine. Ainsi, ces méthodes ne sont pas réellement indépendantes des sources de données. Par ailleurs, les méthodes à base de connaissances font une utilisation implicite de la structure du document. Elles supposent que les documents sont construits de manière à ce que les valeurs d'un même objet se suivent dans le document. Pourtant ceci n'est pas toujours le cas. De plus elles s'appuient sur l'existence de labels permettant d'indiquer la présence d'une valeur d'un type donné. Hors ceci peut ne pas être le cas, notamment lorsque la source de données est très structurée comme pour une source dont les données sont présentées dans un tableau.

1.6 Résumé

Dans ce chapitre nous avons présenté un ensemble représentatif de méthodes d'extraction d'informations à partir du Web. Ces méthodes se classent en quatre catégories : les méthodes à bases de pages étiquetées, les méthodes structurelles, les méthodes à bases d'instances exemples, et les méthodes à base de connaissances. Dans toutes ces approches, sauf les méthodes à bases d'instances exemples, l'objectif est la construction d'un adaptateur. Dans le cas des approches à base de pages étiquetées et les approches structurelles, l'adaptateur s'applique aux données d'une seule source. Dans le cas des approches à base de connaissance l'adaptateur s'applique aux documents issus de sources d'un même domaine. Dans le cas des méthodes utilisant des instances exemples l'objectif est de construire un ensemble de motifs pouvant s'appliquer à toute page Web. Ces différentes méthodes sont présentées dans la table 1.9.

L'objectif d'un adaptateur est d'extraire des documents HTML d'une source le contenu structuré qu'ils contiennent. Les différentes méthodes présentées permettent l'extraction de tels contenus. Toutefois le format, notamment l'ordre, dans lequel se retrouvent les données après extraction est lié au docu-

Système	Approche	Auteurs	Année	Entrées	VM	O	H
WIEN	Pages étiquetées	N. Kushmerick	1997	Pages étiquetés	N	N	N
STALKER	Pages étiquetées	I. Muslea	1998	Pages étiquetés	O	O	O
SOFTMEALY	Pages étiquetées	C.N. Hsu	1998	Pages étiquetés	O	O	N
IEPAD	Structurelle	C.H. Chang	2000	Pages brutes	N	N	N
ROADRUNNER	Structurelle	V. Crescenzi et al.	2001	Pages brutes	O	N	O
DIPRE	Instances exemples	S. Brin	1998	Instances exemples et pages brutes	O	O	N
–	A base de connaissances	X. Gao et L. Sterling	1999	Connaissances	O	O	O
XTROS	A base de connaissances	H. Seo et al.	2001	Connaissances	O	N	N

où VM,O et H signifient :

VM Traitement des valeurs manquantes

O Traitement d'ordres de données différents

H Traitement de données hiérarchiques

Table 1.9 – Comparaison des systèmes d'extraction d'informations à partir du Web

ment. Hors, l'un des objectifs de l'extraction est de permettre d'intégrer différentes sources de données. Une telle intégration nécessite que les données à intégrer soient présentées dans un même format. Par exemple, pour l'intégration de données de sites de DVD ce format pourrait être exprimé par le schéma : *dvd(titre, acteur_principal, prix)*. Les différentes approches proposées ne permettent pas de spécifier le format dans lequel devront se trouver les données après extraction, mais imposent de suivre la structure logique du document. Par exemple, si dans le document le prix se trouve avant le titre, dans les données résultats ce même ordre sera respecté. Ainsi, les adaptateurs qui sont obtenus par les méthodes décrites permettent de transformer des données dont la structure est inconnue dans un format dont la structure est mieux connue mais qui n'est pas forcément le format définitif. Il est alors nécessaire de transformer à nouveau les données extraites dans le format final que souhaite l'utilisateur. Toutefois, cette transformation est largement simplifiée dans le cas des méthodes à base de pages étiquetées et dans le cas d'adaptateur à base de connaissances car un nom d'attribut ou de concept est associé aux valeurs extraites. Dans ce cas il suffit de donné l'ordre dans lequel les attributs ou concepts devront apparaître. Par contre, dans le cas des approches structurelles, aucune information n'est associée aux valeurs extraites. L'obtention de cette information nécessite l'intervention de l'utilisateur.

En plus du problème de seconde transformation des données, les approches actuelles ne facilitent pas la tâche à l'utilisateur. Les méthodes à base de pages exemples totalement étiquetées imposent un travail d'étiquetage qui peut être long et fastidieux. Les approches structurelles sont guidées par la structure des documents desquels elles cherchent à extraire le contenu et nécessitent des traitements de sélection et de transformation sur les données extraites. De plus, l'utilisateur doit choisir parmi des motifs qu'il ne comprend pas forcément. Finalement dans le cas des adaptateurs à base de connaissances l'utilisateur se retrouve à faire une étude des pages pour pouvoir construire une base de connaissances sur le domaine duquel il souhaite extraire. Il n'est pas évident que, dans la construction d'un adaptateur, cette tâche permette un gain de temps conséquent par rapport aux autres méthodes notamment si, pour chaque nouvelle source, de nouvelles connaissances doivent être intégrées. Cette tâche peut être longue et difficile. De

plus, les connaissances s'expriment dans un langage qui n'est pas forcément connu de l'utilisateur et qu'il devra apprendre.

Un autre problème des adaptateurs est celui de leur mise à jour du à l'évolution constante du Web. En effet, les sites Web sont fréquemment mis à jour. Lors d'une mise à jour, la présentation, l'organisation, ainsi que la terminologie utilisée par le site peuvent changer. Dans ce cas toutes les méthodes sont affectées. Les méthodes à base de pages exemples devront à nouveau être appliquées pour construire un nouvel adaptateur, ce qui nécessite l'étiquetage d'un ensemble de pages exemples du site mise à jour. Bien que certainement moins coûteuse, dans le cas des méthodes à base de connaissances, une étude des documents de la source mise à jour sera nécessaire pour ajouter les éventuels nouveaux labels. D'autre part les post-traitements permettant d'obtenir les données dans le format que souhaite l'utilisateur devront être réadaptés.

1.7 Ouvertures de recherche

Dans ce chapitre nous avons décrit un ensemble de méthodes de construction d'adaptateurs. Ces adaptateurs permettent, d'extraire de documents issus du Web, les données structurées qu'ils contiennent. Cependant, en se plaçant du point de vue du concepteur d'un système faisant appel à de tels adaptateurs plusieurs problèmes subsistent. D'une part la construction d'un adaptateur est fastidieuse et difficilement automatisable. D'autre part, les adaptateurs permettent d'extraire les données à partir de documents, mais ne permettent pas de définir comment ils sont obtenus.

1.7.1 Extraction d'informations à partir d'instances exemples

Toutes ces méthodes sont une avancées dans le domaine de l'extraction d'informations. Ces méthodes simplifient la tâche de construction d'un adaptateur, en évitant à l'utilisateur une analyse des documents HTML desquels il souhaite extraire des informations. Toutefois, cette tâche d'analyse est remplacée selon les cas par un étiquetage manuel fastidieux d'un ensemble de pages exemples, une sélection d'un ensemble de motifs dont la sémantique n'est pas évidente, l'analyse d'un nouveau format de données dans lequel l'adaptateur donne ses résultats ou l'écriture d'une base de connaissances qui nécessite toutefois une analyse partielle des documents. Dans certains cas plusieurs de ces tâches devront être effectuées. Seule la méthode de Brin permet d'éviter toutes ces tâches. En effet, il suffit à l'utilisateur d'exprimer ses besoins en donnant quelques exemples de la relation qu'il souhaite extraire, dans le format dans lequel il souhaite les trouvés, et d'appliquer la méthode sur un ensemble de documents. En résultat, il obtient une réponse à ses besoins. De plus, dans ce cas l'expression des besoins peut se faire avant même d'avoir exploré toutes les sources desquelles l'utilisateur pourrait souhaiter extraire des données. Toutefois, la spécificité des motifs générés fait que l'application de cette méthode ne permet pas l'obtention d'une réponse complète, sauf à donner un grand nombre d'instances.

Un premier problème consiste alors à trouver une méthode de construction d'adaptateurs pour des sources de données Web sans imposer ces contraintes fastidieuses à l'utilisateur. L'expression des besoins de l'utilisateur devra pouvoir se faire simplement en ne donnant que le minimum d'informations requises pour l'obtention d'un adaptateur permettant d'extraire correctement toutes les données d'une source Web. La méthode de Brin montre que simplement à partir d'un petit ensemble d'instances exemples, il est possible d'extraire un grand nombre d'instances d'une relation. Une première ouverture de recherche consiste donc, à étudier la possibilité de construire des adaptateurs efficaces en imposant uniquement à l'utilisateur de donner une poignée d'instances exemples.

1.7.2 Formalisme pour le développement d'applications Web

L'un des objectifs de l'extraction d'informations est de permettre l'automatisation de l'accès à des sources Web. Ceci peut être dans l'objectif, d'intégrer une source Web dans un médiateur d'informations, d'extraire d'un site les informations qu'il contient dans l'objectif de construire une base de données structurée, *etc.* La construction d'adaptateurs permettant d'extraire de pages HTML les données qu'elles contiennent, ne répond que partiellement à ce problème. En effet, accéder à une source de données en ligne telle que Amazon, nécessite d'une part de remplir un formulaire, de soumettre le formulaire rempli, de télécharger une première page de résultats, d'en extraire les données par l'application d'un adaptateur, et de parcourir les autres pages résultats en extrayant les données qu'elles contiennent. Ce simple exemple, montre que l'application d'un adaptateur, n'est qu'une étape du processus même si elle est essentielle. Il est évident que l'étape la plus difficile à mettre en oeuvre dans le processus est la tâche d'extraction. Toutefois décrire le processus lui-même n'est pas une tâche évidente. Ceci est d'autant plus vrai, qu'il n'est pas immédiat qu'un schéma classique puisse émerger pour l'ensemble des sources du Web.

Ainsi un second problème est celui de la construction d'applications accédant à des sources de données Web. Ceci nécessite de mettre en avant les similarités et les différences pouvant apparaître dans différentes sources. La mise en avant des similarités pourra alors permettre la construction d'un formalisme permettant de décrire simplement des applications Web. Un tel formalisme devra notamment être suffisamment flexible pour prendre en compte la rapide évolution du Web. En effet, il arrive souvent que le format de données utilisé par une source Web soit modifié (ce qui nécessite le remplacement de l'adaptateur). Il en est de même pour le parcours menant à l'information souhaitée.

CHAPITRE 2

Extraction par généralisation de contextes

Dans le chapitre précédent nous avons vu différentes méthodes permettant d'extraire des informations d'un ensemble de pages. Nous avons mis en avant les limites rencontrées : l'étiquetage exhaustif d'un ensemble de pages pour chacune des sources desquelles des données doivent être extraites, l'analyse structurelle du document qui, bien que non supervisée, ne permet pas d'extraire d'information à l'intérieur de parties textuelles et la nécessité d'un post-traitement permettant d'associer une sémantique aux données extraites. De plus, bien que les méthodes à base de connaissances permettent le traitement de plusieurs sources avec un unique adaptateur, une analyse de chacune des sources de données est nécessaire et les adaptateurs obtenus ne permettent pas toujours d'extraire toute l'information souhaitée.

Dans ce chapitre nous proposons une nouvelle méthode permettant de répondre au problème d'extraction d'information à partir du Web. Tout comme les méthodes inductives, la méthode que nous proposons s'appuie sur un ensemble d'exemples. Toutefois ces exemples ne sont pas des *pages étiquetées* dans lesquelles les informations à extraire sont marquées mais des *instances* d'une relation que l'utilisateur cherche à extraire. La relation à extraire constitue le besoin d'information de l'utilisateur. Ce dernier l'exprime simplement en donnant quelques instances exemples. Dans cette méthode, les exemples et les résultats extraits s'expriment de la même manière : un ensemble de tuples d'une relation. D'autre part les exemples sont détachés des pages, ce qui les rend indépendants du format dans lequel ils sont présentés. De plus notre méthode ne fait pas l'hypothèse que toutes les instances contenues dans les pages exemples ont été données. Ces trois caractéristiques permettent d'avoir une méthode qui, simplifie la tâche de l'utilisateur dans la construction d'un adaptateur en ne lui demandant de donner que quelques instances exemples et permet l'automatisation de la reconstruction d'un adaptateur lorsque la source change de format mais pas de contenu.

Considérons, par exemple, que l'utilisateur cherche à construire un adaptateur pour une source telle que `amazon.fr`. L'objectif de l'utilisateur est que l'adaptateur permette d'extraire une liste de DVD des pages résultant d'une requête sur `amazon.fr`. Les informations souhaitées sont le titre des DVD, l'acteur principal du film, ainsi que le prix du DVD. La relation à extraire est alors la relation `dvd(titre, acteur, prix)`. Cette relation est simplement exprimée par l'utilisateur en donnant quelques instances exemples. La figure 2.1 donne un exemple de page résultat issue d'une requête demandant les DVD où "Depardieu" apparaît comme acteur. Sur cette page trois instances que l'utilisateur souhaite extraire sont entièrement visibles :

- ("Tais-toi", "Jean Reno", "24,98")
- ("Astérix & Obélix : Mission Cléopâtre (Edition Simple)", "Christian Clavier", "12,99")

- ("Cyrano de Bergerac", "Gérard Depardieu", "12,99")

L'adaptateur qui doit être construit doit permettre d'extraire ces trois instances ainsi que toute les instances ayant une présentation similaire, par exemple :

- ("La Chèvre", "Pierre Richard", "18,50")



Figure 2.1 – Première page de résultats de la requête "Depardieu" sur Amazon.fr

Dans l'approche que nous proposons, l'utilisateur construit son adaptateur à partir d'un ensemble de pages issues d'une requête donnée et de quelques instances exemples contenues dans ces pages. Aucun étiquetage n'est nécessaire. Les occurrences des instances sont recherchées dans les documents donnant une certaine indépendance des instances vis à vis des documents. En effet, si la source devait changer de format, ce qui est fréquent sur le Web, les mêmes instances pourraient être réutilisées. L'ensemble de page nécessaires serait simplement obtenu en réutilisant la même requête qui, lorsque la source est la même génère le même contenu, même s'il est formaté différemment. Une fois que les occurrences des instances ont été trouvées dans l'ensemble des documents, leurs contextes sont extraits et généralisés afin d'obtenir des motifs généraux permettant d'extraire de nouvelles instances.

2.1 Notions

Dans notre approche du problème d'extraction d'information, différentes notions entrent en jeu. Il s'agit principalement de manipulations des chaînes de symboles qui sont soit des documents HTML soit des sous-chaînes issues de documents HTML. Soit Σ un alphabet (*i.e.* un ensemble de symboles). Nous notons Σ^* l'ensemble des chaînes qu'il est possible de construire sur cet alphabet.

Les données que nous traitons sont des documents issus du Web. D'un point de vue purement formel, un document n'est rien d'autre qu'une chaîne de symboles construite sur un alphabet Σ donné. Toutefois,

les documents qui nous intéressent sont des documents issus du Web. Le format le plus commun qui se trouve sur le Web est HTML. Les documents au format HTML sont composés de texte et des balises entourant certaines parties textuelles et permettant de donner des informations sur la structure du document, sur la présentation du texte, etc. Ainsi un document HTML peut, à la place d'être vu comme une simple chaîne de caractères, être vu comme une séquence de symboles plus complexes pouvant être soit des parties textuelles et soit des balises. Nous pouvons alors distinguer deux alphabets : l'alphabet des symboles textuels Σ_t et l'alphabet des balises HTML Σ_b . Dans ce cas, l'alphabet sur lequel est construit un document HTML est l'alphabet $\Sigma = \Sigma_t \cup \Sigma_b$. Nous nous intéressons ici, uniquement aux données textuelles apparaissant dans les documents HTML. Ainsi les données telles que les images et les liens sont simplement interprétées comme des balises.

Définition 2.1 (Document). *Un document d est un élément particulier de Σ^* . Dans le cas d'un document HTML, l'ensemble Σ peut être décomposé en deux parties : les symboles textuels Σ_t et les symboles de balises Σ_b .*

Une source de données, dans le contexte du Web, se définit comme un ensemble de documents. Ainsi, formellement, une source est un sous-ensemble de Σ^* .

Définition 2.2 (Source). *Une source de données est un ensemble de documents, i.e. un sous-ensemble de Σ^* .*

Les données à extraire d'un document étant des sous-chaînes de ce document, le domaine des valeurs des informations extraites est l'ensemble Σ^* . Ainsi, les relations que cherche à extraire l'utilisateur sont des relations construites sur le domaine Σ^* .

Définition 2.3 (Relation). *Une relation n -aire est un sous-ensemble de $(\Sigma^*)^n$.*

Étant donné une relation, chaque élément appartenant à cette relation est une instance.

Définition 2.4 (Instance). *Un élément t d'une relation n -aire s'appelle une instance.*

Un adaptateur peut alors être défini comme une fonction qui, à partir d'un document $d \in \Sigma^*$, permet d'extraire des tuples de $(\Sigma^*)^n$.

Définition 2.5 (Adaptateur). *Étant donné un alphabet Σ , un adaptateur est une fonction $w : \Sigma^* \rightarrow (\Sigma^*)^n$ permettant d'extraire un ensemble de tuples.*

Lorsqu'un adaptateur est appliqué à un ensemble de documents $S \subset \Sigma^*$, il permet de construire un ensemble de tuples formant une relation $R \subset (\Sigma^*)^n$. Chaque instance $t = \langle v_1, \dots, v_n \rangle$ extraite est composé de n éléments $v_1, \dots, v_n$ de Σ^* . Chacun de ces éléments est appelé une valeur.

Toute apparition de v dans un document d est appelée occurrence de v dans d . Dans un même document v peut avoir plusieurs occurrences. Une occurrence o donnée peut être représentée par un couple (b, e) où b représente la position de début de o dans d et e représente la position de fin de o dans d . La notion d'occurrence d'une valeur peut être étendue à celle d'occurrence d'un tuple de valeurs, i.e. une instance. Dans un document d , une instance $t = (v_1, \dots, v_n) \in R$ peut avoir plusieurs occurrences $s = (s_1, \dots, s_n)$ où chaque s_i est un couple (b_i, e_i) référençant une occurrence de la chaîne v_i dans le document d .

Définition 2.6 (Occurrence). *Une occurrence s dans un document d d'une instance $t = (v_1, \dots, v_n)$ est un tuple $(s_1, \dots, s_n)$ ou chaque élément est un couple de positions (b_i, e_i) tel que pour tout i , $v_i = d[b_i, e_i]$.*

Dans un document, une occurrence a un contexte, par exemple les chaînes qui se trouvent avant ou après l'occurrence ou entre deux valeurs. Ce qui caractérise les instances d'une même relation est le contexte dans lequel apparaissent leurs occurrences. L'hypothèse de base est que deux instances d'une même relation apparaissant dans un document d'une même source risquent très fortement d'apparaître dans des contextes similaires. Ceci est d'autant plus vrai dans le cas de documents Web. Dans une même source, deux instances seront présentées visuellement à l'utilisateur de manière similaire.

Formellement un contexte peut être vu comme une fonction associant à un document un ensemble d'occurrences. Notons O l'ensemble des occurrences et $\mathcal{P}(O)$ l'ensemble des partitions de O .

Définition 2.7 (Contexte). *Un contexte est une fonction $c : \Sigma^* \rightarrow \mathcal{P}(O)$, qui associe à un document $d \in \Sigma^*$ un ensemble $o \in \mathcal{P}(O)$ d'occurrences.*

La description d'un contexte d'apparition se fait à l'aide d'un motif. Un motif doit contenir l'ensemble des informations contextuelles permettant de retrouver les occurrences se trouvant dans le contexte décrit. Les motifs s'appuient sur les sous-chaînes se trouvant avant, entre et après les valeurs des occurrences qu'il permet d'extraire. Ainsi, un motif permettant d'extraire des occurrences de k valeurs sera composé de $k + 1$ parties. Dans ce cas, on parlera de k -motif. La figure 2.2 donne une vision schématique d'une occurrence $s = \langle s_1, \dots, s_k \rangle$ dans un document. Le document correspond à la totalité de la bande en incluant les s_i . Pour qu'un k -motif $m = \langle p_0, \dots, p_k \rangle$ permette d'extraire l'occurrence $s = \langle s_1, \dots, s_k \rangle$, chaque p_i doit représenter le bloc sous lesquels il se trouve dans la figure 2.2. Chaque partie a pour objectif la reconnaissance d'une sous-chaîne : celle se trouvant avant la première valeur, entre deux valeurs, ou après la dernière valeur.

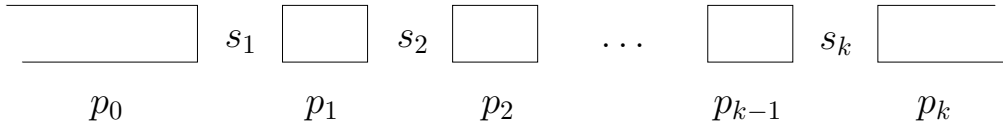


Figure 2.2 – Parties d'un motifs

Définition 2.8 (k -Motif). *Un k -motif est un $(k + 1)$ -uplet $\langle p_0, \dots, p_k \rangle$ où chaque p_i représente une partie d'un motif et où une partie est une chaîne de symboles d'un alphabet $\Sigma_t \cup \Sigma_b \cup \{\star\}$ où $\star$ est un symbole joker qui représente n'importe quel élément de Σ_t .*

L'objectif d'un motif est l'extraction d'occurrences d'un document. Étant donné un k -motif $m = \langle p_0, \dots, p_k \rangle$, l'appliquer à un document $d \in \Sigma^*$ consiste à rechercher successivement chacun des p_i dans l'ordre. On commence par rechercher p_0 dans d , à partir de la fin de l'occurrence de p_0 on recherche p_1 , etc. Si l'un des p_i n'est pas trouvé à partir de la position de fin de p_{i-1} alors la reconnaissance du motif échoue. En cas de succès, l'instance extraite correspond aux valeurs entre les p_i (cf. les s_i de la figure 2.2).

Tous les k -motifs qu'il est possible de construire étant donné un alphabet $\Sigma \cup \{\star\}$ ne sont pas intéressants. Notamment, un motif ayant $\star$ comme symbole débutant ou terminant une partie, ne permet pas une extraction correcte. En effet, il n'est pas possible de déterminer où s'arrête la partie et où commence la valeur de l'occurrence. Afin d'éviter ces motifs nous pouvons définir une notion de validité d'un motif.

Définition 2.9 (k -motif valide). *Un k -motif $m = \langle p_0, \dots, p_k \rangle$ est valide si et seulement si pour tout i , $0 \leq i \leq k$, le symbole $\star$ n'est ni un préfixe, ni un suffixe de p_i .*

Le problème de construction d'un adaptateur dans ce contexte se pose comme suit. Soit R une relation, E un ensemble d'instances exemples de R , S un ensemble de documents. L'objectif est de construire un ensemble de motifs M permettant d'extraire toutes les instances de R ayant une occurrence dans les documents de S . L'expression des besoins de l'utilisateur se fait bien *a priori* par la déclaration d'une relation au travers de quelques instances exemples.

Définition 2.10 (Adaptateur). *Un adaptateur peut être représenté par un ensemble M de k -motifs.*

Un ensemble de motifs M peut être considéré comme un adaptateur. En effet, il s'agit simplement de construire une fonction w_M qui applique l'ensemble des motifs à son entrée et retourne l'ensemble des tuples de valeurs extraits du document en appliquant les motifs. Construire un adaptateur revient donc à déterminer l'ensemble des motifs qui le compose. Ces motifs peuvent être obtenus en généralisant les contextes représentés par des k -motifs.

2.2 Généralisation de contextes

La généralisation des contextes représentés par des k -motifs se fait à plusieurs niveaux. Un k -motif est décomposable en $k + 1$ parties qui elles-mêmes sont décomposables en symboles. Généraliser deux k -motifs entre eux consiste à généraliser, deux à deux, les paires de parties de mêmes positions dans les deux motifs. Ensuite, généraliser deux parties, qui sont chacune une suite de symboles, consiste à construire une nouvelle suite de symboles permettant de reconnaître les mêmes sous-chaînes que les deux parties initiales. Ceci nécessite de pouvoir généraliser des symboles entre eux deux à deux. Ainsi cette généralisation est hiérarchique.

2.2.1 Généralisation des symboles

Dans le cas de documents HTML, nous avons vu qu'il y avait deux ensembles de symboles : les symboles textuels et les symboles de balise. Ainsi, pour deux symboles donnés s_1 et s_2 , il y a quatre cas de généralisations possibles :

- s_1 est de type balise et s_2 de type texte ;
- s_1 est de type texte et s_2 est de type balise ;
- s_1 et s_2 sont de type balise ;
- s_1 et s_2 sont de type texte.

Dans les deux premiers cas, la généralisation n'a pas de sens. Afin qu'elle soit tout de même définie dans ce cas, nous considérerons qu'elle résulte en un symbole d'échec $\top$. Les balises d'un document HTML sont les parties représentant la structure du document. Ainsi, elles jouent un rôle important dans la reconnaissance d'un format donné. Généraliser deux balises qui ne sont pas identiques supprimerait des informations structurelles, ce qui n'est pas souhaitable. Dans ce cas, la généralisation aboutira au symbole $\top$. Lorsque, les balises sont les mêmes la généralisation renverra cette même balise comme résultat.

A l'inverse des balises, les parties textuelles ne jouent, *a priori*, pas de rôle structurel. Par contre, il arrive que du texte puisse être utile pour délimiter une valeur. Par exemple, un prix pourra parfois être précédé de la chaîne de texte "Prix :" ou suivi de "EUR". De telles chaînes jouent alors un rôle structurel et il est nécessaire de les conserver pour une extraction correcte des informations. Il est important de ne pas généraliser une chaîne de texte précédant ou suivant immédiatement une valeur, sinon cette valeur ne pourra plus être extraite correctement. Dans un ce cas, en effet, une chaîne précédant ou suivant une

valeur joue un rôle de délimiteur. Par exemple, si "Prix : 35,20 EUR" est une sous-chaîne du document et que l'on cherche à extraire "35,20" pour déterminer le début de ce prix, il nous faudra délimiter le début de "35,20" avec un suffixe non vide de "Prix : ". Ainsi on ne peut pas généraliser ": " car cette partie de la chaîne "Prix : " est essentielle pour déterminer le début de "35,20". La généralisation de "Prix :" avec "Prix" est également impossible car aucun suffixe commun jouant le rôle de délimiteur n'existe. De manière générale, la généralisation de deux symboles précédant une valeur doit conserver un suffixe commun aux deux symboles. De la même façon, la généralisation de deux symboles suivant une valeur doit conserver un préfixe commun.

Nous avons ainsi trois généralisations possibles pour les symboles textuels : la généralisation de deux symboles précédant une valeur (généralisation suffixe) la généralisation de deux symboles suivant une valeur (généralisation préfixe) et la généralisation de deux symboles ne se trouvant ni avant ni après deux valeurs (généralisation neutre).

Notons $\cdot$ l'opérateur de concaténation sur les chaînes. Étant donnée une chaîne s , nous notons $\text{prefixes}(s)$ l'ensemble de ses préfixes et $\text{suffixes}(s)$ l'ensemble des ses suffixes.

Définition 2.11 (Généralisation suffixe). *La généralisation suffixe de deux symboles s_1 et s_2 est égale à*

$$gs(s_1, s_2) = \begin{cases} s_1 & \text{si } s_1 = s_2 \\ \star \cdot s & \text{si } s = \arg\max_{s' \in \text{suffixes}(s_1) \cap \text{suffixes}(s_2)} (|s'|) \wedge |s| > 0 \\ \top & \text{autrement} \end{cases} \quad (2.1)$$

Définition 2.12 (Généralisation préfixe). *La généralisation préfixe de deux symboles s_1 et s_2 est égale à*

$$gp(s_1, s_2) = \begin{cases} s_1 & \text{si } s_1 = s_2 \\ p \cdot \star & \text{si } p = \arg\max_{p' \in \text{prefixes}(s_1) \cap \text{prefixes}(s_2)} (|p'|) \wedge |p| > 0 \\ \top & \text{autrement} \end{cases} \quad (2.2)$$

Définition 2.13 (Généralisation neutre). *La généralisation neutre de deux symboles s_1 et s_2 est égale à*

$$gn(s_1, s_2) = \begin{cases} s_1 & \text{si } s_1 = s_2 \\ \star & \text{autrement} \end{cases} \quad (2.3)$$

2.2.2 Généralisation des parties

La généralisation de deux parties est dépendante de leur position dans leurs motifs respectifs. En effet, qu'il s'agisse de parties en tête de motifs, de parties en queue de motifs ou de parties au milieu de motifs, elles ne jouent pas le même rôle. Une partie en tête permet de déterminer le début de la première valeur des tuples à extraire, la partie en queue permet de déterminer la fin de la dernière valeur des tuples alors que les parties au milieu permettent de déterminer la fin d'une valeur *et* le début de la suivante.

Dans le cas de la généralisation de parties au milieu, seules deux parties de même longueur (*i.e.* ayant le même nombre de symboles) peuvent être généralisées. Lorsque deux parties n'ont pas la même longueur, c'est qu'elles ne possèdent pas le même nombre de symboles balises et ainsi, qu'elles ont une structure différente. Dans ce cas, les motifs auxquels appartiennent les parties sont des motifs représentant des formats différents. Par contre, dans le cas de deux parties p_1 et p_2 ayant la même longueur, généraliser p_1 et p_2 consiste à construire une partie p composée de la généralisation des symboles de même position dans chacune des deux parties initiales. Cependant les premiers et derniers symboles sont

généralisés en utilisant respectivement gs et gp alors que les autres symboles sont généralisés en utilisant gn . L'objectif est que dans tous les cas le symbole $\star$ ne se retrouve pas en tout début de partie ou tout à la fin de la partie. Cependant, afin de simplifier la présentation, dans tous ces cas, nous noterons $g(s_1, s_2)$ la généralisation de deux symboles s_1 et s_2 . Par ailleurs, lorsque la généralisation d'une des paires de symboles échoue (renvoie $\top$), la généralisation de la partie échoue elle aussi (renvoie $\top$). Dans la suite, étant donnée une partie p , $p[i]$ dénote le symbole de position i dans p , $|p|$ dénote la longueur de la partie (i.e. le nombre de symboles dont elle est composée).

Définition 2.14 (Généralisation milieu). *La généralisation milieu de deux parties p_1 et p_2 est égale à*

$$generalise(p_1, p_2) = \begin{cases} \top & \text{si } |p_1| \neq |p_2| \\ \top & \text{si } |p_1| = |p_2| \wedge \\ & \exists j \in [1, |p_1|] / g(p_1[j], p_2[j]) = \top \\ g(p_1[1], p_2[1]) \cdot \dots \cdot g(p_1[|p_1|], p_2[|p_1|]) & \text{autrement} \end{cases} \quad (2.4)$$

Dans la reconnaissance d'une position de début par une partie, le rôle le plus important est joué par les symboles en fin de parties et notamment le dernier. Ainsi, lors de la généralisation d'une partie en tête, il est nécessaire de ne conserver qu'une généralisation d'un des suffixes de la partie. On peut alors définir la généralisation tête de deux parties p_1 et p_2 comme suit.

Définition 2.15 (Généralisation tête). *La généralisation tête de deux parties p_1 et p_2 est le plus long p , $|p| \neq 0$, $p \neq \top$ tel que :*

$$\exists p'_1 \in \text{suffixes}(p_1), \exists p'_2 \in \text{suffixes}(p_2), p = generalise(p'_1, p'_2) \quad (2.5)$$

De manière analogue, lors de la généralisation de deux parties en queue, il sera suffisant que la partie généralisant les deux parties initiales ne soit une généralisation que d'un préfixe de chacune des parties. Ainsi on peut définir la généralisation queue de deux parties comme suit.

Définition 2.16 (Généralisation queue). *La généralisation queue de deux parties p_1 et p_2 est le plus long p , $|p| \neq 0$, $p \neq \top$ tel que :*

$$\exists p'_1 \in \text{prefixes}(p_1), \exists p'_2 \in \text{prefixes}(p_2), p = generalise(p'_1, p'_2) \quad (2.6)$$

2.2.3 Généralisation des motifs

La généralisation de deux k -motifs $m = \langle p_0, \dots, p_k \rangle$ et $m' = \langle p'_0, \dots, p'_k \rangle$ consiste à généraliser deux à deux les parties de même position. Dans le cas où la généralisation de deux parties p_i et p'_i échoue (leur généralisation est égale à $\top$), la généralisation obtenue pour m et m' est aussi $\top$. Sinon, la généralisation obtenue est un nouveau k -motif $m^g = \langle p_1^g, \dots, p_k^g \rangle$. Chacune des parties p_i^g est obtenue par la généralisation de p_i et p'_i .

2.2.4 Propriétés de la généralisation

La généralisation que nous venons de définir possède plusieurs propriétés intéressantes qui seront exploitées dans les algorithmes de notre méthode. Notons $g(x, y)$ la généralisation de deux motifs x et y . g est alors réflexive, commutative et associative.

Proposition 1 (Réflexivité de g). Soit m un motif. Nous avons alors $g(m, m) = m$.

Proposition 2 (Commutativité de g). Soit m_1 et m_2 deux motifs. Nous avons alors $g(m_1, m_2) = g(m_2, m_1)$.

Proposition 3 (Associativité de g). Soit m_1, m_2 et m_3 trois motifs. Nous avons $g(m_1, g(m_2, m_3)) = g(g(m_1, m_2), m_3)$.

Les motifs et les parties ont la particularité d'être des n -uplets d'éléments d'un même ensemble. Les généralisations de parties et de motifs ont la particularité d'être construit à partir d'un opérateur s'appliquant dans les deux cas à des paires d'éléments de même position. Ainsi, à partir d'un opérateur binaire $\circ$ s'appliquant sur deux éléments d'un domaine D , nous avons induit un n -opérateur $\circ^n$ s'appliquant sur deux éléments du domaine D^n de tel sorte que :

$$\langle a_1, \dots, a_n \rangle \circ^n \langle b_1, \dots, b_n \rangle = \langle a_1 \circ b_1, \dots, a_n \circ b_n \rangle \quad (2.7)$$

où $\forall i, a_i \in D$ et $\forall i, b_i \in D$.

Soit $\circ$ un opérateur binaire sur D et $\circ^n$ le n -opérateur binaire induit à partir de $\circ$ sur D^n .

Proposition 4. Si $\circ$ est un opérateur réflexif alors $\circ^n$ est réflexif.

Démonstration. Soit $\langle a_1, \dots, a_n \rangle \in D^n$. Soit $\langle b_1, \dots, b_n \rangle \in D^n$. Nous avons :

$$\langle a_1, \dots, a_n \rangle \circ^n \langle a_1, \dots, a_n \rangle = \langle a_1 \circ a_1, \dots, a_n \circ a_n \rangle \quad (2.8)$$

$$= \langle a_1, \dots, a_n \rangle \quad (2.9)$$

□

Proposition 5. Si $\circ$ est commutatif alors $\circ^n$ est commutatif.

Démonstration. Soit $\langle a_1, \dots, a_n \rangle \in D^n$. Soit $\langle b_1, \dots, b_n \rangle \in D^n$. Nous avons :

$$\langle a_1, \dots, a_n \rangle \circ^n \langle b_1, \dots, b_n \rangle = \langle a_1 \circ b_1, \dots, a_n \circ b_n \rangle \quad (2.10)$$

$$= \langle b_1 \circ a_1, \dots, b_n \circ a_n \rangle \quad (2.11)$$

$$= \langle b_1, \dots, b_n \rangle \circ^n \langle a_1, \dots, a_n \rangle \quad (2.12)$$

□

Proposition 6. Si $\circ$ est associatif alors $\circ^n$ est associatif.

Démonstration. Soit $\langle a_1, \dots, a_n \rangle \in D^n$. Soit $\langle b_1, \dots, b_n \rangle \in D^n$. Soit $\langle c_1, \dots, c_n \rangle \in D^n$. Nous avons :

$$\langle a_1, \dots, a_n \rangle \circ^n (\langle b_1, \dots, b_n \rangle \circ^n \langle c_1, \dots, c_n \rangle) = \langle a_1, \dots, a_n \rangle \circ^n \langle b_1 \circ c_1, \dots, b_n \circ c_n \rangle \quad (2.13)$$

$$= \langle a_1 \circ (b_1 \circ c_1), \dots, a_n \circ (b_n \circ c_n) \rangle \quad (2.14)$$

$$= \langle (a_1 \circ b_1) \circ c_1, \dots, (a_n \circ b_n) \circ c_n \rangle \quad (2.15)$$

$$= \langle a_1 \circ b_1, \dots, a_n \circ b_n \rangle \circ^n \langle c_1, \dots, c_n \rangle \quad (2.16)$$

$$= (\langle a_1, \dots, a_n \rangle \circ^n \langle b_1, \dots, b_n \rangle) \circ^n \langle c_1, \dots, c_n \rangle \quad (2.17)$$

□

La généralisation des parties, lorsqu'elle s'applique à deux parties de même longueur, est un n -opérateur induit à partir de l'opérateur de généralisation des symboles et où n est la longueur des deux parties à généraliser. Étant donné que la généralisation des symboles est réflexive, commutative et associative, la généralisation de deux parties de même longueur est réflexive, associative et commutative. On peut immédiatement en conclure que la généralisation des parties est réflexive (une partie est toujours de même longueur qu'elle-même). D'autre part, dans le cas de deux parties de tailles différentes on obtient $\top$ dans un sens comme dans l'autre conservant ainsi associativité et commutativité.

La généralisation de deux k -motifs est un k -opérateur induit à partir de la généralisation des parties. Comme la généralisation des parties est réflexive, associative et commutative, il en est de même pour la généralisation de deux motifs.

Étant donné les propriétés d'associativité et de commutativité de la généralisation, la généralisation d'un ensemble de motifs est unique. Son calcul ne dépend pas de l'ordre dans lequel sont effectuées les généralisations. Par abus de notation, nous pourrions noter $g(m_1, \dots, m_n)$ la généralisation de n parties $m_1, \dots, m_n$.

Proposition 7. Soit n k -motifs, $m_1, \dots, m_n$. Nous avons alors pour toute permutation σ sur $\{1, \dots, n\}$:

$$g(m_1, \dots, m_n) = g(m_{\sigma(1)}, \dots, m_{\sigma(n)}) \quad (2.18)$$

2.3 Méthode

Nous proposons d'utiliser cette généralisation afin de construire des adaptateurs à partir d'instances exemples. Dans notre méthode, l'utilisateur doit fournir, d'une part, la source de laquelle il souhaite extraire la relation et d'autre part une description de la relation qu'il souhaite extraire. Cette description se fait simplement en donnant quelques instances exemples de la relation.

Par exemple, soit la source D donnée par la figure 2.3. Cette source contient la relation R donnée dans la figure 2.4. Considérons qu'un utilisateur souhaite extraire cette relation. Il lui suffit d'exprimer ses besoins en donnant quelques instances exemples de cette relation, par exemple les deuxième et troisième instances de la relation. La méthode devra alors à partir de ces deux instances exemples construire un motif qui sera capable d'extraire ces exemples ainsi que les trois autres instances de R .

Pour effectuer cette tâche notre méthode se décompose en plusieurs phases :

- Prétraitement de la source de données ;
- Recherche des occurrences des instances exemples ;
- Extraction de leurs contextes ;
- Généralisation des contextes ;
- Extraction des occurrences à l'aide des contextes généralisés.

La figure 2.5 décrit ce processus. Dans cette figure les losanges correspondent à des données, alors que les carrés correspondent à des fonctions. Le processus se définit ainsi par les fonctions suivantes :

- *modéliser* qui, à partir de la chaîne brute d'un document, en construit un modèle abstrait (prétraitement) ;
- *rechercher* recherche des occurrences dans le modèle de document ;
- *generer* extrait à partir des occurrences une description de leur contextes : les motifs initiaux ;
- *generaliser* construit des motifs généralisés à partir des motifs initiaux ;
- *extraire* applique les motifs généralisés obtenus au modèle de document permettant d'extraire les instances correspondantes.

```

1 <html><body>
2 <h1>Prix du Distributeur de Café</h1>
3 <table>
4 <tr><td><font color="red">Café</font></td>
5 <td><b>0.40</b> EUR (2.62 F)</td></tr>
6 <tr><td><font color="blue">Chocolat</font></td>
7 <td><b>0.45</b> EUR (2.95F)</td></tr>
8 <tr><td><font color="red">Soda</font></td>
9 <td><b>0.75</b> EUR (4.92 F)</td></tr>
10 <tr><td colspan="2"><hr></td></tr>
11 <tr><td><font color="blue">
12 <a href="gateaux.html">Gateau</a></font>
13 </td>
14 <td><b>0.50</b> EUR (3.28 F)</td></tr>
15 <tr><td>
16 <font color="red">
17 <a href="sandwichs.html">Sandwich</a>
18 </font>
19 </td>
20 <td><b>2.00</b> EUR (13.12 F)</td></tr>
21 </table>
22 </body></html>

```

Figure 2.3 – Un exemple de document source

Type	PriceEUR	PriceFF
Café	0.40	2.62
Chocolat	0.45	2.95
Soda	0.75	4.92
Gateau	0.50	3.28
Sandwich	2.00	13.12

Figure 2.4 – Relation à extraire

2.3.1 Prétraitement de la source

Une première phase consiste à appliquer à la source un prétraitement. Ceci consiste à analyser le document et en construire un modèle facilitant les traitements. Nous considérons qu'un document est simplement une succession de symboles. Les symboles considérés sont de deux types : (1) des balises HTML ou (2) des caractères se trouvant en dehors des balises. Les balises sont considérées comme des symboles uniques et sont codées. Les balises sont non seulement distinguées par leur nom mais leurs attributs sont également pris en compte. Ainsi les chaînes `<td>` et `<td colspan="2">` sont considérées comme deux symboles distincts.

Par ailleurs, les balises sont divisées en deux classes : les balises de haut niveau et les balises de bas niveau. Les balises de bas niveau sont les balises, telles que les balises `<i>` ou `<b>`, qui permettent de modifier l'apparence d'une sous-chaîne de texte mais ne contenant pas d'information structurale. Les balises de haut niveau, telles que `<p>` ou `<td>`, donne de l'information sur la structure de la présentation des données. Étant donné que l'information contextuelle est contenue dans la structure de présentation, nous pouvons nous passer des balises de bas niveau. Les usages du langage HTML ont fait que certaines

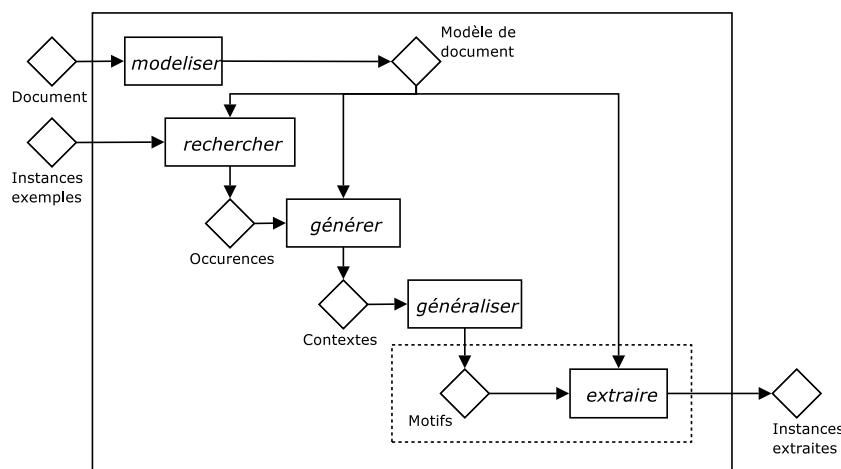


Figure 2.5 – Description du processus d'extraction d'information basé sur la généralisation des contextes

balises de bas niveau peuvent être utilisées comme indicateurs de structure. Dans ce cas, il peut être intéressant de conserver certaines de ces balises. Par défaut, la méthode élimine les balises de bas niveau, mais lorsque la méthode ne trouve pas de motifs permettant d'extraire les résultats, l'utilisateur a la possibilité de conserver certaines d'entre elles.

Ainsi, le modèle de document choisi est obtenu à partir de la chaîne de symboles brute du document d'origine en appliquant un nettoyage des espaces qui se suivent (regroupement en un seul espace), en supprimant les balises de bas niveau et en codant chacune des balises avec ses attributs en un nouveau symbole unique. L'ensemble de ces traitements permet d'obtenir un document codé. La version codée du document de la figure 2.3 est donnée dans la figure 2.6. Dans cette figure, les chiffres en gras correspondent aux balises encodées. Par exemple **1**, **8** et **10** correspondent respectivement aux balises `<html>`, `</td>`, and `<td colspan="2">` du document d'origine. La table 2.1 donne les correspondances entre les codes et les balises.

1-2-3 Prix du Distributeur de Café **4-5-6-7** Café **8-7** 0.40 EUR (2.62 F) **8-9-6-7** Chocolat **8-7** 0.45 EUR (2.95F)
8-9-6-7 Soda **8-7** 0.75 EUR (4.92 F) **8-9-6-10-11-8-9-6-7** Gateau **8-7** 0.50 EUR (3.28 F) **8-9-6-7** Sandwich **8-7**
 2.00 EUR (13.12 F)**8-9-12-13-14**

Figure 2.6 – Exemple de document codé

Code	Balise	Code	Balise
1	<code><html></code>	8	<code></td></code>
2	<code><body></code>	9	<code></tr></code>
3	<code><h1></code>	10	<code><td colspan="2"></code>
4	<code></h1></code>	11	<code><hr></code>
5	<code><table></code>	12	<code></table></code>
6	<code><tr></code>	13	<code></body></code>
7	<code><td></code>	14	<code></html></code>

Table 2.1 – Table des correspondances des codes et des balises pour l'exemple

2.3.2 Recherche des occurrences

L'étape suivante consiste à extraire les occurrences des instances exemples de la relation. Celles-ci sont recherchées dans le document encodé en cherchant successivement chacune des valeurs des instances. Nous considérons qu'une occurrence est représentée par un tuple de couples d'indices $s = ((b_1, e_1), \dots, (b_n, e_n))$ où chaque couple (b_i, e_i) correspond aux indices de début et de fin de la valeur v_i de l'instance correspondante. Pour notre exemple, supposons que les instances exemples sont celles correspondant aux produits "Café", "Soda" et "Gateau". Nous aurons alors pour la première, deux occurrences (en effet il y a deux occurrences de la chaîne "Café") et pour chacune des deux autres instances, une unique occurrence. La deuxième occurrence de la première instance est le tuple $((36, 42), (44, 48), (55, 59))$.

On peut remarquer que le fait d'extraire deux occurrences pour la première instance est discutable car, à l'évidence, c'est la deuxième qui nous intéresse. En effet, l'occurrence de la valeur "Café" de la première faisant partie de la chaîne "Prix du Distributeur de Café", ne correspond pas à une occurrence d'un attribut *produit* de la relation que nous cherchons à extraire. Cependant la valeur "Café" appartient bien à cet attribut dans les exemples et extraire une instance en utilisant cette valeur ne génère pas d'instances n'appartenant pas à la relation recherchée.

En pratique, une telle "fausse" occurrence donnera lieu à un motif ne pouvant généralement pas se généraliser avec les motifs issus des autres instances exemples. Ainsi le motif en question aura un contexte très spécifique ne permettant de n'extraire que l'instance de laquelle il est issu. Sa prise en compte ne donnera pas lieu à l'extraction d'instances erronées.

