N° d'ordre : 56 Année 2012

UNIVERSITE ABDELMALEK ESSAADI
FACULTE DES SCIENCES et TECHNIQUES
TANGER

UFR : Valorisation Biotechnologique des Micro-Organismes

THESE

Présentée

Pour l'obtention du

DOCTORAT EN SCIENCES ET TECHNIQUES

Par :
Abdelaali BRIACHE

Discipline : Bioinformatique

Spécialité : Bioinformatique

Médiation à base d'ontologie pour l'intégration sémantique des données de Saccharomyces cerevisiae

Soutenue le 20/12/2012 devant le Jury

Pr. Khalid LAIRINI	**Faculté des Sciences et Techniques – Tanger**	**Président**
Pr. Omar El BEQQALI	**Faculté des Sciences – Fès**	**Rapporteur**
Pr. M'hamed AIT KBIR	**Faculté des Sciences et Techniques – Tanger**	**Rapporteur**
Pr. Ismael Navas DELGADO	**E.T.S. Ingeniería Informática – Málaga**	**Rapporteur**
Pr. Mohamed ETTAYEBI	**Faculté des Sciences – Fès**	**Examinateur**
Pr. Said BARRIJAL	**Faculté des Sciences et Techniques – Tanger.**	**Examinateur**
Pr. José F. Aldana MONTES	**E.T.S. Ingeniería Informática – Málaga**	**Co-Directeur**
Pr. Badr Din ROSSI HASSANI	**Faculté des Sciences et Techniques – Tanger.**	**Co-Directeur**

Médiation à base d'ontologie pour l'intégration sémantique des données de Saccharomyces cerevisiae

RESUME

Les sources de données biologiques contiennent une très grande richesse d'informations et sont hautement complémentaires. Les biologistes sont systématiquement amenés à consulter ces sources afin d'analyser les résultats de leurs expérimentations et de tirer des conclusions et des hypothèses qui pourraient leur aider à avancer dans leurs recherches. Cependant, le nombre et le contenu de ces sources aujourd'hui disponibles sur le Web ne cessent de croître de façon considérable. Ces sources de données sont à la fois réparties et très hétérogènes : chaque source a sa propre structure et format de représentation de données ; les langages d'interrogation sont très variés; la manière d'accéder aux données change pour chaque source ; les termes scientifiques utilisés pour décrire les données diffèrent d'une source à l'autre...etc. Cela complique la tâche d'interrogation de ces sources et la recherche d'informations par les biologistes. Ce qui rend l'intégration des données biologiques une nécessité afin d'aider les biologistes de remédier à ces problèmes, de bien exploiter les sources de données et d'y tirer le maximum d'avantages.

D'autre part, la levure *Saccharomyces cerevisiae* fut le premier eucaryote à entrer au panthéon des organismes entièrement séquencés. La puissance de sa génétique et l'importance de la communauté scientifique internationale qui s'y consacre ont fait de cet organisme un modèle pour les études fonctionnelles post-séquençage aussi bien qu'une plate-forme expérimentale pour développer et valider de nouvelles approches et stratégies génomiques. Plusieurs sources de données biologiques, qui stockent et organisent les données de la levure *Saccharomyces cerevisiae*, sont publiquement disponibles et accessibles via le Web. Ces sources de données héritent les mêmes problèmes rencontrés dans les bases de données biologiques d'une manière générale.

Au cours de ce travail de thèse, nous avons eu pour objectif la proposition d'un prototype d'intégration de données biologiques de la levure *Saccharomyces cerevisiae*. Il s'agit d'un système, appelé *YeastMed*, à base de médiateur faisant appel à une ontologie pour supporter les requêtes des utilisateurs. *YeastMed* reçoit des requêtes des biologistes à travers une interface Web. Il fait usage ensuite d'un module de réécriture de requêtes pour décomposer la requête utilisateur en un ensemble de sous-requêtes. Ces dernières seront à destination

des sources de données susceptibles de participer à la construction d'une ou plusieurs réponses à la requête utilisateur. Les sources de données sont accédées, par le système, à travers un ensemble de services Web. Le système est doté d'un service Web pour chaque source de données. Ainsi, *YeastMed* fourni un accès transparent et simultané à plusieurs sources de données de la levure *Saccharomyces cerevisiae* via une interface unique.