2.3.2.1 Heuristiques de recherches

La recherche de toutes les occurrences d'un tuple de valeurs $t = (v_1, \dots, v_n)$ consiste d'abord à rechercher toutes les occurrences de chacun des v_i . Soit $occ(v_i)$ l'ensemble des occurrences de v_i . Les occurrences du tuple t correspondent au produit cartésien $occ(v_1) \times \dots \times occ(v_n)$. Celles-ci peuvent s'avérer nombreuses si l'une des valeurs a plusieurs occurrences dans la page. Par exemple dans la figure 2.3, nous avons vu qu'il y avait deux occurrences de la chaîne "Café" qui engendraient deux occurrences pour l'instance ("Café", "0.40", "2.62") : la première est $((28, 32), (44, 48), (55, 59))$ et la seconde est $((36, 42), (44, 48), (55, 59))$. Seule la deuxième correspond réellement à une instance à extraire. Par ailleurs, si le prix associé à "Chocolat" avait aussi été "0.40" (et donc un prix en franc de "2.62", il y aurait eu $2 \times 2 \times 2$ soit 8 occurrences de l'instance ("Café", "0.40", "2.62"). En effet, dans toute la page il y aurait eu 2 occurrences de "Café", 2 occurrences de "0.40" et deux occurrences de "2.62". Chaque combinaison est un candidat possible comme occurrence réel de "Café", "0.40", "2.62". Cependant, celle qui a la plus grande probabilité d'être l'occurrence correspondant réellement aux données à extraire est celle dont les valeurs sont les plus proches. Ainsi, certaines heuristiques et contraintes sur les valeurs des exemples permettent de réduire le nombre d'occurrences générées pour les valeurs d'un tuple. Par exemple, il est possible de définir une notion de proximité des valeurs et de considérer que plus les valeurs sont proches, plus il est probable qu'elles appartiennent à un même tuple.

Ainsi une première heuristique consiste à ne considérer que les occurrences où les valeurs sont "proches" les unes des autres. Ceci peut être obtenu en obligeant deux valeurs successive d'une même occurrence à apparaître dans une fenêtre de taille fixée. Dans notre méthode nous exploitons cette heuristique en ne considérant comme occurrences que celles dont deux valeurs successives figurent dans une fenêtre de taille donnée : en pratique une fenêtre de taille 300 est suffisante dans la majorité des cas.

Dans la plupart des cas, l'ordre des données fournies par l'utilisateur est le même que celui dans le-

quel elles sont présentées dans le document. Ainsi nous imposons que l'ordre des valeurs des occurrences soit le même que celui des instances. Ceci permet d'effectuer la recherche des valeurs des instances dans le document dans l'ordre où elles apparaissent dans l'instance, réduisant d'autant la complexité de la recherche d'une occurrence.

2.3.3 Extraction des contextes

Contextes de ('Café', '0.40', '2.96')	
m_1	1-2-3Prix du Distributeur de ☐ 4-5-6-7Café8-7 ☐ EUR (☐ F)8-9-6-7Soda8-7...
m_2	1-2-3Prix du Distributeur de Café4-5-6-7 ☐ 8-7 ☐ EUR (☐ F)8-9-6-7Soda8-7...
Contexte de ('Soda', '0.75', '4.92')	
m_3	...70.40 EUR (2.29 F)8-9-6-7 ☐ 8-7 ☐ EUR (☐ F)8-9-6-10-11-8-9-6-7Gateau8-7...
Contexte de ('Gateau', '0.50', '3.28')	
m_4	...(4.92 F) 8-9-6-10-11-8-9-6-7 ☐ 8-7 ☐ EUR (☐ F) 8-9-6-7 Sandwich 8-7...
Contexte généralisé	
$g(m_2, m_3, m_4)$	6-7 ☐ 8-7 ☐ EUR (☐ F)8-9-6

Figure 2.7 – Généralisation des contextes

À partir de chaque occurrence il est possible d'en décrire le contexte. Celui-ci se construit à partir des sous-chaînes se trouvant avant l'occurrence, entre les valeurs de l'occurrence et après l'occurrence. Un contexte doit permettre de retrouver (au minimum) l'occurrence qui l'a généré. De plus on doit pouvoir généraliser un ensemble de contextes similaires (*i.e.* ayant des parties communes) en un contexte généralisé qui permet de retrouver (au minimum) les occurrences à partir desquelles il a été construit. Dans cette méthode, nous avons choisi d'utiliser comme contexte d'une occurrence un k -motif. Initialement il est constitué de (1) la sous-chaîne du document se trouvant avant l'occurrence, (2) chacune des sous-chaînes se trouvant entre deux valeurs de l'occurrence et dans l'ordre du document, ainsi que (3) la sous-chaîne se trouvant après l'occurrence. La figure 2.7 donne les contextes des occurrences des instances exemples obtenus à partir du document encodé de la figure 2.6. Le symbole ☐ représente les valeurs qui seront extraites lors de la recherche du contexte dans le document. La figure 2.7 donne aussi le contexte généralisé obtenu à partir des trois contextes m_2 , m_3 et m_4 . Ce motif est obtenu par la généralisation des contextes que nous allons maintenant décrire.

2.3.4 Généralisation des contextes

L'extraction des contextes permet d'obtenir un ensemble de motifs initiaux M_0 . Ceux-ci peuvent être généralisés afin de permettre l'extraction de nouvelles instances. Il s'agit d'appliquer à chaque couple de motifs initiaux, la méthode de généralisation préalablement décrite. L'objectif est d'obtenir à partir de M_0 un nouvel ensemble de motifs M contenant l'ensemble des généralisations de M_0 différentes de $\top$. Ceci peut se faire en sélectionnant un motif m dans M_0 et en généralisant m au fur et à mesure avec chaque autre motif m' de M_0 . Lorsque la généralisation de m et m' est possible (*i.e.* $g(m, m') \neq \top$), m est remplacé par la généralisation et m' est supprimé de M_0 . Cette opération est valide car dans les deux cas $g(m, m')$ est plus générale que m et m' . Ainsi, tout ce qui est extrait par m et m' est aussi extrait par $g(m, m')$.

2.3.5 Résumé

En résumé, notre méthode s'appuie sur quatre étapes : un prétraitement de la source de données qui consiste à encoder les balises du document, une recherche des occurrences de l'ensemble des instances exemples, l'extraction des contextes de ces occurrences et finalement leur généralisation. Nous allons maintenant décrire comment implémenter cette méthode en donnant les différents algorithmes que nous avons mis en oeuvre.

2.4 Algorithmes

Dans cette section nous décrivons les différents algorithmes développés pour réaliser les différents traitements. La première étape est l'extraction des motifs initiaux en utilisant les exemples. A la différence des méthodes de construction d'adaptateurs à base de pages étiquetées, ces exemples ne donnent qu'une information sur les valeurs à extraire, sans donner d'indication de la position de ces valeurs.

Procédure *rechercher_occurrence*

Entrées

- Un document d
- Une instance $t = (v_1, \dots, v_n)$
- Une position de début k

Sorties

- Un ensemble P de tuples de positions.

```

fini ← faux
tantque  $k \leftarrow \text{rechercher}(v_1, d, k)$  et non fini faire
     $P' \leftarrow \text{rechercher\_occurrence}(d, (v_2, \dots, v_n), k + |v_1|)$ 
    si  $P' \neq \emptyset$  alors
        pour tout  $p' \in P'$  faire
             $P \cup ((k, k + |v_1|), p'[1], \dots, p'[n - 1])$ 
        fin pour
    sinon
        fini ← vrai
    fin si
fin tantque
retourne  $P$ 

```

Algorithme 2.1 – Algorithme de recherche des occurrences d'une instances

2.4.1 Recherche des occurrences et extraction des contextes

Ainsi la première étape consiste à rechercher l'ensemble des occurrences de chaque instance dans l'ensemble des documents de la source. L'algorithme 2.1 décrit la méthode de recherche de l'ensemble des occurrences d'une instance que nous avons mise en oeuvre. A partir d'un tuple de la relation $t = (v_1, \dots, v_n)$, les occurrences de v_1 sont recherchées dans le document. Pour chaque occurrence trouvée, l'ensemble des occurrences du sous-tuple $(v_2, \dots, v_n)$ sont recherchées de façon récursive à partir de

la position de fin de l'occurrence de v_1 . Chaque tuple résultat est complété en ajoutant le couple des positions de début et de fin de v_1 en tête et formant ainsi une occurrence résultat. Cette procédure fait appel à la fonction $rechercher(s_1, s_2, k)$ qui recherche dans une chaîne s_1 la position de la première occurrence d'une autre chaîne s_2 en commençant à la position k . Ici n'importe quel algorithme classique de recherche d'une chaîne dans une autre peut être utilisé[16]. Dans notre prototype, nous construisons une expression régulière pour chaque instance et l'appliquons aux documents exemples.

Ensuite, à partir de l'ensemble des occurrences de toutes les instances, les motifs initiaux sont construits à partir des contextes des occurrences. A partir d'un tuple d'occurrence $p = ((b_1, e_1), \dots, (b_n, e_n))$ dans un document d de longueur N , un motif est construit. Pour une occurrence de n éléments, un motif est alors un $(n + 1)$ -uplet avec une partie préfixe, $n - 1$ parties milieu est une partie suffixe. Le motif c de p est alors obtenu en prenant $c = (d[0, b_1], d[e_1, b_2], \dots, d[e_{n-1}, b_n], d[e_n, N])$, où $d[i, k]$ dénote la sous-chaîne de d commençant à la position i et se terminant à la position k .

2.4.2 Généralisation des contextes

Procédure *generalise_motif*

Entrées

- Deux motifs $c = (c_1, \dots, c_n)$ et $c' = (c'_1, \dots, c'_n)$

Sorties

- Un motif généralisé $g = (g_1, \dots, g_n)$

si non compatible(c_1, c'_1) **alors**

 return **echec**;

fin si

$g_1 = \text{generalise_tete}(c_1, c'_1)$

pour tout $k \in [2, n - 1]$ **faire**

$g_k = \text{generalise_milieu}(c_k, c'_k)$

fin pour

$g_n = \text{generalise_queue}(c_n, c'_n)$

return g

Algorithme 2.2 – Fonction de généralisation de deux motifs

Les motifs, représentant les contextes des occurrences à partir desquels ils ont été obtenus, ainsi formés sont ensuite généralisés. L'algorithme de généralisation prend en entrée deux motifs (généralisés ou non) et applique la méthode de généralisation décrite précédemment. Lorsque c'est possible (*i.e.* leur généralisation est différente de $\top$), les deux motifs sont remplacés par leur généralisation. Sinon, les deux motifs d'origine sont conservés et pourront éventuellement être généralisés avec d'autres motifs. Étant donné deux motifs à n parties $c_1 = (p_{1,1}, \dots, p_{1,n})$ et $c_2 = (p_{2,1}, \dots, p_{2,n})$, les parties de même indices sont généralisées deux à deux (*i.e.* pour tout i , $p_{1,i}$ est généralisé avec $p_{2,i}$). Les parties sont elles-mêmes divisées en unités lexicales (symbole de balise ou séquence de texte) et généralisées. La figure 2.8 donne une représentation de la décomposition d'un motif.

La fonction de généralisation g des motifs fait appel à une fonction de généralisation des parties g_p qui elle-même fait appel à une fonction de généralisation des symboles textuels g_t . La généralisation de deux motifs à n parties $m_1 = (p_{1,1}, \dots, p_{1,n})$ et $m_2 = (p_{2,1}, \dots, p_{2,n})$ est le résultat de la

Procédure *generalise_milieu***Entrées**

- Deux parties $p = (s_1, \dots, s_n)$ et $p' = (s'_1, \dots, s'_n)$

Sorties

- Une partie généralisée $\hat{s} = (\hat{s}_1, \dots, \hat{s}_n)$

```

pour tout  $k \in [1, n]$  faire
  si  $type(s_k) \neq type(s'_k)$  alors
    retourne echec
  fin si
  si  $type(s_k) = \text{"balise"}$  alors
    si  $s_k = s'_k$  alors
       $\hat{s}_k \leftarrow s_k$ 
    sinon
      retourne echec
    fin si
  sinon
     $\hat{s}_k \leftarrow g_t(s_k, s'_k)$ 
  fin si
fin pour
retourne  $\hat{s}$ 

```

Algorithme 2.3 – Fonction de généralisation de deux parties

généralisation deux à deux des parties au mêmes indices. Si g est la fonction de généralisation alors $g(m_1, m_2) = (g_p(p_{1,1}, p_{2,1}), \dots, g_p(p_{1,n}, p_{2,n}))$. L'algorithme 2.2 décrit la généralisation de deux motifs que nous avons proposée. En pratique, les généralisations des parties en tête (i.e. $p_{1,1}$ et $p_{2,1}$) et en queue (i.e. $p_{1,n}$ et $p_{2,n}$) de motifs sont traitées à part. Ceci est nécessaire pour éliminer des préfixes et suffixes trop spécifiques. Dans le cas d'une partie en tête (resp. en queue), les parties sont alignées à droite (resp. à gauche) et une partie généralisée issue d'un préfixe (resp. suffixe) commun des deux parties initiales est recherchée. Pour obtenir un tel préfixe, les parties sont parcourues simultanément de droite à gauche (resp. de gauche à droite) jusqu'à ce que les symboles courants ne puissent se généraliser entre eux (selon la procédure de généralisation des symboles décrite ci-après). Dans l'objectif d'obtenir des motifs généralisés permettant d'extraire de nouvelles instances, les parties en tête (resp. en queue) d'un motif n'ont besoin de conserver qu'un préfixe (resp. suffixe) commun. L'algorithme 2.3 précise la généralisation de deux parties de milieu (la généralisation des parties en tête et en queue se fait de manière similaire). Étant données deux parties $p = (s_1, \dots, s_n)$ et $p' = (s'_1, \dots, s'_n)$, il construit une partie généralisée $\hat{p} = (\hat{s}_1, \dots, \hat{s}_n)$. Pour ce faire, pour chaque i , il cherche à généraliser s_i avec s'_i selon le type de s_i et de s'_i . La fonction $type(s)$ renvoie pour un symbole donné s son type : balise ou texte. Lorsque s_i et s'_i sont différents, la généralisation de p avec p' échoue. Lorsque s_i et s'_i sont une même balise alors la valeur de $\hat{s}_i$ est aussi cette même balise. En revanche si s_i et s'_i sont des balises différentes, la généralisation de p avec p' échoue. Finalement, lorsque s_i et s'_i sont de type texte, on applique la généralisation de deux parties textuelles g_t .

Procédure apprendre_motifs

Entrées

- un ensemble E_R d'exemples de la relation R
- un ensemble de documents D desquels extraire une partie de R

Sorties

- un ensemble M de motifs

$M \leftarrow \emptyset$ {Extraction des contextes}

pour tout $d \in D$ **faire**

pour tout $e \in E_R$ **faire**

$M \leftarrow M \cup \mathcal{C}_d(e)$

fin pour

fin pour

{Généralisation des contextes et construction des motifs}

pour tout $c_1 \in M$ **faire**

pour tout $c_2 \in M, c_1 \neq c_2$ **faire**

$c \leftarrow \mathcal{G}(c_1, c_2)$

si $c \neq *$ **alors**

$C \leftarrow M \setminus \{c_1, c_2\}$

$C \leftarrow M \cup \{c\}$

fin si

fin pour

fin pour

Algorithme 2.4 – Extraction de contextes et construction d'un motif

Procédure extraire**Entrées**

- D le document duquel extraire les instances
- M un ensemble de motifs
- f la fonction de codage utilisée par la construction des motifs

Sorties

- Un ensemble d'instances I

```

 $I \leftarrow \emptyset$ 
{Encodage du document}
Soit  $D_e \leftarrow f(D)$ 
{Application des motifs}
pour tout  $m \in M$  faire
  Soit  $(m_1, \dots, m_n) = \text{parties}(m)$ 
  Soit  $trouve \leftarrow \text{vrai}$ 
  tantque  $trouve$  faire
    Soit  $i \leftarrow 1$ 
    tantque  $trouve$  et  $i \geq n$  faire
      Soit  $(d, f) = \text{rechercher}(D, m_i)$ 
       $trouve \leftarrow d \geq 0$ 
      si  $trouve$  alors
        Soit  $t_i \leftarrow D_e[d, f]$ 
      fin si
       $i \leftarrow i + 1$ 
    fin tantque
    si  $trouve$  alors
       $I \leftarrow I \cup (t_1, \dots, t_n)$ 
    fin si
  fin tantque
fin pour
retourne  $I$ 

```

Algorithme 2.5 – Application de l'adaptateur généré à un document

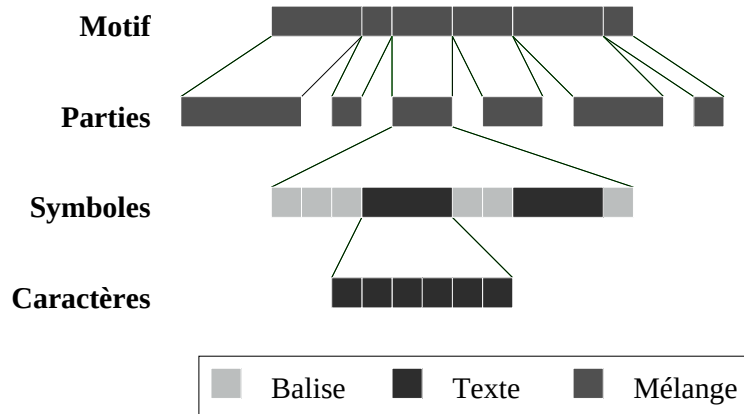


Figure 2.8 – Décomposition d'un motif

2.4.3 Construction d'un adaptateur

Précédemment, nous avons vu qu'un adaptateur est représenté par un ensemble de motifs. Le calcul de l'ensemble des motifs généralisés à partir des motifs initiaux est décrit dans l'algorithme 2.4. Pour obtenir l'ensemble final de motifs, chacun des motifs initiaux doit être généralisé avec les autres motifs initiaux. Cependant la fonction de généralisation étant associative, il n'est pas nécessaire de généraliser chaque paire de motifs explicitement. En effet, il suffit que pour une paire de motifs, le premier ait été généralisé soit directement avec le second, soit avec une généralisation du second. Ainsi lorsque deux motifs sont généralisables, il suffit de les remplacer par leur généralisation dans l'ensemble des motifs. L'algorithme s'arrête lorsqu'il n'y a plus de motifs généralisables.

La figure 2.9 donne le schéma d'une exécution de la généralisation d'un ensemble de motifs.

Les motifs qui ont été extraits à partir des instances exemples permettent la construction d'un adaptateur pour la source et la relation choisie par l'utilisateur. Cet adaptateur doit pouvoir prendre en entrée un document de la source et rendre en sortie les instances de la relation contenues dans ce document. Une telle procédure devra encoder le document à traiter avec le même codage utilisé pour la construction des motifs et appliquer l'ensemble des motifs au document encodé. L'algorithme générique 2.5 décrit la procédure d'extraction qui peut alors être construite à partir du codage (une fonction f) utilisé et de l'ensemble des motifs M . Son application à un document D de la source consiste à encoder D en lui appliquant la fonction de codage f puis à appliquer successivement chacun des motifs pour extraire les instances contenues dans D . L'ensemble des instances est alors retourné.

2.4.4 Exemples négatifs

Dans la plupart des cas, la généralisation des contextes d'occurrences des instances exemples suffit à trouver un motif permettant d'extraire correctement les instances souhaitées par l'utilisateur. Cependant, dans certains cas, les motifs obtenus sont trop généraux et extraient de manière erronée des instances non souhaitées, soit parce que certaines des valeurs sont incorrectement extraites, soit parce qu'il s'agit de tuples de valeurs ne formant pas une instance de la relation. Dans ce cas, il est intéressant de considérer les instances erronées extraites comme des exemples négatifs. De la même manière qu'un exemple positif, les exemples négatifs ont un contexte qui forme un motif. Il est possible de définir une relation de couverture entre deux motifs m_1 et m_2 qui permet de déterminer à partir de la représentation en n -uplet

pour contexte : $c_3 = \dots 32-35-2 \square 2-3 \square 24-36 \dots$

Ce contexte est couvert par le motif obtenu en généralisant les deux instances exemples. Ainsi si l'exemple n a été donné comme exemple négatif, il empêchera la généralisation des contextes c_1 et c_2 .

L'objectif étant d'éviter les sur-généralisations, nous pouvons modifier notre méthode de généralisation de la manière suivante. On construit un ensemble de contextes négatifs, correspondant aux contextes issus des exemples négatifs. Si un motif couvre un de ces contextes négatifs alors on sait que ce motif va extraire au moins une instance négative. Pour éviter d'obtenir ce genre de motif, il suffit lors de la généralisation de deux motifs m_1 et m_2 , de tester si le motif obtenu m' couvre un des contextes négatifs. Si l'un d'eux est couvert par m' , alors celui-ci ne devra pas être retourné comme résultat. On procédera, dans ce cas comme si la généralisation de m_1 et m_2 avait échoué.

L'algorithme 2.6 donne la méthode permettant de prendre en compte des exemples négatifs. Il introduit la fonction *couvre_motif*(g, m) qui teste si un motif g couvre un autre motif m . Lorsque deux motifs sont généralisés et que la généralisation des deux couvre l'un des motifs négatifs, celle-ci est rejetée. Ceci évite de générer des motifs qui extraient des instances erronées.

Procédure *generalise_motif*

Entrées

- Deux motifs $c = (c_1, \dots, c_n)$ et $c' = (c'_1, \dots, c'_n)$
- Un ensemble de contextes négatifs N

Sorties

- Un motif généralisé $g = (g_1, \dots, g_n)$

```

si non compatible( $c_1, c'_1$ ) alors
  retourne echec;
fin si
 $g_1 = \text{generalise\_tete}(c_1, c'_1)$ 
pour tout  $k \in [2, n - 1]$  faire
   $g_k = \text{generalise\_milieu}(c_k, c'_k)$ 
fin pour
 $g_n = \text{generalise\_queue}(c_n, c'_n)$ 
si  $\exists m \in N, \text{couvre\_motif}(g, m)$  alors
  retourne echec
sinon
  return  $g$ 
fin si

```

Algorithme 2.6 – Fonction de généralisation de deux motifs en tenant compte des exemples négatifs

2.5 Analyse

Dans cette section nous nous intéressons à la complexité en temps d'exécution des différentes fonctions que nous avons définies. Nous verrons dans le chapitre suivant qu'en pratique notre méthode est efficace sur des données réelles. Dans notre méthode, deux complexités sont intéressantes : celle de la recherche des occurrences des instances et celle de la généralisation des contextes. La recherche des

occurrences de j instances exemples d'une relation à k attributs dans un document d est au pire en $O(jk|d|)$. La généralisation d'un ensemble C de contextes initiaux issus d'un document d est au pire en $O\left(\frac{|C|^2 - |C|}{2}|d|\right)$.

2.5.1 Complexité de la recherche des occurrences et extraction des contextes

Soit d un document et $|d|$ sa longueur (i.e. le nombre de caractères). Soit k la taille d'un tuple de la relation à extraire (i.e. le nombre d'attributs). Étant donnée une instance $t = (v_1, \dots, v_k)$, la somme des longueurs ne peut pas être supérieure à celle du document car les valeurs des tuples à extraire ne peuvent pas se chevaucher. Ainsi : $\sum_{i=1}^k |v_i| \leq |d|$. La recherche d'une sous-chaîne c dans une chaîne s se fait en $O(|c| + |s|)$ [16]. La recherche d'une instance v_i dans d se fait en $O(|v_i| + |d|)$. La recherche de toutes les valeurs v_i d'une instance se fait en $O(\sum_{i=1}^k (|v_i| + |d|)) = O(\sum_{i=1}^k |v_i| + k|d|)$ ce qui revient à $O(|d| + k|d|) = O(k|d|)$. Si j correspond au nombre d'instances exemples, la recherche des occurrences de toutes les instances se fait en $O(jk|d|)$.

2.5.2 Complexité de la généralisation des contextes

Soit C l'ensemble des motifs, k la taille des tuples de la relation à extraire. Il y a $k + 1$ parties pour chaque motif. On choisit un motif c dans C et on cherche à le généraliser avec chacun des autres. Lorsque la généralisation de c avec un autre motif c' de C réussit, on remplace c et c' par cette généralisation dans C . Le plus grand nombre de généralisations effectuées est obtenu lorsque toutes les généralisations sont des échecs. Dans ce cas, on cherche à généraliser $\frac{(|C| - 1)(|C|)}{2}$ paires de motifs entre elles. La généralisation de deux parties n'est qu'un parcours de deux chaînes simultanément. En effet, pour chaque partie, on parcourt les deux chaînes en même temps en regardant si les symboles de même position peuvent se généraliser. Si p est la longueur de la plus courte des deux parties, ce parcours est en $O(p)$. Dans le pire des cas, la longueur d'un motif, qui correspond à la somme des longueurs de ses parties, est égale à la longueur du document $|d|$ (ou du document le plus long s'il y en a plusieurs).

Soit $c = (p_0, \dots, p_k)$ et $c' = (p'_0, \dots, p'_k)$ deux motifs. On a (1) $\sum_{i=0}^k p_i \leq |d|$ et (2) $\sum_{i=0}^k p'_i \leq |d|$. La généralisation de chaque paire (p_i, p'_i) est au pire en $O(\min(p_i, p'_i))$. Avec (1) et (2) et du fait que $\forall i, \min(p_i, p'_i) \leq p_i$ on a $\sum_{i=0}^k \min(p_i, p'_i) \leq |d|$. Ainsi, la généralisation de deux motifs est au pire en $O(|d|)$. La complexité au pire de l'algorithme de généralisation de tous les motifs est alors en $O\left(\frac{|C|^2 - |C|}{2}|d|\right)$.

2.5.3 Analyse comparative

Nous présentons ci-après les caractéristiques de notre méthode par rapport aux méthodes existantes. La principale différence de notre approche est l'utilisation d'instances exemples dans la construction des motifs. Ceci permet à l'utilisateur d'exprimer son besoin *a priori* à la différence des méthodes structurales telles que IEPAD et ROADRUNNER. À la manière de WIEN et SOFTYMEALY notre approche utilise une méthode inductive. Seulement, notre méthode s'appuie sur un ensemble d'instances exemples et non de pages étiquetées exemples.

Une implication qui découle directement du fait d'utiliser des instances comme exemples plutôt que des pages étiquetées est que l'utilisateur n'est plus obligé d'étiqueter entièrement un sous-ensemble de pages de la source. Il lui suffit de donner deux ou trois exemples de chaque "format" apparaissant

dans les pages de la source. Cet avantage est notable lorsque les pages résultats contiennent beaucoup d'instances. Par exemple, considérons une source ayant 30 instances par page avec un ou deux formats de présentation. Dans le cas où des pages exemples sont utilisées, l'utilisateur aura à étiqueter 3 ou 4 pages exemples soit 90 à 120 instances. Dans ce même cas, notre méthode basée sur des instances exemples, ne nécessite que 4 ou 5 instances.

Par ailleurs, un aspect important d'une méthode de construction d'adaptateurs est l'ensemble des motifs qu'elle permet de construire. Les méthodes WIEN, STALKER et SOFTMEALY cherchent toutes à délimiter les valeurs d'un tuple de manière indépendante à partir du contexte local de chaque valeur. Ceci pose plusieurs problèmes. Par exemple, considérons le contexte suivant :

```
1 ...  
2 <tr><td>Café</td><td>0.40</td></tr>  
3 <tr><td>Les gateaux</td></tr>  
4 <tr><td>Cookie</td><td>1.20</td></tr>  
5 ...
```

Supposons que nous cherchions à extraire ("Café", "0.40"), ("Cookie", "1.20") et toutes les instances du même format. Supposons que les seules parties communes avant et après tout nom ou prix d'un produit soient respectivement `<td>` et `</td>`. Dans notre méthode nous aurions tout naturellement un motif de la forme (3,4-3,4) où 3 et 4 correspondent respectivement aux codes de `<td>` et `</td>`.

Avec des méthodes à base de délimiteurs tel que WIEN ou STALKER nous avons un problème pour choisir le délimiteur droit des noms de produits et le délimiteur gauche des prix des produits. En effet, ils doivent respectivement être un préfixe et un suffixe de `</td><td>` sans toutefois se chevaucher. Par exemple, on ne pourra avoir en même temps `<td></` comme délimiteur droit des nom et `></td>` comme délimiteur gauche des prix. En effet, la recherche d'un nouveau délimiteur commence là où le précédent a terminé. Comment ce choix est effectué par les méthodes citées n'est pas évident. Dans le cas de SOFTMEALY la notion de séparateur n'impose pas de telles contraintes sur les séparateurs gauche et droit de valeurs successives.

Un autre aspect important dans la génération d'adaptateurs est la méthode de construction des motifs. Les méthodes les plus proches de la méthode basée sur la généralisation de contextes que nous proposons sont les méthodes inductives WIEN, STALKER et SOFTMEALY. Dans la méthode WIEN, l'algorithme proposé consiste à générer un ensemble de délimiteurs candidats et de tester leurs validités par rapport à un ensemble de contraintes (cf. chapitre 1). En pratique cette méthode s'avère peu efficace en temps d'exécution. Elle peut être largement améliorée en constatant que les contraintes imposent simplement qu'un délimiteur gauche (resp. droit) doit être un suffixe (resp. préfixe) de chaque sous-chaine apparaissant avant (resp. après) la valeur à délimiter. Par exemple, l'algorithme de calcul d'un suffixe (resp. préfixe) commun de notre méthode peut être utilisé. Dans le cas de STALKER, la méthode proposée fonctionne par raffinements successifs. Initialement, un motif constitué d'un unique symbole est construit. Ensuite, tout motif délimitant une position non étiquetée est raffiné. Le raffinement consiste à ajouter un symbole au motif. De la même manière que pour WIEN cette méthode s'avère prendre un temps considérable et nécessite la génération d'un nombre élevé de motifs avant d'obtenir un motif acceptable. Notamment, cette méthode génère un grand nombre de candidats ne couvrant aucun exemple. Tester la validité de tels motifs nécessite un parcours des documents exemples. Les propriétés de notre méthode par généralisation des contextes, font qu'à aucun moment un motif ne couvrant aucune instance n'est généré. Ceci se traduit par un gain notable en temps d'exécution. Le système SOFTMEALY utilise une méthode qui est aussi basée sur la généralisation. Celle-ci est comparable à celle que nous utilisons.

De manière similaire à la méthode proposée par Brin et implémentée par le système DIPRE, nous cherchons à répondre à un besoin qui est la construction d'une relation à partir de données issues du Web. Toutefois notre objectif se distingue de celui de Brin de plusieurs manières. D'une part les relations à extraire ne sont pas des relations générales telles que la relation (*auteur, titre*). D'autre part, l'objectif n'est pas d'extraire *quelques* instances de la relation apparaissant dans tout le Web, mais *toutes* les instances de la relation apparaissant dans une source donnée. Dans le cas du système DIPRE, nous cherchons à construire un ensemble de motifs très simples pouvant s'appliquer rapidement à un grand ensemble de pages. À l'inverse notre méthode cherche à construire un ensemble de motifs, qui peuvent être complexes et capables d'extraire toutes les instances d'un ensemble de pages d'une même source. Ainsi, les motifs générés par la méthodes de Brin s'avèrent souvent trop simples pour permettre une extraction de manière complète de toutes les instances d'un ensemble de pages. Notre méthode propose de construire des motifs plus complexes permettant de répondre à ce problème d'extraction.

De manière générale, la méthode présentée ici permet d'obtenir un adaptateur rapidement (ie. avec peu d'exemples) sans étiqueter des pages entières. Nous verrons dans le chapitre suivant, différentes expérimentations qui le confirme.

2.6 Conclusion

Dans ce chapitre nous avons présenté une nouvelle approche à l'extraction d'information à partir du Web. L'utilisateur propose un ensemble d'instances exemples d'une relation qu'il souhaite extraire d'une source de données qu'il a préalablement sélectionnée. À partir de ces exemples, notre méthode construit un adaptateur pour la source qui permettra d'en extraire les autres instances de la relation cible. Ainsi, l'adaptateur construit permet de répondre au besoin d'information exprimée par l'utilisateur en l'appliquant aux pages de la source.

La méthode présentée permet à l'utilisateur d'exprimer simplement son besoin d'informations en donnant quelques instances exemples. Par rapport aux méthodes à base de connaissances, l'utilisateur n'a pas à faire une analyse des documents desquels il souhaite extraire des informations. À la différence des méthodes à base de pages étiquetées notre méthode ne fait pas l'hypothèse que toutes les instances d'une page ont été fournies. De plus, un étiquetage lourd et fastidieux de plusieurs pages n'est pas imposé. Mis à part le fait de délivrer l'utilisateur de cette tâche, le fait de ne donner que des instances permet d'avoir une méthode dont les exemples sont indépendants des documents. Ceci permet notamment leur réutilisation lors du changement de format d'une source. Par ailleurs, bien que cette contrainte soit levée, la précision des adaptateurs obtenus par notre méthode est comparable voire supérieur aux méthodes existantes. De plus, en utilisant la généralisation, notre méthode garantie que tout candidat permet au minimum l'extraction d'une instance. Ainsi elle est plus efficace que les méthodes pouvant générer comme candidats des motifs ne permettant d'extraire aucune instance. Par rapport aux méthodes structurales, les données extraites se trouvent dans le format connu par l'utilisateur, le même format que celui dans lequel ont été donnés les exemples. De plus, les motifs que nous construisons peuvent extraire des valeurs à l'intérieur même des parties textuelles sans tout extraire comme le font les méthodes structurales.

Comme nous le verrons dans le chapitre suivant, cette méthode a été implémentée et évaluée sur divers domaines dont les moteurs de recherche, les annuaires en lignes et les catalogues de produits. Sur chacun de ces domaines notre approche donne de très bon résultats. Ces résultats peuvent s'expliquer par le fait que pour conserver un taux de précision élevé la méthode cherche à conserver des motifs suffisamment spécifiques. Les taux de rappel sont obtenus par le fait que la méthode permet d'avoir de multiples motifs, couvrant les différents formats pouvant être présents dans une même source.

Expérimentations du système IEREL

Dans ce chapitre nous présentons l'implémentation de notre méthode par le système IEREL et l'évaluation empirique qui a été effectuée grâce à ce système. Les données sur lesquelles nous avons évalué notre méthode sont issues de trois domaines différents : les moteurs de recherche, les annuaires en ligne et les sites de commerce électronique. Dans une première étude, nous avons évalué les performances de notre méthode sur plusieurs sources de chaque domaine, en cherchant à construire un adaptateur permettant d'extraire un maximum d'instances avec un minimum d'instances exemple. Une seconde étude nous a permis de mettre en avant l'influence du nombre et de la qualité des exemples sur la rapidité d'obtention d'un adaptateur parfait.

Ce chapitre est organisé comme suit. Dans un premier temps nous présentons le système IEREL et ses caractéristiques. Ensuite nous décrivons la méthodologie générale que nous avons utilisée lors des évaluations. Nous présentons alors, une évaluation pour chacun des trois domaines. Avant de conclure, nous présentons une étude de l'influence de la qualité et du nombre d'exemples sur la généralisation.

3.1 Caractéristiques de IEREL

Le système IEREL permet, à partir d'un ensemble de pages d'une source et d'un ensemble d'instances exemple d'une relation à extraire, de construire un adaptateur pour la source. Une fois l'adaptateur construit, il est possible de l'appliquer sur l'ensemble des pages de la source et d'extraire la relation donnée.

Six paramètres peuvent être utilisés afin d'optimiser les phases d'extraction des contextes et des nouvelles instances. Ces paramètres sont les tailles minimales et maximales que peuvent prendre :

1. les différentes parties des contextes entourant les exemples ;
2. les parties ignorées des motifs généralisés ;
3. les instances à extraire.

La figure 3.1 donne un aperçu de l'interface utilisateur de l'application. Elle est divisée en deux parties : la partie haute permet la saisie des données et des paramètres et la partie basse affiche soit l'une des pages à partir de laquelle l'utilisateur cherche à construire un adaptateur soit le résultat de l'exécution de la génération de l'adaptateur et de l'extraction des autres instances.

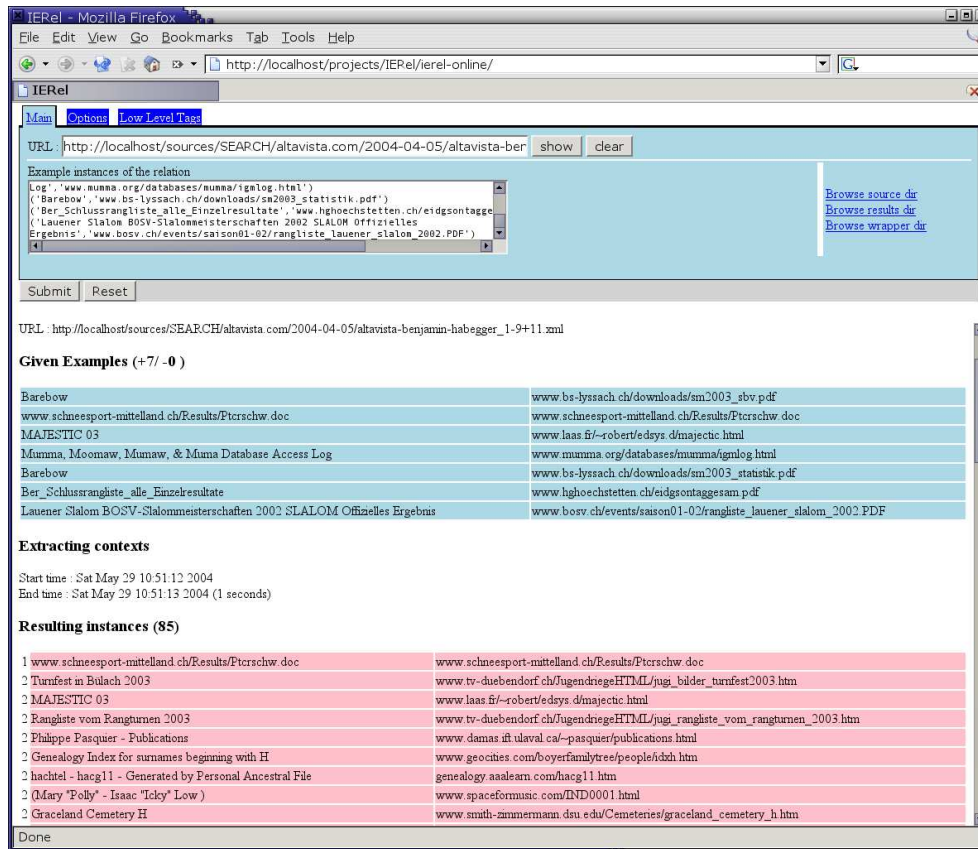


Figure 3.1 – Interface de l'application

3.2 Méthodologie

Nous avons évalué notre méthode sur trois domaines différents : les annuaires en ligne, les moteurs de recherche et les catalogues de produits. Pour chaque domaine, nous avons plusieurs sources de données différentes. Pour chaque source, nous avons rassemblé un ensemble de documents. Ensuite, un ensemble d'instances exemple a été constitué pour chaque source. Chaque ensemble d'exemples a été construit interactivement en utilisant IEREL. Par exemple, en donnant deux ou trois exemples initiaux d'un ensemble de pages d'une source, un certain nombre de nouvelles instances a été extrait. Lorsque certaines instances souhaitées n'étaient pas extraites, quelques unes des instances manquantes ont été ajoutées comme exemples. Ce processus itératif a été répété à chaque fois jusqu'à l'obtention d'un adaptateur parfait (*i.e.* permettant d'extraire toutes les instances).

Les tables 3.1, 3.5 et 3.6 donnent les résultats sur les sources des différents domaines. Dans ces tables, les informations suivantes apparaissent :

Source le nom de la source.

URL l'URL de la source.

Inst. le nombre d'instances de la relation à extraire contenu dans la source.

Ex. le nombre d'instances exemple données.

Mot. le nombre de motifs obtenus.

Récup. le nombre d’instances extraites.

Rap. le taux de rappel.

Prec le taux de précision.

Tmps. le temps d’exécution totale de l’ensemble des généralisations effectuées.

Les colonnes *Inst.* et *Ex.* contiennent, en plus, un nombre entre parenthèses. Celui-ci correspond respectivement au nombre de pages composant la source et au nombre de pages desquelles sont issus les exemples. Les taux de rappel et de précision utilisent les formules habituelles de la communauté d’extraction d’information : étant donné n_r le nombre d’instances extraites, n_i le nombre d’instances réelles et n_c le nombre d’instances correctement extraites (ie. appartenant à la relation souhaitée), le taux de rappel est alors $r = n_c/n_i$ et la précision $p = n_c/n_r$. Toutefois, étant donné l’aspect itératif de la construction, le taux de rappel est normalement de 1.0. En effet, le processus itératif de construction des motifs s’arrête lorsque toutes les instances ont été extraites. Ainsi, les valeurs intéressantes sont alors la précision (qui est aussi souvent de 1.0) et le nombre d’exemples nécessaires.

Afin de vérifier que les instances extraites par chaque adaptateur généré sont correctes, nous avons procédé de la manière suivante. Un adaptateur dont la fiabilité est certaine est construit manuellement¹. Cet adaptateur est appliqué à l’ensemble des pages de la source générant un ensemble d’instances. Ensuite un adaptateur est généré automatiquement par notre méthode à partir d’instances exemple. Les différences des deux ensembles d’instances sont alors calculés. Lorsque les deux adaptateurs génèrent le même ensemble d’instances, c’est que l’adaptateur construit par notre méthode est aussi fiable que celui généré manuellement.

Dans tous les cas, l’évaluation d’une source s’est faite sur un ensemble de pages. Dans le cas des annuaires en ligne, ces ensembles de documents proviennent du dépôt RISE², mis en place par Muslea. Pour les autres méthodes, nous avons développé différents outils permettant de faciliter la constitution de bases de documents issus d’une même source. Un premier outil permet de télécharger un ensemble de documents liés entre eux par un lien “suivant” à partir d’un document initiale. Les liens peuvent être sélectionnés de deux façons. La première consiste à évaluer la similarité de l’URL du lien par rapport à celle du document initial. La seconde consiste à donner un chemin permettant de suivre un lien suivant sur chacune des pages. Dans certains cas nous avons utilisé WEBSOURCE décrit dans la deuxième partie de cette thèse pour constituer cet ensemble. Dans tous les cas, nous avons suivi la méthodologie suivante pour constituer une base de documents pour une source. Une ou plusieurs requêtes ont été formulées et envoyées à la source. L’ensemble des documents est téléchargé en utilisant l’URL obtenue comme URL initiale.

Toutes les expérimentations se sont déroulées sous GNU/Linux sur un PC avec un processeur AMD Athlon XP 1700+ avec une fréquence de 1076 MHz et avec 256Mo de mémoire.

3.3 Annuaires en ligne

Un premier domaine sur lequel nous avons testé notre méthode est celui des annuaires en ligne. Contrairement aux autres domaines, les données sont issues du dépôt RISE constitué par I. Muslea. Nous nous sommes intéressés plus précisément aux trois sources les plus citées dans la littérature : BigBook, OKRA et AddressFinder. Dans toutes les expérimentations sur les annuaires, le paramétrage de IEREL était celui par défaut. Ainsi, les balises de bas niveau ont été éliminées, la taille maximale d’une valeur

¹ En pratique nous avons utilisé des feuilles des styles XSLT

² <http://www.isi.edu/info-agents/RISE/>

a été fixée à 300, la longueur maximale de contexte ou de texte pouvant être ignoré entre deux valeurs a été fixé à 1000.

<i>Source</i>	<i>Inst.</i>	<i>Ex.</i>	<i>Mot.</i>	<i>Recup.</i>	<i>Rap.</i>	<i>Prec</i>	<i>Tmps.</i>
BigBook	4299 (235)	5 (1)	1	4299	1.0	1.0	<1s
OKRA	3335 (252)	4 (2)	1	3335	1.0	1.0	1s
AddressFinder (1)	88 (10)	5 (1)	2	88	1.0	1.0	<1s
AddressFinder (2)	53 (10)	12 (6)	6	88	1.0	1.0	<1s

Table 3.1 – Résultats pour les *Annuaire en ligne*

3.3.1 BigBook

La première source sur laquelle nous avons testé notre méthode est BigBook. Celle-ci est composée de 235 documents contenant chacun de 1 à 20 instances. L'ensemble des documents contient 4299 instances. La relation que nous cherchons à extraire de cette source est la relation *adresse(nom,adresse,ville,tel)*. Cette source est bien structurée et contient un unique format. Ainsi, seulement 3 exemples issus d'une seule page ont suffi à construire un adaptateur permettant d'extraire l'ensemble des instances de tous les documents. Cet adaptateur est composé d'un unique motif.

3.3.2 OKRA

Dans le cas d'OKRA, la source est composée de 252 documents et un total de 3335 instances (dont 214 doublons). Les documents de la source sont composés d'instances ayant 4 ou 5 champs : un score, un nom, une date d'enregistrement, une adresse e-mail et éventuellement une organisation. Cependant la relation que nous cherchons à extraire est la relation *email(nom,email)*. Par contre, les contenus et la présence des autres champs influencent le nombre d'exemples nécessaires. En effet, pour construire un adaptateur capable d'extraire la totalité des 3335 instances, il est nécessaire d'avoir au minimum une instance ayant une valeur pour le champ organisation, au minimum une instance n'ayant pas ce champ et au minimum deux instances ayant la valeur du champ score qui diffère. Nous avons obtenu un adaptateur permettant d'extraire les 3335 instances en donnant 4 instances exemple sur 2 pages différentes. La figure 3.2 donne l'affichage issu des occurrences des 4 instances.

3.3.3 AddressFinder

La dernière source sur les adresses est AddressFinder. Celle-ci est moins conséquente. Elle est composée de 10 pages contenant chacune entre 2 et 10 entrées et totalisant 88 entrées sur les 10 pages. Les entrées qui peuvent être composées de plusieurs champs ont comme format :

Name, AltName?, Email, Organization?, LastUpdate, (Organization|ServiceProvider)?

où *Name* : est le nom de la personne, *AltName* est un nom alternatif, *Email* est l'adresse e-mail, *Organization* dans le premier cas est l'organisation à laquelle appartient la personne et dans le second cas, l'organisation qui diffuse l'entrée, *LastUpdate* est une date de dernière mise à jour et *ServiceProvider* est un nom de la source diffusant l'information.

Score: 1 Name: Ingo Bruss Email: bruss@wbkst15.mach.uni-karlsruhe.de	First Entered: 03/06/96
Score: 1 Name: Trevor Bruss Email: bruss@pa621a.inland.com	First Entered: 03/06/96
Score: 0.6 Name: Adam Bruss Email: amcb@getnet.com	First Entered: 03/06/96
Score: 1 Name: Xavier Beaume Email: beaume@eiffel.com Organization: Interactive Software Engineering	First Entered: 09/16/96

Figure 3.2 – Occurrences des exemples pour OKRA

Dans une première expérimentation nous avons cherché à extraire la relation *email(nom,email)*. À partir de 5 exemples issus de la première page, nous avons obtenu 2 motifs capables d’extraire les 88 instances. Un premier motif correspond à un premier sous-ensemble d’entrées qui contiennent un nom alternatif. Le second motif correspond aux autres entrées, celles pour lesquelles aucun nom alternatif n’apparaît. Les autres champs se trouvant à la fin des entrées ne viennent pas perturber la généralisation.

Dans un deuxième temps nous avons souhaité extraire la relation *email(nom,email,organisation)* où l’organisation est le second champ *Organization*. Dans la source, 53 entrées possèdent ce champ. Ces entrées peuvent être classées en 4 catégories : celles possédant les trois champs obligatoires (i.e. *Nom*, *Email* et *LastUpdate*) et le second champ organisation (soit *X* cet ensemble), celles possédant les trois champs et uniquement en plus le premier champ organisation (soit *O* cet ensemble), celles possédant les trois champs et uniquement en plus le champ nom alternatif (soit *A* cet ensemble) et celles possédant les trois champs, le premier champ organisation ainsi que le champ alternatif (soit *AO* cet ensemble). La répartition des 53 entrées est la suivante : $|X| = 28$, $|A| = 1$, $|O| = 19$ et $|AO| = 4$. Un exemple d’entrée de chaque classe est donné dans la table 3.2.