REMERCIEMENTS

Je tiens à remercier très sincèrement M. le professeur Badr Din Rossi Hassani pour m'avoir accueilli au sein de son équipe afin de mener à bien cette thèse. Merci pour m'avoir permis de faire mes premiers pas dans le monde de la recherche et m'avoir aidé à poursuivre dans cette voie.

Toute ma reconnaissance va à M. José F. Aldana Montes professeur à l'université de Malaga et chef du groupe Khaos, avec lequel j'ai beaucoup appris, pour son soutien sans faille dans la codirection de cette thèse et pour avoir guidé ce travail en conjuguant habilement disponibilité, conseils et critiques constructives tout en laissant libre court à ma créativité.

Je tiens aussi à remercier tous les membres du jury qui m'ont fait l'honneur de porter intérêt à ma thèse et d'avoir accepté de juger ce travail.

Je remercie sincèrement tous les membres du groupe KHAOS, et particulièrement M. Ismael Navas Delgado. Merci pour la confiance qu'il m'a toujours témoigné, ainsi que pour ses conseils, ses encouragements et sa patience.

Je remercie mes parents qui toujours m'ont soutenue, mes frères et mes sœurs pour leur amour inconditionnel et sans limites, pour leurs encouragements et pour leurs conseils. Merci d'avoir cru en moi et m'avoir soutenue tout ou long de ce travail.

Je souhaite remercier ma chère fiancée qui m'a aidé avec indéfectible patience et a sacrifié son temps pour que je puisse finaliser ma thèse.

Finalement, je tiens à remercier du fond du cœur ma famille, mes belles-sœurs, la famille MARRAKCHI, mes beaux-parents pour leurs compréhension et encouragement.

Merci à tous ceux qui ont participé de près ou de loin à l'aboutissement de ce travail.

Abdelaali BRIACHE.

DEDICACE

A ceux qui ont attendu avec patience les fruits de leur bonne éducation...

A ceux qui sont la raison pour laquelle je me lève tous les jours de ma vie…

A mes chers parents.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLES

LISTE DES ABREVIATIONS

ADN	Acide DésoxyriboNucléique
APO	Ascomycete Phenotype Ontology
ARN	Acide RiboNucléique
ASN 1.0	Abstract Syntax Notation One
BioGRID	Biological General Repository for Interaction Datasets
CHEBI	Chemical entities of biological interest
ChIP	Chromatin Immunoprecipitation
CYGD	Comprehensive Yeast Genome Database
EBI	European Bioinformatics Institute
EC	Enzyme Classification
FMA	Foundational Model of Anatomy
FTP	File Transfer Protocol
FunCat	Functional Catalogue
GEO	Gene Expression Omnibus
GO	Gene Ontology
HGP	Human Genome Project
HMM	Hidden Markov Model
HTML	Hypertext Martkup Language
ICD	International Classification of Diseases
IGD-GIS	Integrated Genomic Database - Genome Information System
KIF	Knowledge Interchange Format
MGD	Mouse Genome Database
MIPS	Munich Information Center for Protein Sequences
NCBI	National Center for Biotechnology Information
OBO	Open Biomedical Ontologies
ORF	Open Reading Frame
P2P	Peer-to-Peer
PDMS	Peer Data Management System
PPI	Protein-Protein Interaction
PSI-MI	Proteomics Standard Initiative - Molecular Interactions
SB-KOM	System Biology Khaos Ontology-based Mediator
SGBD	Système de Géstion de Bases de Données
SGD	Saccharomyces Genome Database

SMA	Systeme Multi-Agent
SPELL	Serial Pattern of Expression Levels
UMLS	Unified Medical Language Système
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
W3C	World Wide Web Consortium
WTSI	The Wellcome Trust Sanger Institute
YEASTRACT	YEAst Search for Transcriptional Tegulators And Consensus Tracking
YTPDB	Yeast Transport Protein DB

NOTE AU LECTEUR

Dans la suite du document, les termes marqués par * seront définis dans le glossaire.