Un adaptateur constitué de 6 motifs a été obtenu par IEREL à partir de 12 exemples. On constate qu’à chaque classe correspond un ensemble de motifs. La table 3.3 donne la couverture de chaque motif. Pour la classe *X*, à partir de trois exemples, un unique motif pour la classe a été généré. Il permet d’extraire toutes les instances de cette classe. Pour la classe *AO* trois instances, sur les quatre présentes, ont été nécessaires pour construire un motif permettant de toutes les extraire. Pour la classe *A*, une seule instance apparaît dans l’ensemble de la source. En la donnant comme exemple, ceci permet de l’extraire à nouveau. Toutefois, le motif généré est très spécifique. Ainsi, si dans un nouveau document de la source, une nouvelle instance de cette même classe devait apparaître, il y a fort peu de chance que celle-ci soit extraite. Dans le cas de la classe *O*, un premier motif permettant d’extraire 17 instances est obtenu par la généralisation des contextes de 3 instances exemple. Les occurrences des 2 instances manquantes ont la particularité de se trouver en bas de page. Ainsi la dernière partie de leur contexte est différente de celle des autres instances. Ajouter ces deux instances comme exemples permet de les extraire, mais chacune avec un motif spécifique.

Classe	Exemple d'entrée
<i>X</i>	1. Name: 'Lithium' J Smith E-Mail: aulmer@u.washington.edu Last Update: 08/01/95 Organization: University of Washington
<i>O</i>	1. Name: Chris Kushmerick E-Mail: ckushmerick@ccmail.sunysb.edu Organization: state university of new york Last Update: 02/17/96 Organization: SUNY at Stony Brook
<i>AO</i>	2. Name: 'Sir Brand' Gregrobin Smith Alt. Name: Smith Gregrobin E-Mail: sirbrand@u.washington.edu Organization: university of washington Last Update: 06/21/96 Organization: University of Washington
<i>A</i>	4. Name: - Richard Smith Alt. Name: Richard E-Mail: GBORDERS@SFASU.EDU Last Update: 11/12/95 Organization: Stephen F. Austin State University

Table 3.2 – Exemples d'entrées pour chaque classe dans AddressFinder

3.3.4 Synthèse

Le tableau 3.1 récapitule les résultats obtenus pour les trois sources d'adresses. Dans tous les cas nous avons pu obtenir un adaptateur permettant d'extraire toutes les instances de la source. Dans toutes les expérimentations, le temps de calcul nécessaire à la construction des motifs est inférieur à une seconde. La méthode s'avère donc être efficace sur ce domaine. L'exemple de AddressFinder montre que selon la relation à extraire et les formats apparaissant dans la source, la généralisation peut nécessiter plus ou moins d'exemples. De plus, dans le cas de format multiples, il est nécessaire d'avoir quelques exemples pour chaque format. Ainsi si un format n'apparaît pas dans les documents exemples, aucun motif ne pourra être généré pour ce format et lors de l'extraction, les instances dans ce format ne pourront pas être extraites.

Motif	Classe	Nb exemples	Nb instances	Couverture
1	<i>X</i>	3	29	$29/29 = 1.00$
2	<i>AO</i>	3	4	$4/4 = 1.00$
3	<i>O</i>	1	1	$1/19 = 0.052$
4	<i>O</i>	3	17	$17/19 = 0.895$
5	<i>O</i>	1	1	$1/19 = 0.052$
6	<i>A</i>	1	1	$1/1 = 1.00$
Total		12	53	$53/53 = 1.00$

Table 3.3 – Couverture des motifs d’extraction de la relation *email(nom, email, organisation)* sur AddressFinder

3.4 Moteurs de recherche

Les moteurs de recherche sont l’une des sources d’information les plus répandues et indispensables sur internet. Automatiser l’accès à leurs résultats ouvre la porte à de nombreuses applications telles que la mise en oeuvre de méta-moteurs de recherche, la création de moteurs spécialisés, *etc.* Nous avons testé IEREL sur 8 moteurs de recherche différents. Dans chacun des cas, la relation à extraire est la relation *document(titre, url)*. Dans la version actuelle de IEREL, il n’est pas possible d’accéder aux attributs des balises notamment l’attribut *href* des balises *a*. Cependant, les moteurs de recherche affichent généralement cette URL. Ainsi l’URL que nous cherchons à extraire n’est pas celle dans le liens mais celle affichée dans la page résultat.

3.4.1 AltaVista

Nous avons appliqué IEREL sur l’ensemble des pages obtenues en soumettant la requête “Benjamin Habegger” au moteur AltaVista. Cet ensemble est constitué de 63 pages contenant chacune 10 résultats à l’exception de la dernière qui n’en contient que 6. Nous avons ainsi 626 instances au total. Les résultats des pages peuvent être divisés en deux formats : les résultats menant sur des documents PDF, et les résultats menant sur les documents HTML. Dans le cas de documents PDF une ligne en plus apparaît dans le résultat. Celle-ci propose de télécharger un logiciel permettant la lecture des documents PDF.

Un premier adaptateur a été construit en ne considérant que les 9 premières pages. 5 exemples sur 5 pages ont permis de construire un adaptateur parfait pour ces 10 pages. Toutefois lorsque cet adaptateur est appliqué à l’ensemble des 63 pages, seules 621 instances sont extraites. Les 5 manquantes sont toutes des instances dans le même format, celui des documents PDF. D’autre part, ces cinq instances ont toutes la particularité d’être le premier résultat d’une page. Les 9 premières pages n’ayant pas de tels résultats, les motifs générés n’ont pas permis de couvrir les 5 instances. L’ajout de l’une de ces instances à l’ensemble des exemples permet d’obtenir un adaptateur permettant d’extraire les 626 instances.

3.4.2 Excite

Les données pour Excite ont été obtenues à partir de quatre requêtes différentes donnant un total de 325 instances contenues dans 19 pages. Dans les données, trois formats apparaissent. Le plus fréquent contient un titre, un extrait puis une URL. Le second format contient en plus, entre l’extrait et l’URL, une catégorie. Le dernier format n’apparaît que pour une instance. Dans ce cas, aucun extrait n’apparaît.

La figure 3.3 donne un exemple de résultat obtenu par Excite dans chacun des trois formats.

Pour extraire la totalité des instances, 10 exemples répartis sur 3 pages ont été nécessaires. A partir de ces exemples trois motifs ont été générés. Chaque motif permet d’extraire les valeurs d’un des formats. Cependant, pour permettre l’extraction de l’unique instance dans le troisième format, il a été nécessaire de donner celle-ci en exemple. C’est à partir de cet exemple que le troisième motif a été généré. Étant issu d’un seul exemple, il s’agit d’un motif très spécifique.

13.	DBLP: Mohamed Quafafou Benjamin Habegger , Mohamed Quafafou: Multi-Pattern Wrappers for Relation Extraction from the Web. ECAI 2002: 395-399 http://www.acm.org/sigs/sigmod/dblp/db/indices/a-tree/q/Quaf...
66.	The Nashville Banner: Hunchback Wails Review by Lisa DuBois of the premier of The Hunchback: of Notre Dame musical in Nashville. (September 1, 1997) Top/Arts/Music/Bands_and_Artists/D/DeYoung_Dennis/Articles_... http://www.styxnet.com/deyoung/print/rev_hbwails.html
112.	DuBois volunteer fire department http://www.mindspring.com/~gibbles

Figure 3.3 – Exemples de résultats dans chacun des trois formats du moteur de recherche Excite

3.4.3 Galaxy

Pour Galaxy, nous avons construit une base de 51 documents à partir de 3 requêtes “Dubois”, “XML” et “Web Services”. Chaque document contient 9 résultats sauf un qui n’en contient que 7 totalisant ainsi 457 instances. Un adaptateur a été construit à partir de l’ensemble des 11 documents résultants de la requête “Dubois”. Un adaptateur parfait pour cet ensemble constitué de 4 motifs a été obtenu avec 9 exemples. Appliqué aux deux autres ensembles, il a permis d’extraire toutes les instances. Cependant, certains des formats apparaissant dans l’ensemble issu de la requête “Dubois” n’apparaissaient pas dans les deux autres ensembles. En effet, dans la plupart des cas, les réponses de Galaxy donnent la catégorie à laquelle le résultat appartient. Dans le cas de la requête “Dubois”, certains des résultats n’avaient pas de catégorie. Ainsi, le choix de l’un des deux autres ensembles comme ensemble d’apprentissage n’aurait pas permis de trouver tous les motifs permettant l’extraction de tous les résultats de l’ensemble “Dubois”.

3.4.4 Overture

Un autre moteur de recherche sur lequel nous avons évalué notre méthode est Overture. L’ensemble de données que nous avons constitué pour les expérimentations contient 1145 documents issus de deux requêtes “Benjamin Habegger” (15 pages) et “Web Services” (1130 pages). Overture présente ses résultats par pages de 40 entrées chacune. La dernière page de la requête “Benjamin Habegger” n’en contient que 36. Ainsi nous avons au total 45796 instances.

Les résultats d’une requête sont de deux types : les liens sponsorisés et les liens additionnels. Cependant, le format dans lequel ces deux types de liens sont présentés diffère très peu. Ainsi, il a été possible de construire un adaptateur à partir de deux exemples (un lien de chaque type) permettant d’extraire 45789 instances (dont 16 doublons), soit une grande majorité des instances. Les 7 manquantes sont des instances dont les entrées correspondantes ne possèdent pas de descriptif. Elles sont réparties sur 7 pages différentes. Un adaptateur parfait a pu être obtenu en ajoutant 3 pages et 3 instances exemple. L’adaptateur final contient 3 motifs.

3.4.5 Metacrawler

L'ensemble de données constitué pour évaluer notre méthode sur le moteur Metacrawler est composé de 30 documents issus des trois requêtes : “Benjamin Habegger”, “XML” et “Web Services”. Chaque document contient 20 réponses du moteur sauf les dernières pages de chaque requête qui en comptent respectivement 6, 1 et 1. Ainsi nous avons au total 548 instances.

Deux formats apparaissent dans les données : un pour les résultats ayant une catégorie associée et un pour les résultats sans catégories. De plus, deux types de résultats sont affichés dans les mêmes formats : 159 résultats sponsorisés et 389 résultats non sponsorisés. La figure 3.4 donne un exemple de chacun des deux types de résultats. Un adaptateur pour cette source a été construit à partir de 6 exemples (issus de résultats non sponsorisés) issus de 3 pages. Celui-ci extrait les 548 instances (dont 1 doublon) de l'ensemble des pages.

47.	Microsoft Security Summit 2004 Attend the free Microsoft Security Summit coming to 20 U.S. cities between April and June 2004. Create secure web services for your business. Sign up today. Sponsored by: http://www.microsoft.com
60.	Reba McEntire Reba's Official Web Site. Includes an interactive area called RebaNet with special features and services . Top/Arts/Music/Styles/Country/Bands_and_Artists/M/McEntire,... http://www.reba.com/

Table 3.4 – Exemples de résultats sponsorisés ou non de Metacrawler

Les instances correspondant à des résultats sponsorisés ont la partie “lien” qui est préfixée par “Sponsored by :”. Ceci est dû au fait que les liens des résultats non sponsorisés se trouvent dans le même format que les chaînes préfixées par “Sponsored By :” correspondant au résultats sponsorisés. Ainsi les résultats de l'extraction peuvent être interprétés de trois façons différentes. La première consiste à considérer que l'utilisateur cherche à extraire tous les résultats (sponsorisés ou non) et qu'il considère qu'étant donné que pour les liens sponsorisés l'URL est extraite (bien qu'elle soit préfixée par “Sponsored by :”), ces résultats sont corrects. Dans ce cas, les taux de précision et de rappel sont de 1.0. Le second cas consiste à considérer que l'on ne souhaite extraire que les liens non sponsorisés. Les 159 instances extraites correspondant à des résultats sponsorisés sont alors de trop. Étant donné que tous les résultats souhaités sont extraits le rappel reste à 1.0 mais la précision chute à $389/548 = 0.71$. Le dernier cas consiste à considérer que l'utilisateur souhaite extraire tous les liens mais que les liens sponsorisés sont incorrectement extraits. Avec cette hypothèse, les taux de rappel et de précision sont de $389/548 = 0.71$.

En utilisant l'une des instances contenant “Sponsored by :” comme exemple négatif, aucune instance erronée n'est extraite. Par contre, les contextes des exemples positifs ne se généralisent plus entre eux. Ainsi chaque motif construit ne permet que l'extraction d'une seule instance. Les motifs ne pouvant se généraliser, seules les instances données en exemples sont alors extraites.

3.4.6 Google

Dans le cas du moteur Google l'ensemble de données que nous avons constitué est composé de 83 documents, résultats de la requête “benjamin habegger”. Chaque document contient 10 réponses. Ainsi nous avons un total de 830 instances. Les formats présents dans les résultats de Google sont très divers :

- un résultat peut être un résultat principal ou secondaire où un résultat secondaire est présenté avec une plus grande marge à gauche ;

- certains résultats proposent un lien “More pages from ...” avec des résultats complémentaires du site auquel appartient le résultat ;
- les résultats peuvent proposer une version sauvegardée en cache, ce qui se traduit par la présence d’un lien “Cache” vers cette page ;
- lorsque la page résultat n’est pas dans la langue courante, un lien de traduction est proposé ;
- les résultats peuvent être des documents autres que du HTML et dans ce cas, ils sont notamment précédés du type de document (par exemple [PDF] dans le cas de documents PDF) avec d’autres variations de formats ;
- la plupart du temps, pour les résultats dans d’autres formats que le HTML, une version HTML est proposée ;
- la plupart des résultats ont un extrait du document résultat.

Toutes les combinaisons de ces caractéristiques sont possibles. Ainsi le nombre de formats est élevé.

Pour nos expérimentations, nous avons mis à part les 9 premières pages pour la construction des motifs. Dans une première expérience, nous avons choisi de manière itérative 16 exemples permettant d’obtenir un ensemble de 6 motifs couvrant la totalité des 90 résultats présents sur les 9 premières pages. En appliquant les motifs à l’ensemble des 83 pages, 775 instances ont été extraites. Toutefois, 2 instances erronées ont été extraites. Cette erreur provient de la proximité de deux résultats avec le même titre comme le montre la figure 3.4. Les deux premiers résultats ont le même titre mais ont des URL différentes. L’instance correspondant au second résultat a été donnée comme exemple. Lors de la recherche d’occurrences de cette instance, l’occurrence composée du titre du premier résultat et de l’URL du second a été trouvée (les parties grisées dans la figure 3.4). La généralisation de son contexte avec celui d’une autre occurrence similaire a donné lieu à un motif erroné. La figure 3.5 montre une occurrence erronée qui a été extraite par ce motif. La suppression du motif erroné a permis de corriger l’adaptateur qui extrait alors 773 instances correctement.

Dans une seconde expérimentation, l’un des exemples ambigus a été remplacé par un autre exemple du même format mais non ambigu. Ceci a permis de générer un nouvel adaptateur ne contenant plus de motif erroné. Celui-ci a même permis d’extraire plus d’instances. Ainsi 778 instances correctes ont été extraites dans cette seconde expérimentation. Finalement, en ajoutant 2 exemples supplémentaires sur une page de plus, les 830 instances ont été extraites.

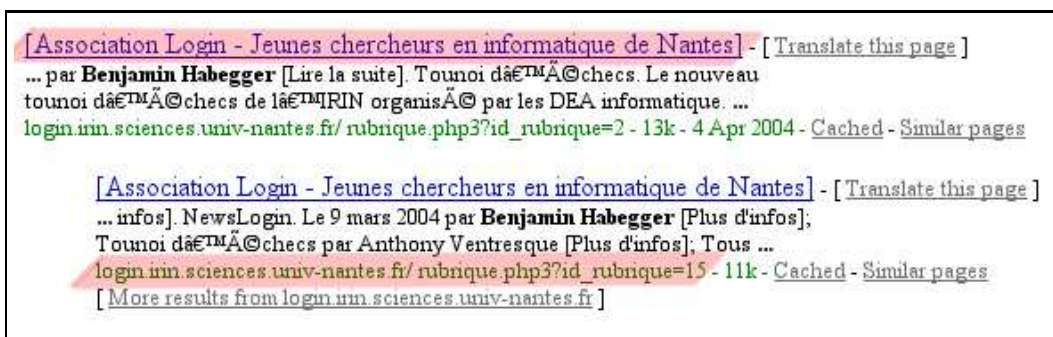


Figure 3.4 – Recherche d’occurrence erronée



Figure 3.5 – Extraction erronée

3.5 Catalogues de produits

Le commerce électronique est un domaine où la création d’adaptateurs peut jouer un rôle important. Par exemple, ils sont indispensables pour la mise en place de comparateurs de prix, de systèmes de veille économique, *etc.* En effet, créer de tels systèmes nécessite d’être capable d’extraire de différents sites de commerce électronique concurrents les informations sur les prix des mêmes produits ou de produits similaires. Ainsi, nous avons testé notre méthode sur de tels sites. IEREL a donc été évalué sur quatre sites de commerces électronique français différents : les sites de la FNAC, de Alapage, de Amazon et de Conforama.

Dans le cas des trois premiers, nous avons cherché à créer un adaptateur permettant d’extraire les produits de la catégorie DVD. Dans ces trois cas, la relation à extraire était *dvd(titre, acteur_principal, prix)*. L’ensemble de données constitué pour chaque source a été obtenu à partir de la requête “Depardieu”. Les sites Alapage, Amazon et FNAC ont respectivement généré 5, 7 et 4 pages de résultats.

3.5.1 Alapage

Dans le cas de Alapage, les pages contiennent respectivement 20, 20, 20, 19 et 5 instances, soit un total de 84 instances. Dans la troisième page, l’un des résultats n’a pas été comptabilisé car il n’avait pas de prix. Tous les résultats de Alapage sont présentés dans le même format, sauf pour les trois premiers de la première page. En effet, ils correspondent aux produits les plus consultés et sont mis en avant par une présentation différente. À partir de 6 instances exemple, dont 5 sur la première page et une sur la troisième page, nous avons obtenu un adaptateur composé de 3 motifs et permettant d’extraire la totalité des instances. Un premier motif a été obtenu à partir de 2 des exemples correspondant aux premiers résultats de la première page. Un second motif a été obtenu à partir de 3 autres exemples issus, eux aussi, de la première page. Finalement, une dernière instance a été ajoutée car elle n’était pas reconnue car son format était légèrement différent : c’était la seule à avoir une liste d’acteurs à un unique élément.

3.5.2 Amazon

Pour Amazon, les documents sont respectivement composés de 20, 16, 9, 13, 20, 8 et 13 instances, soit un total de 99 instances. Le fait que certaines pages aient moins de 20 instances s’explique par le fait que certains résultats n’ont pas été comptabilisés car ils correspondaient à des titres pour lesquels il n’y avait plus de stocks et dont le prix n’apparaissait pas. Globalement les résultats de d’Amazon ont une présentation régulière. Malgré tout, deux éléments n’apparaissent pas dans tous les résultats : d’une part

Source URL	Inst.	Ex.	Mot.	Recup.	Rap.	Prec	Tmps.
AltaVista http://www.altavista.com/	626 (63)	7 (7)	2	626	1.00	1.00	1s
Excite http://www.excite.com/	325 (19)	10 (3)	3	325	1.00	1.00	1s
Galaxy http://www.galaxy.com/	457 (51)	9 (4)	4	457	1.00	1.00	<1s
Overture http://www.overture.com/	45789 (1145)	5 (4)	3	45789	1.00	1.00	1s
Metacrawler (interprétation 1) http://www.metacawler.com/	548 (30)	6 (3)	2	548	1.00	1.00	1s
Metacrawler (interprétation 2) http://www.metacawler.com/	389 (30)	6 (3)	2	548	1.00	0.71	1s
Metacrawler (interprétation 3) http://www.metacawler.com/	548 (30)	6 (3)	2	389	0.71	0.71	1s
Google (expérience 1) http://www.google.com/	830 (83)	16 (9)	6	773/775	0.93	0.997	1s
Google (expérience 2) http://www.google.com/	830 (83)	16 (9)	6	778	0.937	1.00	1s
Google (expérience 3) http://www.google.com/	830 (83)	18 (10)	6	803	0.967	1.00	1s

Table 3.5 – Résultats pour les *moteurs de recherche*

la disponibilité du produit et d'autre part, un commentaire sur le DVD. Il est alors nécessaire de fournir un ensemble d'exemples suffisamment grand pour permettre l'abstraction de ces différentes informations. Un adaptateur composé de deux motifs et permettant d'extraire les 99 instances a été obtenu en donnant 8 instances issues de 5 pages.

3.5.3 FNAC

L'ensemble des documents utilisés pour évaluer le site de la FNAC contient au total 76 instances. Les trois premières pages contiennent chacune 20 instances et la dernière les 16 autres. Les résultats du site prennent des formats très variés. Tous contiennent un titre, une liste d'acteurs, une information sur le type de DVD (nombre de zones), une date de sortie et un prix. Selon les cas, une combinaison des informations suivantes peut apparaître en plus :

- un lien contenant une image et référençant une page d'avis sur le film ;
- des informations complémentaires sur le DVD (par exemple s'il s'agit d'une édition spéciale) ;
- des informations sur le public à qui s'adresse le film ;
- des informations sur la disponibilité du film.

De plus, pour chacune de ces informations, plusieurs valeurs sont possibles. Ceci a rendu la tâche d'extraction assez difficile.

Dans une première expérimentation, nous avons obtenu un adaptateur composé de 8 motifs et permettant d'extraire les 76 instances avec 21 exemples. Cependant, 17 instances erronées supplémentaires

Source	URL	Inst.	Ex.	Mot.	Récup.	Rap.	Prec.	Tmps.
Alapage	http://www.alapage.com/	84 (5)	6 (2)	3	84	1.00	1.00	2 s
Amazon	http://www.amazon.fr/	99 (7)	8 (5)	2	99	1.00	1.00	<1 s
Fnac (a)	http://www.fnac.com/	76 (4)	21 (4)	8	93	1.00	0.82	1 s
Fnac (b)	http://www.fnac.com/	76 (4)	38 (4)	20	76	1.00	1.00	5 s
Conforama	http://www.conforama.fr/	94 (26)	3 (2)	1	94	1.00	1.00	1 s

Table 3.6 – Résultats sur les catalogues

3.6 Nombre et qualité des exemples

Les évaluations précédentes nous montrent que le nombre d'exemples et la qualité de ceux-ci jouent sur la complétude de l'adaptateur obtenu et la rapidité d'obtention d'un tel adaptateur. Nous étudions ici le comportement de notre algorithme sur un même ensemble de documents dans l'objectif d'extraire la même relation mais en faisant des choix d'exemples différents.

Afin d'évaluer le comportement de l'algorithme selon les exemples qui lui sont fournis, nous avons procédé de la manière suivante pour chacune des sources étudiées. L'ensemble des instances de la source a été extrait en utilisant un adaptateur, soit créé à la main, soit en utilisant IEREL en choisissant manuellement les exemples. Nous avons vérifié que toutes les instances étaient extraites et que celles-ci étaient correctes. Ensuite, plusieurs séquences d'exemples permettant de construire un adaptateur parfait ont été construites. Deux méthodes permettant de constituer cette séquence ont été utilisées. Les deux méthodes cherchent à construire de manière itérative une séquence d'exemples. A chaque étape, un nouvel exemple est ajouté à la séquence parmi un ensemble d'exemples restant. Ensuite, un ensemble de motifs est construit en utilisant la séquence comme ensemble d'exemples. Les motifs obtenus sont appliqués aux documents de la source. Lorsque toutes les instances sont extraites, l'itération s'arrête. La différence entre les deux méthodes est l'ensemble dans lequel les nouveaux exemples sont choisis. Dans la première méthode, les exemples sont choisis parmi l'ensemble des instances non encore choisies. Dans la seconde méthode, les exemples sont choisis dans la différence entre l'ensemble des instances initiales et l'ensemble des instances extraites. Autrement dit, les nouveaux exemples sont choisis parmi les instances non couvertes.

3.6.1 Amazon

Une première source sur laquelle nous avons mené une étude du nombre d'exemples nécessaires à l'obtention d'un adaptateur parfait est Amazon. Les données traitées sont les mêmes que celle présentées précédemment dans la section 3.5 composées de 99 instances sur 7 pages. Nous avons appliqué une fois une sélection aléatoire des instances exemple sans suppression des instances couvertes. Cette exécution a permis d'obtenir un adaptateur avec 20 instances exemple. De plus, nous avons appliqué 5 fois la construction itérative d'un adaptateur en supprimant à chaque étape les instances couvertes. A chaque fois, nous avons obtenu 2 motifs permettant l'extraction de l'ensemble des 99 instances. Dans trois cas, 7 instances ont été nécessaires et, dans deux cas, 8 instances ont été nécessaires.

La figure 3.7 décrit, pour chaque exécution, l'évolution du nombre d'instances extraites par rapport au nombre d'exemples donnés. La courbe **réf** correspond à l'exécution sans suppression des instances couvertes. Sur cette courbe apparaissent des plateaux montrant que certaines instances exemple n'ont pas permis l'extraction de nouvelles instances. Il s'agissait d'instances déjà couvertes par un des motifs. Les cinq autres courbes correspondent aux cinq exécutions effectuées en supprimant au fur et à mesure

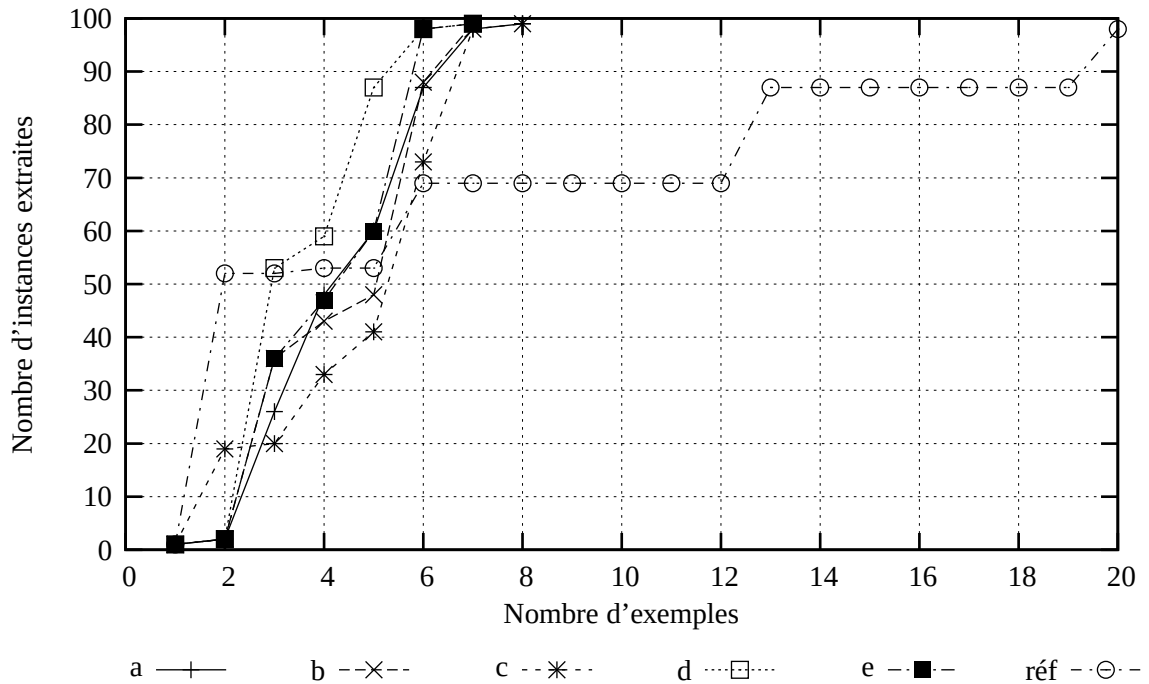


Figure 3.7 – Évaluation du nombre d'exemples nécessaire pour Amazon

les exemples couverts. En observant ces courbes nous pouvons constater qu'elles peuvent toutes se décomposer en environ 4 grands pics où un nouvel exemple permet l'extraction de 10 à 50 instances de plus. Dans les cinq courbes, les pics sont environ de même taille. Ces pics de même taille correspondent à l'apparition d'instances d'un même format dans chacune des exécutions.

Pour évaluer l'influence de la qualité des exemples, nous avons repris les 8 exemples utilisés par la troisième exécution précédente (courbe **c** dans la figure 3.7). Cinq nouvelles exécutions de notre système ont été réalisées avec 8 permutations aléatoires de ces exemples. Les exécutions ont respectivement utilisé 6, 5, 6, 5, 5 instances. Les courbes de ces 5 exécutions sont données dans la figure 3.8 avec en rappel la courbe **c**. Cette expérience montre que certains exemples permettent la construction plus rapide des motifs. En observant les courbes, on constate que **c-1** et **c-5** sont très proches, sauf sur la fin, où à partir du quatrième exemple une instance de moins est extraite par **c-1**. Par ailleurs, la courbe l'exécution **c-1** nécessite un exemple de plus. Les séquences d'exemples utilisées pour **c-1** et **c-5** sont données respectivement dans les tables 3.7 et 3.8. En observant les séquences, on constate que tous les exemples de **c-5** apparaissent dans ceux de **c-1** sauf celui du film "Je vous aime". Cette instance est de moindre qualité car en la supprimant le même adaptateur est généré.

3.6.2 Fnac

Nous avons effectué les mêmes expérimentations sur les données issues du site de la FNAC. Celles-ci ont été effectuées en conservant les balises de liens (*i.e.* les balises **a**). Une première expérience à consister à exécuter 4 fois la construction itérative d'un adaptateur pour la source. Les courbes de l'évolution du nombre d'instances extraites en fonction du nombre d'instances exemples sont données dans la figure 3.9. La première exécution (**a** dans la figure) a utilisé 41 instances exemple alors que les

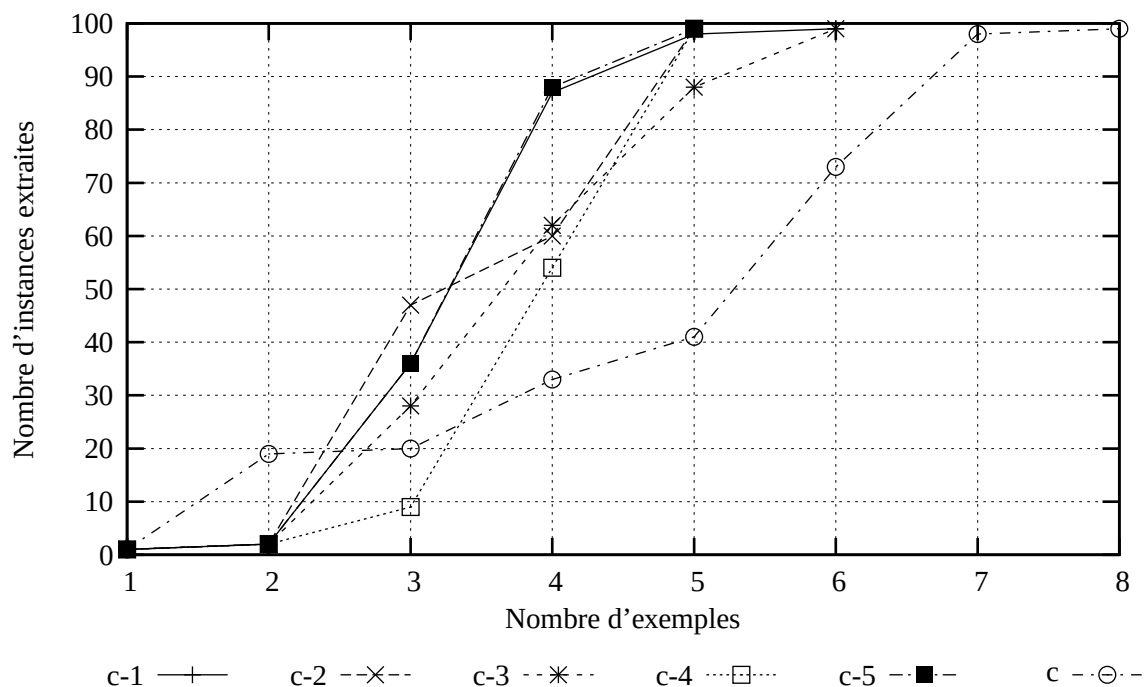


Figure 3.8 – Évaluation de la qualité des exemples

1	('Je vous aime', 'Catherine Deneuve', '12,99')
2	('Aime ton père', 'Gérard Depardieu', '22,99')
3	('Les Fugitifs', 'Pierre Richard', '26,99')
4	('On ne meurt que deux fois', 'Michel Serrault', '20,98')
5	('Le Pharmacien de garde', 'Vincent Perez', '30,35')
6	('Elisa - Edition Collector', 'Vanessa Paradis', '26,43')

Table 3.7 – Séquence d'exemples pour **c-1**

trois autres en ont utilisé 38.

La seconde expérience, consistant à évaluer l'influence de l'ordre des exemples et donc de leur qualité, a été réalisée en prenant comme ensemble d'exemples ceux utilisés lors de l'exécution **a**. Trois nouvelles exécutions ont été réalisées utilisant respectivement 36, 38 et 37 instances exemple. La figure 3.10 donne les courbes issues de ces exécutions.

Dans les courbes pour la Fnac, il est intéressant de noter que nous ne retrouvons pas des pics aussi prononcés que pour Amazon. Ceci s'explique par le fait que les données pour la Fnac contiennent un plus grand nombre de formats différents. Ainsi, il est plus difficile de construire des motifs très généraux permettant d'extraire un grand nombre d'instances.

- 1 ('Les Fugitifs', 'Pierre Richard', '26,99')
- 2 ('On ne meurt que deux fois', 'Michel Serrault', '20,98')
- 3 ('Aime ton père', 'Gérard Depardieu', '22,99')
- 4 ('Elisa - Edition Collector', 'Vanessa Paradis', '26,43')
- 5 ('Le Pharmacien de garde', 'Vincent Perez', '30,35')

Table 3.8 – Séquence d'exemples pour c-5

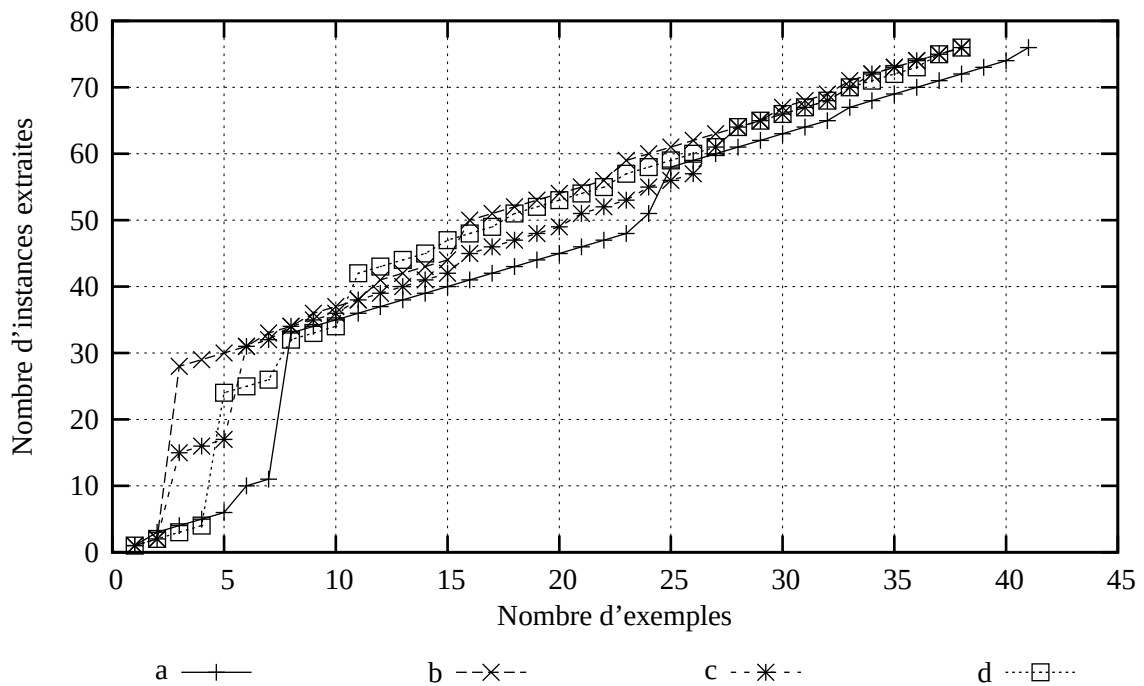


Figure 3.9 – Fnac - évaluation de l'influence du nombre d'exemples

3.7 Conclusion

Nous avons évalué notre méthode sur des données issues de différents domaines. Les résultats que nous avons obtenus montrent que notre méthode permet, dans la grande majorité des cas, de construire un adaptateur rapidement à l'aide d'une poignée d'instances exemple. De plus, la plupart du temps, cet adaptateur est parfait dans le sens où il permet d'extraire correctement toutes les instances et uniquement celles-ci. L'extraction d'instances erronées peut être corrigée à l'aide d'exemples négatifs ou du changement d'interprétation des liens hypertexte. D'autre part, selon les différents formats présents dans la source, la construction d'un motif peut s'avérer plus ou moins difficile. Nous avons montré que le nombre et la qualité des exemples jouent un rôle important dans la rapidité d'obtention d'un adaptateur parfait.

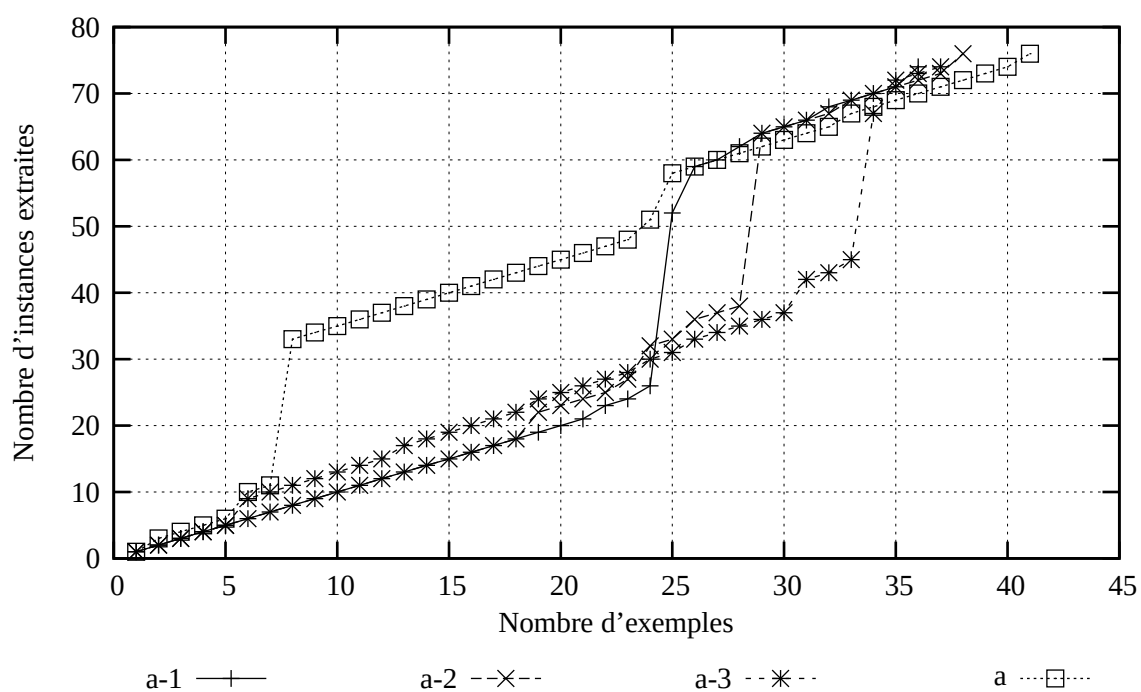


Figure 3.10 – Fnac - évaluation de la qualité des exemples

Conclusion

Dans cette partie nous avons étudié le problème de l'extraction d'informations à partir de données issues du Web. Différentes approches du problème ont d'abord été présentées :

la construction manuelle d'un adaptateur - Dans ce cas les méthodes définissent un langage souvent très riche mais laissant à l'utilisateur les tâches d'analyse des pages et d'écriture des règles. Dans certains cas, les méthodes proposent des outils d'aide à la construction des règles.

l'induction d'adaptateurs à base de pages exemple étiquetées - Ces méthodes proposent à l'utilisateur de construire un adaptateur pour une source donnée en étiquetant un ensemble de pages exemple. Elles utilisent ensuite différentes méthodes d'induction pour la construction de l'adaptateur.

l'analyse structurelle des pages - À partir d'une ou plusieurs pages, ces méthodes utilisent différentes techniques de recherche des régularités dans les pages pour la construction de motifs. Ces motifs permettent d'extraire les parties structurées d'une page correspondant souvent au contenu intéressant.

l'utilisation de connaissances domaine - Certaines méthodes utilisent une connaissance du domaine fournie par l'utilisateur. Cette connaissance décrit les formats dans lesquels apparaissent les concepts élémentaires du domaine. La connaissance et l'utilisation d'un algorithme général permettent d'identifier des concepts composés.

l'extraction à base d'instances exemple - Dans ce cas l'objectif est l'extraction d'une relation à partir du Web dans sa globalité. La méthode utilise un ensemble d'instances exemple pour construire un ensemble de motifs permettant de nouvelles extractions.

Nous avons montré que ces différentes approches ne sont pas satisfaisantes. Selon le cas, elles imposent différentes contraintes : un étiquetage fastidieux et complet d'un ensemble de pages exemples, l'analyse manuelle d'un ensemble de pages pour la construction des règles ou de connaissances et/ou la transformation des données après extraction afin de les mettre dans le format souhaité par l'utilisateur. Seule la méthode DIPRE n'impose pas de tels contraintes : les données fournies par l'utilisateur sont un petit ensemble d'instances exemple, elles sont automatiquement recherchées dans un ensemble de pages et permettent de générer un ensemble de motifs permettant l'extraction de nouvelles instances. Par contre, cette dernière méthode n'est pas adaptée à la construction d'un adaptateur. En effet, les motifs obtenus sont très spécifiques, ce qui rend difficile la construction d'un adaptateur permettant d'extraire la totalité des instances d'une source.

Nous avons proposé une nouvelle méthode qui reprend l'idée de Brin d'utiliser un ensemble d'instances exemple pour la construction de motifs. Par contre, la méthode de généralisation des contextes que nous proposons, permet la génération de motifs suffisamment généraux pour construire un adaptateur pour une source de données. Ainsi, dans la plus part des cas, un poignée d'instances exemple (souvent moins d'une dizaine pour une source) sont suffisants pour construire un adaptateur parfait pour une source donnée. L'utilisateur ne se retrouve plus face aux problèmes d'étiquetage fastidieux et complet d'un ensemble de pages. De même, il n'est pas nécessaire de faire une analyse des pages, même si un choix judicieux des exemples permet de générer plus rapidement un adaptateur parfait. Pour finir, le format des instances extraites est connu à l'avance car il s'agit du même format que les instances données en exemple.

Notre méthode a été évaluée sur différentes sources de différents domaines. Cette évaluation montre que notre méthode permet, dans la grande majorité des cas, d'obtenir un adaptateur fiable. D'autre part, elle s'adapte aux besoins de l'utilisateur en lui permettant de spécifier la relation qu'il souhaite extraire. Par ailleurs, une étude a été menée permettant de déterminer l'influence du choix des exemples sur la rapidité d'obtention d'un adaptateur.

PARTIE II

Construction d'applications d'extraction d'informations à partir du Web

Introduction

Nous avons vu jusqu'ici comment construire un programme permettant d'extraire d'un ensemble de pages Web issues d'une même source de données, les informations qu'elle contiennent. Cependant, ceci ne permet pas de donner un accès totalement automatisé. En effet, l'extraction n'est qu'une partie du problème. Il est aussi nécessaire de décrire comment interroger la source et comment parcourir l'ensemble des résultats. Selon les sources de données, ceci peut nécessiter une "infrastructure" plus ou moins complexe. Par exemple, dans le cas d'une source classique, il est nécessaire (1) de remplir un formulaire, (2) de soumettre le formulaire et télécharger la première page de résultats, (3) d'extraire de cette page les résultats, (4) suivre un lien menant à une nouvelle page de résultats, *etc.* Par contre, pour certaines sources, ce schéma d'exécution est trop rigide.

Dans cette partie, nous étudions le problème de la construction d'applications Web permettant d'accéder de manière automatisée à un ensemble d'informations sur le Web. Pour ce faire, nous avons étudié comment accéder à un ensemble de sources de données Web. Cette étude nous a permis de mettre en avant un ensemble d'opérateurs génériques pour la construction d'applications Web. Nous avons alors défini un langage, WETDL, permettant de décrire une tâche d'extraction d'informations en la décomposant en un ensemble d'opérateurs. Ceux-ci sont composés en un réseau qui pourra être exécuté dans le but de réaliser la tâche décrite. Ainsi, nous proposons deux algorithmes permettant d'effectuer cette exécution. Le langage proposé s'appuyant sur les technologies XML et la généralité des opérateurs proposés rendent le langage très flexible. Ainsi, n'importe quelle tâche peut être simplement décrite. Par ailleurs, nous avons implémenté le système WEBSOURCE permettant l'exécution de toute tâche décrite en WETDL. Ainsi, en utilisant ce système, un accès automatisé à différentes sources de données Web a été réalisé.

De plus, nous montrons comment le langage WETDL et le modèle de composition d'opérateurs peuvent être adaptés. En effet, nous proposons d'étendre WETDL afin de donner la possibilité d'intégrer l'accès à un Web Service dans une tâche d'extraction. Par ailleurs, nous montrons que le langage WETDL permet non seulement de décrire l'accès à des sources de données Web mais qu'il est aussi possible de décrire une tâche aussi complexe que le parcours intelligent du Web, nécessitant l'accès à plusieurs documents, leur filtrage à différentes étapes, *etc.*

Cette partie est organisée comme suit. Dans le chapitre 4, nous décrivons comment accéder à différentes sources de données Web. Dans le chapitre 5, nous présentons le langage WETDL. Dans le chapitre 6, nous donnons les descriptions WETDL des tâches d'extraction d'informations que nous avons exécutées à l'aide du système WEBSOURCE. Le chapitre 7 étudie le problème de l'intégration des Web Services et de l'extraction d'informations. Finalement, le chapitre 8 propose une méthode de parcours du Web intelligent basée sur des opérateurs WETDL.

CHAPITRE 4

Applications d'extraction d'informations

Dans ce chapitre, nous nous intéressons au problème de la construction d'applications d'extraction d'informations à partir du Web. L'objectif derrière toute application d'extraction d'informations est la réponse à un besoin d'information générique et/ou récurrent ou à un besoin nécessitant un ensemble de traitements répétitifs et générant un ensemble de données volumineuses. Par ailleurs, les données de l'application en entrée comme en sortie devront être dans un format standard afin de faciliter l'intégration de l'application dans un plus gros système.

Il existe plusieurs types de tâches d'extraction d'information. Ces types dépendent de l'objectif de l'application générée. Par exemple, il peut s'agir de l'intégration de l'accès à différents sites Web, tels que Amazon, la FNAC et Alapage, dans un système permettant leurs accès simultanés. Deux types d'applications classiques sont l'intégration d'une source Web dans une application ou dans un service de médiation de données et la construction d'une base de données à partir d'informations extraites du Web.

Un exemple simple de besoin d'information peut être d'avoir le cours actuel d'une valeur boursière. Il est *générique* dans le sens où la valeur boursière n'est pas précisée et *récurrent* dans le sens où l'objectif est de pouvoir régulièrement répondre au besoin dans le temps. D'un autre côté, une des tâches d'extraction de donnée volumineuse est par exemple l'extraction des brevets d'un site diffusant une masse de brevets afin d'effectuer une analyse de ceux-ci. L'objectif dans ce cas est d'automatiser l'extraction d'un grand volume de données. En effet, il est difficilement imaginable de parcourir manuellement l'ensemble des pages d'un tel site et d'en extraire manuellement toutes les données.

Dans ce chapitre, nous allons présenter en détails quelques exemples d'applications d'extraction d'informations à partir du Web. Dans les chapitres suivants, nous verrons comment de telles applications peuvent être construites.

4.1 Accéder à une source Web

Un premier type de tâche d'extraction d'informations à partir du Web est l'automatisation de l'accès à une source de données en ligne. Dans ce cas, la tâche consiste à construire un programme capable de prendre en entrée une requête de l'utilisateur correspondant au domaine d'application de la source et de renvoyer l'ensemble des réponses à cette requête obtenues par l'interrogation de la source. Par contre, les format des requêtes et des résultats définis par l'utilisateur sont très probablement différents de ceux de la source de données. Notamment le format des données résultats prend la forme d'un ensemble de pages HTML alors que l'utilisateur souhaite la plupart du temps les avoir dans un format plus structuré

(eg. un format XML qu'il à lui-même défini). Ainsi, il faut pouvoir, d'une part, transformer les requêtes de l'utilisateur dans un format compréhensible par la source et, d'autre part, transformer les réponses de la source dans le format choisi par l'utilisateur.

Dans le cas d'une source de données Web typique, le point d'entrée de la source est une page contenant un formulaire HTML. Le formulaire rempli est alors traduit en une requête qui est envoyée au serveur de la source. La réponse à cette requête est un document HTML contenant les premières réponses à la requête posée. Sur la première page résultat et les suivantes se trouve un lien permettant d'accéder à une nouvelle page de résultats. Beaucoup de sources Web suivent ce même schéma d'exécution.

Automatiser l'accès à une source nécessite de construire une requête que le serveur de la source sache interpréter à partir des données fournies par l'utilisateur. L'exécution de cette requête donnera alors lieu à la génération d'un document contenant les premiers résultats et un lien vers un document contenant les résultats suivants. Il faut donc extraire de ce document, d'une part, les résultats qu'il contient et, d'autre part, le lien vers le document résultat suivant. De ce nouveau document, il est aussi nécessaire d'extraire les résultats et le lien "suivant", etc. Le processus s'arrête lorsque la dernière page ne contenant pas de lien "suivant" est rencontrée. Automatiser l'accès à de telles sources revient alors à définir :

1. comment transformer les données de l'utilisateur en une requête pour la source ;
2. comment extraire les résultats d'un document généré à partir de la requête ;
3. comment extraire l'URL de la page de résultats suivante.

Les sources Web utilisent le protocole HTTP. Une requête respectant ce protocole est composée de trois parties : une URL, une méthode et un ensemble paramètres. L'URL est une référence vers le programme distant qui en s'exécutant avec les paramètres de la requête génère les pages résultats. La méthode permet de définir le moyen utilisé pour le transfert des paramètres entre le client et le serveur. La méthode est soit GET, soit POST¹. Les paramètres sont un ensemble de couples clé-valeur. Ils permettent aux utilisateurs de spécifier leur besoin. Lors d'un accès manuel à la source, la valeur associée à chaque clé est obtenue via les champs d'un formulaire HTML. Dans le cas d'une automatisation de l'accès à la source, les valeurs des paramètres devront être obtenues à partir des données fournies par l'utilisateur. En supposant que ces données prennent aussi la forme d'un ensemble de clé-valeur il est nécessaire de savoir comment faire la correspondance entre les clés des données de l'utilisateur et les clés utilisées par la source.

Ainsi une source simple peut être décrite par les éléments suivants :

- l'URL de la source ;
- la méthode de la source ;
- une fonction de correspondance entre les données de l'utilisateur et les paramètres de la source ;
- une fonction d'extraction du lien "suivant" dans une page de résultat ;
- un adaptateur pour la source permettant d'extraire les résultats.

Étant donné ces informations, il est possible de définir un algorithme générique d'accès à une source simple. L'algorithme 4.1 décrit comment cet accès générique s'effectue. Il fait appel à deux fonctions auxiliaires :

- *construire_requete* qui à partir d'une URL, d'une méthode, et d'un ensemble de paramètres construit une requête HTTP ;
- *executer* qui à partir d'une requête HTTP, l'exécute, c'est à dire qu'elle télécharge la page référencée par la requête.

D'autre part, deux fonctions dépendantes de la sources sont utilisées :

¹Le protocole HTTP permet d'autres valeurs pour la méthode tels que PUT, mais ces méthodes ne sont pas utiles pour ce qui nous intéresse : obtenir des documents contenant des informations à extraire

Procédure *query_source***Entrées**

- L'URL u de la source à accéder
- La méthode m HTTP utilisé par la source
- Une fonction de correspondance c
- Une fonction n permettant de trouver le lien "suivant"
- Un adaptateur ω pour la source
- Une requête q de l'utilisateur

Sorties

- Une liste de résultats R

Soit $r \leftarrow \text{construire_requete}(u, m, c(q))$

Soit $fini \leftarrow 0$

tantque $\neg fini$ **faire**

 Soit $d \leftarrow \text{executer}(r)$

$R \leftarrow R \cup \omega(d)$

$l \leftarrow n(d)$

$fini \leftarrow (l = \emptyset)$

si $\neg fini$ **alors**

$r \leftarrow \text{premier}(l)$

fin si

fin tantque

Algorithme 4.1 – Accès générique à une source classique

- n qui permet d'extraire d'une page de résultats le lien vers la page de résultats suivante ;
- ω un adaptateur pour la source qui permet d'extraire d'une page les résultats qu'elle contient.

Avec cet algorithme, l'accès à une source Web est facilité. Il suffit alors de déterminer les paramètres spécifiques à la source. Par contre, bien que ces paramètres soient suffisants pour décrire beaucoup de sources de données Web, ils ne sont pas suffisants pour décrire toutes les sources Web existantes. En effet, ce schéma d'exécution n'est pas suffisamment générique. Toutes les sources ne présentent pas forcément leurs données sur un ensemble de pages résultats reliées par un lien "suivant". Par exemple, certaines génèrent comme première page un index de tous les résultats, avec chaque résultat sur une page séparée. D'autres nécessitent avant toute exécution d'une requête, de suivre une série de liens afin de conserver des informations liées à une session. Ainsi, pour permettre de construire des applications Web, il est nécessaire de prendre en compte tous ces aspects. Dans la section suivante, nous décrivons comment s'effectue l'accès à différentes sources de données Web. Ces exemples illustrent les différents problèmes pouvant être rencontrés.

4.1.1 Interrogation d'un moteur de recherche : Google

Le cas simple et classique d'application d'extraction d'informations à partir du Web est celle d'extraire les résultats d'un moteur de recherche tel que Google. La figure 4.1 décrit l'accès manuel au moteur Google. En (1) l'utilisateur saisit sa requête dans le formulaire, en (2) l'utilisateur extrait les résultats, en (3) l'utilisateur clique sur le lien "Suivant" qui l'amène sur une nouvelle page de résultats. Cette tâche

d'extraction est un exemple de tâche d'extraction simple. L'URL de la source est <http://www.google.fr/> et la méthode est de type GET, c'est à dire que les paramètres sont transmis directement dans l'URL. La source n'a qu'un seul paramètre obligatoire : le paramètre `q` dont la valeur est l'ensemble de termes formant la requête de l'utilisateur. La fonction d'extraction, permettant d'extraire les informations qui répondent au besoin de l'utilisateur, peut, par exemple, être construite avec le système IERE l décrit dans la première partie de cette thèse. Finalement le lien "Suivant" est l'unique lien d'une page de résultats contenant la chaîne `Suivant`.

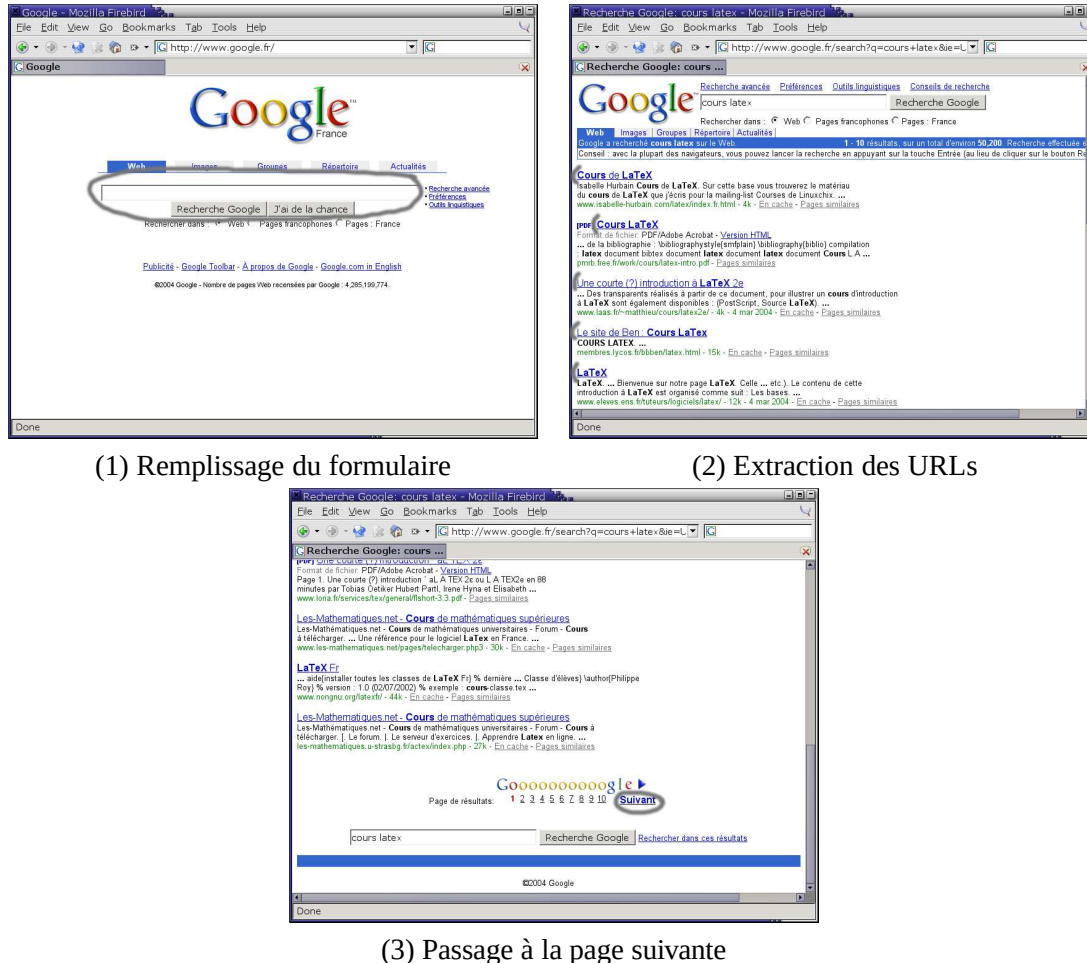


Figure 4.1 – Extraction des réponses de Google exécutée manuellement

Dans la tâche la plus simple, seules les URLs des documents résultats nous intéressent. Cependant, dans certains autres cas, l'utilisateur peut vouloir accéder aux informations concernant le document tels que les extraits proposés par le moteur, la taille du document, *etc.* Certaines de ces informations sont fournies par le moteur de recherche sur les pages résultats. Par contre d'autres informations qui ne sont pas mises à disposition par le moteur peuvent intéresser l'utilisateur. Par exemple, il se peut que l'utilisateur souhaite connaître le type de document que référence l'URL résultat. Cette information peut être obtenue en interrogeant les serveurs sur lesquels chacune des pages se trouve sans télécharger le document lui-même en utilisant la méthode HEAD du protocole HTTP². En effet, un serveur HTTP interrogé

²Les types que proposent parfois Google pour les documents sont obtenus uniquement sur la base des suffixes des docu-

avec cette méthode ne donne que les informations sur le document demandé sans en donner le contenu. En utilisant cette information, il devient alors même possible de compléter le service rendu par un moteur de recherche en définissant, par exemple, une tâche d'extraction ne revoyant comme résultat d'une requête sur le moteur que les documents PDF (*i.e.* dont le type MIME est `application/pdf`). De telles tâches d'extraction d'informations dérivées ne peuvent être décrites comme des applications simples.

4.1.2 Accéder à la base de DVD chez Amazon.fr

Un autre exemple de tâche d'extraction d'informations classique est l'interrogation d'un site de e-commerce tel que `amazon.fr`. L'objectif, dans ce cas, est de construire une application capable de répondre à une requête en faisant appel au site e-commerce. Par exemple, dans le cas d'`amazon.fr`, l'application devra pourvoir répondre à la requête suivante : “Quels sont les titres, nom de l'acteur principal et prix des DVD disponibles où Depardieu apparaît comme acteur?”. La réponse devra être dans un format structuré, par exemple sous forme d'instances de la relation *dvd*(titre, acteur_principale, prix).

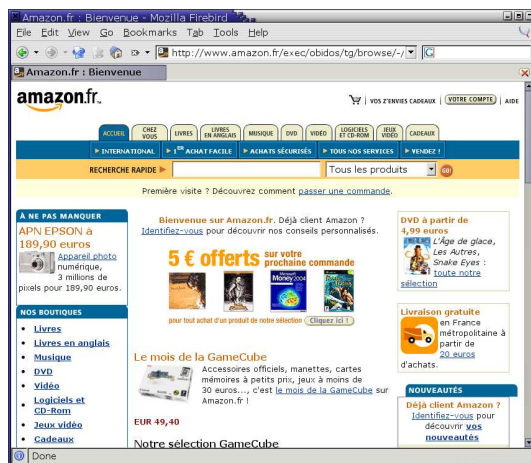
La figure 4.2 montre comment s'effectue cette tâche à la main. En (1) l'utilisateur se connecte sur la page index d'Amazon.fr et sélectionne le lien pointant sur la page DVD. Il est ensuite dirigé vers la page spécifique au DVD (2). Celle-ci ne permettant que la saisie de requêtes sous forme de mots clés, il clique sur le lien permettant d'aller sur la page de requêtes avancées (3). Ensuite il entre les informations exprimant son besoin d'information dans le formulaire proposé. La soumission de ce formulaire, génère une page de résultats (4) de laquelle l'utilisateur pourra alors extraire les informations qu'il souhaite et se constituer la liste des DVD répondant à son besoin d'information.

Dans un premier temps, cette méthode semble être accessible en la décrivant comme une tâche d'extraction classique. En effet, il suffirait de reprendre les informations du formulaire tels que l'URL de la source et les noms des paramètres, de la page de recherche avancée. Cependant le passage initial par la page index du site et le suivi des liens jusqu'à la page du formulaire de requête avancée est nécessaire car Amazon génère une clé de session lors de l'accès à la page index. Toutes les pages suivantes contiennent des liens uniquement valables pour la session. Hors d'une session, les pages d'interrogation du service sont alors inaccessibles. Dans ce cas, il n'est pas possible d'automatiser la tâche comme une tâche simple, bien qu'une fois la page du formulaire atteint le schéma d'exécution est semblable à celui d'une tâche simple. Dans ce cas, pour automatiser l'accès à la source, une solution consiste à écrire un programme qui effectue le parcours de la page index jusqu'à la page de requête avancée, en extrait l'URL, et construit la requête HTTP permettant d'obtenir la première page de résultats et effectue le cycle d'extraction et de suivi de liens “Suivant”.

4.1.3 Accéder à une base d'offres de mobilier chez Conforama

Le site `conforma.fr` propose la mise en ligne des produits vendus par le magasin Conforama. L'accès aux informations sur ce site nécessite un schéma d'exécution complexe différent du schéma classique. La figure 4.3 décrit comment est effectué la recherche d'un produit sur ce site. Dans l'exemple le produit recherché est un réfrigérateur. En (1) l'utilisateur accède à la page index du site sur laquelle se trouve un formulaire. Il remplit le formulaire avec sa requête “réfrigérateur”. En soumettant le formulaire il est emmené sur la page (2) qui lui présente différentes catégories contenant des réponses à sa requête. Sur cette page, chaque catégorie est associée à un lien menant sur une page de résultats. Par exemple

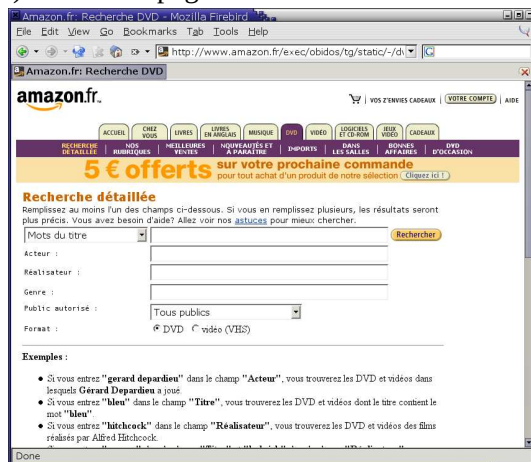
ments. Hors pour connaître le type réel d'un document généré, par un script CGI par exemple, il est nécessaire d'interroger le serveur qui le contient.



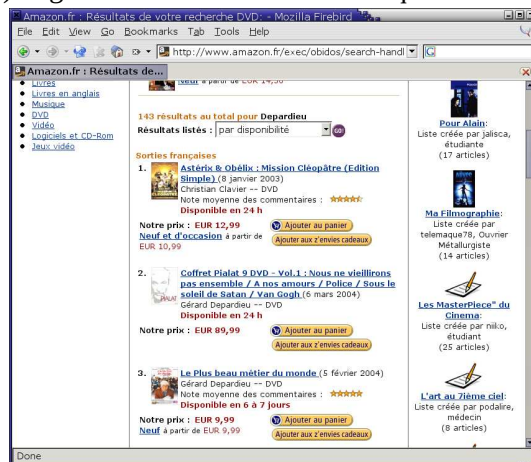
(1) Accès à la page index et suivi du lien “DVD”



(2) Page “DVD” et suivi du lien “requête avancée”

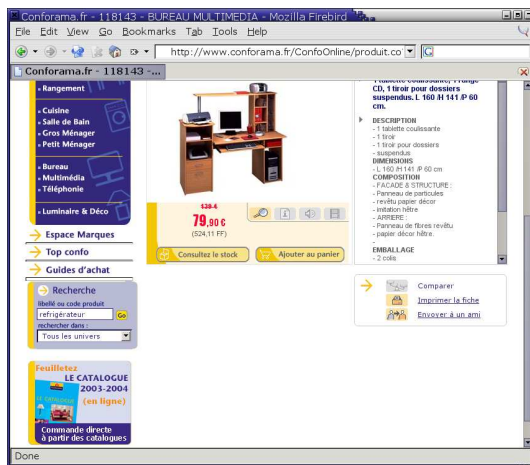


(3) Remplissage du formulaire et soumission

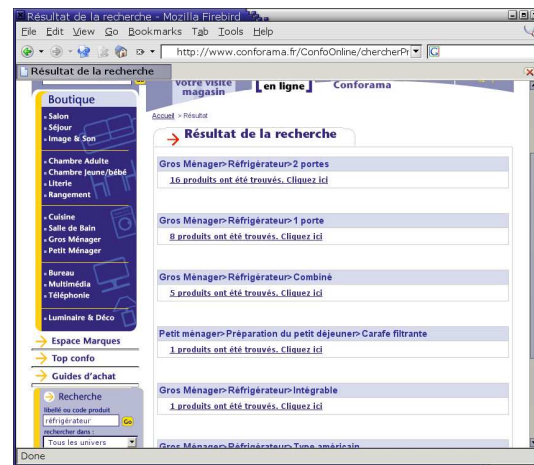


(4) Extraction des résultats

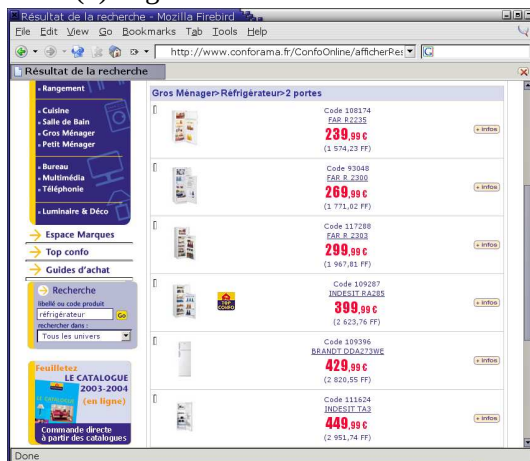
Figure 4.2 – Accès manuel à Amazon.fr



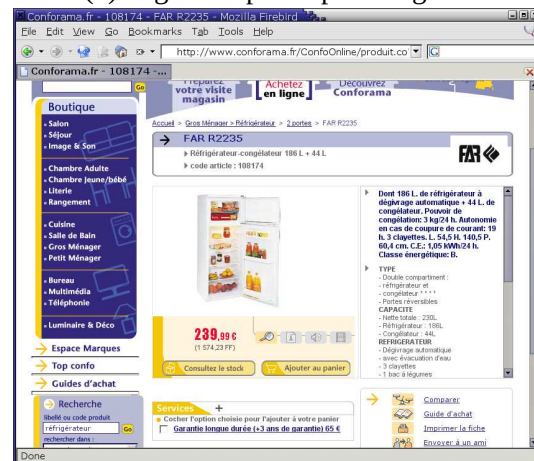
(1) Page index avec un formulaire



(2) Page de réponse par catégorie



(3) Début de liste de résultats pour une catégorie



(4) Détails d'un résultat

Figure 4.3 – Accès manuel à Conforama.fr

l'utilisateur clique sur le lien associé à la première catégorie. Ceci l'emmène sur une page contenant une liste des 10 premiers résultats pour la catégorie. Un lien permet d'accéder à la page de résultats suivants. Chaque résultat, ne présente qu'un descriptif court du produit. Un lien associé permet d'avoir le détail de chaque produit.

Ainsi, pour accéder à cette source il est nécessaire de :

1. Simuler le remplissage du formulaire (construction d'une requête HTTP) ;
2. Soumettre le formulaire (exécuter la requête HTTP) ;
3. Extraire de la page index des catégories de résultats, les liens vers les pages de résultats pour chaque catégorie ;
4. Télécharger chaque page initiale de résultats ;
5. Extraire des pages résultats les liens vers les pages de détails des produits ;
6. Extraire des pages résultats le lien vers la page de résultats suivante.

Ainsi, de la même manière que les précédentes, cette tâche ne peut être décrite à l'aide d'un schéma classique. En effet, les pages de résultats ne sont pas directement accessibles car il est nécessaire de

passer par une page index. Ensuite, les détails des produits ne se trouvent pas directement sur la page des résultats : à chaque fois il est nécessaire de suivre un lien supplémentaire.

4.1.4 Synthèse

La construction d'un accès à une source de données Web n'est pas évidente. En effet, plusieurs problèmes peuvent se poser. Selon la manière dont a été organisé le site, l'accès, d'une part, au formulaire et, d'autre part, aux données résultats peut ne pas être immédiat. Par exemple, dans le cas d'Amazon, il est nécessaire d'effectuer un parcours depuis la page index du site jusqu'au formulaire. Dans le cas de Conforama, si les résultats de la requête appartiennent à plusieurs catégories, une page présentant ces catégories est d'abord proposée à l'utilisateur. Ainsi, l'automatisation de l'accès à de tels sites ne peut se décrire comme un simple formulaire et une succession de pages résultats. Construire un accès à de telles sources, nécessite de pouvoir faire des parcours plus complexes.

4.2 Extraction d'informations pour la constitution de bases de données

Une tâche d'extraction d'informations peut avoir comme objectif l'extraction et la mise en forme d'un nombre important de données disponibles sur internet afin d'en constituer une base de données. Les objectifs de la construction d'une telle base de données peuvent être divers. Par exemple, on peut avoir comme objectif de faire une analyse des données extraites, de stocker les informations dans une base de connaissances pour une exploitation future, *etc.* Pour des raisons de temps et de coûts évidents le volume de données à extraire rend difficilement envisageable d'effectuer manuellement des telles opérations.

4.2.1 Extraction de brevets

Un premier exemple est l'extraction d'informations sur des brevets pour leur analyse. L'objectif est de construire une base de données de brevets en utilisant une source disponible sur internet. Dans cette source, chaque brevet est décrit sur une page contenant diverses informations telles que le titre du brevet, les inventeurs, la date de dépôt, la classification internationale, *etc.* Ces pages sont rendues accessibles au travers d'un formulaire HTML. Cependant l'accès aux pages n'est pas direct. Il se fait par l'intermédiaire d'un ensemble de pages contenant un lien vers les pages brevet. Ainsi pour construire une base de données à partir d'une telle source il est nécessaire :

1. De remplir le formulaire HTML.
2. De soumettre ce formulaire.
3. Pour chaque page index, de suivre chaque lien vers une page de description de brevet.
4. Pour chaque page index, de suivre le lien vers la page index suivante.
5. D'extraire de chaque page de description de brevet, les informations souhaitées.
6. D'intégrer les informations extraites dans la base de données.

Réaliser cette tâche manuellement, que ce soit en partie ou totalement, est fastidieux et sujet à erreur dû au temps nécessaire pour parcourir l'ensemble des pages. En effet, pour l'unique classification internationale "B27N" 163 réponses existent. Pour extraire les descriptions des 163 brevets, il est nécessaire de parcourir 167 pages, 4 pages indexes et 163 pages de description de brevets. Une autre classe "G09B"

contient 1287 brevets et nécessite donc le parcours de 1312 pages. Une autre solution est la construction d'un script ad-hoc permettant d'automatiser cette extraction. Il devra alors, à partir d'une requête donnée, permettre d'effectuer les différentes étapes décrites précédemment.

4.2.2 CIA world fact book

Sur le site de la CIA est proposé le World Fact Book, un “livre des faits” sur les pays du monde. Il s'agit d'une source de donnée souvent citée en exemple dans les communautés d'extraction et de recherche d'informations. Elle propose diverses informations géographiques, politiques *etc.*. Comme toutes les autres sources d'informations du Web, les données sont présentées sur plusieurs pages : une page index et une page par pays. Avoir un accès à ce type de site peut être intéressant dans l'objectif de la construction d'une base de connaissances. Dans le cas du World Fact Book, ceci nécessite de parcourir chacune des pages décrivant un pays. Le site propose une page index avec pour chaque pays un lien vers la page décrivant ce pays. Dans ce cas, il est nécessaire :

1. D'extraire de la page index l'ensemble des liens vers les pages de description de pays.
2. D'extraire de chaque page de description de pays les informations souhaitées.

De la même façon que pour les brevets, extraire manuellement les informations d'un tel site peut être long et fastidieux dû au nombre de pages. De plus, lorsque les données sont mises à jour il est nécessaire d'extraire à nouveau les données du site. Ainsi il est intéressant d'automatiser l'accès à un tel site pour constituer une base de connaissances et la maintenir à jour. Ceci peut être fait en écrivant un script permettant de faire l'extraction. Toutefois, toute modification du site nécessitera une réécriture du script.

4.2.3 Construction d'une base de conférences

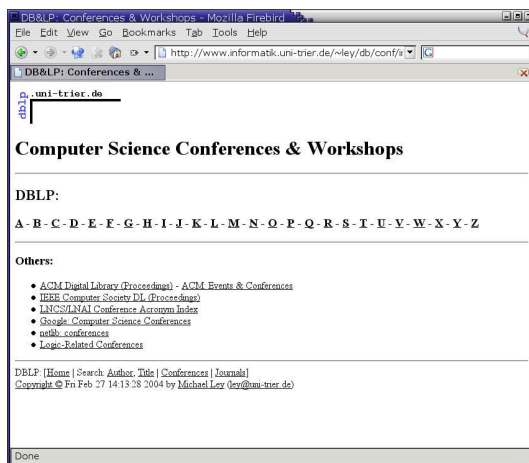
Le site dblp.uni-trier.de est un site d'information bibliographique sur les conférences et journaux majeurs en informatique. Il propose notamment une liste des conférences et une liste des journaux. Pour chaque conférence ou journal, une page contient la liste de leurs éditions et pour chaque édition, une page contenant la liste des articles qui y sont parus avec leurs informations bibliographiques. Ainsi à partir de cette source, il est possible d'en extraire des informations pour créer une base locale. Pour diverses raisons, il peut être intéressant de constituer une base de données sur les conférences.

La figure 4.4 décrit comment accéder manuellement aux descriptions des conférences sur dblp.uni-trier.de. En (1) l'utilisateur accède à une première page index. Cette page contient un lien pour chaque lettre de l'alphabet menant sur une page index spécifique pour cette lettre. En (2) l'utilisateur accède à la page index d'une des lettres. Sur celle-ci se trouve une liste avec un lien par conférence. En suivant ce lien, l'utilisateur est dirigé, en (3), vers une page récapitulant les différentes éditions des conférences.

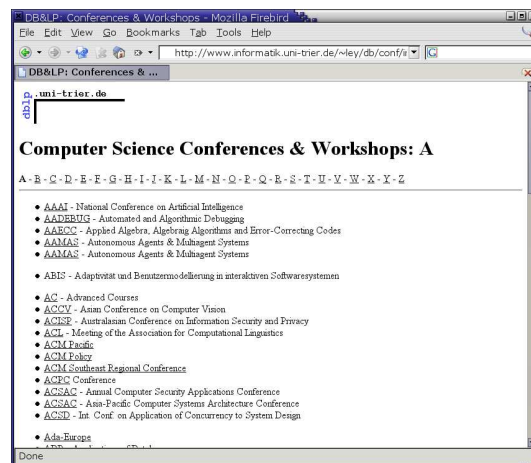
Une fois de plus, effectuer une telle tâche manuellement est fastidieux. De la même manière que pour les deux tâches précédentes, une telle tâche peut être automatisée par l'écriture d'un script spécifique. Toutefois, une restructuration du site nécessitera d'écrire à nouveau le script.

4.2.4 Synthèse

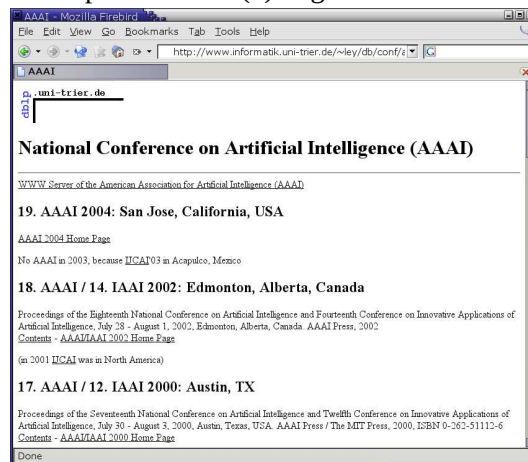
Différentes sources de données riches en informations et régulièrement mises à jour sont proposées sur le Web. Cependant, elles contiennent toutes un grand nombre d'informations qu'il est difficilement envisageable d'extraire manuellement. Une solution consiste à écrire pour chaque site duquel on souhaite extraire des informations, un script permettant d'automatiser l'accès au site et de remplir la base de



(1) Page index par lettre de l'alphabet



(2) Page index des conférences pour une lettre donnée



(3) Les des éditions d'une conférence

Figure 4.4 – Accès manuel à dblp.uni-trier.de

données et toute modification du site nécessitera une réécriture du script. Cette solution peut être viable lorsqu'il s'agit d'automatiser l'accès à quelques sites. Par contre, ceci est difficilement envisageable dans un contexte où un grand nombre d'accès doivent être construits et maintenus à jour.

4.3 Interrogation de service multiples

L'un des objectifs de l'extraction d'informations est de permettre la médiation d'information. Cette médiation peut prendre deux formes : regrouper des sources complémentaires pour offrir un service plus large, ou, au contraire, regrouper des services concurrentiels permettant d'offrir un service plus complet. Dans cette section, nous proposons un exemple de chacun des deux types. L'objectif de la première application est de permettre l'interrogation de deux services d'annuaire en lignes pour deux pays différents. Ceci permet d'offrir un nouveau service permettant la recherche d'adresses dans les deux pays. La seconde application a pour but d'offrir les propositions de ventes de DVD de trois sources de données. Ceci permet à l'utilisateur d'interroger simultanément les trois services et ainsi d'avoir une vision plus globale de l'offre qui lui est faite.

Dans la suite nous représentons graphiquement les schémas des applications. Ces schémas sont décrits à l'aide de représentations graphiques de graphes orientés. Les sommets des graphes peuvent prendre deux formes : celle d'un losange et celle d'un rectangle dont les deux côtés gauches et droits sont doublés. Un losange représentant l'agrégation et la duplication de données : toute donnée arrivant par n'importe quelle entrée est dupliquée et transmise à l'ensemble des sorties. Lorsqu'un nombre n apparaît dans un losange, une donnée ne sera transmise qu'à sa n -ième apparition en entrée. Un rectangle représente une fonctionnalité qui est exécutée lorsque des données arrivent en entrée et qui transmet les résultats à sa sortie. Les arcs orientés représentent la transmission de données d'un sommet vers un autre. Dans certains cas une condition peut apparaître sur l'arc. Dans ce cas la donnée n'est transmise que si elle vérifie la condition.

4.3.1 Annuaire en ligne

L'objectif ici est l'interrogation de sources de données multiples offrant un service complémentaire. Par exemple le site pagejaunes.fr propose un annuaire en ligne pour la France alors que superpages.com propose un annuaire pour l'ensemble des États-Unis. Combiner ces annuaires permettrait d'obtenir un service proposant un annuaire pour la France et les États-Unis. En ajoutant l'accès à des annuaires d'autres pays il devient possible de proposer un service d'annuaire composé de sources existantes.

Donner un tel accès nécessite de définir un langage de requête initial commun. Dans le cas d'un annuaire le langage de requête devra permettre de spécifier les nom, prénom, ville, province/état, et pays de la personne recherchée. Selon la valeur du champ pays tel ou tel source devra être interrogée. D'autre part chacun des services devra retourner ses résultats dans le même format. Ceci peut être obtenu soit en effectuant pour chaque source une extraction qui donne des résultats structurés dans le format final ou en passant par un format intermédiaire qui sera transformé dans un format final.

La figure 4.5 exprime le schéma de l'application permettant l'interrogation des deux services. En entrée de l'application, une requête est soumise par l'utilisateur. Selon le pays celle-ci est envoyée soit à pagesjaunes.fr soit à superpages.com. Les résultats sont ensuite retournés par l'application.

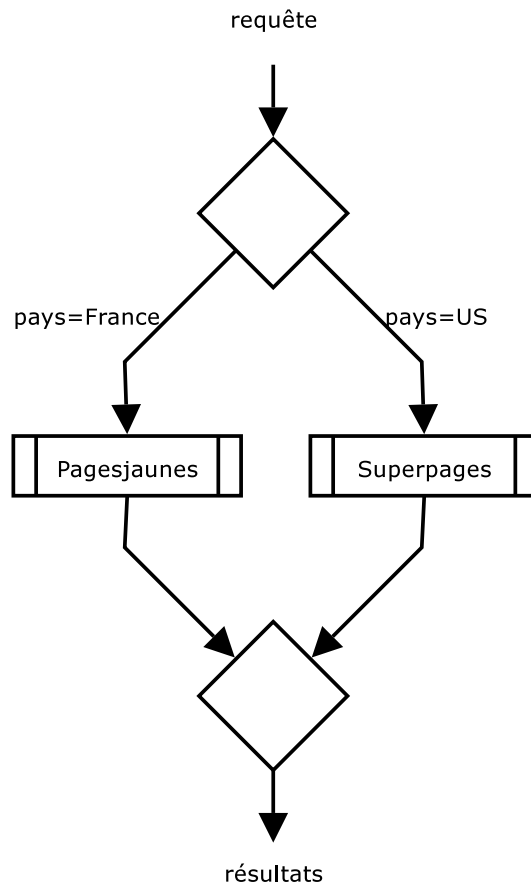


Figure 4.5 – Combinaison de l'accès à deux annuaires en ligne

4.3.2 Comparateur de prix de DVD

Un autre exemple d'application interrogeant de multiples services est celui d'un comparateur de prix. Un exemple d'application permettant d'obtenir pour une requête donnée les prix des DVD de sources telles que amazon.fr, alapage.com et fnac.com. Toutefois chacune de ces sources possède son propre formulaire avec son ensemble de champs et de valeurs possibles pour ces champs. Ainsi, pour pouvoir interroger les sources de manière identique, il sera nécessaire de définir un langage commun. Une requête dans ce langage devra alors être transformée pour chaque source en un ensemble de couples clé-valeurs permettant de remplir les champs du formulaire. Ensuite, en supposant qu'une application permettant l'accès à chacune des sources a été préalablement créée, les requêtes transformées devront être transmises à chaque application. Les résultats obtenus de chaque source pouvant être dans des formats différents, il sera nécessaire de transformer ces données dans un même format.

La figure 4.6 donne le schéma d'une telle application. En entrée, arrive la requête de l'utilisateur. Celle-ci est transmise à chacune des trois sources. Les résultats obtenus par chacune des sources sont alors regroupés et retournés.

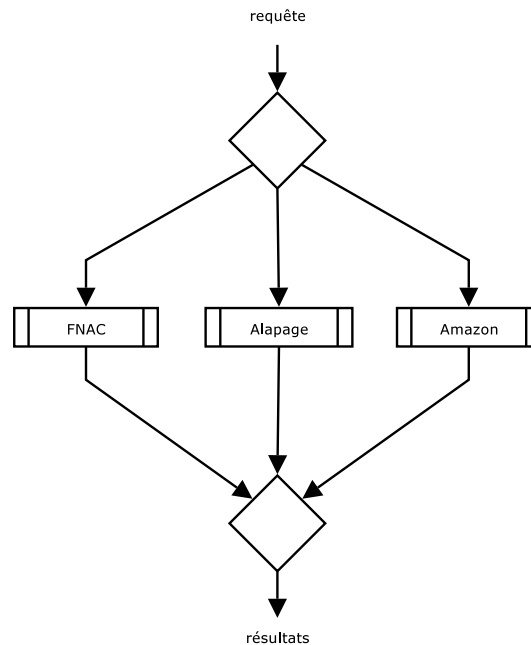


Figure 4.6 – Mise en oeuvre d'un comparateur de prix de DVD

4.3.3 Meta-moteur de recherche

Un meta-moteur de recherche est un moteur de recherche combinant les résultats de plusieurs autres moteurs de recherche. Par exemple, un meta-moteur peut faire appel aux moteurs de recherches Google et AltaVista et ne renvoyer que les réponses qui sont fournies par les deux moteurs. Un autre exemple de tâche pourrait être d'accéder à n moteurs de recherches et de ne renvoyer que les réponses obtenues par un pourcentage p donné de moteurs. Pour définir un tel moteur il est d'abord nécessaire de construire une application d'extraction pour chacun des moteurs de recherche.

La figure 4.7 décrit une application utilisant trois moteurs : Google, AltaVista et Yahoo. Au départ une requête est transmise à l'application. Celle-ci est alors redirigée vers chacun des moteurs. Chaque moteur calcule la liste des résultats. Ensuite, les résultats apparus plus de trois fois sont retournés.

4.3.4 Conclusion

Sur le Web, beaucoup de sources offrent des services similaires ou complémentaires. Combiner l'accès à de telles sources permet de multiples applications intéressantes. Les opérations nécessaires sont récurrentes. Il est nécessaire de pouvoir diffuser des données à plusieurs sources, d'agréger les résultats provenant de ces mêmes sources et dans certains cas filtrer les données.

4.4 Conclusion

Dans ce chapitre, nous avons décrit différentes applications d'extraction d'informations. La diversité de ces applications ne permet pas d'avoir un schéma classique unique d'accès aux sources Web. Toutefois, nous avons constaté qu'un certain nombre d'opérations apparaissent de manière récurrente dans

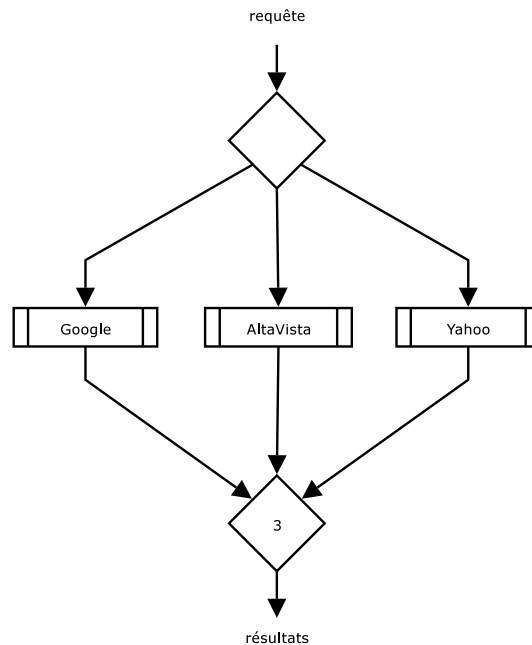


Figure 4.7 – Mise en oeuvre d'un méta moteur de recherche

chacune des tâches. Par exemple, toutes ces applications nécessitent le téléchargement de pages Web, l'extraction d'informations contenues dans ces documents, le filtrage des informations extraites, *etc.* Les opérations s'enchaînent les unes après les autres traitant les données résultant de l'opération précédente.

Malgré ces similitudes, dans l'état actuel, construire de telles applications n'est pas une chose évidente. La seule possibilité de mise en oeuvre de ces tâches est d'écrire des scripts spécifiques pour chaque tâche. Malheureusement, de tels scripts sont difficilement adaptables pour l'écriture de nouvelles tâches. Ceci est dû au fait que chacune des tâches possède ses spécificités propres. Toutefois, mieux exploiter les similitudes mais en permettant des différences, permettrait de bénéficier du potentiel du Web comme la plus grande base de données mondiale.

CHAPITRE 5

Langage WetDL

Dans le chapitre précédent nous avons décrit un ensemble d'applications d'extraction d'informations. Nous avons vu que pour créer de telles applications il n'est pas suffisant de construire un adaptateur pour chacune des sources. En effet, il est aussi nécessaire de mettre en oeuvre une certaine "infrastructure" autour de l'extraction. Dans le cas d'Amazon, par exemple, nous avons vu qu'il était nécessaire d'effectuer plusieurs sous-tâches telles que télécharger la page index, suivre un parcours de liens jusqu'à un formulaire, remplir le formulaire, *etc.*

Une solution immédiate pour construire une telle infrastructure est de la programmer dans un langage quelconque. Cette programmation n'est pas évidente et nécessite une connaissance précise du fonctionnement du Web et peut être longue et fastidieuse. De plus, programmer en "dur" une application d'extraction d'informations permet difficilement de l'adapter. Dans le cadre d'un système nécessitant un accès régulier à de nombreux sites, la maintenance de simples scripts devient d'autant plus difficile.

Nous proposons pour la construction de tâche d'extraction d'informations de décomposer l'application en un ensemble d'opérateurs d'extraction d'informations génériques et paramétrables. La coordination de ces opérateurs permet alors d'obtenir une application d'extraction d'informations opérationnelle. Pour décrire de telles tâches nous proposons la définition d'un langage permettant de décrire une tâche d'extraction d'informations. Celui-ci doit permettre de décrire chacune des opérations de l'application ainsi que leur coordination. Nous proposons un langage XML nommé WETDL (Web Extraction Task Description Language). L'apport principal de ce langage est qu'il permet de décrire des tâches d'extractions d'information à un niveau d'abstraction plus élevé qu'un langage de programmation classique. Ainsi, l'utilisateur du langage n'a pas à se soucier de détails d'implémentation tels que comment doit s'effectuer le téléchargement, la gestion d'éventuels informations de session *etc.* Le langage s'appuie sur la notion d'opérateurs de base permettant d'effectuer les tâches de bas niveau et sur la coordination de ces opérateurs. Ainsi pour décrire une tâche, il suffit de déterminer l'ensemble des opérateurs nécessaires, leur paramétrage et leur coordination. Cette solution permet au concepteur de l'application d'extraction d'informations de se concentrer sur les aspects spécifiques à une source, les parties génériques étant déjà implémentées dans les opérateurs. Par exemple, pour l'extraction d'un lien suivant il lui suffira de préciser une expression permettant de retrouver ce lien dans une page sans avoir à se préoccuper de comment il sera extrait.

Les exemples du chapitre précédent montrent que plusieurs opérations sont récurrentes. Parmi celles-ci nous avons le téléchargement de documents, l'extraction de sous-parties de documents, *etc.* Une caractéristique commune de ces opérations est qu'elles prennent toutes un objet (un document, une URL, *etc.*) en entrée et renvoient plusieurs objets en sortie. D'autre part, ces opérateurs s'enchaînent les uns après les autres en formant un certain schéma d'exécution. Le langage WETDL permet de décrire les opérateurs d'une tâche, de les paramétrer pour cette tâche et de décrire le schéma d'exécution de la tâche.

5.1 Types, objets et données

Afin de pouvoir définir nos opérateurs d'extraction d'informations, il est d'abord nécessaire de spécifier les types de données que nous allons manipuler. De manière générale, nous utiliserons le terme *objet* pour faire référence à un élément d'un des types définis ci-après.

Un premier type d'objets que nous serons amené à manipuler sont des URL. Il s'agit de références vers des documents accessibles sur le Web. Certaines chaînes de caractères bien formées pourront être transformées en URL. Ceci sera par exemple le cas lorsqu'un lien "suivant" sera extrait d'une page Web.

Définition 5.1 (URL). *Le type `URL` est l'ensemble des objets faisant référence à un document sur le Web. Un tel objet peut être décomposé en cinq parties : un protocole (`HTTP`, `FTP`, etc.), une adresse de serveur, un répertoire, un nom de document et une partie requête.*

Les requêtes HTTP sont les messages qu'envoient les clients HTTP à un serveur HTTP. Ceux-ci se composent d'une adresse (une URL sans partie requête), d'un type de méthode (parmi `GET`, `POST` et `HEAD`) et d'une partie requête. Toute URL peut être transformée en une requête HTTP.

Définition 5.2 (HTTPRequest). *Le type `HTTPRequest` est l'ensemble des tuples $\langle B, M, Q \rangle$ formant une requête HTTP. B est une URL sans partie requête, M est une méthode (`GET`, `POST`, `HEAD`), Q est un ensemble de paramètres sous la forme de couples clé-valeur.*

La réponse résultant d'une requête HTTP prend la forme d'un document. Ce document peut être dans différents formats. Toutefois, nous nous intéressons principalement aux documents au format HTML ou son dérivé le XHTML. Par la suite, nous ne considérerons que les documents XHTML respectant les règles syntaxiques du XML permettant de ce fait l'utilisation d'outils basés sur des standards développés pour XML tel que le modèle objet document (DOM).

Nous considérons deux représentations des documents : l'une "sérialisée" prenant la forme d'une chaîne de symboles respectant les règles syntaxiques du XML, l'autre prenant la forme d'un arbre DOM en mémoire. Ainsi nous avons deux types de documents : les documents XML et les documents DOM selon la représentation.

Définition 5.3 (DocumentXML). *Le type `DocumentXML` contient toute chaîne de caractères respectant les règles syntaxiques de la spécification XML.*

Définition 5.4 (DocumentDOM). *Le type `DocumentDOM` contient les représentations sous forme d'arbres DOM des documents XML.*

Dans la spécification DOM, un document est composé de parties appelées des noeuds. Étant donné un document XML, tout élément, attribut, partie textuelle, partie CDATA, commentaire est un noeud. Dans ce qui suit, nous ne nous intéresserons qu'aux parties communes regroupées sous le type de `NoeudDOM` qui correspond à la classe `Node` de la spécification DOM.