Introduction Général

La biologie moléculaire est une discipline qui génère et manipule une grande quantité de données relatives aux séquences nucléiques et protéiques des êtres vivants. La plupart de ces données sont habituellement stockées dans des banques et des bases de données disponibles à travers le Web. Le nombre de ces sources ainsi que leur contenu croissent de façon considérable. Ces sources de données sont à la fois réparties et très hétérogènes : chaque source a son propre format de données et sa propre structure, et il est fréquent que les termes scientifiques utilisés pour décrire des données différentes d'une source à l'autre.

Par ailleurs, l'étude des mécanismes du vivant aujourd'hui ne passe plus seulement par des analyses expérimentales biologiques classiques (dites *in vivo* ou *in vitro*) mais aussi par des analyses informatiques (qualifiées d'analyses *in silico*). Les langages informatiques ont offert la possibilité de développer des outils bioinformatiques permettant le traitement des données. Les outils d'analyses bioinformatiques disponibles sont eux aussi de plus en plus nombreux et leur utilisation permet de produire de nouvelles informations qui sont à leur tour stockées dans des sources de données.

Les sources de données biologiques contiennent néanmoins une très grande richesse d'informations. Les biologistes sont amenés fréquemment à les interroger dans le but de chercher des données qui leur permettent d'analyser et de comparer les résultats de leurs expérimentations. Exploiter ce volume et cette diversité d'informations, réparties, complémentaires et fortement hétérogènes, est un réel défi à relever. Ainsi, la diversité et l'hétérogénéité que présentent ces sources de données sont les difficultés principales rencontrées par les biologistes. Cette hétérogénéité provient essentiellement des aspects suivants :

- différents modèles et schémas de données (modèle relationnel, orienté objet, semi-structuré…etc.) ont été utilisé pour décrire les données ;
- différents concepts de modèle peuvent être utilisés pour décrire le même objet même si le même modèle est utilisé ;
- une grande variété de formats ont été utilisé pour représenter les données (ASN,1, XML, HTML). Pour que le biologiste puisse utiliser ou interpréter les données, il est nécessaire qu'il soit au courant de ces formats ;

- les sources de données rendent leurs données accessibles par différentes manière : Alors que certaines sources de données sont accessibles par le biais des formulaires de recherche Web, d'autres sont disponibles sous forme de fichiers plats via le mécanisme FTP, et quelques autres fournissent un ensemble d'APIs pour permettre un accès par programmation à leurs données ;
- aussi, le langage de requête utilisé par une banque de données (comme GenBank) est souvent une simple combinaison de mots à rechercher dans les textes, tandis que les bases de données relationnelles, par exemple, peuvent être interrogées en SQL. Certains projets orientée-objet d'entrepôts de données (Gedaw par exemple) offrent la possibilité de soumettre des requêtes OQL sur leurs schémas ;
- une autre forme d'hétérogénéité est l'utilisation non uniforme de la nomenclature. Plusieurs termes différents sont souvent utilisés pour désigner un même concept dans une même source, mais aussi à travers différentes sources et espèces.

Malgré toutes ces hétérogénéités, les biologistes, comme mentionné avant, sont amenés à consulter systématiquement un ensemble de ces sources afin d'analyser les résultats de leurs expériences. Ils doivent effectuer au moins les tâches suivantes lors de la recherche d'informations:

- identifier les sources convenables et leurs emplacements ;
- identifier le focus de chaque source. Les sources de données biologiques se caractérisent par un focus bien définis. Alors que certaines se focalisent sur les gènes, d'autres peuvent se focaliser sur les protéines et d'autres sur une espèce donnée ;
- interroger chaque source appropriée de façon indépendante en utilisant sa méthode d'accès et son langage de requête spécifiques ;
- naviguer à travers les sources pour obtenir des données complémentaires ;
- puis fusionner manuellement les données obtenues à partir de différentes sources.

Ceci place une charge sur les biologistes, dont la plupart ne sont pas des experts en bioinformatique, et limite l'utilisation qui peut être faite sur les données disponibles. Par conséquent, il rend nécessaire le développement des systèmes d'intégration de données qui peuvent apporter des solutions aux biologistes dans leurs processus d'interrogation des sources.