Définition 5.5 (NoeudDOM). *Un `NoeudDOM` est une partie d'un `DocumentDOM`. Il peut lui-même être décomposé en sous-parties qui sont aussi des `NoeudDOM`.*

5.2 Décomposition d'une tâche d'extraction d'informations

Pour décrire une tâche d'extraction d'informations en WETDL il est d'abord nécessaire d'identifier les opérateurs nécessaires. En observant les diverses applications d'extraction d'information qu'il est

possible de construire, dont certaines ont été illustrés dans le chapitre précédent, il est possible de définir un ensemble d'opérateurs typiques. Nous en avons identifié plusieurs :

- la construction de requête HTTP à partir d'une requête utilisateur ;
- le téléchargement d'une page référencée par une requête HTTP ou une URL ;
- l'extraction de parties d'une page ;
- la transformation de données issues d'une page ;
- le filtrage de données.

Chacun de ces opérateurs a la particularité d'appliquer un traitement à un objet en entrée et de retourner une liste d'objets en sortie. De plus, elles sont génériques dans le sens où elles sont paramétrables pour une tâche donnée. Par exemple, la tâche d'extraction prend un document HTML en entrée et renvoie une liste ordonnée des parties vérifiant une expression donnée en paramètre. Dans la suite, nous décrivons en détail ses paramètres.

5.2.1 Construction d'un requête HTTP : opérateur query

Entrée Une requête dans un langage Q_U (langage utilisateur) ;

Sortie Une `HTTPRequest` pour la source

Paramètres

- L'URL de la source ;
- La méthode utilisée par la source ;
- Une fonction t de transformations de Q_U en un ensemble de couples clé-valeur compréhensibles par la source ;
- Un ensemble de valeurs par défaut pour la requête.

L'interrogation manuelle d'une source Web se fait généralement via un formulaire HTML. Ce formulaire est ensuite interprété par le navigateur en une requête HTTP. Le formulaire permet la saisie d'un ensemble de paires clé-valeur. D'autre part, il définit la méthode HTTP à utiliser (GET ou POST) ainsi que l'URL du script qui donnera la réponse à la requête. Cet ensemble d'informations est nécessaire pour la construction de la requête HTTP.

Un opérateur de construction de requêtes HTTP permet la simulation du remplissage d'un formulaire et de son interprétation. Les valeurs permettant de remplir les champs du formulaire peuvent être soit spécifiées par les paramètres de l'opérateur, soit être obtenues par les données en entrée de l'opérateur. Généralement, ces valeurs sont issues d'une requête transmise à l'application d'extraction.

La plupart des applications d'extraction d'informations cherchent à extraire des informations répondant à une requête de l'utilisateur. Le langage dans lequel est exprimée cette requête peut dépendre de l'application ou du contexte dans lequel l'application sera utilisée. Par exemple, dans le cas de la recherche de DVD sur Amazon, une requête pourra être spécifiée en donnant un nom d'acteur, un titre de film, etc. Seulement, le langage utilisé par l'application n'est généralement pas le même que celui utilisé par la source Web. De plus, il peut être nécessaire d'avoir des paramètres avec des valeurs par défaut pour une source Web donnée. Par exemple, certains sites proposent des produits de catégories différentes. Dans le cadre d'une application précise seule une de ces catégories intéresse l'utilisateur. Dans ce cas, cette catégorie pourra être spécifiée à l'aide d'un paramètre.

5.2.2 Exécution d’une requête HTTP : opérateur `fetch`

Entrée Une `HTTPRequest`.

Sortie Une liste vide si la requête a échoué ou une liste contenant le `DocumentXML` résultat.

Afin d’accéder au contenu d’une page résultant d’une requête ou référencée par une URL, il est nécessaire de télécharger cette page. Ceci nécessite l’exécution d’une requête HTTP. La réponse à cette requête est un document.

L’opérateur d’exécution d’une requête HTTP `fetch` permet, à partir d’une URL ou d’une requête HTTP, de télécharger le document référencé. Cet opérateur ne nécessite aucune information autre que l’URL ou la requête HTTP. La séparation explicite de la construction d’une requête HTTP de son exécution permet de réutiliser l’opérateur d’exécution. Ceci permet notamment d’avoir un même opérateur téléchargeant la première page d’une requête et les suivantes. En effet, autant la première page est issue d’un formulaire que les autres sont simplement des liens extraits des pages résultats précédentes.

5.2.3 Analyse d’un document : opérateur `parse`

Entrée Un `DocumentXML`.

Sortie Une liste vide si l’analyse du document échoue ou une liste contenant le `DocumentDOM` obtenu par l’analyse.

Les documents renvoyés par l’opérateur `fetch` prennent la forme d’une chaîne brute. A fin de pouvoir leur appliquer des traitements, telle que l’extraction de certaines parties ou leur transformation, il est nécessaire d’en avoir une version décomposée en parties.

Le rôle de l’opérateur `parse` est d’analyser un document XML sous forme de chaîne et de retourner sa représentation DOM. Cette représentation permettra d’accéder et de traiter les différentes parties du document.

5.2.4 Extraction de sous-parties : opérateur `extract`

Entrée Un `DocumentDOM`.

Sortie Une liste de `NoeudDOM`, parties référencées du document.

Paramètres

- Une fonction *s* de sélection des parties intéressant l’utilisateur.

A partir d’un document, seules certaines parties du document peuvent intéresser l’utilisateur. Par exemple, dans le cas de l’extraction de DVD d’Amazon, l’utilisateur peut ne s’intéresser qu’aux descriptions des DVD. L’objectif d’une tâche d’extraction est de décomposer le document qu’il reçoit en entrée en une liste de sous parties. Ces sous-parties sont celles qui correspondent au besoin de l’utilisateur, à des informations pouvant être utiles dans la suite d’une tâche, *etc.*. Par exemple, une tâche d’extraction peut être définie pour extraire un lien “page suivante” dans le cas de sites proposant leurs résultats sur plusieurs pages.

Il existe plusieurs manières de faire référence aux sous parties d’un document. Dans le cas de documents XML, il existe des langages permettant de faire référence à de telles parties. Par exemple, le langage XPath permet de donner une expression de chemin spécifiant un ensemble nœuds de l’arbre associé au document XML.

5.2.5 Transformation de données : opérateurs `transform`

Entrée Un `DocumentDOM`.

Sortie Un `DocumentDOM` résultat de la transformation.

Paramètres • Une fonction de transformation t

Lors d'une tâche d'extraction d'informations à partir du Web, il arrive souvent qu'il soit nécessaire d'opérer des changements de formats. Le plus souvent, il s'agit de transformer les résultats dans le format que souhaite l'utilisateur. Cependant il peut arriver que les résultats d'une sous-tâche nécessitent d'être transformés pour pouvoir être utilisés par une autre sous-tâche. Le rôle d'une tâche de transformation est d'effectuer ces changements de formats.

Dans le cas de données XML, le moyen le plus simple de décrire une transformation est de la décrire sous forme d'une feuille de style XSLT. Ainsi, la transformation effectuée par un opérateur `transform` peut être décrite à l'aide d'une feuille de style XSLT.

5.2.6 Filtrage de données : opérateur `filter`

Entrée Un `NoeudDOM`.

Sortie Une liste vide ou une liste contenant le `NoeudDOM`.

Paramètres

- un prédicat p déterminant si le noeud en entrée doit être conservé ou non.

Dans certains cas, tous les résultats d'une tâche ne sont pas utiles pour les tâches suivantes ou comme résultat final. Ainsi, des tâches de filtrage peuvent être utiles afin de sélectionner les résultats pertinents. Par exemple, dans le cas de l'interrogation d'un moteur de recherche, l'utilisateur peut n'être intéressé que par les documents de type PDF. Dans ce cas, une tâche de filtrage des résultats après leur extraction devra être effectuée. Un filtre peut se définir par un prédicat qui teste la pertinence ou non d'un objet. Étant donné un prédicat p , l'opération de filtrage consiste à appliquer p à son entrée et selon la vérité du prédicat, retourner ou non l'objet en entrée. Décrire un opérateur de filtrage revient à décrire son prédicat. Celui-ci peut, par exemple, être défini par une série de tests que doivent vérifier les objets.

5.2.7 Modules externes : opérateur `external`

Entrée Dépendant de l'opérateur

Sortie Dépendant de l'opérateur

Paramètres Dépendant de l'opérateur

Afin de pouvoir étendre plus facilement notre système, nous avons prévu d'inclure un opérateur spécial, l'opérateur `external`. Celui-ci est défini par le nom du module qui sera chargé de traiter les données envoyées à l'opérateur. Dans le cadre de la construction d'applications d'extraction d'information, ce mécanisme de modules externes nous permettra, par exemple, d'utiliser des modules spécifiques générés par IEREL.

5.2.8 Synthèse

Dans cette section, nous avons défini un ensemble d'opérateurs permettant la description de tâches d'extraction d'informations. La table 5.1 donne un récapitulatif de ces opérateurs, leurs paramètres, leurs entrées et leurs sorties. Dans la section suivante, nous verrons comment coordonner de tels opérateurs dans l'objectif de réaliser une tâche d'extraction.

Nom	Paramètres	Entrées	Sorties
query	URL, méthode, paramètres par défaut	Query	HTTPRequest
fetch	–	HTTPRequest	DocumentXML
parse	–	DocumentXML	NoeudDOM
extract	expression XPath	NoeudDOM	NoeudDOM
filter	ensemble de tests	NoeudDOM	NoeudDOM
transform	feuille de style	NoeudDOM	NoeudDOM
external	dépendant de l'opérateur	dépendant de l'opérateur	dépendant de l'opérateur

Table 5.1 – Opérateurs d'extraction d'informations

5.3 Coordination des tâches

Les opérateurs précédemment décrits sont les bases pour la construction d'une chaîne de traitements permettant d'accomplir une tâche d'extraction d'informations. Une fois ces opérateurs définis et paramétrés, il est nécessaire de les coordonner. Chaque opérateur reçoit des données, leur applique un traitement et renvoie une liste de résultats qui deviendront à leur tour des données pour une ou plusieurs opérations dans la chaîne de traitement. Par exemple, dans le cas de l'application d'interrogation de Google, la donnée initiale est un ensemble de mots clés que l'utilisateur souhaite soumettre au moteur. La première étape dans la chaîne de traitement de la tâche est de construire une requête HTTP à laquelle Google sache répondre. Cette requête HTTP constitue une nouvelle donnée qui devra être transmise à Google via un opérateur de téléchargement. En exécutant la requête HTTP, l'opérateur de téléchargement obtient un document HTML en réponse. A celui-ci doivent être appliquées deux traitements séparés, l'extraction des résultats qu'il contient et l'extraction du lien vers la page de résultat suivante.

5.4 Description des opérateurs

Afin de décrire un réseau d'opérateurs permettant d'effectuer une tâche d'extraction d'informations, nous proposons un nouveau langage XML : WETDL. Une description dans ce langage se compose d'un ensemble de descriptions d'opérateurs, de leur paramétrage et de leur coordination. Dans WETDL, chacun des opérateurs est décrit par un élément XML dont le nom correspond au type d'opérateur. Chacun des éléments possède des attributs communs à tous les opérateurs (`name` et `forward-to`) et d'autres spécifiques au type d'opérateurs qu'ils décrivent. L'annexe B donne le détail de la DTD définissant le langage WETDL.

5.4.1 query : opérateur de construction de requête

```
1 <wt:query name="q" method="GET" forward-to="fetcher">
2   <base>http://www.google.fr/search</base>
3   <parameters>
4     <param name="q" value-of="requete" />
5   </parameters>
6 </wt:query>
```

Figure 5.1 – Exemple de description d’un opérateur de construction de requête HTTP

L’élément `query` permet de spécifier un opérateur de construction de requêtes HTTP. Pour ce faire, il est nécessaire de préciser comment obtenir les trois informations permettant de construire une requête HTTP : l’URL, la méthode et les paramètres. L’élément `query`, en plus des attributs communs à tous les éléments décrivant des opérateurs, possède :

- un attribut `method` permet de spécifier la méthode à utiliser ;
- un sous-élément `base` permet de spécifier l’URL de la source ;
- un sous-élément `parameters` contenant un ou plusieurs éléments `param` permet de définir les paramètres de la requête HTTP et d’où sont issues les informations.

Chaque élément `param` contient un attribut `name` dont la valeur sera le nom d’une clé des paramètres de la requête HTTP. Deux autres attributs peuvent apparaître `default` et `value-of`. L’attribut `default` permet de spécifier une valeur par défaut pour le paramètre. L’attribut `value-of` permet de décrire la partie de l’objet en entrée qui sera utilisée comme valeur pour le paramètre.

La figure 5.1 donne un exemple de déclaration d’un opérateur de construction de requête HTTP. Dans cet exemple, il s’agit d’un opérateur prenant en entrée une requête n’ayant qu’un couple clé-valeur dont le nom de la clé est “requete” et la valeur est la liste des termes avec lesquelles l’utilisateur souhaite interroger Google. La requête générée aura comme méthode GET, l’URL de la source où la requête devra être envoyée est `http://www.google.fr/search` et la requête HTTP aura un unique paramètre dont la clé sera nommée “q” et dont la valeur sera celle du paramètre “requete” issue des données en entrée de l’opérateur.

5.4.2 fetch : opérateur de téléchargement

```
1 <wt:fetch name="fetch-query-page"
2   forward-to="parse-query-page" />
```

Figure 5.2 – Exemple de description d’un opérateur de téléchargement

L’élément `fetch` permet de décrire un opérateur de téléchargement. Mis à part les attributs `name` et `forward-to` communs à tous les opérateurs, cet opérateur ne nécessite aucun paramétrage. Toutefois, plusieurs opérateurs `fetch` peuvent être utiles dans une même tâche d’extraction. En effet, toutes les données obtenues en entrée sont dirigées vers une sortie déterminée par l’attribut `forward-to`, or le traitement que doit subir le document téléchargé dépend du type de document. Par exemple, un l’opérateur de téléchargement d’une page index devra transmettre ses résultats à un opérateur d’extraction des liens alors que l’opérateur téléchargement d’une page résultat devra transmettre ses résultats à un opé-

rateur d'extraction des résultats. La figure 5.2 donne un exemple de déclaration d'un opérateur `fetch` nommé "fetch-query-page" qui transmet ses résultats à l'opérateur nommé "parse-query-page".

5.4.3 parser : opérateur d'analyse XML

```
1 <wt:parser name="parser" forward-to="next"/>
```

Figure 5.3 – Exemple de description d'un opérateur d'analyse syntaxique

La description un opérateur d'analyse syntaxique d'un document XML se fait à l'aide de l'élément `parser`. À la manière d'un opérateur `fetch`, un opérateur `parser` n'a pas de paramétrage particulier. Plusieurs opérateurs d'analyse différents peuvent toutefois être nécessaires dans une même chaîne de traitement, pour les même raisons évoquées pour l'opérateur `fetch`. La figure 5.2 donne un exemple de déclaration d'un opérateur `fetch` nommé "parser" qui transmet ses résultats à l'opérateur nommé "next".

5.4.4 filter : opération de filtrage

```
1 <ws:filter name="link-filter" type="tests" forward-to="url-extract">
2   <test select="." match="publication" action="reject" />
3   <test select="." match="call for papers" />
4   <test select="@href" match="cfp" />
5   <test select="@href" match="call" />
6 </ws:filter>
```

Figure 5.4 – Exemple de description d'un opérateur de filtrage

Un opérateur de filtrage est décrit par un élément `filter`. Un filtre est composé d'un ensemble de tests. Ceux-ci sont décrits par l'élément `test` sous l'élément `filter`. Un teste donné se compose de deux parties : une expression XPath décrite par l'attribut `select` et permettant de sélectionner une partie de l'objet reçu en entrée et une expression régulière décrite à l'aide de l'attribut `match` qui doit reconnaître la valeur issue de l'application de l'expression XPath pour que le test soit réussi. D'autre part, pour chaque test, deux actions sont possibles lorsque le test réussit : conserver l'objet reçu en entrée ou le rejeter. Cette action est spécifiée à l'aide de l'attribut `action`. Par défaut, l'action est de conserver l'objet si le test réussit et dans ce cas il est inutile de spécifier la valeur de l'attribut `action`.

La figure 5.4 donne un exemple de description d'un opérateur de filtrage. L'opérateur décrit permet de sélectionner dans un document les liens faisant référence à un appel à communication pour une conférence à l'aide de tests heuristiques. Les objets reçus en entrée sont des liens (ie. des éléments `a` ayant un attribut `href`). Dans l'exemple, quatre tests sont décrits. Les deux premiers utilisent le contenu du texte du lien pour décider si l'objet est un lien vers une page contenant à appel à communication alors que les deux autres testent le contenu de l'URL référencée. En effet, l'expression XPath "." lorsqu'elle est interprétée comme une chaîne renvoie le contenu textuel de l'élément courant (le lien dans notre cas) et l'expression "@href" renvoie la valeur de l'attribut `href` de l'élément courant. Ainsi le premier test

vérifie si le lien contient comme sous-chaîne la chaîne "publication" auquel cas le lien n'est pas conservé. Le second test vérifie si le lien contient la chaîne "call for papers" et l'accepte le cas échéant. Les deux autres tests vérifient si l'URL référencée par le lien contient respectivement la chaîne "cfp" ou la chaîne "call".

5.4.5 **extract** : opérateur d'extraction

```
1 <wt:extract name="query-page-finder" forward-to="fetch-query-page">  
2   <path>//a[contains(., "recherche")]/@href</path>  
3 </wt:extract>
```

Figure 5.5 – Exemple de description d'un opérateur d'extraction

L'élément **extract** permet de décrire un opérateur d'extraction. La spécification des parties à extraire se fait à l'aide d'une expression XPath contenue dans un élément **path** sous l'élément **extract**.

La figure 5.5 donne un exemple de description d'un opérateur d'extraction. L'opérateur décrit recherche dans tout objet reçu en entrée les sous-parties vérifiant l'expression suivante :

`//a[contains(., "recherche")]/@href.`

C'est à dire qu'il extrait l'attribut **href** de tous les éléments **a**, descendant de la racine du document auquel appartient l'objet en entrée et dont le contenu contient la chaîne "recherche".

5.4.6 **transform** : opérateur de transformation

L'élément **transform** permet de spécifier un opérateur de transformation. Sous cet élément se trouve une feuille de style XSL. La feuille de style sera appliquée à tout objet reçu en entrée par l'opérateur. Le résultat généré sera retourné par l'opérateur. La figure 5.6 donne un exemple de description d'un opérateur de transformation. Les objets en entrées sont composés d'un élément racine nommé **result** et contiennent différents attributs tels que les attributs **Last-Modified**, **type**, **baseuri**, *etc.* Ces objets sont alors transformés en un élément **page-info** ayant un sous élément **url** obtenu à partir de l'attribut **baseuri**, puis un élément **type** obtenu à partir de l'attribut **type**. Si l'élément **result** contient un attribut **Last-Modified** alors un élément **last-modified** est ajouté avec comme contenu la valeur de l'attribut. De même, si l'élément **result** contient un attribut **Content-Length** alors un élément **size** est ajouté sous l'élément **page-info** avec comme contenu la valeur de l'attribut **Content-Length** divisée par 1024 pour obtenir une valeur en kilo-octets, la valeur originale étant en octets.

5.4.7 **external** : opérateur externe

L'élément **external** permet de déclarer un opérateur externe. La spécification du module à utiliser se fait à l'aide de l'attribut **module**. Le paramétrage de l'opérateur se fait à l'aide d'éléments se trouvant sous l'élément **external** et d'éventuels attributs. Évidemment ces paramètres dépendent du module utilisé. Ainsi, la forme exacte de la description des paramètres est dépendante du module. La figure 5.7 donne un exemple de description d'un opérateur externe. Le nom du module qui devra être

```

1 <wt:transform name="select" forward-to="output"
2   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:stylesheet version="1.0">
4     <xsl:output method="xml" indent="yes" />
5     <xsl:template match="result">
6       <page-info>
7         <url><xsl:value-of select="@baseuri" /></url>
8         <type><xsl:value-of select="@type" /></type>
9         <xsl:if test="@Last-Modified != "">
10          <last-modified><xsl:value-of select="@Last-Modified" /></last-modified>
11        </xsl:if>
12        <xsl:if test="@Content-Length != "">
13          <size><xsl:value-of select="number(@Content-Length) div 1024" /></size>
14        </xsl:if>
15      </page-info>
16    </xsl:template>
17  </xsl:stylesheet>
18 </wt:transform>

```

Figure 5.6 – Exemple de description d'un opérateur de transformation

chargé par le système est `WebFox::TermExtractor`. Les paramètres dans ce cas sont spécifiés à l'aide d'un élément nommé `parameters`. Toutefois cette convention est spécifique à module externe `WebFox::TermExtractor`.

5.5 Coordination des opérateurs

La coordination des opérateur se fait par la définition d'une fonction de transmission des résultats. Cette fonction associe à un opérateur σ un ensemble d'opérateurs $\mu(\sigma)$ auxquels les résultats de σ devront être transmis. Par exemple, soit les opérateurs σ_f , σ_e et σ_n permettant respectivement de télécharger une page, d'extraire les résultats d'une page et d'extraire le lien suivant. Pour que les pages référencées par les URL reçues par σ_f soit transmises à σ_e et σ_n et que les liens extraits par σ_n soient transmis à σ_f , μ doit être défini de la manière suivante :

- $\mu(\sigma_f) = \{\sigma_e, \sigma_n\}$
- $\mu(\sigma_e) = \{\}$
- $\mu(\sigma_n) = \{\sigma_f\}$

Ainsi, pour définir la fonction μ d'une tâche d'extraction, il suffit d'associer à chaque opérateur l'ensemble des opérateurs auxquels il devra transmettre ses résultats. L'attribut `forward-to` que tout opérateur possède indépendamment de son type joue ce rôle. La description de la tâche fictive précédente est donnée dans la figure 5.8.

A partir d'une fonction de transitions μ donnée et d'un opérateur σ_i , il est possible de définir deux ensembles : l'ensemble $\text{cons}(\sigma_i)$ (qui correspond à $\mu(\sigma_i)$) des consommateurs de σ_i et l'ensemble $\text{prod}(\sigma_i)$ des producteurs de σ_i . Ces deux notions se définissent formellement comme suit :

Définition 5.6 (Consommateurs). *Étant donné un opérateur σ_i et une fonction de transition μ l'ensemble*

```

1 <ws:external name="term-extract"
2   module="WebFox::TermExtractor" forward-to="format">
3   <parameters>
4     <param name="query">
5       <ws:set-attribute name="value" value-of="query" />
6     </param>
7     <param name="min-col-occur" value="5" />
8     <param name="min-occur" value="3.5" />
9     <param name="max-occur" value="10000" />
10    <param name="min-perc-doc" value="0.4" />
11  </parameters>
12 </ws:external>

```

Figure 5.7 – Exemple de description d'un opérateur de traitement externe

```

1 <wt:task name="example">
2   <ws:fetch name="f" forward-to="e n" />
3   <ws:extract name="n" forward-to="f" />
4   <ws:extract name="e" forward-to="" />
5 </wt:task>

```

Figure 5.8 – Description d'une tâche fictive

des consommateurs de σ_i est l'ensemble

$$\text{cons}(\sigma_i) = \mu(\sigma_i)$$

Définition 5.7 (Producteurs). *Étant donné un opérateur σ_i et une fonction de transition μ l'ensemble des producteurs de σ_i est l'ensemble*

$$\text{prod}(\sigma_i) = \{\sigma_j | \sigma_i \in \mu(\sigma_j)\}$$

5.6 Définition d'un réseau WetDL

Une tâche d'extraction d'informations peut donc être décrite par un réseau d'opérateur interagissants. Formellement, un réseaux WetDL est un cinq-uplet $\langle D, S, i, f, \mu \rangle$ où D est un domaine de valeurs, $S = \{\sigma_1, \dots, \sigma_n\}$ est un ensemble d'opérateurs, i est l'indice de l'opérateur initial et f est l'indice de l'opérateur final et $\mu : S \rightarrow \mathcal{P}(S)$ est une fonction de transition. Chacun des opérateurs σ_i prend ses valeurs dans D et renvoie une séquence d'éléments de D , i.e. $\sigma_i : D \rightarrow L(D)$ où $L(X)$ représente le domaine des listes construites à partir d'éléments de l'ensemble X .

5.7 Modèle d'exécution

Un réseau WetDL représente un flux de données. Le flux de contrôle est déterminé par le modèle d'exécution. Un modèle d'exécution détermine comment une tâche décrite par un réseau WETDL sera

réalisé. À un instant donné, plusieurs opérations peuvent être effectuées. Le modèle d'exécution permet de déterminer comment sont faits ces choix. Nous proposons deux modèles d'exécutions. La philosophie de premier consiste à n'exécuter une opération que lorsque celle-ci est nécessaire : la philosophie "paresseuse". L'autre consiste à effectuer toute opération dès que suffisamment de données sont disponibles et de transmettre immédiatement les informations. Selon les impératifs de la tâche d'exécution (*i.e.* nombre de réponses souhaitées, mode d'accès, *etc.*) l'une ou l'autre des philosophies pourra être utilisées. Nous proposons un algorithme permettant l'exécution d'un réseau WetDL utilisant chacune des deux philosophies.

5.7.1 État d'un réseau WETDL

Afin de pouvoir construire des algorithmes exécutant un réseau WETDL il est nécessaire de définir la notion d'état du réseau. A chaque opérateur σ_i est associée une file de résultats s_i . L'état du réseau WETDL est défini par les contenus des s_i . Lorsqu'un opérateur reçoit des données, il les traite en leur appliquant l'opération et les stocke dans la file de résultats associés à l'opérateur. Par exemple si un opérateur σ_i reçoit une donnée x alors la file s_i associée à σ_i devient $s_i \cdot \sigma_i(x)$ où $\cdot$ est l'opérateur de concaténation sur les listes. Exécuter une tâche d'extraction d'information décrite par un réseau WETDL consiste à faire évoluer l'état du réseau jusqu'à ce que tous les résultats souhaités aient été obtenus à la sortie du réseau.

5.7.2 Algorithme paresseux

Un premier algorithme d'exécution d'un réseau WETDL est l'algorithme paresseux. Le principe de cet algorithme est de ne transmettre les données que lorsqu'elles sont nécessaires. Pour ce faire, il suffit de définir une procédure *genere* qui appliquée à un opérateur génère un résultat. L'algorithme 5.1 donne la définition de cette procédure.

Lorsque un opérateur σ_i reçoit une demande de production (la procédure *genere* est appelée avec son indice), la file correspondante s_i peut soit être vide soit contenir des résultats. Si elle contient au moins un résultat le premier élément de la file est envoyé à chacun des consommateurs de σ_i . Autrement dit pour chaque opérateur σ_j consommateur de σ_i on applique l'opérateur σ_j à cet élément et on ajoute les résultats de l'application à la fin de la file s_j . Si par contre s_i est vide, il faut demander aux producteurs de σ_i des résultats en appliquant *genere* récursivement. Cette demande peut aboutir à trois cas :

1. Aucun producteur n'a pu produire de résultat.
2. Certains producteurs ont produit des résultats mais l'application de σ_i sur ceux-ci a toujours retourné une liste vide.
3. Certains producteurs ont produit des résultats et l'application de σ_i sur l'un d'entre eux a au moins rendu une liste non vide.

Dans le premier cas, σ_i n'a plus de possibilité pour générer des résultats : il renvoie donc une liste vide. Dans le second cas, l'application de σ_i à chacun des résultats des producteurs, n'a pas permis d'obtenir des résultats pour σ_i . Toutefois, il est possible que les producteurs puissent encore générer de nouveaux résultats. Il faut donc les interroger à nouveau. Dans le troisième cas, de nouveaux résultats se trouvent dans la file s_i . En effet, tout résultat obtenu après application de σ_i est ajouté à la file s_i . Il suffit donc de transmettre le premier résultat de la file aux consommateurs de σ_i et de le retourner à l'appelant.

Soit :

- Un ensemble d'opérateurs $\sigma_1, \dots, \sigma_n$
- Un ensemble de files $s_1, \dots, s_n$
- Une fonction de transmission μ

Procédure *genere*

Entrées

- Un indice d'opérateur i

Sorties

- Une liste de résultats R

```

fini  $\leftarrow 0$ 
tantque  $s_n = [] \wedge \neg fini$  faire
     $L \leftarrow []$ 
    fini  $\leftarrow 1$ ;
    pour tout  $\sigma_j$  tel que  $\sigma_i \in \mu(\sigma_j)$  faire
         $T \leftarrow genere(j)$ 
        si  $T \neq []$  alors
            { $T$  contient un unique élément}
             $s_n \leftarrow concat(s_n, \sigma_i(premier(T)))$ 
            fini  $\leftarrow 0$ 
        fin si
    fin pour
fin tantque
si  $s_n = []$  alors
    retourne  $[]$ 
sinon
    retourne  $[premier(s_n)]$ 
fin si

```

Algorithme 5.1 – Algorithme d'exécution d'un réseau WETDL paresseuse

5.7.3 Algorithme par saturation

Une seconde méthode d'exécution d'un réseau WETDL est une méthode par saturation. Dans ce cas, dès qu'il est possible de transmettre une valeur d'un opérateur à ses consommateurs, celle-ci est transmise et les opérateurs auxquels elle a été transmise doivent eux-mêmes être saturés. L'algorithme 5.2 décrit cette méthode. Lorsqu'un opérateur σ_i reçoit une demande de saturation il vide sa file d'attente s_i en transmettant les éléments à chacun de ses consommateurs. De plus après avoir transmis un résultat à un consommateur, une demande de saturation est aussi envoyée au consommateur.

Soit :

- Un ensemble d'opérateurs $\sigma_1, \dots, \sigma_n$
- Un ensemble de files $s_1, \dots, s_n$
- Une fonction de transmission μ

Procédure *staturer*

Entrées

- Un indice d'opérateur i

```

si  $\mu(\sigma_i) \neq \emptyset$  alors
  tantque  $s_i \neq []$  faire
    pour tout  $\sigma_j \in \mu(\sigma_i)$  faire
       $R \leftarrow \sigma_j(\text{premier}(s_i))$ 
       $s_j \leftarrow \text{concat}(s_j, R)$ 
      staturer( $j$ )
    fin pour
  fin tantque
fin si

```

Algorithme 5.2 – Algorithme d'exécution d'un réseau WETDL par saturation

5.8 Interprétation d'un réseau WetDL

Le modèle de réseau d'opérateurs WETDL permet de spécifier en toute simplicité une tâche d'extraction d'informations à partir du Web. Toutefois, il peut être intéressant de voir quelles relations celui-ci entretient avec d'autres types de modélisations. Dans cette section, nous nous intéressons à faire le parallèle entre un réseau WETDL et d'autres formes de modélisation. Notamment, nous nous intéressons à la traduction d'un réseau WETDL en un programme logique. Ainsi nous pourrons alors comparer WETDL avec d'autres propositions utilisant une description en logique du premier ordre tels que LIXTO [4] et le cadre proposé par May et al. [80] par exemple. Par ailleurs, nous montrerons comment faire le parallèle entre un réseau WETDL et un réseau de Pétri.

5.8.1 Comme un système déductif

Étant donné un réseau WETDL $\langle D, S, i, f, \mu \rangle$, on peut construire un programme logique en définissant pour chaque opérateur σ_k les prédicats Res_k et Val_k . $Res_k(X, Y)$ calcule l'ensemble des résultats du réseau à la sortie de σ_k pour un X donné. Autrement dit, il calcule pour tout élément X en entrée

du réseaux, l'ensemble des Y qui seront obtenus à la sortie de σ_k . $Val_k(X, Y)$, calcule les résultats Y obtenus en appliquant σ_k à X . Le prédicat Val_k peut se définir à partir de définitions génériques de chaque type d'opérateur (*i.e.* fetch, extract, transform, *etc.*).

Pour chaque type d'opérateur, il est nécessaire de définir un prédicat ternaire associé, ayant comme premier argument un paramétrage, comme second argument la valeur sur laquelle l'opérateur doit être appliqué et comme troisième paramètre une variable qui sera unifiée avec chacun des résultats de l'opération. Par exemple, considérons un opérateur d'extraction σ_k avec comme paramétrage $path("/a/@href")$. Si $extract(P, X, Y)$ est le prédicat correspondant aux opérateurs d'extraction, alors le prédicat $Val_k(X, Y)$ associé à l'opérateur σ_k se définit comme suit :

$$Val_k(X, Y) \leftarrow extract(path("/a/@href"), X, Y)$$

La définition de Res_k se fait à partir des producteurs de σ_k (*i.e.* à partir de l'ensemble des σ_j tels que $\mu(\sigma_j) = \sigma_k$). Le prédicat Res_k représente tout ce qui sort de σ_k . Ceci peut être défini à partir de tout ce qui entre en σ_k . En effet ce qui est obtenu à la sortie de σ_k est construit à partir de l'application de σ_k sur ce qu'il a obtenu en entrée, autrement dit, ce qui sort de l'ensemble des producteurs de σ_k . Pour chaque producteur σ_j de l'opérateur σ_k on ajoute à la définition de Res_k une clause de la forme :

$$Res_k(X, Y) \leftarrow Res_j(X, Z) \wedge Val_k(Z, Y). \quad (5.1)$$

D'autre part si σ_k est l'opérateur initial (*i.e.* $k = i$) alors on ajoute une clause de la forme :

$$Res_k(X, Y) \leftarrow Val_k(X, Y).$$

La tâche d'extraction se définit alors comme un prédicat binaire $Task$. L'expression $Task(X, Y)$ calcule les Y qui ressortent du réseau avec X comme entrées du réseau. Ce prédicat se définit à partir du prédicat Res_f associé à l'opérateur finale du réseau σ_f :

$$Task(X, Y) \leftarrow Res_f(X, Y).$$

La figure 5.9 donne la description du réseau fictif de la figure 5.8 sous forme de programme logique.

$$Res_f(X, Y) \leftarrow Val_f(X, Y). \quad (5.2)$$

$$Res_f(X, Y) \leftarrow Res_n(X, Z) \wedge Val_f(Z, Y). \quad (5.3)$$

$$Res_n(X, Y) \leftarrow Res_f(X, Z) \wedge Val_n(Z, Y). \quad (5.4)$$

$$Res_e(X, Y) \leftarrow Res_f(X, Z) \wedge Val_e(Z, Y). \quad (5.5)$$

$$Task(X, Y) \leftarrow Res_e(X, Y). \quad (5.6)$$

Figure 5.9 – Exemple programme logique issu d'une description d'un réseau WetDL

Ainsi, il est possible de traduire tout réseau WETDL en un programme logique. Cependant, le langage WETDL, en permettant une description à un niveau d'abstraction plus élevé, facilite l'écriture d'une tâche.

5.8.2 Comme un réseau de pétri

Les réseaux de pétri sont un outil graphique et mathématique permettant de modéliser des systèmes. Ils ont différentes applications [87], mais ils servent notamment dans la modélisation de systèmes parallèles. Toutefois, il existe des similarités dans le modèle WETDL et les réseaux de pétri notamment lorsqu'ils sont utilisés pour modéliser le flot de données d'un système.

Un réseau de pétri est un graphe orienté bipartite dans lequel les noeuds sont, soit des “places”, soit des “transitions”. Les places sont représentées par des cercles alors que les transitions sont représentées par des rectangles. Le graphe étant bipartite, les transitions peuvent uniquement être liées à des places et inversement. A un réseau de pétri est associé un marquage. Un marquage associe à chaque transition un nombre de jetons. Lorsque toutes les places en amont d'une transition contiennent au moins un jeton, la transition est dite *tirable*. Un pas dans l'évolution d'un réseau de pétri se fait par le *tirage* d'une transition. *Tirer* une transition consiste à supprimer de chaque place en amont un jeton et d'ajouter à chaque place en aval un jeton. Plusieurs interprétations du tirage d'une transition sont possibles, la plus classique étant de considérer qu'une transition est un événement, que les places sont des conditions. Dans ce contexte, un événement peut avoir lieu lorsque plusieurs conditions sont vérifiées (présence d'un jeton dans chaque place en amont) et lorsque l'événement a lieu, de nouvelles conditions sont vérifiées (ajout de jetons aux places en aval).

Les réseaux de pétri permettent notamment de représenter le flot de données. Dans ce cas, les jetons représentent à la fois la présence de données ainsi que leur valeur. Par exemple, la figure 5.10 donne un exemple de réseau de pétri calculant la valeur $x = (a + b)/(a - b)$.

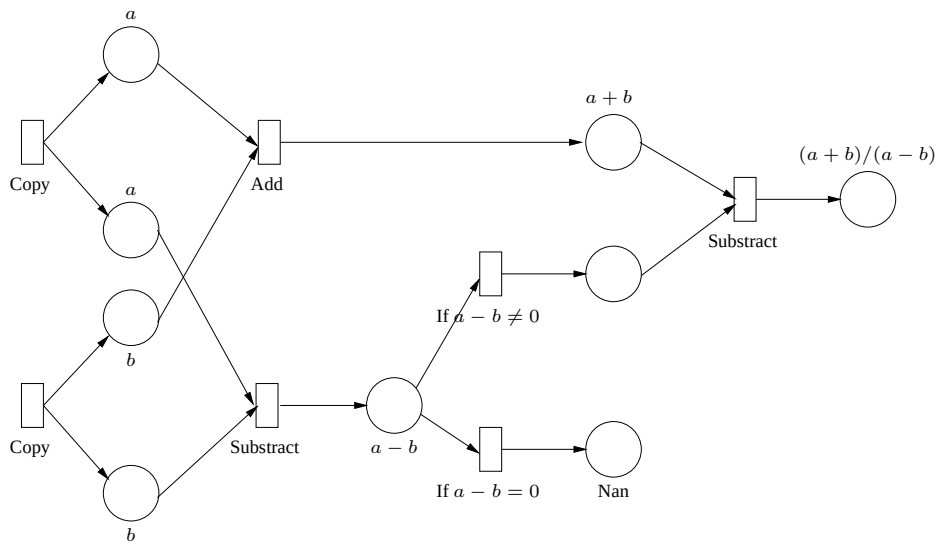


Figure 5.10 – Exemple de réseau de pétri calculant $x = (a + b)/(a - b)$

À première vue, un réseau WETDL pourrait être interprété comme un réseau de pétri représentant un flux de données avec des places implicites. Rappelons qu'un réseau WETDL est un ensemble d'opérateurs Σ liés par une fonction de transition μ . Chaque opérateur transmet à l'ensemble de ses consommateurs les résultats qu'il produit. Dans un réseau de pétri, une place transmet son jeton à une seule transition alors qu'une transition, lorsqu'elle est tirée, transmet un jeton à l'ensemble des places en aval. Ainsi, on pourrait imaginer modéliser un opérateur WETDL par un couple place-transition : la place représentant la disponibilité de données pour l'opérateur et la transition l'application de l'opérateur à

une donnée. Cependant, dans un réseau de pétri, toute transition recevant un jeton, place un unique jeton dans chaque place en aval lorsqu'elle est tirée. Au contraire, lorsqu'il reçoit une donnée, un opérateur WetDL peut, soit ne générer aucune donnée, soit générer une ou plusieurs données. Ainsi pour traduire un réseau WETDL en un réseau de pétri il n'est pas suffisant de traduire les opérateurs en un couple place-transition.

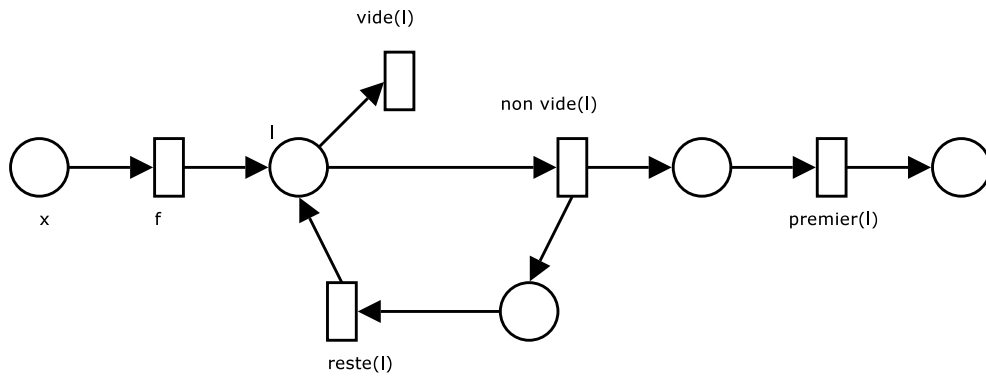


Figure 5.11 – Un opérateur WETDL traduit en réseau de petri

Afin de pouvoir considérer qu'un opérateur WETDL génère un unique résultat, il devient nécessaire d'explicitier l'utilisation d'une liste. Ainsi lorsqu'un opérateur est appliqué à une donnée, il générera dans tous les cas un résultat : une liste pouvant être vide, ne contenir qu'un seul résultat ou plusieurs. Dans ce cas tout opérateur f peut être décrit comme un réseau de pétri tel que représenté dans la figure 5.11, où chaque transition effectue une opération et où les opérations $vide(x)$, $premier(x)$ et $reste(x)$ calculent respectivement si la liste est vide, le premier élément de la liste et le reste de la liste. La transition associée à l'opérateur f applique l'opérateur et retourne une liste.

Traduire un réseau WETDL en un réseau de pétri consiste alors à traduire chacun opérateur en un réseau de pétri équivalent. Ensuite pour chaque paire d'opérateurs σ_i et σ_j tels que $\sigma_j = \mu(\sigma_i)$ (i.e. σ_i transmet ses résultats à σ_j) la place en sortie de σ_j est confondue avec la place en entrée de σ_i .

5.9 Conclusion

Dans ce chapitre, nous avons décrit un ensemble d'opérateurs génériques pour l'extraction d'information à partir du Web. Ceux-ci nous ont permis de définir WETDL, un langage permettant de décrire des tâches d'extraction d'informations. Ce langage s'appuie sur une modélisation en réseau d'opérateurs d'une tâche. Nous avons proposé deux algorithmes d'exécution d'une tâche ainsi décrite. De plus nous avons montré comment interpréter un réseau WETDL comme un système logique déductif et un réseau de pétri.

Expérimentations du système WEBSOURCE

Dans ce chapitre, nous nous intéressons à la description des différentes applications d'extraction d'informations à partir du Web. Pour chaque exemple, nous montrons comment décomposer l'application en un ensemble d'opérateurs et comment ces opérateurs se coordonnent. La description de chaque application en WETDL est donnée en annexe C. Ces différents exemples montrent l'expressivité du langage WETDL et la facilité de description des applications qu'il permet. Toutes les descriptions WETDL données ont été exécutées à l'aide du système WEBSOURCE qui permet de réaliser toute tâche décrite à l'aide de WETDL.

Nous reprenons, pour ce chapitre, l'organisation du chapitre 4. Dans un premier temps, nous montrons comment décrire l'accès à une source Web. Dans un second temps, nous proposons la description de tâches d'extraction massive de données. Finalement, nous montrons comment il est possible d'utiliser WETDL pour décrire une tâche combinant plusieurs sources.

6.1 Interrogation de sources Web

Un premier type de tâche d'extraction d'information est l'interrogation d'une source de donnée Web. Dans un tel cas, l'objectif est à partir d'une requête exprimée dans le langage de l'application, d'interroger la source de données Web, en extraire les résultats et les transformer dans le format de l'application.

6.1.1 Amazon.fr : un site de e-commerce

Le site `amazon.fr` propose entre autres la vente de DVD en ligne. Automatiser l'accès à la base de DVD en ligne d'Amazon peut mener sur plusieurs applications : l'intégration de la source dans un médiateur, proposer le suivi tarifaire d'un produit, etc. Nous avons vu précédemment qu'il n'était pas possible d'accéder directement à la source de DVD sans faire un parcours initial. Ce parcours nécessite le passage par la page index d'Amazon, de suivre le lien menant à la page index des DVD et de suivre le lien menant au formulaire avancé. Ces trois étapes, peuvent être décrites à l'aide de WETDL à partir de plusieurs opérateurs. Premièrement, il est nécessaire d'avoir un opérateur de téléchargement *init* qui aura comme donnée initiale l'adresse de la page index d'Amazon et qui permettra d'obtenir cette page et d'initier une session. A partir de la page index il faut un opérateur d'extraction *dvd – link* qui permette d'extraire de cette page le lien qui mène à la page index des DVD pour la session. Un nouvel opérateur de téléchargement *dvd – page* permettant d'obtenir la page index des DVD est nécessaire. Sur cette page se trouve un lien menant à une page contenant un formulaire de requêtes avancées. Pour accéder à cette page il faut un opérateur d'extraction *qpage – link* permettant d'extraire le lien menant à cette page et

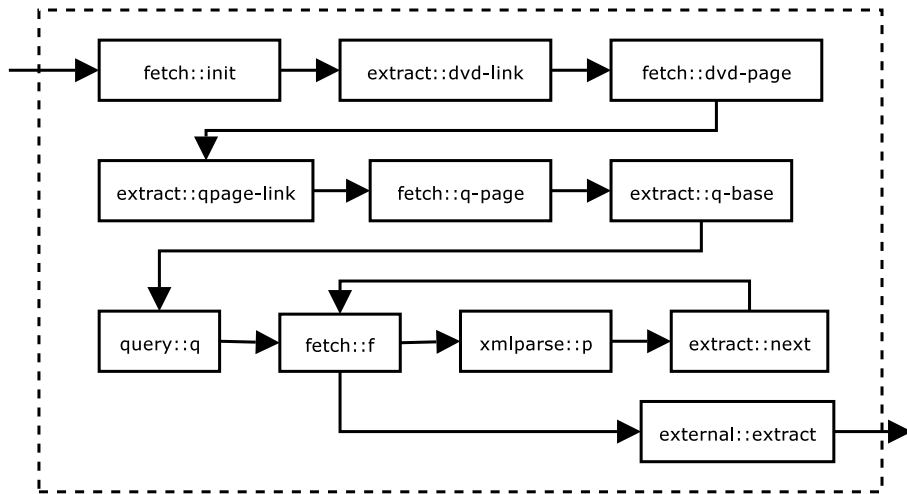


Figure 6.1 – Réseaux correspondant à la tâche d'extraction Amazon

un opérateur *q – page* permettant de télécharger cette page. L'objectif étant d'obtenir l'adresse du point d'entrée de la source, il est nécessaire d'avoir un opérateur d'extraction *q – base* permettant d'extraire cette adresse du formulaire avancée.

À partir du moment, où l'adresse du point d'entrée pour la source est connue, la source est accessible comment une source simple. Il faut un opérateur de construction de requête *q* qui transmet ses données à un opérateur de téléchargement *f*. Celui-ci transmet ses résultats à deux opérateurs : l'un *next* pour l'extraction du lien menant à la page de résultats suivante et l'autre *extract* permettant d'extraire les résultats de la page. La figure 6.1 décrit le réseau d'opérateurs nécessaire pour la tâche Amazon. La description complète de la tâche d'extraction de DVD d'Amazon est donnée en annexe dans la section C.1.

6.1.2 Pagesjaunes.fr : un annuaire en ligne

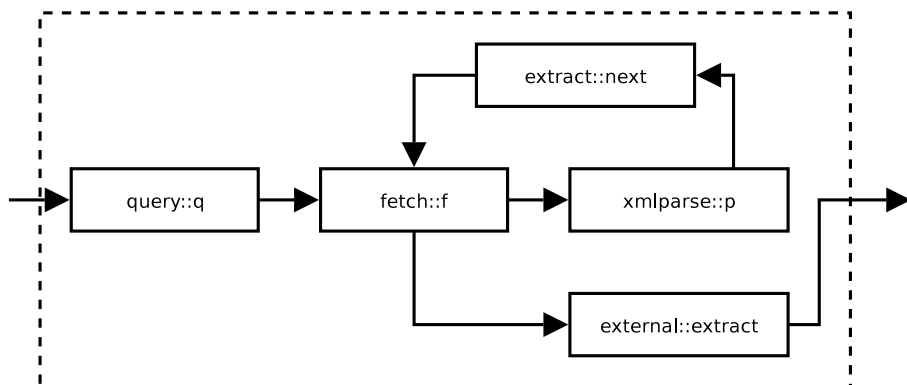


Figure 6.2 – Réseau correspondant à la tâche d'extraction Pagesjaunes.fr

L'annuaire en ligne de *pagesjaunes.fr* est un cas de source classique. La tâche qui consiste à partir d'une requête, à interroger la source et d'en extraire les résultats se fait à l'aide de cinq opérateurs :

- un opérateur de construction de requête *q* ;
- un opérateur de téléchargement *f* ;
- un opérateur *extract* pour l'extraction des résultats ;
- un opérateur *p* d'analyse syntaxique permettant d'obtenir une version DOM d'un document ;
- un opérateur d'extraction *next* permettant d'obtenir l'URL de la page de résultats suivants.

Le réseau de ces opérateurs est décrit dans la figure 6.2. Initialement, la requête de l'utilisateur est transmise à l'opérateur *q*. Celui-ci produit une requête HTTP qui est transmise à *f*. Le document issu de la requête HTTP est alors transmis, d'une part, à l'opérateur *p* et, d'autre part, à l'opérateur *extract*. L'opérateur *p* transmet une version DOM du document reçu à l'opérateur *next* qui permet d'obtenir la requête HTTP de la page suivante. Dans le cas de pagesjaunes.fr, la référence vers la page suivante prend la forme d'un formulaire HTML nommé `form_suite` sans champs. Ainsi la requête HTTP est obtenue en analysant ce formulaire. Cette requête est alors transmise à l'opérateur *f*. L'opérateur *extract* est un module externe faisant appel à un adaptateur construit en utilisant le système IEREL pour pagesjaunes.fr. Les résultats extraits par *extract* sont alors retournés à l'utilisateur.

6.2 Extraction massive de données

L'objectif d'une tâche d'extraction d'information peut être la collecte massive de données disponibles sur le Web. Dans le chapitre 4, nous avons présenté deux applications de ce type : l'extraction d'informations sur les pays du monde, mise à disposition par la CIA et l'extraction d'informations sur des brevets. Nous décrivons ici, la description de ces tâches.

6.2.1 CIA World Fact Book

L'extraction d'informations du "CIA World Fact Book" nécessite peu d'opérateurs pour sa description. De plus, dans ce cas, la description ne contient pas de boucles. Il suffit d'un opérateur de téléchargement permettant d'obtenir la page index, d'un opérateur d'extraction permettant d'obtenir les URL de chaque page décrivant un pays, d'un opérateur de téléchargement permettant de télécharger les documents correspondant à chaque pays et finalement un opérateur de transformation permettant d'obtenir les informations utiles de chaque page. La figure 6.3 donne le réseau associé à la tâche. La description WETDL est donnée en annexe à la section C.3

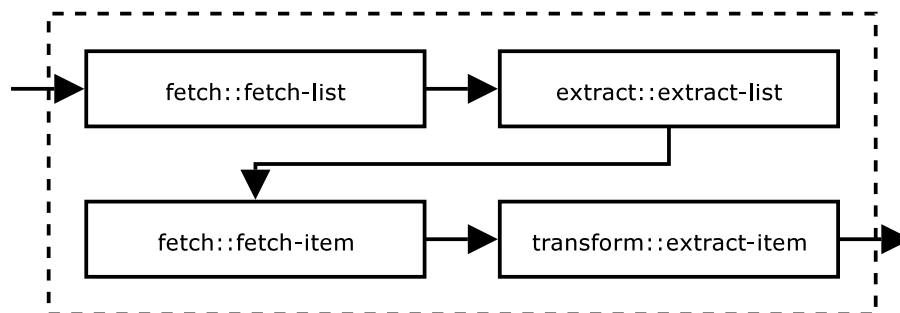


Figure 6.3 – Réseau correspondant à la tâche d'extraction CIA World Fact Book

domaine	classes	brevets
electronics	26	73 958
paper	21	17 471
pharmacy	14	18 916
textile	33	28 599
total	94	138 944

Table 6.1 – Données sur les brevets extraits

6.2.2 Extraction de brevets

Nous avons aussi utilisé WEBSOURCE pour l'extraction d'informations sur les brevets. L'objectif de la tâche d'extraction d'information dans ce cas était l'analyse des données des brevets. L'information à notre disposition était la classification internationale (ICL) des brevets souhaités pour l'analyse. Afin d'extraire les brevets souhaités, nous avons mis en place une tâche d'extraction permettant d'extraire tous les brevets d'une ICL donnée.

Cette extraction peut s'effectuer avec six opérateurs :

- un opérateur de requêtage prenant comme donnée une ICL ;
- un opérateur de téléchargement effectuant le téléchargement des pages résultats ;
- un opérateur d'extraction permettant d'obtenir la liste des URL des pages de brevets correspondant à la requête ;
- un autre opérateur d'extraction permettant d'obtenir l'URL de la page de résultats suivante ;
- un autre opérateur de téléchargement permettant de récupérer les pages de brevets ;
- un opérateur de transformation permettant de transformer une page de brevet en XML.

Au final un document XML contenant l'ensemble des brevets de l'ICL fourni en entrée est obtenu. Pour chaque ICL, une simple transformation du document ainsi obtenu en un ensemble de requêtes SQL d'insertion a permis de constituer la base de données de brevets. La base de données obtenues est décrite dans la figure 6.4.

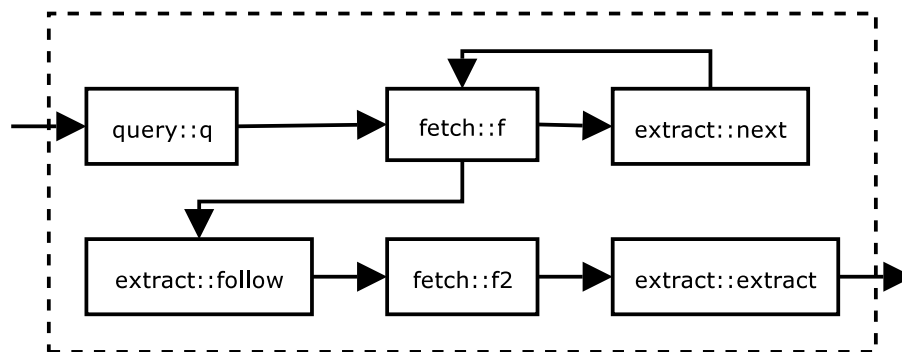


Figure 6.4 – Tâche d'extraction d'un ensemble de brevets

6.2.3 Extraction de la base de conférences de DBLP

La tâche consistant à extraire les descriptions de conférences contenues sur dblp.uni-trier.de s'exprime de manière similaire à la tâche d'extraction des données du CIA World Fact Book. Il est toutefois

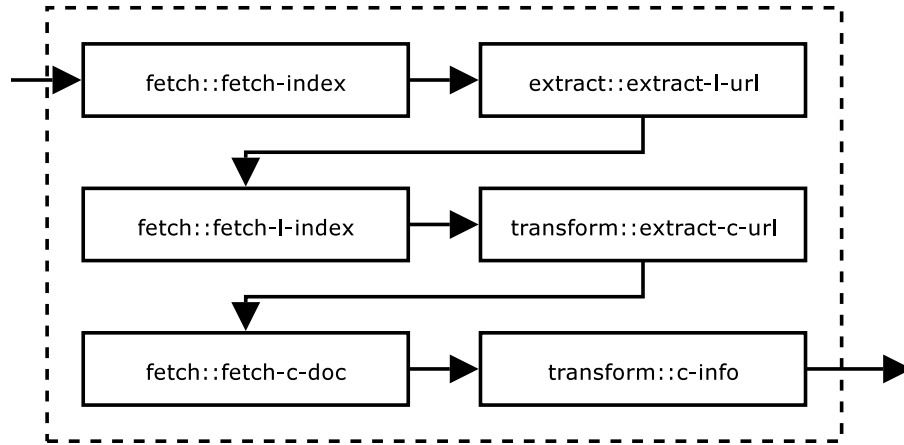


Figure 6.5 – Réseau correspondant à la tâche d'extraction DBLP

nécessaire de traverser deux indexes : un premier index contenant une référence vers un sous-index (un par lettre de l'alphabet) contenant la liste des URL vers les pages contenant l'information sur une conférence. A partir de l'URL de l'index u il est nécessaire de :

- télécharger le document d référencé par u (*fetch-index*) ;
- d'extraire de d l'ensemble L des URL des sous-indexes (*extract-l-url*) ;
- pour chaque $l_i \in L$ télécharger le document dl_i correspondant (*fetch-l-index*) ;
- extraire de chaque dl_i l'ensemble C_i des URL des pages de description de conférence (*extract-c-url*) ;
- pour chaque $C_{ij} \in C_i$ télécharger le document dc_{ij} correspondant (*fetch-c-doc*) ;
- appliquer à chaque dc_{ij} la transformation permettant d'obtenir les informations souhaitées (*c-info*).

A chacune de ces actions correspond un opérateur indiqué entre parenthèses. La figure 6.5 donne le réseau correspondant à cette tâche. La description WETDL de la tâche est donnée en annexe C.4.

6.3 Combinaison de sources

Le langage WETDL permet de décrire des applications faisant de la médiation d'information. Nous proposons ici un exemple : la description d'un accès à deux sources de données offrant de manière complémentaire le même service.

6.3.1 Annuaire multiples

Il arrive souvent que dans chaque pays du monde soit proposé un service d'annuaire en ligne permettant d'obtenir les coordonnées téléphoniques et postales d'une personne donnée. Par exemple, en France, ce service est proposé par *pagesjaunes.fr* et aux États Unis l'une des offres est celle de *superpages.com*. Les données fournies par l'utilisateur sont des informations sur la personne dont il cherche les coordonnées. Par exemple, le nom de la personne recherchée, la ville dans laquelle il habite et le pays de résidence. Afin de pouvoir interroger les deux sources de manière similaire, il est nécessaire d'avoir un langage commun.

Pour cette tâche nous pouvons définir un langage XML simple permettant de décrire les requêtes comme un ensemble de couples clé-valeur. La figure 6.6 montre le format que prend une requête dans ce langage simple mais suffisant. Un élément racine *query* contient un ou plusieurs sous-éléments parmi *firstname*, *lastname*, *city*, *province* et *country* permettant de fournir le prénom de la personne recherchée, son nom, la ville où elle réside, le département ou l'état, et le pays.

```

1 <query>
2   <firstname value="_prenom_" />
3   <lastname value="_nom_" />
4   <city value="_ville_" />
5   <province value="_etat_ou_departement_" />
6   <country value="_pays_" />
7 </query>

```

Figure 6.6 – Description d'une requête pour l'interrogation d'un annuaire multiple

Étant donné une requête dans ce format, la première étape consiste à transmettre cette requête à la bonne source (*i.e.* *pagesjaunes.fr* ou *superpages.com*). En WETDL ceci peut être obtenu en transmettant la requête à deux filtres : l'un conservant les requêtes destinées à *pagesjaunes.fr* et l'autre celles destinées à *superpages.com*.

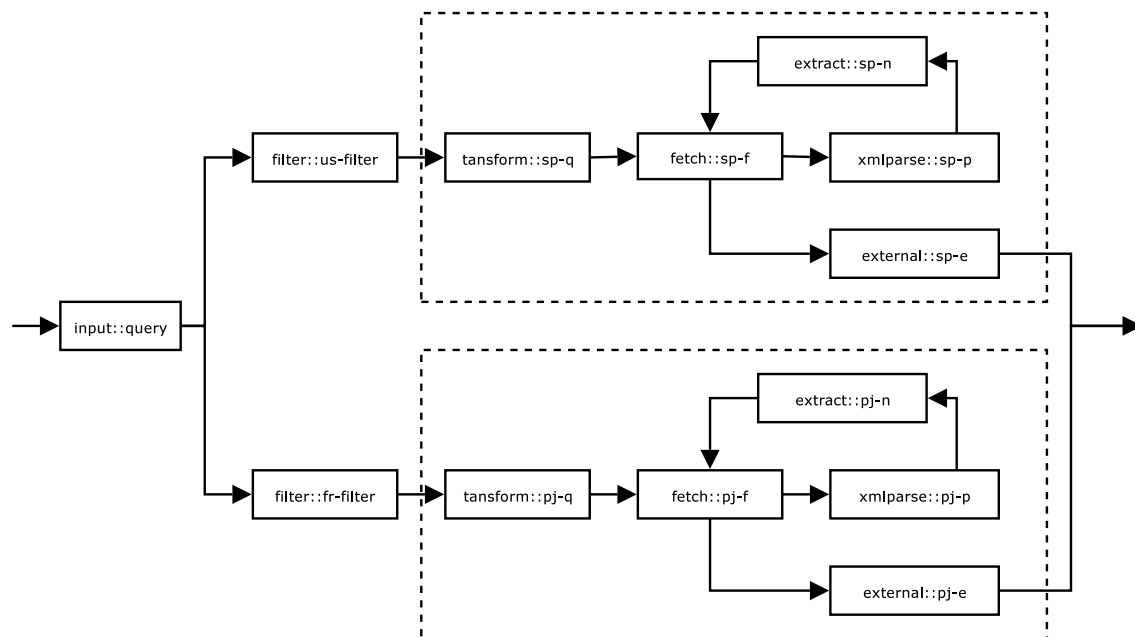


Figure 6.7 – Réseau décrivant la tâche d'interrogation de plusieurs annuaires en ligne

Une fois le filtrage effectué, la chaîne de traitements qui suit est la même que celle pour une source unique. Par exemple pour *pagesjaunes.fr* il faut transformer la requête dans le langage précédemment décrit en une requête HTTP compréhensible par *pagesjaunes.fr*. L'exécution de cette requête par un opérateur de téléchargement permet alors d'obtenir une première page de résultats. Ceux-ci sont alors extraits en appliquant un opérateur d'extraction spécifique à *pagesjaunes.fr*. Ensuite de cette même

page est extrait le lien vers la page de résultats suivante. La chaîne de traitements d'une requête destinée à `superpages.com` est similaire à celle-ci mais avec des opérateurs spécifiques à `superpages.com`. La figure 6.7 donne le schéma décrivant le réseau complet pour la tâche d'interrogation multiple. Dans la section C.6 en annexe est donnée la description WETDL complète de cette tâche.

6.4 Conclusion

Dans ce chapitre, nous avons décrit les différentes expérimentations du système WEBSOURCE que nous avons réalisées. Pour chaque application, nous avons donné la décomposition de l'application en opérateurs WETDL et leur composition en un réseau d'opérateurs. Les descriptions en WETDL sont données en annexe. Par ailleurs, l'exécution correcte de chacune des tâches a été vérifiée à l'aide du système WEBSOURCE. Ces exemples, montrent que le langage permet décrire simplement de nombreuses applications Web. Ainsi, en combinant le système IEREL pour la construction d'un adaptateur et une description WETDL toute source de donné Web peut alors facilement être accédée de manière totalement automatisée.

Web Services et extraction d'informations

Récemment un nouveau courant de recherche s'est intéressé à l'utilisation des technologies du Web pour permettre la communication et les échanges entre des application distantes. Dans cet objectif certains sites proposent maintenant des Web Services. Un Web Service est un ensemble de fonctions qui sont accessibles à distance. Par exemple, Google propose un accès aux résultats de son moteur de recherche via un Web Service. À la différence des données classiquement diffusées via le Web, les données échangées au travers de Web Services ont une structure et une sémantique connues à l'avance. Les données accessibles via un Web Service ne posent plus les mêmes problèmes que les données du Web classique. Notamment, les données seront toutes présentées dans un format standard, les mécanismes d'accès aux sources sont standardisés et à chaque source est associée une description de son accès.

L'apparition de Web Services peut remettre en question la nécessité de systèmes d'extractions d'informations à partir du Web. En effet, si une information est disponible de manière structurée au travers d'un Web Service, il est alors inutile de chercher cette même information sur un site Web classique. Toutefois, dans l'état actuel, la grande majorité des informations disponibles via le Web le sont au travers de documents HTML et non de Web Service. De plus, il n'est pas certain que tout le Web classique évolue vers un accès au travers de Web Service. En effet les Web Services sont orientés plutôt vers la communication entre organisations (Business to Business) alors que le Web classique reste la solution idéale pour communiquer avec l'utilisateur final (Business to Client). Ainsi, l'apparition des Web Services n'a pas pour vocation de remplacer les sources de données Web existantes.

Par contre, deux aspects rendent intéressant l'interaction entre les Web Services et l'extraction d'informations. D'une part, l'extraction d'informations à pour vocation de rendre accessible aux machines l'information destinée initialement aux utilisateurs. Une solution de mise en oeuvre rapide d'un Web Service donnée, consiste à utiliser un service Web classique existant et de transformer son accès en Web Service. Par exemple, un Web Service permettant de faire de la commande en ligne de DVD chez Amazon, peut être implémenté comme une tâche d'extraction d'information accédant au service en ligne d'Amazon. D'autre part, les Web Services existants et à venir ont vocation à devenir des sources de données riches en informations. Il est donc raisonnable de permettre leur accès dans un processus d'extraction d'informations. En intégrant, ainsi Web Services et extraction d'informations, il devient alors naturel de créer de nouveaux Web Services enrichissant des Web Services existants en utilisant des informations extraites via le Web classique.

Par ailleurs, dans le contexte de l'extraction d'informations, la flexibilité est essentielle pour répondre aux besoins des utilisateurs et/ou des appareils utilisés (*e.g.* technologie mobile). Ainsi, dans ce contexte construire des Web Services qui sont implémentés dans un langage de programmation donné une fois

pour toute, tel que c'est le cas actuellement, n'est pas une solution viable. Cette programmation "en dur" rend difficile la modification, le raffinement et la réutilisation de Web Services existants. De plus, du point de vue de l'utilisateur, un Web Service particulier peut ne pas répondre complètement à son besoin. Par exemple Google offre un Web Service permettant, à partir d'un ensemble de termes, d'obtenir une liste des documents contenant ces termes. On peut facilement imaginer un utilisateur souhaitant utiliser ce service au travers d'un téléphone mobile mais ne souhaitant obtenir que les documents adaptés à ce type d'appareil; ce que ne permet pas ce service. De plus, l'utilisation malgré tout du service génère des surcoûts dûs au téléchargement de documents non adaptés. Ainsi, il est nécessaire de pouvoir créer des services d'extraction d'informations adaptées.

Dans ce chapitre nous nous intéressons aux Web Services, leur intégration dans un processus d'extraction d'informations et de la description de Web Services d'extraction d'informations.

7.1 Web Services

Un Web Service (WS) est un ensemble de fonctions qui peuvent être invoquées à distance en s'appuyant sur les technologies d'internet. Ainsi on peut voir un Web Service comme une bibliothèque de fonctions. Cependant les fonctions ne se trouvent pas sur la machine sur laquelle s'exécute le programme mais sont sur une machine distante.

Les Web Services s'appuient sur des technologies existantes telles que le protocole HTTP et des nouvelles technologies créées spécifiquement telles que SOAP, WSDL, UDDI. SOAP est un langage XML permettant de décrire des messages qui seront transmis entre l'application cliente et l'application implémentant le Web Service. WSDL est un langage XML permettant de spécifier l'interface d'accès aux Web Services. Il permet notamment de définir un ensemble de types de messages qui pourront être échangés, le nom des méthodes pouvant être appelées et éventuellement une adresse spécifiant la localisation du service (*i.e.* où envoyer le message). UDDI est un annuaire de Web Services, il permet de rechercher un type de service particulier.

Dans l'état actuel de la technologie des Web Services, l'utilisation effective d'un Web Service se fait de la manière suivante : un développeur souhaitant intégrer un Web Service à son application fait une recherche du Web Service dans un annuaire UDDI. Celui-ci donne accès à une description informelle du service et à la description de son interface au travers d'un document WSDL. En utilisant la description informelle du service et le document WSDL, l'utilisateur peut intégrer à son programme des appels aux méthodes décrites dans le document WSDL. Selon la librairie utilisée, les appels à un Web Service peuvent être plus ou moins transparents. Dans certains cas il est possible à partir d'une référence vers un fichier WSDL, d'obtenir une référence vers un objet qui rend les appels des méthodes du Web Service distant similaires à des appels de méthodes locaux.

7.1.1 Protocole SOAP

Le protocole SOAP est un protocole d'échange de données basé sur XML. Le plus souvent il est utilisé avec le protocole HTTP. Toute donnée qui doit être transmise d'une application vers une autre est encodée dans une enveloppe SOAP. Une telle enveloppe est composée de deux parties : un entête optionnel et un corps. Le corps contient le message à envoyer ou reçu. Il peut éventuellement contenir un message d'erreur. La figure 7.1 donne un exemple d'un message SOAP. Dans cet exemple, il s'agit de l'appel de la méthode *getQuote* d'un Web Service de cotations proposé par XMethodes. La cotation demandée est "MSN".

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <SOAP-ENV:Envelope
3   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
4   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
5   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
6   xmlns:xsd="http://www.w3.org/1999/XMLSchema"
7   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
8   <SOAP-ENV:Body>
9     <namespace1:getQuote xmlns:namespace1="urn:xmethods-delayed-quotes">
10       <symbol xsi:type="xsd:string">MSN</symbol>
11     </namespace1:getQuote>
12   </SOAP-ENV:Body>
13 </SOAP-ENV:Envelope>
```

Figure 7.1 – Exemple de message SOAP

7.1.2 Description de services en WSDL

WSDL (Web Service Description Language) est un langage XML permettant de décrire l'interface d'accès un Web Service. Un document WSDL permet de décrire comment accéder à un service donné. Il est défini en trois temps : la description des messages, la description des opérations et les informations de liaison entre ces descriptions et l'implémentation du service.

La figure 7.2 donne la description en WSDL¹ du service de cotations proposé par XMethods. Dans cet exemple, deux types de messages sont décrits : `getQuoteResponse1` et `getQuoteRequest1`. Le premier contient un paramètre `Result` de type `float` et le second contient un paramètre `symbol` de type `string`. Le WSDL décrit un unique groupe de procédure (`portType`) contenant une unique opération `getQuote`. Cette opération prend en entrée un message de type `symbol` qui correspond au symbole pour lequel on souhaite obtenir la cotation) et donne comme résultat un message de type `Result` qui correspond à la valeur de la cotation du symbole donnée en entrée. La partie `binding` permet de décrire quel protocole de communication utiliser pour accéder au service et quels encodages utiliser pour la transmission des messages. La partie `service` permet de décrire où se trouve physiquement le service.

7.2 Composition de WS

L'une des problématiques de recherche majeure soulevée par les Web Services est celle de leur composition. Différents langages permettant la description de compositions de Web Services sont apparus. Parmi ceux-ci figurent XLANG[109] proposé par Thatte de chez Microsoft, WSFL[74] (Web Services Flow Language) proposé par Leymann de chez IBM. De ces deux langages est issu BPEL4WS [2] (Business Process Execution Language for Web Services).

Définir un service composé consiste à définir l'ensemble de fonctions du nouveau service à partir des fonctions de services existants. Par exemple, la fonction *commandeVoyage* d'une agence de voyage se compose d'un appel à la fonction *commandeVol* et *commandeChambre* envoyé respectivement à une compagnie aérienne et un hôtel. Ainsi le Web Service de l'agence de voyage sera composé à partir

¹ Le fichier WSDL original est disponible à l'adresse <http://www.xmethods.net/sd/StockQuoteService.wsdl>

```

1  <?xml version='1.0' encoding='UTF-8'?>
2  <definitions name='net.xmethods.services.stockquote.StockQuote'
3    targetNamespace='http://www.themindelectric.com/wsdl/net.xmethods.services.stockquote.StockQuote/'
4    xmlns:tns='http://www.themindelectric.com/wsdl/net.xmethods.services.stockquote.StockQuote/'
5    xmlns:electric='http://www.themindelectric.com/'
6    xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
7    xmlns:xsd='http://www.w3.org/2001/XMLSchema'
8    xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'
9    xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
10   xmlns='http://schemas.xmlsoap.org/wsdl/'>
11
12   <message name='getQuoteResponse1'>
13     <part name='Result' type='xsd:float'>
14   </message>
15
16   <message name='getQuoteRequest1'>
17     <part name='symbol' type='xsd:string'>
18   </message>
19
20   <portType name='net.xmethods.services.stockquote.StockQuotePortType'>
21     <operation name='getQuote' parameterOrder='symbol'>
22       <input message='tns:getQuoteRequest1'>
23       <output message='tns:getQuoteResponse1'>
24     </operation>
25   </portType>
26
27   <binding name='net.xmethods.services.stockquote.StockQuoteBinding'
28     type='tns:net.xmethods.services.stockquote.StockQuotePortType'>
29     <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http'>
30     <operation name='getQuote'>
31       <soap:operation soapAction='urn:xmethods-delayed-quotes#getQuote'>
32         <input>
33           <soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'
34             encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
35         </input>
36         <output>
37           <soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'
38             encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
39         </output>
40       </operation>
41     </binding>
42
43   <service name='net.xmethods.services.stockquote.StockQuoteService'>
44     <documentation>net.xmethods.services.stockquote.StockQuote web service</documentation>
45     <port name='net.xmethods.services.stockquote.StockQuotePort'
46       binding='tns:net.xmethods.services.stockquote.StockQuoteBinding'>
47       <soap:address location='http://services.xmethods.net/soap/'>
48     </port>
49   </service>
50 </definitions>

```

Figure 7.2 – Description du service de cotation en ligne proposé par XMethods

des Web Services respectifs de la compagnie aérienne et de l'hôtel. Décrire une composition consiste à définir comment se déroule l'exécution d'une fonction du service composé à partir des fonctions des services utilisés.

Le langage WSFL (Web Source Flow Language) permet de décrire d'une part le flot correspondant à service et d'autre part la composition de différents services. Une description d'un Web Service en WSFL se fait par la description d'un graphe d'activité. Chaque activité correspond à une tâche unitaire à effectuer. Par exemple, il peut s'agir de traiter une commande, de traiter un paiement, d'envoyer une demande de prêt, *etc.* Les activités sont liées entre elles en définissant un flux de contrôle. Celui-ci permet de déterminer quelles activités peuvent avoir lieu à la suite d'une activité donnée.

Les langages XLANG et BPEL4WS permettent de décrire la composition de Web Services en utilisant les structures de contrôles habituelles des langages de programmation classiques. Par exemple, il est possible de définir des séquences d'instructions, des boucles (*while*), des conditionnels (*if*), *etc.* D'autre part, BPEL4WS permet l'exécution en parallèle de certaines séquences.

7.3 Composition automatique de Web Services

Dans [81], les auteurs proposent une méthode de composition automatique de Web Services. Elle se base sur la description de services en utilisant une ontologie et un langage de composition (CSSL) de haut niveau permettant de décrire le service composite souhaité.

Dans sa forme actuelle, un document WSDL contient des informations sur les types de données qui seront utilisés, le format des messages qui seront échangés, la spécification des opérateurs que propose le service, *etc.* Cependant, une telle description ne contient pas d'informations sur la sémantique des opérateurs, *i.e.* quels sont les objectifs d'une opération, à quel domaine elles appartiennent, *etc.* Les auteurs proposent d'ajouter de telles informations sémantiques à cette description afin de permettre la composition automatique de Web Services.

Dans leur formalisation, un Web Service se définit par une description (une définition informelle de ce que fait le service) et d'un ensemble d'opérateurs. Un opérateur se définit comme dans WSDL par une *description*, un message en *entrée*, un message en *sortie*, un mode ("sens unique", "notification", "demande de réponse", "requête-réponse"). De plus, trois informations sémantiques sont associées à chaque opérateur : un *objectif*, une *catégorie* et une *qualité*. Un objectif se définit par un nom de *fonction* (ce que fait l'opérateur), un ensemble de *synonymes* de la fonction et un ensemble de *caractéristiques* de la fonction. Une catégorie se définit par un nom de domaine (associé à l'opérateur), un ensemble de *synonymes* du domaine et un ensemble de *caractéristiques* du domaine. La qualité est définie par un *coût* de l'exécution de l'opération, un booléen *sécurité* permettant de déterminer si l'opération se déroule dans un mode sécurisé, un booléen *confidentiel* permettant de déterminer si des informations sensibles sont transmises par l'opérateur. Les auteurs proposent une extension du langage WSDL, le CSSL (Composite Service Specification Language), permettant de décrire les Web Services dans ce formalisme.

A partir des informations syntaxiques et sémantiques, une notion de composabilité des opérateurs peut être définie. Celle-ci peut être décomposée en la composabilité syntaxique et la composabilité sémantique. Deux opérations sont syntaxiquement composables si elles sont de mode opposé : si l'une est en mode "sens unique" l'autre doit être en mode "notification" et inversement et si l'une est en mode "requête-réponse" l'autre doit être en mode "demande de réponse". Par ailleurs le message en entrée (resp. en sortie) d'une opération doit être de même type que le message en sortie (resp. en entrée) de l'autre opération. Deux opérations sont sémantiquement composables lorsqu'elles ont des objectifs, catégories et qualités similaires.

7.4 Web Service pour l'extraction d'informations

Dans l'intégration des Web Services et de l'extraction d'informations deux aspects sont intéressants. Le premier consiste à considérer un Web Service comme une nouvelle source d'informations pouvant être exploitée dans l'exécution d'une tâche d'extraction d'informations. Le second consiste à considérer la tâche elle-même comme un Web Service. Dans cette section, nous nous intéressons à ces deux aspects.

7.4.1 Web Services comme source d'informations

Un Web Service peut être considéré au même titre qu'un site Web comme une source d'informations. Par exemple, Google propose un accès à son moteur de recherche via un Web Service. Accéder à un Web Service consiste à construire un message SOAP correspondant à la requête soumise au Web Services et de poster ce message via un protocole de transport (HTTP, SMTP, *etc.*) à un endroit désigné. L'utilisation d'un document WSDL permet de s'abstraire des détails techniques de la sorte. Ainsi, avec une référence (URL) du WSDL du service, accéder à celui-ci revient à faire un appel de méthode avec un ensemble de paramètres.

Pour permettre l'accès à un Web Service par une tâche d'extraction d'informations il suffit alors d'ajouter un opérateur **soap** faisant appel à un Web Service. Une tel opérateur se spécifie par les paramètres suivants :

- une référence vers la description en WSDL du Web Service ;
- le nom de la méthode à appeler ;
- un ensemble de valeurs pour les paramètres de la méthode.

A la manière de l'opérateur **query** les paramètres de la méthode peuvent être soit issus des données en entrée lors de l'exécution de l'opérateur, soit des valeurs prédéfinies lors de la déclaration de l'opérateur. L'opérateur **soap** donne en sortie un document XML correspondant à l'enveloppe SOAP générée pour l'appel au Web Service. Celle-ci pourra alors être traitée à la manière d'un document HTML obtenu par l'exécution d'une requête HTTP consistant à sélectionner les éléments à extraire du document, puis appliquer différents opérateurs de filtrage, transformation, *etc.*

La description XML d'un opérateur **soap** se fait par l'utilisation d'un élément **soap**. Celui-ci contient trois sous-éléments : **service**, **method** et **parameters**. L'élément **service** permet de spécifier via un attribut **href** l'URL du fichier WSDL décrivant le service à appeler. L'élément **method** permet de spécifier via un attribut **name** le nom de la méthode du Web Service à appeler et via un attribut **params** la liste des noms de paramètres dans l'ordre ou ils doivent apparaître dans l'appel. Cette liste des noms peut correspondre, soit à un nom existant dans les données en entrée, soit à un nom de paramètre spécifié avec un élément **param** sous la balise **parameters**. La figure 7.3 donne un exemple d'opérateur **soap** décrit en WETDL.

7.4.2 Génération de Web Service

Un Web Service peut être simplement vu comme une bibliothèque de fonctions pouvant être appelées à distance. Or, une tâche d'extraction d'informations, peut être considérée comme l'application d'une fonction à un ensemble de données qui forment la requête de l'utilisateur. De ce point de vue il devient naturel de décrire une tâche d'extraction d'informations comme un Web Service. Ceci permet de rendre disponible l'exécution d'une tâche d'extraction d'informations à distance.

```
1 <wt:soap name="google-service" forward-to="parser">
2   <service href="http://api.google.com/GoogleSearch.wsdl" />
3   <method name="doGoogleSearch"
4     params="key data start maxResults filter restrict safeSearch lr ie oe" />
5   <parameters>
6     <param name="key"      value="jUM7k/xQFHK/FhCd6bXRZ2usrV6IFGBY" />
7     <param name="start"    value="0" />
8     <param name="maxResults" value="10" />
9     <param name="filter"   value="0"/>
10    <param name="restrict" value=""/>
11    <param name="safeSearch" value="0"/>
12    <param name="lr"       value=""/>
13    <param name="ie"       value="utf8"/>
14    <param name="oe"       value="utf8"/>
15  </parameters>
16 </wt:soap>
```

Figure 7.3 – Exemple de description d'un opérateur de construction de requête HTTP

7.5 Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'intégration des Web Services et de l'extraction d'informations à partir du Web. Les Web Services peuvent être vus comme des nouvelles sources d'informations diffusées via le Web. À la différence des pages Web ils sont destinés à être accédés par des machines. Toutefois, dans le cadre de l'extraction d'informations, qui consiste à mettre en oeuvre un système permettant de répondre à un besoin spécifique, ceux-ci, bien que facilitant la tâche, ne répondent pas toujours directement au problème. En effet, certains traitements complémentaires sont nécessaires pour répondre pleinement aux attentes de l'utilisateur et/ou des appareils utilisés. L'intégration des Web Services à un système d'extraction d'informations permet de répondre complément aux besoins de l'utilisateur tout en optimisant la tâche d'extraction par l'utilisation de Web Services. En intégrant l'accès aux Web Services dans le langage WETDL, il devient alors possible de composer les opérateurs d'extraction d'informations avec des Web Services. Ceci permet notamment de compléter et spécialiser les services proposés et de répondre plus précisément aux besoins en informations de l'utilisateur.

Par ailleurs, simplement en donnant un accès sous forme de Web Service à une tâche d'extraction d'information décrite en WETDL, il devient possible d'utiliser WETDL dans le but de composer des Web Services. De plus, les différents opérateurs définis, permettent d'intégrer les adaptations nécessaires pour faire communiquer deux Web Services n'utilisant pas forcément les mêmes messages. Notamment, transformer une requête dans un langage spécifique au Web Service composite dans les langages de chacun des services composés se fait naturellement à l'aide d'un opérateur de transformation.

Parcours intelligent du Web

Dans les chapitre précédents nous avons vu comment construire des applications d'extraction d'informations à partir du Web tels que l'accès à un site e-commerce ou l'interrogation d'un moteur de recherche. Nous avons proposé le langage WETDL qui permet de décrire de tels systèmes en définissant un ensemble d'opérateurs d'extraction d'informations et leur composition. Dans ce chapitre nous montrons qu'il est possible de décrire avec WETDL un type de tâche différent. Nous nous intéressons au problème de parcours intelligent du Web et montrons comment il est possible de décrire une telle tâche en WETDL.

La recherche classique de documents sur le Web telle qu'elle est faite par les moteurs de recherche tels que `google.com` ou `altavista.com`, s'appuie sur une base indexée des documents. Une telle recherche marche en deux temps : création d'un index puis recherche dans l'index. Cette méthode pose plusieurs problèmes : seule une partie infime du Web est indexée par les moteurs de recherches et les pages indexées ne sont plus forcément à jour. D'une part, une grande majorité des documents du Web sont générés par des programmes et ne sont accessibles qu'au travers de formulaires que les programmes indexant le Web ne savent pas remplir et donc ne suivent pas. D'autre part, le contenu des pages sur le Web peut évoluer très rapidement et pose alors le problème de la réindexation des documents. Cette tâche est connue pour être difficile. La recherche de document n'utilisant qu'un index préalablement construit s'appelle recherche statique, dans le sens où elle s'effectue sur une version statique du Web à un instant t .

La recherche dynamique de documents s'oppose à la recherche statique qu'elle construit sa réponse à partir de l'état courant du Web. Elle peut partir des réponses d'une recherche statique en effectuant une exploration du Web au moment de la recherche afin d'en vérifier et d'en compléter les résultats. Dans le cas d'une recherche dynamique, le problème majeur qui se pose est le nombre de page téléchargées et analysées.

Dans le cas des deux approches, un parcours du Web est nécessaire à un moment donné : celui-ci peut se faire avant la recherche (pour une recherche statique) ou pendant la recherche (pour une recherche dynamique). Toutefois le type de parcours diffère selon qu'il s'agit d'une recherche statique ou dynamique. Dans le premier cas, l'objectif du parcours est de construire un index d'un ensemble conséquent de pages (idéalement toutes les pages) du Web alors que dans le second, l'objectif est d'obtenir un ensemble de pages pertinent par rapport à la requête.

Lors de la construction d'un index se voulant complet un algorithme de parcours "naïf" suffit¹ : l'ensemble des pages devra de toute façon être parcouru. Par contre, dans le cas d'une recherche dynamique, l'objectif n'est plus de parcourir un maximum de pages du Web mais un maximum de pages du

¹ Il peut être toutefois souhaitable de définir des stratégies de parcours même dans le cas de la création d'un index. Par exemple, on peut souhaité indexer d'abord des documents ayant une grande chance d'être retournés dans les requêtes "habituelles"

Web correspondant à la requête initiale. Afin d'éviter qu'une requête dynamique et son parcours du Web monopolisent les ressources du système et que le temps de réponse ne soit pas trop élevé, il est aussi nécessaire d'avoir des méthodes permettant de limiter ce parcours en perdant un minimum de résultats. Pour ce faire, il est nécessaire de mettre en place des méthodes de parcours "intelligentes".

8.1 Parcours exhaustif du Web

L'objectif d'un parcours exhaustif du Web est la construction d'un index des documents disponibles sur le Web. Celui-ci s'effectue en aval de recherches statiques. L'algorithme 8.1 décrit comment effectuer un parcours du Web de manière naïve. Celui-ci se décompose en trois étapes à partir d'une liste d'URL initiale : sélection d'une URL dans la liste, téléchargement du document référencé par l'URL, puis extraction des liens du document résultant et ajout de ceux-ci à la liste d'URL. Afin de ne pas télécharger deux fois un même document un test est effectué sur les URL avant de le télécharger.

Procédure parcours-naïf

Entrées

- Une liste U d'URL

Sorties

- Un ensemble de documents D

```

tantque  $U \neq \emptyset$  faire
   $u \leftarrow \text{choisir}(U)$ 
  si  $\text{deva\_vu}(u)$  alors
     $\text{deva\_vu}(u) \leftarrow 1$ 
     $d \leftarrow \text{telecharger}(u)$ 
    pour tout  $l$  dans  $\text{liens}(d)$  faire
       $U \leftarrow U \cup \{\text{url}(l)\}$ 
    fin pour
  fin si
fin tantque

```

Algorithme 8.1 – Algorithme de parcours naïf du Web

8.2 Parcours intelligent du Web

Dans le cas d'un parcours intelligent, l'objectif est de construire dynamiquement la réponse à une requête. La construction de la réponse s'effectue en parcourant une partie du Web. Pour être efficace cette partie doit se limiter le plus possible au documents répondant à la requête. Ceci nécessite que le parcours passe le plus souvent par des pages répondant à la requête. En effet, dans le parcours du Web, une page parcourue est une page qui doit être téléchargée. Afin de ne pas monopoliser les ressources systèmes et réseaux il faut minimiser le téléchargement de pages ne répondant pas à la requête. Pour ce faire, nous proposons une méthode de parcours "intelligente". Ceci nécessite d'avoir un moyen d'évaluer la pertinence d'un document par rapport à la requête *avant* son téléchargement. Il est possible d'imaginer différentes heuristiques permettant d'évaluer la pertinence d'un document avant son téléchargement.

Celles-ci se basent sur des informations accessibles au système : par exemple la forme ou la structure de l'URL qui référence le document, le contexte dans lequel apparaît le lien pointant vers ce document, *etc.* L'exploitation de ces informations permet de filtrer l'ensemble des documents qui sont parcourus.

La première information permettant d'évaluer la pertinence d'un document avant d'avoir accès à son contenu est la structure de l'URL qui référence. En effet, le Web est structuré en serveurs Web qui eux même publient une arborescence de documents décomposés en répertoires et sous-répertoires. La structure d'une URL permet de connaître le serveur sur lequel se trouve une page ainsi que le répertoire dans lequel la page se trouve sur ce serveur. Souvent sur un même site se trouvent des informations de même nature. De plus, les répertoires sont souvent organisés par thématiques. Ainsi une classification des serveurs et des répertoires peut-être utile pour déterminer la pertinence des pages qui s'y trouvent. Une page appartenant à un répertoire contenant des documents qui ont jusqu'alors été peu pertinents aura de très forte chance d'être peu pertinente elle aussi.

La seconde information qui peut être utile à l'évaluation *a priori* d'une page est le contexte d'apparition du lien qui pointe vers cette page. A chaque lien est associé un bout de texte dont le contenu à un rapport direct avec le document référencé par le lien. L'ensemble des termes composant ce texte peut alors être exploité pour évaluer la pertinence du document. Pour ce faire, chaque terme se voit associer une mesure reflétant la pertinence des documents vers lesquels les liens dont ils font partie pointent.

L'exploitation de chacune de ces informations permet de déterminer si un document sera prometteur ou non. Cette information permet d'effectuer un filtrage des documents qui seront téléchargés ou non. En résumé, nous avons deux filtres : un filtre structurel qui évalue un document selon la structure de l'URL qui le référence, et un filtre contextuel qui évalue un document selon les termes apparaissant dans le contexte du lien qui le référence.

Dans le cas des deux filtres, l'évaluation d'un document se fait à partir d'une modélisation préalable. Dans notre approche celle-ci est calculée à partir de la pertinence des documents déjà téléchargés. Ainsi la finesse des modèles sera améliorée avec le nombre de documents téléchargés. Ceci nécessite d'avoir une méthode d'évaluation du document permettant de dire si un document est pertinent ou non. Dans un premier temps nous pouvons supposer que nous avons un oracle en notre possession permettant d'indiquer pour un document donné s'il est pertinent ou non. A la différence des deux filtres, l'oracle a accès au contenu du document. Selon les réponses de l'oracle les modèles des filtres pourront être améliorés : lorsque l'oracle détermine qu'un document D n'est pas pertinent les filtres structurels et contextuels peuvent mettre à jour leurs modèles en fonction. Par exemple, dans ce cas du filtre structurel, la note associée au répertoire auquel appartient D peut être abaissée. De manière similaire, dans le cas du filtre contextuel, les notes associées aux termes du lien ayant mené au téléchargement de D seront revues à la baisse.

L'objectif du parcours est d'obtenir un ensemble de documents pertinents par rapport à une requête. Dans ce contexte, l'oracle joue aussi le rôle de filtre. En effet, c'est d'après ses réponses que nous pouvons déterminer si un document répond ou non à la requête. De ce fait, il a un rôle central dans tout le processus de parcours et de recherche. D'autre part, étant donné qu'il s'appuie sur le contenu du document, nous pouvons le considérer comme un filtre de contenu.

En résumé, notre méthode de parcours s'appuie sur trois filtres : un filtre structurel se basant sur la structure de l'URL, un filtre contextuel se basant sur le contexte du lien hypertexte et un filtre de contenu se basant sur le contenu du document. Les filtres contextuel et structurel construisent leur modèle interne à partir des réponses positives et négatives du filtre de contenu. Le filtre de contenu est un oracle qui dépend de la tâche à effectuer.

8.2.1 Architecture du crawler

Nous pouvons maintenant envisager de construire l'architecture globale de notre système de parcours intelligent du Web à partir de ces trois filtres. Chaque filtre constitue un composant de l'architecture du système. Nous nous intéressons maintenant aux autres composants. En étudiant l'algorithme 8.1 de parcours naïf nous pouvons constater qu'il y a trois principales fonctionnalités : le téléchargement de documents (fonction *telecharger*), l'extraction de liens (fonction *liens*) et l'extraction de l'URL d'un lien (fonction *url*).

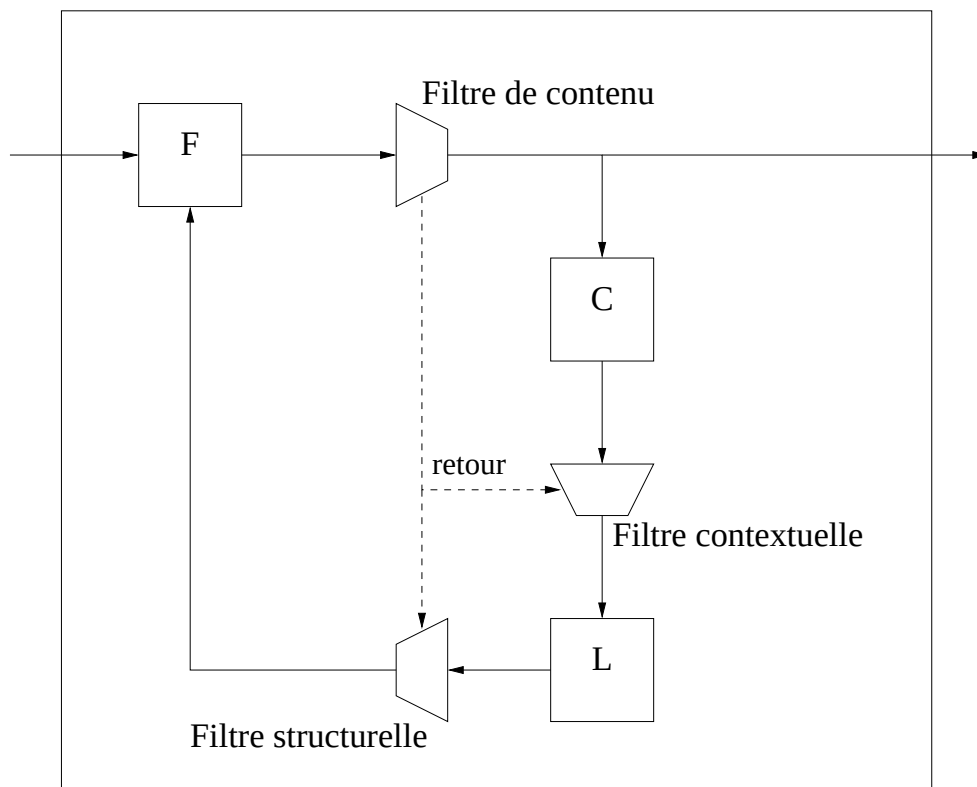


Figure 8.1 – Architecture d'un crawler intelligent

Dans notre architecture chacune de ces fonctionnalités devient un composant : un composant de téléchargement F qui reçoit des URL et renvoie des documents, un composant d'extraction de liens C qui reçoit des documents et en extrait les liens avec leurs contextes et un composant d'extraction d'URL L qui reçoit des liens et renvoie des URL. Le processus d'extraction est constitué d'un cycle commençant par un ensemble d'URL qui sont envoyés à F qui télécharge les documents correspondants. Ensuite ces documents sont dirigés vers C qui recherche les liens dans les documents et en extrait le contexte. Par la suite, ceux-ci sont envoyés à L qui extrait les URL pour à nouveau les envoyer sur F et recommencer un cycle. Le processus s'arrête lorsque les seules URL produites par L sont des URL que F a déjà téléchargées. Dans ce cycle il reste à placer les filtres permettant de réduire le nombre de documents téléchargés. Ces trois filtres se placent naturellement : le filtre de contenu entre F et C ce qui permet de ne filtrer les documents qui ne sont pas pertinents par rapport à la requête, le filtre de contexte entre C et L ce qui permet de filtrer les liens d'après les termes de leur contexte et le filtre de structure entre L et F pour filtrer les URL d'après leur structure. La figure 8.1 schématise le processus. Les carrés représentent

chacun des composants F , C et L alors que les trapèzes représentent des filtres ne laissant passer que les données prometteuses et/ou pertinentes. Le document résultant du processus sont ceux qui ont réussi à passer tous les filtres, notamment le filtre de contenu.

Dans l'algorithme 8.2, nous décrivons la mise en oeuvre de ce processus. Les fonctions *oracle*, *filtrage_struct* et *filtrage_context* correspondent respectivement aux filtres de contenu, structurel et contextuel. Chacune renvoie un booléen indiquant si le document, contexte, ou lien est pertinent par rapport au domaine. Le calcul de pertinence effectué par chaque filtre est décrit ci-après.

Procédure parcours-intelligent

Entrées

- Un ensemble U d'URL initiales

Sorties

- Un oracle permettant de déterminer la pertinence d'un document
- Un ensemble de documents D pertinents selon *oracle*