Le domaine de l'intégration d'informations vise à développer des solutions offrant un accès uniforme à des sources de données multiples, réparties et hétérogènes. Ces solutions permettent à un utilisateur d'interroger plusieurs sources de données avec un langage de requêtes unique, sans qu'il ait à se soucier de la structure interne de chacune des sources.

Le problème d'intégrer des sources de données hétérogènes et de les interroger via une seule interface de requêtes ne date pas d'aujourd'hui. Depuis l'arrivée et l'adoption des bases de données dans les années soixante, leur partage et leur combinaison sont naturellement devenus indispensables. Cette combinaison peut être effectuée de

différentes façons et à différents niveaux de l'architecture du système d'intégration de données. Deux approches principales pour la conception de ces systèmes ont été définies en se fondant sur la localisation des données gérées par le système :

- l'intégration virtuelle de données, où la vue unifiée est virtuelle et les données restent stockées dans les sources d'origine. L'architecture type pour l'intégration virtuelle de données est l'architecture médiateur ;
- l'intégration matérialisée de données, où la vue unifiée des données est matérialisée et les données sont rapatriées des sources d'origine et stockées dans un entrepôt de données.

Cadre et but de travail

Saccharomyces cerevisiae est l'un des principaux modèles cellulaires eucaryotiques dans la recherche fondamentale en biologie du fait de la performance des outils génétiques et de biologie moléculaire et cellulaire qui y ont été développés. Elle est considérée comme un outil biotechnologique pour la production de protéines d'intérêt commercial. Après l'achèvement du séquençage total de son génome en 1996, les chercheurs y ont repéré très vite des gènes proches à ceux qui, chez l'homme, provoquent des maladies génétiques. Il restait à comprendre le rôle des protéines correspondantes et c'est là que la levure a révélé ses qualités d'outil en biologie moléculaire.

L'importance de la levure dans les recherches biologiques a généré une grande quantité d'information. Ceci a donné lieu à plusieurs sources et bases de données. Il s'agit de sources de données publiques ou privées qui se sont constituées autour d'une thématique biologique spécifique de *Saccharomyces cerevisiae*, afin de satisfaire des besoins particuliers. Elles sont qualifiées de sources de données spécialisées par opposition aux sources de données généralistes. Ces sources présentent aussi les caractéristiques précédemment et qui rendent leur interrogation complexe et fastidieuse, et par conséquent leur intégration une nécessité pour faciliter leur exploitation.

Au cours de mon travail de thèse, mon objectif était de fournir une solution d'intégration de données tenant compte des défis mentionnés ci-dessus et adaptée au contexte d'intégration des données biologiques de *Saccharomyces cerevisiae*. L'enjeu était double :

- Intégrer des informations biologiques à partir des sources hétérogènes et réconcilier ces données afin d'avoir une vue unifiée des informations disponibles sur un gène donné.
- Fournir à l'utilisateur la possibilité de choisir les sources de données à intégrer et de garder la traçabilité des informations collectées.

La première contribution de notre travail se place dans le contexte d'une approche médiateur. Nous avons développé un système médiateur nommé *YeastMed* qui permet aux utilisateurs, intéressés à *Saccharomyces cerevisiae*, d'accéder via une interface unique aux sources de données diverses, hétérogènes et réparties sur le Web. Ce système d'intégration est le fruit d'une collaboration scientifique entre le groupe de recherche LABIPHABE de la

faculté des sciences et techniques de Tanger, et le groupe KHAOS de l'institut supérieure d'ingénierie de Malaga en Espagne.

La deuxième défit vise à résoudre le problème de choix de sources. Lorsque l'utilisateur de *YeastMed* est face à des informations divergentes, il est le plus souvent amené à privilégier les données issues des sources en lesquelles il a le plus de confiance. Le biologiste a, en effet, des préférences sur le type de sources qu'il interroge. Pour un biologiste, connaître l'origine des données récupérées est donc crucial.