```

tantque  $U \neq \emptyset$  faire
   $u \leftarrow$  un élément de  $U$ 
   $U \leftarrow U \setminus \{u\}$ 
  si filtrage_struct( $u$ ) alors
     $d \leftarrow$  telecharger( $u$ )
    si oracle( $d$ ) alors
      pour tout  $l$  dans liens( $d$ ) faire
        si filtrage_context( $l$ ) alors
           $U \leftarrow U \cup \{url(l)\}$ 
        fin si
      fin pour
    fin si
  fin si
fin tantque

```

Algorithme 8.2 – Algorithme “intelligent” de parcours du Web

8.2.2 Filtre de contenu

L'objectif d'un filtre de contenu est de déterminer la pertinence d'un document par rapport à une requête de l'utilisateur. Une solution classique consiste à procéder en plusieurs étapes. Premièrement un vecteur des termes importants est calculée. Un terme est considéré comme important lorsqu'il est fréquent dans le document et qu'il est discriminant par rapport à la collection. Deuxièmement, l'intersection du vecteur de termes du document et de l'ensemble des mots clés est calculé. Troisièmement, selon la taille de l'intersection le document est conservé ou non.

8.2.3 Filtre contextuel

L'objectif du filtre contextuel est de déterminer à partir des termes contenus dans un lien si le document référencé sera pertinent ou non. Si nous considérons que le filtre de contenu joue le rôle d'oracle et

que les prédictions de ce dernier sont correctes alors nous pouvons supposer que le problème consiste à prédire le résultat du filtre de contenu. Le problème ainsi posé revient alors à de l'apprentissage inductif classique.

L'information qu'utilise le filtre contextuel est l'ensemble de termes qui est associé à un lien. Son objectif est de prédire si le document référencé par le lien sera considéré par le filtre de contenu comme pertinent ou non. L'approche que nous proposons pour faire cette évaluation est une approche probabiliste. Le problème peut alors se poser comme suit : étant donné n termes $t_1, \dots, t_n$ nous cherchons à déterminer la probabilité que ces termes mènent à un succès. Nous avons alors $n+1$ événements probabilistes : l'événement s associé à un succès et les événements $t_1, \dots, t_n$ associés à chaque terme. L'objectif est alors de calculer la probabilité que l'on ait un succès sachant que nous avons les termes $t_1, \dots, t_n$. Autrement dit nous cherchons à estimer $p = P(s/t_1, \dots, t_n)$, qui d'après Bayes peut se réécrire comme suit :

$$P(s/t_1, \dots, t_n) = \frac{P(s, t_1, \dots, t_n)}{P(t_1, \dots, t_n)} \quad (8.1)$$

En supposant que les événements s, t_i et les événements s, t_j sont indépendants nous pouvons écrire :

$$P(s/t_1, \dots, t_n) \simeq \frac{P(s, t_1)P(s, t_2) \dots P(s, t_{n-1})P(s, t_n)}{P(t_1, \dots, t_n)} \quad (8.2)$$

En décomposant $P(s/t_1, \dots, t_n)$ ainsi, il devient alors aisé d'en obtenir une estimation à partir des retours du filtre de contenu. En effet, celui-ci donne son retour sous la forme d'un $n+1$ -uplet $r = \langle t_1, \dots, t_n, v \rangle$ où les t_i sont des termes et v est un booléen indiquant si le document associé à ces termes était pertinent ou non. A partir de k retours du filtre de contenu nous pouvons estimer la probabilité $P(t_i)$ d'apparition d'un terme t_i de la manière suivante :

$$P_e(t_i) = f_{t_i}/k \quad (8.3)$$

où f_{t_i} correspond au nombre de retours où t_i apparaît.

L'estimation de $P(s, t_i)$ s'évalue comme suit :

$$P_e(s, t_i) = f_{v=1, t_i}/f_{t_i} \quad (8.4)$$

où $f_{v=1, t_i}$ correspond au nombre de retours dans lequel t_i apparaît et tel que $v = 1$.

8.2.4 Filtre structurel

Le filtre structurel permet d'évaluer la pertinence d'un document avant son téléchargement d'après les informations sur la structure de l'URL qui y fait référence. Par exemple, les documents issus d'un site Web pour lequel tous ou une grande majorité des documents ont été rejetés par le filtre de contenu auront peu de chance d'être retenus. De la même façon que pour le filtre de contexte, le filtre de contenu peut jouer le rôle d'oracle pour que le filtre structurel puisse construire un modèle interne permettant d'évaluer une URL par rapport aux résultats obtenus par des URL du même répertoire ou du même site.

Le filtre de contenu cherche à évaluer pour une URL u donnée, la probabilité que le document référencé par u sera accepté par le filtre de contenu (événement $A(u)$) sachant que u est une URL dont le site est S (événement $S(u)$). Notons cette probabilité $P(A(u)/S(u))$. De manière analogue, on peut chercher à calculer la probabilité que u soit acceptée sachant que u appartient au répertoire R

(événement $R(u)$), i.e. $P(A(u)/R(u))$. Cette probabilité peut être estimée par la fréquence d'acceptation des documents d'un site ou d'un répertoire donné. Celle-ci peut être calculée à partir des retours du filtre de contenu. En effet, lorsque le filtre de contenu accepte ou rejette un document, l'URL de ce document et le résultat de son acceptation peuvent être retournés au filtre structurel. Ainsi, en conservant le nombre total de documents t_S issue d'un site (ou d'un répertoire) et le nombre de documents acceptés a_S pour ce même site, la fréquence d'acceptation des documents est a_S/t_S . Cette valeur est une estimation de la probabilité qu'un document quelconque du site S soit accepté par le filtre de contenu, autrement dit la probabilité que le document soit accepté sachant qu'il appartient au site.

8.2.5 Description à l'aide de WETDL

Ce processus se décrit très naturellement en WETDL. Les opérateurs F , C et L de la figure 8.1 peuvent être respectivement décrits à l'aide d'un opérateur de téléchargement et de deux opérateurs d'extraction. Le premier opérateur d'extraction extrait les liens avec leur contexte : ceci se traduit concrètement par l'extraction des noeuds éléments a ayant un attribut `href` qui s'exprime par l'expression XPath `//a[@href]`. Le second opérateur d'extraction reçoit les éléments a extraits par le premier opérateur d'extraction. L'objectif de cet opérateur est d'extraire l'URL référencée par un élément a reçu en entrée. Ceci s'exprime en XPath par l'expression `@href`.

Une façon efficace d'initialiser l'ensemble des résultats d'une recherche dynamique consiste à faire une première recherche statique sur un ou plusieurs moteurs de recherche. Nous avons vu qu'il était aisé en WETDL de faire une telle recherche et même de combiner plusieurs moteurs. La tâche qui consiste à effectuer une première recherche statique sur plusieurs moteurs puis de compléter ses résultats par une recherche dynamique s'exprime très naturellement en WETDL. La figure 8.2 donne le réseau WETDL correspondant à une telle tâche.

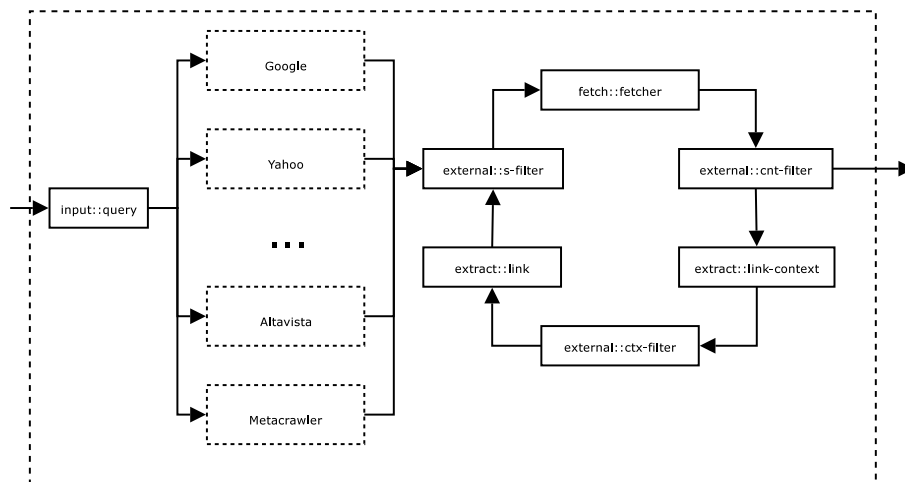


Figure 8.2 – Réseaux WETDL d'une tâche de recherche dynamique

8.3 Application: le système E-awacs

Cette approche du parcours intelligent du Web a été utilisée dans une application concrète : le système E-AWACS. Les objectifs de ce système sont les suivant :

- l'intégration d'un parcours intelligent et incrémental du web et d'une recherche statique sur plusieurs moteurs ;
- libérer les utilisateurs du parcours de longues listes de documents retournées par les moteurs de recherche ;
- l'extraction des documents de motifs compréhensibles et de connaissances utiles ;
- proposer un support visuel pour l'exploration de collections de documents ;
- l'interrogation des documents à plusieurs niveaux.

Dans ce système, les documents acceptés par le filtre de contenu sont envoyés à un analyseur permettant de visualiser graphiquement l'ensemble de mots issus des documents ainsi que les dépendances qu'ils entretiennent entre eux. Le graphe est mis à jour en continu avec l'arrivée de nouveaux documents et l'évolution du calcul de l'analyse des dépendances. Une caractéristique du système d'analyse est qu'il est fortement lié à la visualisation et inversement. En effet, les étapes de l'analyse dépendent de la proximité graphique des mots et des liens entre eux alors que l'analyse même produit met à jour ces informations. Un prototype du système e-awacs a été mis en oeuvre en collaboration avec Vincent Dubois. Les détails des expérimentations de la partie analyse du système se trouvent dans sa thèse de doctorat [26].

8.4 Conclusion

Dans ce chapitre, nous avons montré comment décrire le problème de la construction d'un système de parcours intelligent du Web. Ce parcours est rendu efficace par l'intégration de filtres à plusieurs niveaux de la tâche : un filtre de contenu permet d'évaluer les documents une fois téléchargés, un filtre contextuel permet de filtrer les documents avant leur téléchargement à partir du contexte des liens les référençant et finalement, un filtre structurel permet de filtrer les documents avant leur téléchargement en analysant la structure de leur URL. Afin d'améliorer le pouvoir filtrant, initialement nul des filtres structurels et contextuels, le filtre de contenu joue le rôle d'oracle pour les deux autres filtres. Ainsi au fur et à mesure de l'évolution de la recherche dynamique, les filtres structurels et contextuels deviennent plus efficaces réduisant le nombre de documents non pertinents téléchargés. Ce système se décrit naturellement en WETDL en définissant les filtres spécifiques comme des opérateurs externes et en utilisant un opérateur de téléchargement ainsi que deux opérateurs d'extraction. Ceci montre l'expressivité et l'utilité d'un langage comme WETDL.

Conclusion

Dans cette partie nous avons étudié le problème de la construction d'applications d'extraction d'informations à partir du Web. Dans un premier temps nous avons décrit plusieurs tâches classiques comme l'accès à une source de donnée Web, l'extraction massive des données d'un site, et la combinaisons d'accès à plusieurs sites. L'étude de ces différentes tâches nous a permis de mettre en avant différents opérateurs génériques pour l'extraction d'information permettant, par exemple, la construction d'une requête pour une source donnée, le téléchargement d'une page, l'extraction de données, *etc.* Le paramétrage et la combinaison de ces opérateurs permet la construction d'applications d'extraction d'informations à partir du Web. Nous avons défini le langage WETDL permettant de décrire de tels applications en paramétrant et composant un ensemble d'opérateurs génériques. Nous avons montré l'expressivité et la généralité du langage WETDL en décrivant, à l'aide de ce langage, de multiples tâches d'extractions d'informations. Les différences entre ces tâches montrent que le langage WETDL est souple et peut être facilement adapté. Par ailleurs, nous avons étudié le problème de l'intégration des Web Services et des tâches d'extraction d'informations. D'une part, les Web Services constituent une nouvelle source d'information qui peut être utile pour répondre à un besoin d'information. D'autre part, une tâche d'extraction d'information peut être interprétée comme un nouveau Web Service. Nous avons montré comment l'accès à un Web Service peut être décrit en WETDL. Finalement, nous avons décrit comment à l'aide du langage WETDL il était possible de décrire simplement une tâche aussi complexe que le parcours intelligent du Web.

Conclusion générale

Bilan

Dans cette thèse nous nous sommes intéressés au problème de l'extraction d'informations à partir du Web. La première partie de cette thèse a été consacrée à au problème de la construction semi-automatique d'adaptateurs pour des sources de données Web. Dans la seconde partie de cette thèse nous avons étudiés le problème de la construction d'applications d'extraction d'informations à partir du Web.

Dans le cas du problème de la construction des adaptateurs, les méthodes existantes d'extraction d'informations posent plusieurs problèmes. Les méthodes à base de pages exemples nécessitent un étiquetage complet, fastidieux et sans erreurs d'un ensemble de pages exemples. Les méthodes structurelles permettent de détecter certaines régularités de manière automatique mais nécessitent l'intervention de l'utilisateur pour choisir le motif d'extraction qu'il souhaite utiliser et de définir comment transformer les données extraites dans le format dans lequel il souhaite les avoir. Finalement, les méthodes à base de connaissances nécessitent la construction d'une base de connaissance contenant l'ensemble des formats dans lesquelles peuvent se trouver les données.

La solution que nous avons proposé pour la construction d'un adaptateur pour une source de données Web s'appuie sur un ensemble d'instances exemples d'une relation à extraire de la source. Les contextes de ces instances sont alors extraits d'un ensemble de pages de la source. La généralisation des contextes génère des motifs permettant l'extraction de nouvelles instances. Cette méthode a plusieurs avantages par rapport aux méthodes précédentes. Elle permet à l'utilisateur d'exprimer simplement son besoin d'information comme un ensemble d'instances d'une relation à extraire. Le nombre d'instances exemples à donner est souvent réduit : généralement deux ou trois instances par formats présents dans la source suffisent. Le format des données extraites est le même que celui des instances. Ceci permet d'utiliser les instances extraites comme exemples lorsque la reconstruction d'un adaptateur devient nécessaire à cause d'un changement de format de la source. La méthode est efficace : dans nos expérimentations la généralisation des motifs se fait en quelques secondes. Ainsi, l'obtention d'un adaptateur pour une source est aisée et rapide. De plus, la reconstruction automatique d'un nouvel adaptateur est facilitée.

L'autre aspect auquel nous nous sommes intéressés est la construction d'applications d'extraction d'informations. Actuellement, la réalisation de telles applications passe par leur implémentation dans un langage de programmation. Ceci n'est pas satisfaisant pour plusieurs raisons. Décrire l'accès à de nombreuses sources de données nécessite l'écriture pour chaque source d'un programme spécifique car la variété des sources ne permet pas l'écriture d'un programme générique d'accès à une source. Toutefois, les opérations réalisées dans chacun des programmes sont très similaires. Par ailleurs, les sources de données Web évoluent rapidement ce qui nécessite la réécriture des applications d'accès.

Pour répondre à ces problèmes nous proposons de décomposer chaque application en un ensemble d'opérations génériques. Le paramétrage et la composition permettent de construire rapidement des applications Web. Pour décrire de telles compositions nous avons définis le langage WETDL. Ce langage permet de décrire simplement une tâche d'extraction d'informations. Nous avons montré comment exprimer diverses applications en utilisant ce langage. La diversité des applications montre l'utilité et l'expressivité du langage. De plus, nous avons montré comment intégrer très simplement l'accès à des Web Services dans une tâche d'extraction d'informations.

Les idées développées dans cette thèse ont été mises en oeuvre et expérimentées. Notamment, deux systèmes ont été implémentés : le système IEREL permettant de construire un adaptateur pour une source donnée et le système WEBSOURCE permettant d'exécuter une tâche décrite en WETDL. L'utilisation de ces deux systèmes permet de construire rapidement des applications accédant à une ou plusieurs sources de données Web. Dans un premier temps, IEREL est utilisé pour construire un adaptateur permettant d'extraire les résultats d'une page de la source. Dans un second temps, l'infrastructure permettant d'interroger la source, d'obtenir l'ensemble de pages résultats et appliquant l'adaptateur obtenue est décrit en WETDL. La description obtenue peut alors être exécutée en utilisant WEBSOURCE pour toute requête donnée en entrée. Ainsi, nous donnons une réponse complète au problème de l'accès à une source de données Web telles qu'elles existent à ce jour. Cette accès permet d'intégrer les sources de données Web dans une application quelconque et notamment un système de médiation de données.

Perspectives

Amélioration du système IEREL

Dans l'état actuel le système IEREL impose que l'ordre des valeurs des exemples soit le même que l'ordre des occurrences des valeurs dans le document. Il est envisageable de lever cette contrainte. Dans ce cas il sera nécessaire d'adapter les heuristiques de recherche afin qu'elles permettent de ne retrouver que les occurrences "valides".

Le langage des motifs utilisé actuellement par IEREL peut s'avérer dans certains cas un peu rigide. Il serait intéressant d'étudier d'autres langages de motifs. Par exemple, dans l'état actuel de la méthode, deux balises distinctes ne sont pas généralisables. Il pourrait être envisageable de généraliser deux balises de même nom mais ayant des attributs différents. Ceci permettrait, dans certains cas, de réduire le nombre de motifs et d'obtenir des motifs plus généraux.

Les contextes utilisés par IEREL sont basés sur les chaînes se trouvant avant, entre et après les valeurs des occurrences. Il est envisageable de considérer d'autres représentations des contextes en changeant la modélisation du document. Par exemple, il est possible de représenter le document non par une succession de symboles mais comme un arbre DOM. Le contexte pourrait être un ensemble de chemins XPath permettant de déterminer une occurrence d'une instance et ses valeurs. Une autre possibilité, permettant en plus la prise en compte de plusieurs représentations d'un même document, serait l'utilisation de la logique du premier ordre pour décrire les occurrences. Dans ce cas les techniques de programmation logique inductive pourraient être utilisées pour la généralisation des contextes.

Construction automatique de l'accès à une source

L'une des motivations derrière nos recherches sur l'extraction d'informations à partir du Web était la volonté de permettre une intégration facile de plusieurs sources de données Web. Dans l'idéal, cette intégration doit pouvoir se faire simplement en donnant une référence vers le site (*i.e.* son URL). Par exemple, l'intégration de sources telles que la FNAC, Amazon et Alapage doit pouvoir se faire en spécifiant simplement respectivement www.fnac.com, www.amazon.fr, www.alapage.com au système. Dans l'état actuel de nos recherches, ceci peut être simplement obtenu en associant à chaque URL une description WETDL. Cependant, ceci nécessite, d'une part, un travail en amont consistant à construire la description, et d'autre part une maintenance de cette description. Différents sujets de recherches découlent immédiatement de la volonté d'automatiser ces deux tâches.

Un premier problème est celui de construire automatiquement un adaptateur pour une source. Les recherches sur les adaptateurs à base de connaissances sont une première solution dans cette direction. En effet, dans l'idéal, une fois la connaissance acquise, aucune intervention humaine n'est nécessaire. Cependant, les méthodes à base de connaissances existantes montrent que la connaissance construite nécessite d'être tenue à jour et qu'elle ne permet pas en pratique une extraction de qualité (cf. section 1.5 du chapitre 1). Une solution que nous pensons qu'il serait intéressant d'étudier, est d'adapter notre méthode basée sur la généralisation des contextes. Plusieurs adaptations sont nécessaires. D'une part, on pourra permettre la descriptions d'exemples génériques en remplaçant les valeurs spécifiques à une source, comme un prix, par une expression représentant une telle valeur. D'autre part, la contrainte d'ordre sur les valeurs d'une instance doit être levée. Ces deux adaptations risquent très probablement de donné lieu à des occurrences "erronées". Ainsi, il sera nécessaire de définir une méthode de sélection des occurrences correctes. Avec ces adaptations, il sera alors possible de spécifier un ensemble d'exemples génériques s'appliquant sur n'importe quelle source. En généralisant les occurrences de ces exemples on pourra alors construire un adaptateur pour chaque source.

Un second problème consiste à construire de manière automatique le réseaux d'opérateurs permettant, à partir d'une requête, d'interroger la source et d'appliquer l'adaptateur sur chacune des pages. Pour cela, il est nécessaire (1) de déterminer l'ensemble des opérateurs nécessaires, (2) de déterminer leur paramétrage et (3) de déterminer leur coordination.

Extensions de WETDL

Dans son état actuel, le langage WETDL permet de répondre à un besoin spécifique : la construction d'applications d'extraction d'information à partir du Web. Cependant, le modèle de réseaux d'opérateur peut certainement s'appliquer dans d'autres domaines. Dans ce cas il sera nécessaire de définir les opérateurs spécifiques au domaine.

Par ailleurs, tous les opérateurs définis pour l'extraction d'informations sont des opérateurs de décomposition ou de transformation. Dans certains cas il peut être intéressant de recomposer un objet à partir des entrées de l'opérateur. Il serait intéressant d'étudier l'impact de l'introduction d'un tel opérateur. En effet, dans ce cas, un tel opérateur peut ne pas générer de données en sortie car toutes les données nécessaires à produire le résultat ne lui sont pas encore parvenues.

Les deux modèles d'exécution que nous avons proposé ne considèrent qu'un seul flux de contrôle. Or la décomposition en opérateurs suggère qu'une parallélisation du flux de contrôle est envisageable. Dans ce cas, un flux de contrôle est associé à chaque opérateur. Une question qui se pose alors est celle de la synchronisation des flux.

Bibliographie

- [1] Brad ADELBERG. NoDoSE—a tool for semi-automatically extracting structured and semistructured data from text documents. In Laura M. HAAS et Ashutosh TIWARY, réds., *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 283–294, Seattle, Washington, USA, June 1998. ACM Press.
- [2] Tony ANDREWS, Francisco CURBERA, Hitesh DHOLAKIA, Yaron GOLAND, Johannes KLEIN, Frank LEYMANN, Kevin LIU, Dieter ROLLER, Doug SMITH, Satish THATTE, Ivana TRICKOVIC et Sanjiva WEERAWARANA. Business Process Execution Language for Web Services Version 1.1, 2003. Available at <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>,
- [3] N. ASHISH et C. KNOBLOCK. Wrapper Generation for Semi-structured Internet Sources. In *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, AZ, USA, 1997.
- [4] Robert BAUMGARTNER, Sergio FLESCA et Georg GOTTLÖB. Declarative Information Extraction, Web Crawling, and Recursive Wrapping with Lixto. In Thomas EITER, Wolfgang FABER et Mirosław TRUSCZYŃSKI, réds., *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, LNCS/LNAI, pages 21–40, Vienna, Austria, September 2001. Springer Verlag.
- [5] Robert BAUMGARTNER, Sergio FLESCA et Georg GOTTLÖB. Visual Web Information Extraction with Lixto. In Peter M. G. APERS, Paolo ATZENI, Stefano CERI, Stefano PARABOSCHI, Kotagiri RAMAMOCHANARAO et Richard T. SNODGRASS, réds., *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases*, pages 119–128, Roma, Italy, August 2001. Morgan Kaufmann.
- [6] Krishna BHARAT. SearchPad: Explicit Capture of Search Context to Support Web Search. In *Proceedings of the WWW9 Conference*. Elsevier Science, 2000.
- [7] I. BOURNAUD et J.-G. GANASCIA. Conceptual Clustering of Complex Objects: A Generalisation Space based Approach. In Gerard ELLIS, Robert LEVINSON, William RICH et John F. SOWA, réds., *Conceptual Structures: Applications, Implementation and Theory, Third International Conference on Conceptual Structures, ICCS'95*, number 954 in LNCS, pages 172–187, Santa Cruz, California, USA, August 1995. Springer Verlag.
- [8] Brian E. BREWINGTON et George CYBENKO. How dynamic is the Web? In *Proceedings of the WWW9 Conference*, pages 257–276. Elsevier Science, 2000.
- [9] Laura BRIGHT, Jean-Robert GRUSER, Louiqa RASCHID et Maria Esther VIDAL. A wrapper generation toolkit to specify and construct wrappers for Web accessible data sources (WebSources). *International Journal of Computer Systems Science and Engineering*, 1999.
- [10] Sergey BRIN. Extracting Patterns and Relations from the World Wide Web. In Paolo ATZENI, Alberto O. MENDELZON et Giansalvatore MECCA, réds., *Proceedings of The World Wide Web and Databases, International Workshop WebDB'98*, number 1590 in LNCS, Valencia, Spain, March 1998. Springer Verlag.
- [11] Julien CARME, Aurélien LEMAY et Joachim NIEHREN. Learning Node Selecting Tree Transducer from Completely Annotated Examples. In *International Conference on Grammar Induction*, pages 29–102, 2004.

- [12] Jeromy CARRIÈRE et Rick KAZMAN. WebQuery: Searching and Visualizing the Web through Connectivity. In *Proceedings of the 6th International World-Wide Web Conference*, Santa Clara, California, USA, 1997.
- [13] Soumen CHAKRABARTI, Martin van den BERG et Byron DOM. Focused crawling: a new approach to topic-specific Web resource discovery. In *Proceeding of the 8th WWW Conference*, pages 545–562, Toronto, Canada, 1999.
- [14] Chia-Hui CHANG, Chun-Nan HSU et Shao-Chen LUI. Automatic Information Extraction from Semi-Structured Web Pages by Pattern Discovery. *Decision Support Systems Journal*, 35(1), April 2003.
- [15] Chia-Hui CHANG et Shao-Chen LUI. IEPAD : Information Extraction based on Pattern Discovery. In *Proceedings of the ACM WWW10 Conference*. ACM Press, 2001.
- [16] Christian CHARRAS et Thierry LECROQ. *Handbook of Exact String Matching Algorithms*. King's College Publications, 2004.
- [17] William W. COHEN. Automatically Extracting Features for Concept Learning from the Web. In Pat LANGLEY, réd., *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 159–166, Stanford, CA, USA, July 2000. Morgan Kaufmann.
- [18] William W. COHEN, Matthew HURST et Lee S. JENSEN. A Flexible Learning System for Wrapping Tables and Lists in HTML. In *Proceedings of the 11th WWW Conference*, Honolulu, Hawaii, USA, 2002. Elsevier Science.
- [19] J. CONWAY. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.
- [20] Mark CRAVEN, Dan DIPASQUO, Dayne FREITAG, Andrew MCCALLUM, Tom M. MITCHELL, Kamal NIGAM et Sean SLATTERY. Learning to construct knowledge bases from the World Wide. *Artificial Intelligence*, 2000.
- [21] Mark CRAVEN, Dan DIPASQUO, Dayne FREITAG, Andrew K. MCCALLUM, Tom M. MITCHELL, Kamal NIGAM et Seán SLATTERY. Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, pages 509–516, Madison, US, 1998. AAAI Press, Menlo Park, US.
- [22] Valter CRESCENZI, Giansalvatore MECCA et Paolo MERIALDO. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In *The VLDB Journal*, pages 109–118, 2001.
- [23] Luiz A. G. da COSTA, Paulo F. PIRES et Marta MATTOSO. Automatic Composition of Web Services with Contingency Plans. In *Proceedings of the IEEE International Conference on Web Services*, pages 454–461, 2004.
- [24] Michelangelo DILIGENTI, Frans COETZEE, Steve LAWRENCE, C. Lee GILES et Marco GORI. Focused Crawling Using Context Graphs. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 527–534, Caire, Egypt, 2000. Morgan Kaufmann Publishers Inc.
- [25] Document Object Model (DOM), 2001. Available at <http://www.w3.org/DOM/>,
- [26] Vincent DUBOIS. *Apprentissage approximatif et extraction de connaissances à partir de données textuelles*. Thèse de Doctorat, IRIN - Institut de Recherche en Informatique de Nantes, Université de Nantes, 2003.
- [27] Vincent DUBOIS, Mohamed QUAFAROU et Benjamin HABEGGER. Mining crawled data and visualizing discovered knowledge. In *Web Intelligence: Research and Development. Proceedings of The First Asia-Pacific Conference on Web Intelligence*, pages 493–507. Springer-Verlag, 2001. LNAI 2198,

- [28] Line EIKVIL. Information Extraction from World Wide Web - A Survey. Rapport technique, Norweigan Computing Center, 1999.
- [29] David W. EMBLEY, Douglas M. CAMPBELL, Y. S. JIANG, Stephen W. LIDDLE, Yiu-Kai NG, Dallan QUASS et Randy D. SMITH. A Conceptual-Modeling Approach to Extracting Data from the Web. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 78–91, 1998.
- [30] David W. EMBLEY, Douglas M. CAMPBELL, Stephen W. LIDDLE et Randy D. SMITH. Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 52–59, Bethesda, Maryland, USA, 1998. ACM Press.
- [31] David W. EMBLEY et L. XU. Record Location and Reconfiguration in Unstructured Multiple-Record Web Documents. In *WebDB (Informal Proceedings)*, 2000.
- [32] Ronen FELDMAN, Yonatan AUMANN, Moshe FRESKO, Orly LIPSHTAT, Binyamin ROSENFELD et Yonatan SCHLER. Text Mining via Information Extraction. In *Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery*, LNCS, pages 165–173. Springer Verlag, 1999.
- [33] Ronen FELDMAN et Ido DAGAN. Knowledge discovery in textual databases (kdt). In Usama M. FAYYAD et Ramasamy UTHURUSAMY, réds., *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 112–117. AAAI Press, September 1995.
- [34] J. FIELDER et J. HAMMER. Using Mobile Crawlers to Search the Web Efficiently. *International Journal of Computer and Information Science*, 1(1), 2000.
- [35] Dayne FREITAG et Nicholas KUSHMERICK. Boosted Wrapper Induction. In *Proceedings of the 17th National Conference on Artificial Intelligence AAAI-2000*, pages 577–583, 2000.
- [36] Georg FUELLEN. Multiple Alignment. *Complexity International*, 4, 1997.
- [37] Jean-Gabriel GANASCIA. Algebraic Structure of Some Learning Systems. In Klaus P. JANTKE, Shigenobu KOBAYASHI, Etsuji TOMITA et Takashi YOKOMORI, réds., *Proceedings of the Algorithmic Learning Theory, 4th International Workshop, ALT'93*, number 744 in LNCS, pages 398–409, Tokyo, Japan, November 1993. Springer Verlag.
- [38] Jean-Gabriel GANASCIA. TDIS : an Algebraic Formalization. In Ruzena BAJCSY, réd., *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, volume 2, pages 1008–1015, Chambéry, France, September 1993. Morgan Kaufmann.
- [39] Jean-Gabriel GANASCIA, Jérôme THOMAS et Philippe LAUBLET. Integrating Models of Knowledge and Machine Learning. In *Proceedings of Machine Learning: ECML-93, European Conference on Machine Learning*, number 667 in LNCS, pages 396–401, Vienna, Austria, April 1993. Springer Verlag.
- [40] X. GAO et L. STERLING. AutoWrapper: automatic wrapper generation for multiple online services. In *Proceedings of the Asia Pacific Web Conference*, , Hong Kong, September 1999.
- [41] X. GAO et L. STERLING. Semi-Structured Data Extraction from Heterogeneous Sources. In *Second International Workshop on Innovative Internet Information Systems (IIIS'99)*, Copenhagen, Denmark, 1999.

- [42] Paulo Braz GOLGHER, Alberto H. F. LAENDER, Altigran Soares da SILVA et Berthier A. RIBEIRO-NETO. An Example-Based Environment for Wrapper Generation. In *ER Workshops*, 2000.
- [43] Georg GOTTLÖB, Christoph KOCH et Klaus U. SCHULZ. Conjunctive Queries Over Trees. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 189–200, Paris, France, June 2004. ACM Press.
- [44] Stéphane GRUMBACH et Giansalvatore MECCA. In Search of the Lost Schema. In *Proceeding of the 7th International Conference on Database Theory*, LNCS, pages 314–331. Springer Verlag, 1999.
- [45] Jean-Robert GRUSER, Louiqa RASCHID, M. E. VIDAL et Laura BRIGHT. Wrapper Generation for Web Accessible Data Sources. In *Conference on Cooperative Information Systems*, 1998.
- [46] Suhit GUPTA, Gail KAISER, David NEISTADT et Peter GRIMM. DOM-based Content Extraction of HTML Documents. In *Proceedings of the 12th WWW Conference*, Budapest, Hungary, 2003. Elsevier Science.
- [47] D. GUSFIELD. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.
- [48] Benjamin HABEGGER et Mohamed QUAFAROU. A pragmatic approximation operator. In S. HIRANO, M. INUIGUCHI et S. TSUMOTO, réds., *Bulletin of International Rough Set Society. Proceedings of the JSAI International Workshop on Rough Set Theory and Granular Computing*, volume 5, pages 51–56, May 2001.
- [49] Benjamin HABEGGER et Mohamed QUAFAROU. Multi-pattern wrappers for relation extraction. In F. van HARMELAN, réd., *ECAI 2002. Proceedings of the 15th European Conference on Artificial Intelligence*, Amsterdam, 2002. IOS Press.
- [50] Benjamin HABEGGER et Mohamed QUAFAROU. Building web information extraction tasks. In *WI'2004. Proceedings of the ACM/IEEE Web Intelligence Conference*, Beijing, China, 2004.
- [51] Benjamin HABEGGER et Mohamed QUAFAROU. Context generalization for information extraction from the web. In *WI'2004. Proceedings of the ACM/IEEE Web Intelligence Conference*, Beijing, China, 2004.
- [52] Benjamin HABEGGER et Mohamed QUAFAROU. Web services for information extraction from the web. In *ICWS 2004. Proceedings of the IEEE International Conference on Web Services*, 2004.
- [53] Benjamin HABEGGER et Mohamed QUAFAROU. WetDL : A web information extraction language. In *ADVIS'2004. Proceeding of the third international conference on Advances in Information Systems*, LNCS, Izmir, Turkey, 2004.
- [54] Rachid HAMADI et Boualem BENATALLAH. A Petri net-based model for web service composition. In *Proceedings of the Fourteenth Australasian database conference on Database technologies*, volume 17 of *CRPITS*, pages 191–200, Adelaide, Australia, 2003. Australian Computer Society, Inc.
- [55] Joachim HAMMER, Hector GARCIA-MOLINA, Junghoo CHO, Arturo CRESPO et Rohan ARANHA. Extracting Semistructured Information from the Web. In *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, AZ, USA, 1997.
- [56] C. HSU.. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In *Papers from the 1998 Workshop on AI and Information Integration*. AAAI Press, 1998.

- [57] Chun-Nan HSU et Ming-Tzung DUNG. Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web. *Information Systems*, 23(8), 1998.
- [58] Gerald HUCK, Peter FANKHAUSER, Karl ABERER et Erich NEUHOLD. Jedi: Extracting and Synthesizing Information from the Web. In Michael HALPER, réd., *Proc. 3rd IFCIS Intl. Conf. on Cooperative Information Systems, CoopIS'98*, pages 32–43, New York, New York, USA, 1998. IEEE Computer Society Press.
- [59] Jugal K. KALITA et Paritosh ROHILLA. Automated Information Extraction from Web Pages Using An Interactive Learning Agent. In *Proceedings of the 2001 International Conference on Web Intelligence*, 2001.
- [60] Michael KIFER, Georg LAUSEN et James WU. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM (JACM)*, 42(4), 1995.
- [61] Craig KNOBLOCK, Kristina LERMAN, Steven MINTON et Ion MUSLEA. Accurately and Reliably Extracting Data from the Web : A Machine Learning Approach. *Data Engineering Bulletin*, 23(4), 2003.
- [62] R. KOSALA, M. BRUYNNOOGHE, J. V. den BUSSCHE et H. BLOCKEEL. Information Extraction from web documents based on local unranked tree automaton inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003)*, pages 403–408, 2003.
- [63] Raymond KOSALA et Hendrik BLOCKEEL. Web Mining Research : A Survey. *SIGKDD Explorations*, 2(1), 2000.
- [64] Lukasz A. KURGAN, Waldemar SWIERCZ et Krzysztof J. CIOŚ. Semantic Mapping of XML Tags Using Inductive Machine Learning. In M. Arif WANI, Hamid R. ARABINIA, Khalid HAFEEZ et Graham KENDALL, réds., *Proceedings of the 2002 International Conference on Machine Learning and Applications - ICMLA 2002*, pages 99–109, Las Vegas, Nevada, USA, June 2002. CSREA Press.
- [65] Nicholas KUSHMERICK. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 2000.
- [66] Nicholas KUSHMERICK. Learning to Invoke Web Forms. In Robert MEERSMAN, Zahir TARI et Douglas C. SCHMIDT, réds., *CoopIS/DOA/ODBASE*, Lecture Notes in Computer Science, pages 997–1013, Catania, Sicily, Italy, 2003. Springer Verlag.
- [67] Nickolas KUSHMERICK, Daniel S. WELD et Robert B. DOORENBOS. Wrapper Induction for Information Extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.
- [68] Nicolas KUSHMERICK. *Wrapper Induction for Information Extraction*. Thèse de Doctorat, University of Washington, 1997.
- [69] Steve LAWRENCE. Context in Web Search. *IEEE Data Engineering Bulletin*, 23(3), 2000.
- [70] Kristina LERMAN, Craig KNOBLOCK et Steven MINTON. Automatic Data Extraction from Lists and Tables in Web Sources. In *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, Seattle, Washington, August 2001.
- [71] Kristina LERMAN, Steven N. MINTON et Craig A. KNOBLOCK. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research*, 18, 2003.
- [72] Alon Y. LEVY, Anand RAJARAMAN et Joann J. ORDILLE. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the Twenty-second International Conference on Very Large Databases*, pages 251–262, Bombay, India, 1996. VLDB Endowment, Saratoga, Calif.

- [73] Alon Y. LEVY et Marie-Christine ROUSSET. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1-2), September 1998.
- [74] Frank LEYMANN. Web Services Flow Language (WSFL 1.0), 2001. Available at <http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>,
- [75] Jiamao LIU, Juntao CUI et Ning GU. Composing Web services Dynamically and Semantically. In *IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-East'04)*, pages 234–241, 2004.
- [76] Ling LIU. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *Proceedings of the 16th International Conference on Data Engineering*, pages 611–621, San Diego, California, USA, March 2000. IEEE Computer Society.
- [77] Emmanuelle MARTIENNE. *EAGLE : un système pour la découverte interactive de descriptions de concepts basée sur la théorie des ensembles approximatifs*. Thèse de Doctorat, IRIN - Institut de Recherche en Informatique de Nantes, Université de Nantes, 2000.
- [78] Wolfgang MAY, Rainer HIMMERÖDER, Georg LAUSEN et Bertram LUDÄSCHER. A Unified Framework for Wrapping, Mediating and Restructuring Information from the Web. In *International Workshop on the World-Wide Web and Conceptual Modeling (WWWCM'99)*, LNCS, pages 307–320, Paris, France, November 1999. Springer Verlag.
- [79] Wolfgang MAY et Georg LAUSEN. Information extraction from the Web. Rapport technique, Institut für Informatik Albert-Ludwigs-Universität, 2000.
- [80] Wolfgang MAY et Georg LAUSEN. A uniform framework for integration of information from the web. *Information Systems*, 29, March 2004.
- [81] B. MEDJAHED, A. BOUGUETTAYA et A. K. ELMAGARMID. Composing Web services on the Semantic Web. *The VLDB Journal*, 12(4), 2003.
- [82] Gerome MIKLAU et Dan SUCIU. Containment and Equivalence for an XPath Fragment. In *Symposium on Principles of Database Systems*, pages 65–76, Madison, Wisconsin, USA, 2002. ACM Press.
- [83] Tom MITCHELL. *Version Spaces : An approach to Concept Learning*. Thèse de Doctorat, Stanford University, 1978.
- [84] Tom M. MITCHELL. Generalization as Search. *Artificial Intelligence*, 18(2), 1982.
- [85] Tom M. MITCHELL. *Machine Learning*. WCB/McGraw-Hill, 1997.
- [86] Sougata MUKHERJEA. WTMS : A System for Collecting and Analyzing Topic-Specific Web Information. In *Proceedings of the WWW9 Conference*. Elsevier Science, 2000.
- [87] Tadao MURATA. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4), April 1989.
- [88] I. MUSLEA, S. MINTON et C. KNOBLOCK. STALKER: Learning extraction rules for semistructured, Web-based information sources. In *Proceedings of AAAI-98 Workshop on AI and Information Integration*. AAAI Press, 1998.
- [89] Ion MUSLEA. Extraction Patterns for Information Extraction Tasks: A Survey. In *The AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
- [90] Ion MUSLEA, Steven MINTON et Craig A. KNOBLOCK. Hierarchical Wrapper Induction for Semistructured Information Sources. *Autonomous Agents and Multi-Agent System*, 4(1-2), March 2001.

- [91] Ion MUSLEA, Steven MINTON et Craig A. KNOBLOCK. Active + Semi-supervised Learning = Robust Multi-View Learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 435–442. Morgan Kaufmann Publishers Inc., 2002.
- [92] Bart ORRIËNS, Jian YANG et Mike P. PAPAZOGLU. A Framework for Business Rule Driven Service Composition. In *ECOMO, ER 2003 Workshop Proceedings*, number 2814 in LNCS, pages 52–64, 2003.
- [93] Andreas PAEPCKE, Hector GARCIA-MOLINA, Gerard RODRIGUEZ et Junghoo CHO. Beyond Document Similarity: Understanding Value-Based Search and Browsing Technologies. *SIGMOD Records*, 29(1), 2000.
- [94] Giacomo PINCCINELLI et Scott Lane WILLIAMS. Workflow: A Language for Composing Web Services. In *Second International Workshop on Composition Languages*, 2002.
- [95] Sriram RAGHAVAN et Hector GARCIA-MOLINA. Crawling the Hidden Web. In Peter M. G. APERS, Paolo ATZENI, Stefano CERI, Stefano PARABOSCHI, Kotagiri RAMAMOHANARAO et Richard T. SNODGRASS, réds., *Vldb 2001, Proceedings of 27th International Conference on Very Large Data Bases*, pages 129–138, Roma, Italy, August 2001. Morgan Kaufmann.
- [96] G. RAMALHO et J.-G. GANASCIA. Simulating Creativity in Jazz Performance. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 108–113, Seattle, Washington, USA, August 1994. Springer Verlag.
- [97] Bradley L. RICHARDS et Raymond J. MOONEY. Learning Relations by Pathfinding. In *National Conference on Artificial Intelligence*, pages 50–55, 1992.
- [98] Ellen RILLOFF. Automatically Generating Extraction Patterns from Untagged Text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1044–1049, Portland, Oregon, August 1996. AAAI Press / The MIT Press.
- [99] M. ROTH et P. SCHWARZ. Don't Scrap It, Wrap it! A Wrapper Architecture for Legacy Data Sources. In *Prococeedings of the 23 VLDB Conference*, 1997.
- [100] Gunter SABLON. *Iterative Version Spaces with an Application in Inductive Logic Programming*. Thèse de Doctorat, Katholieke Universiteit Leuven, 1995.
- [101] Gunther SABLON, Luc De RAEDT et Maurice BRUYNOOGHE. Iterative versionspaces. *Artificial Intelligence*, 69, 1994.
- [102] Arnaud SAHUGUET et Fabien AZAVANT. Looking at the Web through XML Glasses. In *Proceedings of the Conference on Cooperative Information Systems*, pages 148–159, 1999.
- [103] Arnaud SAHUGUET et Fabien AZAVANT. Web Ecology: Recycling HTML Pages as XML Documents Using W4F. In *WebDB (Informal Proceedings)*, 1999.
- [104] Gwen SALAÜN, Lucas BORDEAUX et Marco SCHAEFER. Proceedings of the IEEE International Conference on Web Services. In *Proceedings of the IEEE International Conference on Web Services*, 2004.
- [105] Heekyoung SEO, Jaeyoung YANG et Joongmin CHOI. Knowledge-based Wrapper Generation by Using XML. In *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, Seattle, Washington, August 2001.
- [106] Seán SLATTERY et Mark CRAVEN. Combining Statistical and Relational Methods for Learning in Hypertext Domains. In David PAGE, réd., *Proceedings of the 8th Int. Conference on Inductive Logic Programming*, number 1446 in LNAI, pages 38–52, Madison, US, 1998. Springer Verlag.

- [107] Stephen SODERLAND. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning*, 1999.
- [108] Neel SUNDARESAN et Jeonghee YI. Mining the Web for relations. In *Proceedings of the WWW9 Conference*, pages 699–711. Elsevier Science, 2000.
- [109] Satish THATTE. XLANG : Web Services for Business Process Design, 2001. Available at http://www.gotdotnet.com/team/xml_wsspecs/clang-c/default.htm,
- [110] Anthony TOMASIC, Louiqa RASCHID et Patrick VALDURIEZ. Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering*, 10(5), 1998.
- [111] Michael USCHOLD. Where Are the Semantics in the Semantic Web. *AI Magazine*, 2003.
- [112] Victor VIANU. A Web Odyssey: From Codd to XML. In *Symposium on Principles of Database Systems*, 2001.
- [113] M. VIDAL et L. RASCHID. WebSrcMed: A Mediator for Scaling up to Multiple Web Accessible Sources (WebSources). Rapport technique, UMIACS, University of Maryland, 1998.
- [114] XML Path Language (XPath), 1999. Available at <http://www.w3.org/TR/xpath>,
- [115] XSL Transformations (XSLT), 1999. Available at <http://www.w3.org/TR/xslt>,
- [116] Yan ZHU, Christof BORNHOEVD, Doris SAUTNER et Alejandro P. BUCHMANN. Materializing Web Data for OLAP and DSS. In *Proc. of the 1st International Conference on Web-Age Information Management (WAIM00)*, LNCS. Springer Verlag, 2000.
- [117] J.-D. ZUCKER et J.-G. GANASCIA. Learning structurally indeterminate clauses. In David PAGE, éd., *Proceedings of the 8th Int. Conference on Inductive Logic Programming*, number 1446 in LNAI, pages 235–244, Madison, US, 1998. Springer Verlag.

Liste des tableaux

Partie I — Extraction d'informations à partir du web

1.1	Résultats publiés pour le système WIEN	14
1.2	Résultats publiés pour le système STALKER	20
1.3	Classification des symboles utilisé dans SOFTMEALY	22
1.4	Résultats de SOFTMEALY publiés par les auteurs	25
1.5	Résultats publiés pour le système IEPAD	31
1.6	Résultats publiés pour le système ROADRUNNER	35
1.7	Résultats de la méthode de Gao et Sterling	43
1.8	Résultats publié pour le système Xtros	47
1.9	Comparaison des systèmes d'extraction d'informations à partir du Web	49
2.1	Table des correspondances des codes et des balises pour l'exemple	63
3.1	Résultats pour les <i>Annuaire en ligne</i>	80
3.2	Exemples d'entrées pour chaque classe dans AddressFinder	82
3.3	Couverture des motifs d'extraction de la relation <i>email(nom, email, organisation)</i> sur AddressFinder	83
3.4	Exemples de résultats sponsorisés ou non de Metacrawler	85
3.5	Résultats pour les <i>moteurs de recherche</i>	88
3.6	Résultats sur les catalogues	90
3.7	Séquence d'exemples pour c-1	92
3.8	Séquence d'exemples pour c-5	93

Partie II — Construction d'applications d'extraction d'informations à partir du Web

5.1	Opérateurs d'extraction d'informations	120
6.1	Données sur les brevets extraits	136

Table des figures

Partie I — Extraction d'informations à partir du web

1.1	Un exemple de document d'une source fictive	8
1.2	Label d'une page contenant M tuples d'une relation à K attributs.	9
1.3	Le label du document de la figure 1.1	9
1.4	Exemple de décomposition d'une page	16
1.5	Forme générale d'un automate à délimiteur simple	17
1.6	Exemples pour l'apprentissage d'une règle gauche pour les occurrence n_i d'un noeud N	17
1.7	Exemple d'occurrence d'un noeud "Nom" à l'intérieur d'une occurrence d'un noeud "TUPLE(Restaurant)"	18
1.8	Raffinements de la règle ($SkipTo(;;)$)	19
1.9	Exemple de transducteur dans SOFTMEALY	23
1.10	Résultat de l'encodage de la source donnée par la figure 1.1	27
1.11	Codes binaires associés aux symboles de la source encodée de la figure 1.10.	27
1.12	Codage en binaire de la source encodée de la figure 1.10	27
1.13	Exemple d'exécution de la fonction <i>match</i> (figure issue de [22])	33
1.14	Principe de la dualité motif-relation dans DIPRE	37
1.15	Construction d'un nouveau tuple (C, D) d'une relation à partir d'un exemple (A, B) en utilisant comme motif intermédiaire (P, M, S)	39
1.16	Exemple de hierarchie de concepts dans la méthode de Gao et Sterling	40
1.17	Abstraction d'un document avec la méthode de Gao et Sterling	43
1.18	Exemple d'application de la méthode Gao et Sterling	44
1.19	Exemple de description d'un prix dans XTROS	46
2.1	Première page de résultats de la requête "Depardieu" sur Amazon.fr	54
2.2	Parties d'un motifs	56
2.3	Un exemple de document source	62
2.4	Relation à extraire	62
2.5	Description du processus d'extraction d'information basé sur la généralisation des contextes	63
2.6	Exemple de document codé	63
2.7	Généralisation des contextes	65
2.8	Décomposition d'un motif	71
2.9	Execution de la méthode de généralisation	72
2.10	Exemples négatifs	72
3.1	Interface de l'application	78
3.2	Occurrences des exemples pour OKRA	81
3.3	Exemples de résultats dans chacun des trois formats du moteur de recherche Excite	84
3.4	Recherche d'occurrence erronée	86

3.5	Extraction erronée	87
3.6	Exemple de résultats du site de la FNAC	89
3.7	Évaluation du nombre d'exemples nécessaire pour Amazon	91
3.8	Évaluation de la qualité des exemples	92
3.9	Fnac - évaluation de l'influence du nombre d'exemples	93
3.10	Fnac - évaluation de la qualité des exemples	94

Partie II — Construction d'applications d'extraction d'informations à partir du Web

4.1	Extraction des réponses de Google exécutée manuellement	104
4.2	Accès manuel à Amazon.fr	106
4.3	Accès manuel à Conforama.fr	107
4.4	Accès manuel à dblp.uni-trier.de	110
4.5	Combinaison de l'accès à deux annuaires en ligne	112
4.6	Mise en oeuvre d'un comparateur de prix de DVD	113
4.7	Mise en oeuvre d'un méta moteur de recherche	114
5.1	Exemple de description d'un opérateur de construction de requête HTTP	121
5.2	Exemple de description d'un opérateur de téléchargement	121
5.3	Exemple de description d'un opérateur d'analyse syntaxique	122
5.4	Exemple de description d'un opérateur de filtrage	122
5.5	Exemple de description d'un opérateur d'extraction	123
5.6	Exemple de description d'un opérateur de transformation	124
5.7	Exemple de description d'un opérateur de traitement externe	125
5.8	Description d'une tâche fictive	125
5.9	Exemple programme logique issu d'une description d'un réseau WetDL	129
5.10	Exemple de réseau de pétri calculant $x = (a + b)/(a - b)$	130
5.11	Un opérateur WETDL traduit en réseau de petri	131
6.1	Réseaux correspondant à la tâche d'extraction Amazon	134
6.2	Réseau correspondant à la tâche d'extraction Pagesjaunes.fr	134
6.3	Réseau correspondant à la tâche d'extraction CIA World Fact Book	135
6.4	Tâche d'extraction d'un ensemble de brevets	136
6.5	Réseau correspondant à la tâche d'extraction DBLP	137
6.6	Description d'une requête pour l'interrogation d'un annuaire multiple	138
6.7	Réseau décrivant la tâche d'interrogation de plusieurs annuaires en ligne	138
7.1	Exemple de message SOAP	143
7.2	Description du service de cotation en ligne proposé par XMethods	144
7.3	Exemple de description d'un opérateur de construction de requête HTTP	147
8.1	Architecture d'un crawler intelligent	152
8.2	Réseaux WETDL d'une tâche de recherche dynamique	155

Liste des algorithmes

1.1	Algorithme <i>exec_{LR}</i>	11
1.2	Induction des délimiteurs gauches pour un adaptateur de la classe <i>LR</i>	13
1.3	Induction d'une disjonction dans STALKER	19
1.4	Construction d'un délimiteur dans STALKER	20
1.5	Algorithme de généralisation de SOFTMEALY	24
1.6	Alignement multiple de <i>n</i> séquence avec l'algorithme <i>center star</i>	30
1.7	Algorithme de recherche du motif le plus fréquent dans XTROS	47
2.1	Algorithme de recherche des occurrences d'une instances	66
2.2	Fonction de généralisation de deux motifs	67
2.3	Fonction de généralisation de deux parties	68
2.4	Extraction de contextes et construction d'un motif	69
2.5	Application de l'adaptateur généré à un document	70
2.6	Fonction de généralisation de deux motifs en tenant compte des exemples négatifs	73
4.1	Accès générique à une source classique	103
5.1	Algorithme d'exécution d'un réseau WETDL paresseuse	127
5.2	Algorithme d'exécution d'un réseau WETDL par saturation	128
8.1	Algorithme de parcours naïf du Web	150
8.2	Algorithme "intelligent" de parcours du Web	153

Table des matières

Introduction	ix
1.1 Problème de l'extraction d'informations	ix
1.2 Illustration du problème d'extraction d'informations	ix
1.3 Contributions de la thèse	xi
1.4 Organisation du mémoire	xii

Partie I — Extraction d'informations à partir du web

Introduction	3
1 Extraction d'informations à partir du Web	5
1.1 Construction manuelle d'adaptateurs	6
1.2 Induction d'adaptateurs à base de pages étiquetées	7
1.2.1 WIEN	7
1.2.2 STALKER	15
1.2.3 SOFTMEALY	21
1.2.4 Synthèse	25
1.3 Extraction de motifs via l'analyse de la structure des documents	26
1.3.1 IEPAD : Chang	26
1.3.2 ROADRUNNER	31
1.3.3 Synthèse	36
1.4 Extraction de relations	36
1.4.1 DIPRE : Brin	36
1.5 Adaptateurs à base de connaissances	39
1.5.1 Gao et Sterling	40
1.5.2 XTROS	45
1.5.3 Synthèse	48
1.6 Résumé	48
1.7 Ouvertures de recherche	50
1.7.1 Extraction d'informations à partir d'instances exemples	50
1.7.2 Formalisme pour le développement d'applications Web	51
2 Extraction par généralisation de contextes	53
2.1 Notions	54
2.2 Généralisation de contextes	57
2.2.1 Généralisation des symboles	57
2.2.2 Généralisation des parties	58
2.2.3 Généralisation des motifs	59
2.2.4 Propriétés de la généralisation	59

2.3	Méthode	61
2.3.1	Prétraitement de la source	62
2.3.2	Recherche des occurrences	64
2.3.3	Extraction des contextes	65
2.3.4	Généralisation des contextes	65
2.3.5	Résumé	66
2.4	Algorithmes	66
2.4.1	Recherche des occurrences et extraction des contextes	66
2.4.2	Généralisation des contextes	67
2.4.3	Construction d'un adaptateur	71
2.4.4	Exemples négatifs	71
2.5	Analyse	73
2.5.1	Complexité de la recherche des occurrences et extraction des contextes	74
2.5.2	Complexité de la généralisation des contextes	74
2.5.3	Analyse comparative	74
2.6	Conclusion	76
3	Expérimentations du système IEREL	77
3.1	Caractéristiques de IEREL	77
3.2	Méthodologie	78
3.3	Annuaire en ligne	79
3.3.1	BigBook	80
3.3.2	OKRA	80
3.3.3	AddressFinder	80
3.3.4	Synthèse	82
3.4	Moteurs de recherche	83
3.4.1	AltaVista	83
3.4.2	Excite	83
3.4.3	Galaxy	84
3.4.4	Overture	84
3.4.5	Metacrawler	85
3.4.6	Google	85
3.5	Catalogues de produits	87
3.5.1	Alapage	87
3.5.2	Amazon	87
3.5.3	FNAC	88
3.5.4	Conforama	89
3.6	Nombre et qualité des exemples	90
3.6.1	Amazon	90
3.6.2	Fnac	91
3.7	Conclusion	93
	Conclusion	95

Partie II — Construction d'applications d'extraction d'informations à partir du Web

Introduction	99
4 Applications d'extraction d'informations	101
4.1 Accéder à une source Web	101
4.1.1 Interrogation d'un moteur de recherche : Google	103
4.1.2 Accéder à la base de DVD chez Amazon.fr	105
4.1.3 Accéder à une base d'offres de mobilier chez Conforama	105
4.1.4 Synthèse	108
4.2 Extraction d'informations pour la constitution de bases de données	108
4.2.1 Extraction de brevets	108
4.2.2 CIA world fact book	109
4.2.3 Construction d'une base de conférences	109
4.2.4 Synthèse	109
4.3 Interrogation de service multiples	111
4.3.1 Annuaire en ligne	111
4.3.2 Comparateur de prix de DVD	112
4.3.3 Meta-moteur de recherche	113
4.3.4 Conclusion	113
4.4 Conclusion	113
5 Langage WetDL	115
5.1 Types, objets et données	116
5.2 Décomposition d'une tâche d'extraction d'informations	116
5.2.1 Construction d'une requête HTTP : opérateur <code>query</code>	117
5.2.2 Exécution d'une requête HTTP : opérateur <code>fetch</code>	118
5.2.3 Analyse d'un document : opérateur <code>parse</code>	118
5.2.4 Extraction de sous-parties : opérateur <code>extract</code>	118
5.2.5 Transformation de données : opérateurs <code>transform</code>	119
5.2.6 Filtrage de données : opérateur <code>filter</code>	119
5.2.7 Modules externes : opérateur <code>external</code>	119
5.2.8 Synthèse	120
5.3 Coordination des tâches	120
5.4 Description des opérateurs	120
5.4.1 <code>query</code> : opérateur de construction de requête	120
5.4.2 <code>fetch</code> : opérateur de téléchargement	121
5.4.3 <code>parser</code> : opérateur d'analyse XML	122
5.4.4 <code>filter</code> : opération de filtrage	122
5.4.5 <code>extract</code> : opérateur d'extraction	123
5.4.6 <code>transform</code> : opérateur de transformation	123
5.4.7 <code>external</code> : opérateur externe	123
5.5 Coordination des opérateurs	124
5.6 Définition d'un réseau WetDL	125
5.7 Modèle d'exécution	125
5.7.1 État d'un réseau WETDL	126

5.7.2	Algorithme paresseux	126
5.7.3	Algorithme par saturation	128
5.8	Interprétation d'un réseau WetDL	128
5.8.1	Comme un système déductif	128
5.8.2	Comme un réseau de pétri	130
5.9	Conclusion	131
6	Expérimentations du système WEBSOURCE	133
6.1	Interrogation de sources Web	133
6.1.1	Amazon.fr : un site de e-commerce	133
6.1.2	Pagesjaunes.fr : un annuaire en ligne	134
6.2	Extraction massive de données	135
6.2.1	CIA World Fact Book	135
6.2.2	Extraction de brevets	136
6.2.3	Extraction de la base de conférences de DBLP	136
6.3	Combinaison de sources	137
6.3.1	Annuaire multiples	137
6.4	Conclusion	139
7	Web Services et extraction d'informations	141
7.1	Web Services	142
7.1.1	Protocole SOAP	142
7.1.2	Description de services en WSDL	143
7.2	Composition de WS	143
7.3	Composition automatique de Web Services	145
7.4	Web Service pour l'extraction d'informations	146
7.4.1	Web Services comme source d'informations	146
7.4.2	Génération de Web Service	146
7.5	Conclusion	147
8	Parcours intelligent du Web	149
8.1	Parcours exhaustif du Web	150
8.2	Parcours intelligent du Web	150
8.2.1	Architecture du crawler	152
8.2.2	Filtre de contenu	153
8.2.3	Filtre contextuel	153
8.2.4	Filtre structurel	154
8.2.5	Description à l'aide de WETDL	155
8.3	Application: le système E-awacs	155
8.4	Conclusion	156
	Conclusion	157
	Conclusion générale	159

Bibliographie	163
Liste des tableaux	171
Table des figures	173
Table des matières	177
A Notations	185
A.1 Chaînes de symboles	185
A.2 Listes	185
B DTD de WetDL	187
C Exemples de descriptions de tâches en WETDL	193
C.1 Extraction de DVD d'Amazon	193
C.2 Interrogation des pages blanches	195
C.3 Extraction d'informations du CIA World Fact Book	197
C.4 Extraction de brevets	198
C.5 Extraction d'informations sur les conférences	203
C.6 Interrogation d'annuaires multiples	205
C.7 Description WETDL permettant de calculer la séquence de nombres de la suite de Fibonacci	208

Annexes

ANNEXE A

Notations

Dans cette annexe nous présentons les notations que nous utilisons au travers de cette thèse.

A.1 Chaînes de symboles

Soit Σ un alphabet. Soit $s, t, u \in \Sigma$.

- $\cdot$ dénote la concaténation de chaînes (par exemple si $s = "abc"$ et $t = "fg"$ alors $s \cdot t = "abcfg"$);
- $|s|$ dénote le nombre de caractères contenus dans s (i.e. la longueur de s);
- $s[i]$ dénote la caractère de position i dans s ;
- $s[i, j]$ dénote la sous-chaîne de s débutant à la position i et se terminant à la position j ;
- $\text{prefixes}(s) = \{s' / \exists w \in \Sigma^*, w = s' \cdot w\}$ dénote l'ensemble des préfixes de s ;
- $\text{suffixes}(s) = \{s' / \exists w, \in \Sigma^*, w = w \cdot s'\}$ dénote l'ensemble des suffixes de s .

A.2 Listes

Soit l une liste non vide. Soit l_1, l_2 deux listes.

- $[]$ dénote la liste vide;
- $\text{premier}(l)$ retourne le premier élément de l ;
- $\text{reste}(l)$ renvoie la liste l privée de son premier élément;
- $\text{concat}(l_1, l_2)$ renvoie la concaténation de la liste l_1 avec la liste l_2 .

ANNEXE B

DTD de WetDL

```
1 <!--
2
3 WetDL DTD Version 2.0
4 Author      : Benjamin Habegger
5 Last modified : 2004-01-19
6
7 -->
8
9 <!--
10 **
11 ** wet:source : root element of a WebSource Description
12 ** - Elements are :
13 **   - an eventual wet:options element containing general options
14 **   - a list of module declarations
15 ** - Attributes :
16 **   - name : gives a name to this Web Source Description
17 -->
18 <!ELEMENT wet:source (
19     wet:options?,
20     (
21         wet:query|wet:extract|wet:fetch|wet:format|wet:filter|
22         wet:dummy|wet:external|wet:xmlparser|wet:cache
23     )*
24 )
25 >
26 <!ATTLIST wet:source
27 name CDATA #REQUIRED
28 xmlns:wet CDATA #FIXED "http://wetdl.free.fr/ns/wetdl"
29 >
30
31 <!--
32 **
33 ** wet:options option declaration (used by wet:set-attribute)
34 **
35 -->
36 <!ELEMENT wet:options (options*)>
37 <!ELEMENT option EMPTY>
38 <!ATTLIST option
39 name CDATA #REQUIRED
40 shortcut CDATA #REQUIRED
41 type CDATA "string"
42 >
```

```

43
44 <!ELEMENT wet:set-attribute EMPTY>
45 <!ATTLIST wet:set-attribute
46   name    CDATA #REQUIRED
47   value-of CDATA #REQUIRED
48 >
49
50 <!--
51 **
52 ** Entity task-attribs defines attributes commun to all modules
53 **
54 -->
55 <!ENTITY % task-attribs
56   "name    ID    #REQUIRED
57   forward-to IDREFS #IMPLIED"
58 >
59
60 <!--
61 **
62 ** parameters : element allowing to declare a set of attribute value pairs
63 **
64 -->
65
66 <!ELEMENT parameters (param+)>
67 <!ELEMENT param (wet:set-attribute?)>
68 <!ATTLIST param
69   name    ID    #REQUIRED
70   default CDATA #IMPLIED
71   type    CDATA #IMPLIED
72 >
73
74 <!--
75 **
76 ** data : element allowing to push initial data into the
77 **       module during initialisation
78 **
79 -->
80 <!ELEMENT data (#PCDATA)>
81
82 <!--
83 *****
84 ** wet:query describes a query task
85 ** - A base element giving the uri of the script to query
86 ** - A list of parameters which are added to the query
87 ** - A method attribut to say which type of HTTP method
88 **   to use
89 *****
90 -->
91 <!ELEMENT wet:query (base,parameters)>
92 <!ATTLIST wet:query
93   method (GET|POST) "GET"
94   %task-attribs;

```

```

95 >
96
97 <![ELEMENT base (#PCDATA)> <!-- Contains the URI of the source -->
98
99 <!--
100 *****
101 ** wet:extract describes an extraction task
102 ** - Default attributes
103 ** - A type attribute to specify the type of extraction
104 **   extraction method used
105 *****
106 -->
107
108 <![ELEMENT wet:extract ANY>
109 <![ATTLIST wet:extract
110 type (xpath|xslt) "xpath"
111 limit CDATA #IMPLIED
112 %task-attrs;
113 >
114 <![ELEMENT path (#PCDATA)>
115
116 <!--
117 *****
118 ** wet:fetch describes a fetching task
119 ** - Default attributes
120 ** - No elements
121 *****
122 -->
123 <![ELEMENT wet:fetch EMPTY>
124 <![ATTLIST wet:fetch
125 %task-attrs;
126 >
127
128 <!--
129 *****
130 ** wet:xmlparser describes a parsing task
131 ** - Default attributes
132 ** - No elements
133 *****
134 -->
135 <![ELEMENT wet:xmlparser EMPTY>
136 <![ATTLIST wet:xmlparser
137 %task-attrs;
138 >
139
140 <!--
141 *****
142 ** wet:filter describes a filtering task
143 ** - Default attributes
144 ** -
145 *****
146 -->

```

```

147 <!ELEMENT wet:filter ANY>
148 <!ATTLIST wet:filter
149 type CDATA #IMPLIED
150 %task-attrs;
151 >
152
153 <!ELEMENT test EMPTY>
154 <!ATTLIST test
155   select CDATA      #REQUIRED
156   match CDATA       #REQUIRED
157   action (keep|reject) "keep"
158   type  (regex|approx) "regex"
159 >
160
161 <!--
162 *****
163 ** wet:cache : describes a caching task
164 ** Attributes :
165 ** - directory : defines in which directory to set results
166 ** - template  : defines the name template to use when
167 **               creating a new file
168 *****
169 -->
170 <!ELEMENT wet:cache EMPTY>
171 <!ATTLIST wet:cache
172 template CDATA "webservice-XXXXX"
173 directory CDATA #IMPLIED
174 %task-attrs;
175 >
176
177 <!--
178 *****
179 ** wet:format describes a formatting task
180 ** - Default attributes
181 ** - A format attribute describing the desired formatting
182 ** - No elements
183 *****
184 -->
185 <!ELEMENT wet:format EMPTY>
186 <!ATTLIST wet:format
187 format (text|xml|details) "xml"
188 %task-attrs;
189 >
190
191 <!--
192 *****
193 ** wet:dummy describes a dummy task (just forwards what it
194 ** receives)
195 ** - Default attributes
196 ** - Eventually initial data

```

```
199 *****
200 -->
201 <!ELEMENT wet:dummy (data*)>
202 <!ATTLIST wet:dummy
203 %task-attrs;
204 >
205
206 <!--
207 *****
208 ** wet:external
209 ** - Default attributes
210 ** - A module attribute specifying the perl module
211 **   which handles this module
212 *****
213 -->
214 <!ELEMENT wet:external ANY>
215 <!ATTLIST wet:external
216 %task-attrs;
217 module CDATA #REQUIRED
218 >
```


Exemples de descriptions de tâches en WETDL

C.1 Extraction de DVD d'Amazon

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <!DOCTYPE ws:source SYSTEM "/usr/local/share/perl/5.8/WebSource/websource.dtd">
3 <ws:source name="Google.fr" xmlns:ws="http://wwwsource.free.fr/ns/websource">
4   <ws:options>
5     <option name="titre" shortcut="t" type="string" />
6     <option name="acteur" shortcut="a" type="string" />
7     <option name="realisateur" shortcut="r" type="string" />
8     <option name="genre" shortcut="g" type="string" />
9     <option name="public" shortcut="p" type="string" />
10    <option name="format" shortcut="f" type="string" />
11  </ws:options>
12
13  <ws:fetch name="init-amazon" type="xml" forward-to="dvd-link-finder">
14    <data>http://www.amazon.fr/</data>
15  </ws:fetch>
16
17  <ws:extract name="dvd-link-finder" forward-to="follow-dvd-link">
18    <path>//area[@alt="DVD"]/@href</path>
19  </ws:extract>
20
21  <ws:fetch name="follow-dvd-link" type="xml" forward-to="query-page-finder" />
22
23  <ws:extract name="query-page-finder" forward-to="fetch-query-page">
24    <path>//a[contains(., "recherche")]/@href</path>
25  </ws:extract>
26
27  <ws:fetch name="fetch-query-page" type="xml" forward-to="extract-query-uri" />
28  <ws:extract name="extract-query-uri" forward-to="q">
29    <path>//table//form/@action</path>
30  </ws:extract>
31
32  <ws:query name="q" method="POST" forward-to="fetcher">
33    <parameters>
34      <param name="qtype" default="at" />

```

```

35 <param name="rank" default="+amzrank" />
36 <param name="field-0" default="title" />
37 <param name="query-0" default="">
38   <ws:set-attribute name="default" value-of="titre" />
39 </param>
40 <param name="field-actor" default="">
41   <ws:set-attribute name="default" value-of="acteur" />
42 </param>
43 <param name="field-director" default="">
44   <ws:set-attribute name="default" value-of="realisateur" />
45 </param>
46 <param name="field-subject" default="">
47   <ws:set-attribute name="default" value-of="genre" />
48 </param>
49 <param name="field-cnc-rating" default="">
50   <ws:set-attribute name="default" value-of="public" /> <!-- possible pg 12 16 18 -->
51 </param>
52 <param name="index" default="dvd-fr">
53   <ws:set-attribute name="default" value-of="format" /> <!-- possible dvd-fr vhs-fr -->
54 </param>
55 </parameters>
56 </ws:query>
57
58 <ws:fetch name="fetcher" forward-to="cache" />
59
60 <ws:cache name="cache" template="amazon-XXXXX" directory="/tmp"
61   forward-to="parser extractor" />
62
63 <ws:xmlparser name="parser" forward-to="next"/>
64 <ws:extract name="next" forward-to="fetcher" method="xpath">
65   <path>//a[img[contains(@src,"button-more-results.gif")]]/@href</path>
66 </ws:extract>
67
68 <ws:external name="extractor" module="amazon_dvd" />
69
70 <!--
71 <ws:extract name="links" method="xpath">
72   <path>//p[@class="g"]/a[1]/@href</path>
73 </ws:extract>
74 -->
75 </ws:source>

```

C.2 Interrogation des pages blanches

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <!DOCTYPE ws:source SYSTEM "/usr/local/share/perl/5.8.2/WebSource/websource.dtd">
3 <ws:source name="pages-blanches-fetcher">
4   <ws:options>
5     <option name="ville" shortcut="v"/>
6     <option name="nom" shortcut="n"/>
7     <option name="prenom" shortcut="p"/>
8     <option name="template" shortcut="t"/>
9     <option name="repertoire" shortcut="r"/>
10  </ws:options>
11
12  <ws:query name="query" method="POST" forward-to="fr-fetch">
13    <base>http://www.pagesjaunes.fr/pb.cgi</base>
14    <parameters>
15      <param type="hidden" name="faire" default="decode_input_image"/>
16      <param type="hidden" name="DEFAULT_ACTION"
17        default="jf_inscriptions_req"/>
18      <param type="hidden" name="mode" default="web"/>
19      <param type="hidden" name="titre_bandeau"
20        default="Votre recherche Pages&#10; Blanches :"/>
21      <param type="hidden" name="lang" default="FR"/>
22      <param type="hidden" name="TYPE_RECHERCHE" default="ZZZ"/>
23      <param type="text" name="FRM_NOM">
24        <ws:set-attribute name="default" value-of="nom" />
25      </param>
26      <param type="text" name="FRM_PRENOM" default="">
27        <ws:set-attribute name="default" value-of="prenom" />
28      </param>
29      <param type="text" name="FRM_ADRESSE" default="">
30      <param type="text" name="FRM_LOCALITE">
31        <ws:set-attribute name="default" value-of="ville" />
32      </param>
33      <param type="text" name="FRM_DEPARTEMENT" default="">
34      <param type="submit" name="JF_INSCRIPTIONS_REQ"
35        default="Lancer la recherche"/>
36    </parameters>
37  </ws:query>
38
39  <ws:fetch name="fr-fetch" forward-to="cache fr-next" />
40
41  <ws:extract name="fr-next" type="form" forward-to="fr-fetch">
42    <parameters>
43      <param name="form" value="form_suite" />
44    </parameters>
45  </ws:extract>
46
47  <ws:cache name="cache">
48    <ws:set-attribute name="template" value-of="template" />
49    <ws:set-attribute name="directory" value-of="repertoire" />

```

50 </ws:cache>
51 </ws:source>

C.3 Extraction d'informations du CIA World Fact Book

```
1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <!DOCTYPE ws:source SYSTEM "/usr/local/share/perl/5.8.2/WebSource/websource.dtd">
3 <ws:source name="CIA World Fact Book"
4   xmlns:ws="http://wwwsource.free.fr/ns/websource"
5   xmlns:xi="http://www.w3.org/2003/XInclude"
6   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
7
8   <ws:fetch name="fetch-list" forward-to="extract-list" type="xml">
9     <data>http://www.cia.gov/cia/publications/factbook/countrylisting.html</data>
10  </ws:fetch>
11
12  <ws:extract name="extract-list" forward-to="fetch-item">
13    <path>//li/a/@href</path>
14  </ws:extract>
15
16  <ws:fetch name="fetch-item" forward-to="extract-item" type="xml" />
17  <ws:extract name="extract-item" type="xslt" forward-to="format">
18    <xi:include href="cia-wfb.xsl" />
19  </ws:extract>
20
21  <ws:format name="format" format="xml"/>
22 </ws:source>
```

C.4 Extraction de brevets

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <ws:source name="bonjour.fr"
3   xmlns:ws="http://wwwsource.free.fr/ns/websource">
4
5   <ws:query name="query" method="GET" forward-to="fetcher">
6     <base>http://patft.uspto.gov/netacgi/nph-Parser</base>
7     <parameters>
8       <param type="hidden" name="Sect1" default="PTO2"/>
9       <param type="hidden" name="Sect2" default="HITOFF"/>
10      <param type="hidden" name="u" default="/netahtml/search-adv.htm"/>
11      <param type="hidden" name="r" default="0"/>
12      <param type="hidden" name="p" default="1"/>
13      <param type="hidden" name="f" default="S"/>
14      <param type="hidden" name="l" default="50"/>
15      <param type="textarea" name="Query"/>
16      <param type="option" name="d" default="ptxt"/>
17    </parameters>
18  </ws:query>
19
20  <ws:fetch name="fetcher" forward-to="follow next" />
21
22  <ws:extract name="next" forward-to="fetcher" method="xpath" format="url">
23    <path>//a[img[@src="/netaicon/PTO/nextlist.gif"]]/@href</path>
24  </ws:extract>
25
26  <ws:extract name="follow" forward-to="fetcher2" method="xpath" format="url">
27    <path>descendant::table[2]/tr[position()=2 and td[1]!="']/td[2]/a/@href</path>
28  </ws:extract>
29
30  <ws:fetch name="fetcher2" forward-to="extract" />
31
32  <ws:extract name="extract" type="xslt" forward-to="format"
33    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
34    <xsl:stylesheet version="1.0">
35      <xsl:output method="xml" indent="yes" />
36      <xsl:template match="/">
37        <patent>
38          <number>
39            <xsl:value-of
40              select="//hr[1]/following-sibling::table[1]/tr[1]/td[2]"
41            />
42          </number>
43          <title>
44            <xsl:value-of
45              select="//hr[2]/following-sibling::font[1]"
46            />
47          </title>
48          <icl>
49            <xsl:value-of
50              select="substring-before(descendant::p/table//tr[2]/td[2]/b/i,' ')"

```

```

51     />
52 </icl>
53 <ccl>
54   <xsl:value-of
55     select="//p/table//tr[1]/td[2]/b"
56   />
57 </ccl>
58 <issue-date>
59   <xsl:value-of
60     select="//hr[1]/following-sibling::table[1]/tr[2]/td[2]"
61   />
62 </issue-date>
63 <primary-examinor>
64   <xsl:value-of
65     select="//i[contains(text(),'Primary Examiner:')]/following-sibling::text()[1]" />
66 </primary-examinor>
67 <assistant-examinor>
68   <xsl:value-of
69     select="//i[contains(text(),'Assistant Examiner:')]/following-sibling::text()[1]" />
70 </assistant-examinor>
71 <agent>
72   <xsl:value-of
73     select="//i[contains(text(),'Attorney, Agent or Firm:')]/following-sibling::text()[1]" />
74 </agent>
75
76 <xsl:apply-templates
77   select="//hr[3]/following-sibling::table[1]/tr"
78   mode="rel-data" />
79
80 <xsl:apply-templates
81   select="//center[contains(b/text(),'U.S. Patent Documents')]/following-sibling::table[1]"
82   mode="references" />
83
84 <xsl:apply-templates
85   select="//center[contains(b/text(),'Foreign Application Priority Data')]/following-sibling::table[1]"
86   mode="fdata" />
87 </patent>
88 </xsl:template>
89
90 <xsl:template match="tr" mode="rel-data">
91   <xsl:choose>
92     <xsl:when test="td[contains(text(),'Inventors')]">
93       <xsl:apply-templates select="td[2]" mode="inventors"
94       />
95     </xsl:when>
96     <xsl:when test="td[contains(text(),'Assignee')]">
97       <xsl:apply-templates select="td[2]" mode="assignee"
98       />
99     </xsl:when>
100    <xsl:when test="td[contains(text(),'Appl. No.')] ">
101      <xsl:apply-templates select="td[2]" mode="appl-num"
102      />

```

```

103     </xsl:when>
104     <xsl:when test="td[normalize-space(text())='Filed:']">
105         <xsl:apply-templates select="td[2]" mode="filed-date"
106             />
107     </xsl:when>
108     <!--
109     <xsl:otherwise>
110         <unknown type="{td[1]}">
111         <xsl:value-of select="td[2]" />
112     </unknown>
113     </xsl:otherwise>
114     -->
115 </xsl:choose>
116 </xsl:template>
117
118 <xsl:template match="td" mode="inventors">
119     <xsl:for-each select="b">
120         <inventor>
121             <name><xsl:value-of select="normalize-space(text())" /></name>
122             <address>
123                 <xsl:value-of
124                     select="substring-before(substring-after(following-sibling::text()[1],',' ),',' )" />
125             </address>
126         </inventor>
127     </xsl:for-each>
128 </xsl:template>
129
130 <xsl:template match="td" mode="assignee">
131     <xsl:for-each select="b">
132         <assignee>
133             <name><xsl:value-of select="normalize-space(text())" /></name>
134             <address>
135                 <xsl:value-of
136                     select="substring-before(substring-after(following-sibling::text()[1],',' ),',' )" />
137             </address>
138         </assignee>
139     </xsl:for-each>
140 </xsl:template>
141
142 <xsl:template match="td" mode="appl-num">
143     <application-number>
144         <xsl:value-of select="normalize-space(.)" />
145     </application-number>
146 </xsl:template>
147
148 <xsl:template match="td" mode="filed-date">
149     <filed>
150         <xsl:value-of select="normalize-space(.)" />
151     </filed>
152 </xsl:template>
153
154 <xsl:template match="table" mode="references">

```

```
155     <refs>
156     <xsl:choose>
157       <xsl:when test="tr[td/@colspan]">
158         <xsl:apply-templates select="tr[td/@colspan]/preceding-sibling::tr" mode="references"/>
159       </xsl:when>
160       <xsl:otherwise>
161         <xsl:apply-templates select="tr" mode="references" />
162       </xsl:otherwise>
163     </xsl:choose>
164   </refs>
165
166   <frefs>
167     <xsl:apply-templates
168       select="tr[td/@colspan]/following-sibling::tr" mode="references" />
169   </frefs>
170 </xsl:template>
171
172 <xsl:template match="tr" mode="references">
173   <ref>
174     <number>
175       <xsl:value-of select="normalize-space(td[1])" />
176     </number>
177     <issue-date>
178       <xsl:value-of select="normalize-space(td[2])" />
179     </issue-date>
180     <inventors>
181       <xsl:value-of select="normalize-space(td[3])" />
182     </inventors>
183     <icl>
184       <xsl:value-of select="normalize-space(td[4])" />
185     </icl>
186   </ref>
187 </xsl:template>
188
189 <xsl:template match="tr" mode="references">
190   <fref>
191     <number>
192       <xsl:value-of select="normalize-space(td[1])" />
193     </number>
194     <issue-date>
195       <xsl:value-of select="normalize-space(td[2])" />
196     </issue-date>
197     <inventor-country>
198       <xsl:value-of select="normalize-space(td[3])" />
199     </inventor-country>
200   </fref>
201 </xsl:template>
202
203 <xsl:template match="table" mode="fdata">
204   <prirs>
205     <xsl:apply-templates select="tr" mode="fdata"/>
206   </prirs>
```

```
207 </xsl:template>
208
209 <xsl:template match="tr" mode="fdata">
210   <prir>
211     <date>
212       <xsl:value-of select="substring-before(td[2]/text(), '[')" />
213     </date>
214     <country>
215       <xsl:value-of select="normalize-space(substring-before(substring-after(td[2]/text(), '['), ']))" />
216     </country>
217     <number>
218       <xsl:value-of select="normalize-space(td[3])" />
219     </number>
220   </prir>
221 </xsl:template>
222 </xsl:stylesheet>
223 </ws:extract>
224
225 <ws:format name="format" format="xml" root="patents"/>
226
227 </ws:source>
```

C.5 Extraction d'informations sur les conférences

```

1 <?xml version="1.0" ?>
2 <!DOCTYPE ws:source SYSTEM "/usr/local/share/perl/5.8.0/WebSource/websource.dtd" >
3 <ws:source name="db-acronymes">
4   <ws:dummy name="init" forward-to="f1">
5     <data>http://www.informatik.uni-trier.de/~ley/db/conf/indexa.html</data>
6   </ws:dummy>
7
8   <ws:fetch name="f1" forward-to="x1"/>
9   <ws:xmlparser name="x1" forward-to="e1" />
10
11   <ws:extract name="e1" forward-to="f2">
12     <!-- extract link for each letter index page -->
13     <path>//h3/a/@href</path>
14   </ws:extract>
15
16   <ws:fetch name="f2" forward-to="x2" />
17   <ws:xmlparser name="x2" forward-to="e2" />
18
19   <ws:extract name="e2" forward-to="f3">
20     <!-- extract conference page from index page -->
21     <path>//li/a/@href</path>
22   </ws:extract>
23
24   <ws:fetch name="f3" forward-to="x3" />
25   <ws:xmlparser name="x3" forward-to="e3" />
26
27   <ws:extract name="e3" type="xslt" forward-to="select">
28     <xi:include xmlns:xi="http://www.w3.org/2001/XInclude" href="dblp-conf-info.xsl" />
29   </ws:extract>
30
31   <ws:extract name="select" format="string" forward-to="split">
32     <path>//edition</path>
33   </ws:extract>
34
35   <ws:extract name="split" type="regexp" forward-to="db">
36     <regexp>(?:[0-9]+\.[ ]?([^\s]*)(?:[ ]?/?(?:[0-9]+\.[ ]?([^\s]*))?)? ([0-9]{4}): ([^\s,]+), (?:([^\s,]+), )?(.*)</regexp>
37     <map>
38       <key>acronyme</key>
39       <key>acronyme2</key>
40       <key>year</key>
41       <key>city</key>
42       <key>province</key>
43       <key>country</key>
44     </map>
45   </ws:extract>
46
47   <ws:database name="db" forward-to="format">
48     <parameters>
49       <param name="db" value="dbmasters" />
50       <param name="user" value="habegger" />

```

```
51    <param name="pass" value="n62Odj" />
52  </parameters>
53  <query>
54    INSERT INTO conference (acronyme, year, city, province, country)
55    VALUES ('$acronyme', $year, '$city', '$province', '$country')
56  </query>
57 </ws:database>
58
59 <ws:format name="format" format="details" />
60 </ws:source>
```

C.6 Interrogation d'annuaires multiples

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <!DOCTYPE ws:source SYSTEM "/usr/local/share/perl/5.8.2/WebSource/websource.dtd">
3 <ws:source name="Google.fr"
4   xmlns:ws="http://wwwsource.free.fr/ns/websource"
5   xmlns:wst="http://wwwsource.free.fr/ns/websource-types"
6   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
7
8   <ws:options>
9     <option name="fname" shortcut="f"/>
10    <option name="lname" shortcut="l"/>
11    <option name="city" shortcut="c"/>
12    <option name="province" shortcut="p"/>
13    <option name="country" shortcut="C" />
14  </ws:options>
15
16  <ws:dummy name="init" forward-to="fr-filter us-filter">
17    <xmldata>
18      <query>
19        <firstname><ws:set-attribute name="value" value-of="fname" /></firstname>
20        <lastname><ws:set-attribute name="value" value-of="lname" /></lastname>
21        <city><ws:set-attribute name="value" value-of="city" /></city>
22        <province><ws:set-attribute name="value" value-of="province" /></province>
23        <country><ws:set-attribute name="value" value-of="country" /></country>
24      </query>
25    </xmldata>
26  </ws:dummy>
27
28
29  <!--
30  Description des pages blanches francaises
31  -->
32
33  <ws:filter type="tests" name="fr-filter" forward-to="fr-trans-query">
34    <test select="country/@value" match="france" />
35  </ws:filter>
36
37  <ws:extract name="fr-trans-query" type="xslt" forward-to="fr-fetch">
38    <xsl:stylesheet version="1.0">
39      <xsl:template match="query">
40        <wst:http-request base="http://www.pagesjaunes.fr/pb.cgi" method="post">
41          <wst:param type="hidden" name="faire" value="decode_input_image"/>
42          <wst:param type="hidden" name="DEFAULT_ACTION"
43            value="jf_inscriptions_req"/>
44          <wst:param type="hidden" name="mode" value="web"/>
45          <wst:param type="hidden" name="titre_bandeau"
46            value="Votre recherche Pages&#10; Blanches :"/>
47          <wst:param type="hidden" name="lang" value="FR"/>
48          <wst:param type="hidden" name="TYPE_RECHERCHE" value="ZZZ"/>
49          <wst:param type="text" name="FRM_NOM" value="{lastname/@value}"/>

```

```

50 <wst:param type="text" name="FRM_PRENOM" value="{firstname/@value}"/>
51 <wst:param type="text" name="FRM_ADRESSE" value="" />
52 <wst:param type="text" name="FRM_LOCALITE" value="{city/@value}" />
53 <wst:param type="text" name="FRM_DEPARTEMENT" value="{province/@value}" />
54 <wst:param type="submit" name="JF_INSCRIPTIONS_REQ"
55 value="Lancer la recherche"/>
56 </wst:http-request>
57 </xsl:template>
58 </xsl:stylesheet>
59 </ws:extract>
60
61 <ws:fetch name="fr-fetch" forward-to="fr-parser fr-next" />
62 <ws:xmlparser name="fr-parser" forward-to="fr-extract" />
63 <ws:extract name="fr-extract" forward-to="fr-format">
64 <path>//table[@class="fdinscr"]</path>
65 </ws:extract>
66
67 <ws:extract name="fr-next" type="form" forward-to="fr-fetch">
68 <parameters>
69 <param name="form" value="form_suite" />
70 </parameters>
71 </ws:extract>
72
73 <ws:extract name="fr-format" type="xslt" forward-to="output">
74 <xsl:stylesheet version="1.0">
75 <xsl:template match="table">
76 <entry>
77 <name><xsl:value-of select="tr[1]/td/b" /></name>
78 </entry>
79 </xsl:template>
80 </xsl:stylesheet>
81 </ws:extract>
82
83 <!--
84 Description des pages blanches US
85 -->
86
87 <ws:filter type="tests" name="us-filter" forward-to="us-trans-query">
88 <test select="country/@value" match="usa" />
89 </ws:filter>
90
91 <ws:extract type="xslt" name="us-trans-query" forward-to="us-fetch">
92 <xsl:stylesheet version="1.0">
93 <xsl:template match="query">
94 <wst:http-request base="http://directory.superpages.com/wp/results.jsp" method="GET">
95 <wst:param type="hidden" name="SRC" value="" />
96 <wst:param type="hidden" name="PS" value="15"/>
97 <wst:param type="hidden" name="PI" value="1"/>
98 <wst:param type="hidden" name="STYPE" value="WS"/>
99 <wst:param type="text" name="WF" value="{firstname/@value}" />
100 <wst:param type="text" name="WL" value="{lastname/@value}" />
101 <wst:param type="checkbox" name="WES" value="" />

```

```
102     <wst:param type="text"    name="T"    value="{city/@value}" />
103     <wst:param type="option"  name="S"    value="{province/@value}" />
104     <wst:param type="checkbox"   name="expand" value=""/>
105     </wst:http-request>
106   </xsl:template>
107 </xsl:stylesheet>
108 </ws:extract>
109
110 <ws:fetch name="us-fetch" forward-to="us-extract us-parse"/>
111
112 <ws:xmlparser name="us-parse" forward-to="us-next"/>
113
114 <ws:extract limit="1" name="us-next" forward-to="us-fetch">
115   <path>//font/a[contains(.,'Next Page')]/@href</path>
116 </ws:extract>
117
118 <ws:external name="us-extract" module="superpages" forward-to="output"/>
119
120 <!--
121 *****
122 -->
123 <ws:format format="xml" name="output" />
124
125 </ws:source>
```

C.7 Description WETDL permettant de calculer la séquence de nombres de la suite de Fibonacci

```
1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <!DOCTYPE ws:source SYSTEM "webservice.dtd">
3 <ws:source name="test" xmlns:ws="http://wwwsource.free.fr/ns/webservice">
4   <ws:options>
5     <option name="number" shortcut="n" type="string" />
6   </ws:options>
7
8   <ws:dummy name="query" forward-to="transform">
9     <xmldata>
10      <data>
11        <ws:set-attribute name="a" value-of="number" />
12        <ws:set-attribute name="b" value-of="number" />
13      </data>
14    </xmldata>
15  </ws:dummy>
16
17  <ws:extract type="xslt" name="transform" forward-to="transform extract">
18    <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
19      <xsl:template match="data">
20        <data>
21          <xsl:attribute name="a"><xsl:value-of select="@b" /></xsl:attribute>
22          <xsl:attribute name="b"><xsl:value-of select="@a + @b" /></xsl:attribute>
23        </data>
24      </xsl:template>
25    </xsl:stylesheet>
26  </ws:extract>
27
28  <ws:extract name="extract">
29    <path>/data/@b</path>
30  </ws:extract>
31 </ws:source>
```


Extraction d'informations à partir du Web

Benjamin HABEGGER

Résumé

Avec l'essor du Web, de nombreuses sources de données en ligne, telles que des annuaires en ligne, des sites immobiliers, des sites de commerce électronique, des moteurs de recherches, *etc.* sont apparues. Cependant ces sources de données ont été conçues pour être parcourues et visualisées par des utilisateurs humains utilisant un navigateur. Bien que riche en contenu, les formats que prennent ces sources sont des formats de présentation et il est alors difficile pour une machine d'utiliser les informations directement. Pourtant, donner à des programmes informatiques un accès à de telles sources ouvre la porte à de nombreuses applications telles que la création d'agents intelligents accédant aux données du Web, à l'intégration de données du Web dans des systèmes de médiation de données, *etc.* Deux problèmes majeurs se posent pour permettre cet accès. Premièrement, il est nécessaire d'extraire les informations que contiennent les pages résultats d'une source et de les mettre dans un format compréhensible par la machine. Deuxièmement, la machine doit savoir comment accéder à la source. En effet, on doit savoir où poster une requête, comment parcourir l'ensemble de pages résultats, *etc.* Pour résoudre le premier problème nous proposons d'utiliser une méthode dans laquelle l'utilisateur décrit les informations qu'il souhaite extraire en donnant quelques instances exemples de celles-ci. Les contextes d'occurrence de ces exemples sont alors recherchés dans les pages résultats de la source et généralisés afin d'extraire de nouvelles instances. Par rapport aux méthodes existantes dans la littérature celle que nous proposons permet d'extraire précisément les données souhaitées sans toutefois avoir à étiqueter manuellement des pages exemples entières. D'autre part, nous proposons une solution au second problème : permettre à la machine d'accéder à une source. Par l'étude de plusieurs sources, nous avons mis en avant un ensemble d'opérateurs récurrents dont le paramétrage et la combinaison permet l'accès aux sources. Nous proposons le langage WETDL afin de décrire les opérateurs et leur combinaisons. Nous proposons aussi plusieurs algorithmes permettant "d'exécuter" ces descriptions et ainsi réaliser une tâche d'extraction.

Mots-clés : extraction d'informations à partir du web, induction d'adaptateurs, Web

Abstract

With the growth of the Web, many on-line sources such as on-line address books, real estate sites, e-commerce sites, *etc.* have appeared. However these data sources are destined to be accessed and viewed by human users. While being content rich, these pages are in a presentational format thus making it difficult for an automated machine access. However, giving such machine access opens the door to many applications such as allowing intelligent agents to make use of Web sources, allowing to include Web sources in data mediation systems, *etc.* In order to give such an access two major problems need to be resolved. First, it is necessary to be able to extract the information contained in the result documents and put this information into a machine understandable format. Second, the machine must know how to access the source, *i.e.* how to build queries the source will understand, where to post the queries, how to navigate through the result pages, *etc.* To resolve the first problem we propose a method in which the user specifies the information he wishes to extract by giving example instances of this information. The contexts of the occurrences of these instances are searched for in the result pages and generalized allowing to extract unseen instances. Compared the methods of the literature this method allows to precisely extract the desired information without having to fully manually label example pages. We also propose a solution to the second problem : allowing the machine to access a source. By the study of multiple on-line sources we put front a set of recurrent operators whose parameter settings and combination allows to access a source. We propose a language WETDL in order to describe the operators and their combination. We also give different algorithms allowing to "execute" such a description thus allowing to realize an extraction task.

Keywords: information extraction from the web, wrapper induction, Web