Structuration du document

Le mémoire et organisé comme suit :

- **Le chapitre 1** présente et met en évidence les caractéristiques des sources de données biologiques. Il dresse aussi un état de l'art des solutions d'intégration majoritairement suivies dans le domaine bioinformatique.
- **Le chapitre 2** introduit le Web sémantique et ses technologies qui sont impliquées dans le domaine de l'intégration de données. Dans ce chapitre, nous nous focaliserons sur les notions d'ontologie et de service Web car sont la base du système *YeastMed* qui sera détaillé dans le chapitre 4.
- **Le chapitre 3** donne une vue détaillée sur l'architecture de l'infrastructure de médiation à base d'ontologie sur laquelle est fondé le système *YeastMed*. Cette infrastructure a pour objectif la fourniture d'un accès aux données en utilisant un modèle de données et un langage de requête communs.
- **Le chapitre 4** présente le système *YeastMed*. Mais avant tout, il dresse une vue générale sur la levure *Saccharomyces cerevisiae* dont les données constituent le focus du *YeastMed*. Le chapitre présente le cycle de vie et les différentes caractéristiques de la levure. Il présente aussi les différentes sources de données intégrées par *YeastMed*, en décrivant pour chaque source son contenu et ses méthodes d'accès. Le chapitre détaille aussi les différentes étapes du développement du système à travers une description détaillées de ses composants.

Puis, nous concluons le travail en ouvrant des perspectives sur nos travaux futurs.

Première Partie

ETAT DE L'ART

Chapitre 1

Intégration des Sources des données biologiques

Sommaire

1.1. Introduction

La biologie moléculaire est une discipline riche en données. Elle manipule un grand nombre de séquences nucléiques et protéiques ainsi que les données qui en sont associées. La plupart de ces données sont actuellement stockées dans des banques et des bases de données accessibles via le Web. Le biologiste, afin d'analyser les résultats de ses expériences, est systématiquement amené à consulter un ensemble de ces sources. Celles-ci lui apportent des informations complémentaires qui pourront lui permettre d'énoncer des hypothèses biologiques et d'avancer dans ses recherches. Néanmoins, du fait de la très grande diversité des sources et de leur hétérogénéité, les interroger indépendamment et combiner manuellement les résultats retournés de chacune d'elles afin d'exploiter la richesse des informations qu'elles contiennent est une tâche complexe. Face à cette diversité, de nombreuses solutions visant à offrir un accès uniforme à des sources multiples, distantes et hétérogènes ont été mises en ouvres. On parle alors des solutions d'intégration de données. Ces solutions permettent à un utilisateur d'interroger plusieurs sources via une interface et un langage de requêtes unique, sans qu'il ait à se soucier de la structure interne de chacune des sources.

Dans ce chapitre, nous allons parler de l'intégration des données dans le domaine de la biologie. Nous allons commencer, dans la section 1.2, par énumérer les caractéristiques des sources de données biologiques qui rendent leur intégration une exigence. Puis dans la section 1.3, nous allons introduire la notion de l'intégration de données avant de donner, dans la section 1.4, une classification des systèmes d'intégration de données développés dans le domaine biologique. La section 1.5 sera consacrée au processus de traitement des requêtes dans un système de médiation.

1.2. Caractéristiques des sources de données biologiques

Les sources de données biologiques présentent des caractéristiques spécifiques qui rendent leur interrogation une tâche complexe et fastidieuse et par conséquent, leur intégration une nécessité pour faciliter leur exploitation et tirer plus de bénéfices. Après un bref historique, nous allons énumérer, dans cette section, les propriétés des sources biologiques les plus connues.

1.2.1. Bref historique

Vers le début des années 80, les premières banques de séquences ont vu le jour sous l'initiative de quelques équipes. Très rapidement avec les évolutions des techniques du séquençage, la collecte et la gestion des données ont nécessité une organisation plus conséquente. Ainsi, plusieurs organismes ont pris en charge la production de telles bases de données. En Europe, financée par l'*EMBO*[1] (*European Moleculary Biology Organisation),* une équipe s'est constituée pour développer une banque de séquences nucléiques appelée *EMBL data library*[2] et en assurer la diffusion (Bertails, et al., 2010). Du côté américain, une

[1] www.embo.org/

[2] http://www.embl.org/

banque nucléique nommée *GenBank*[3] et soutenue par le *NIH*[4] (*National Institute of Health*)a été créée (Berners-Lee, et al., 2001). Cette source de données était à l'origine une base de données relationnelle puis fut diffusée sous la forme de fichiers plats jusqu'à maintenant par le *NCBI*[5] (*National Center for Biotechnology Information*). La collaboration entre ces deux banques a commencé relativement tôt. Elle s'est étendue en 1987 avec la participation de la *DDBJ*[6] (*Dna Data Bank of Japan*) du Japon pour donner naissance finalement en 1990 à un format unique dans la description des caractéristiques biologiques qui accompagnent les séquences dans les banques de données nucléiques[7].

Parallèlement, pour les protéines, deux banques principales ont été créées. La première se nomme *PIR*[8] (*The Protein Information Resource*) (Berners-Lee and Cailliau, 1990). La deuxième, *SwissProt* (Boeckmann, et al., 2003), qui fait maintenant partie de *UniProt*[9] (Apweiler, et al., 2004), a été constituée à l'Université de Genève à partir de 1986 et regroupe entre autres des séquences annotées de *PIR* ainsi que des séquences codantes traduites de *EMBL*. D'autres banques de séquences protéiques sont apparues ensuite comme *GenPept*[10] (actuellement *NCBI-Protein*) et *TrEMBL* (actuellement UniProtKB/TrEMBL) qui sont issues respectivement de parties codantes identifiées dans la base *GenBank* ou *EMBL*.

1.2.2. Des sources multiples

Il existe actuellement un grand nombre de sources de données biologiques, que ce soit d'intérêt biochimique, génétique, pharmaceutique ou génomique pour ne citer que les plus nombreuses. En effet, ces dernières années, les sources de données biologiques disponibles sur le Web se sont multipliées. La 19ème *annual Database Issue* de la revue NAR[11] (*Nucleic Acids Research)*, qui liste chaque année les sources les plus importantes du Web, recense plus de 1380 sources publiques en 2012 (Galperin and Fernández-Suárez, 2012) avec 92 bases de données nouvelles par rapport à la précédente issue (Galperin and Cochrane, 2011). Ces sources étaient environ 968 en 2007 (Galperin, 2007). En l'espace de 5 ans plus de 410 sources de données publiques ont donc vu le jour.

Cohen-Boulakia dans sa thèse (Cohen-Boulakia, 2005) énumère trois éléments d'explication à ce phénomène de multiplication des sources de données biologiques :

[3] http://www.ncbi.nlm.nih.gov/genbank/

[4] http://www.nih.gov/

[5] http://www.ncbi.nlm.nih.gov/

[6] http://ddbj.sakura.ne.jp/

[7] ftp://ftp.ebi.ac.uk/pub/databases/embl/doc/FTv6_6.html

[8] http://pir.georgetown.edu/pirwww/

[9] http://www.uniprot.org/

[10] http://www.ncbi.nlm.nih.gov/protein

[11] http://nar.oxfordjournals.org/

1. D'abord, les projets de séquençage qui se sont extrêmement développés durant ces quinze dernières années. Chacun de ces projets a pour but de séquencer un génome ; il conçoit et développe alors sa propre source de données pour mettre ses résultats à la disposition de tous. Citons le HGP[12] (*Human Genome Project*) (Cantor, 1990) débuté en 1990 et le MGD[13] (*Mouse Genome Database*) (Eppig, et al., 2012) quelques années plus tard comme exemples de projets d'annotation ayant mis en ligne leurs résultats.

2. En parallèle, de nouvelles techniques d'analyse biologique à haut débit ont vu le jour, comme les puces à ADN aujourd'hui incontournables et plus récemment les puces à protéines ou les puces à CGH*. Ces nouvelles techniques ont généré de nouveaux types de données, qui ont été stockés dans de nouvelles sources. Ainsi, les sources GEO[14] (*Gene Expression Omnibus*) (Barrett, et al., 2011) et ArrayExpress[15] (Brazma, et al., 2003) ont été créées pour contenir des données de puces à ADN (ou microarray).

3. La troisième cause est le développement d'outils bioinformatiques. Les données sont aujourd'hui régulièrement analysées et comparées à l'aide d'outils de recherche de similarités de séquence (Blast*), d'alignements multiples, ou encore de détection de gènes dans les séquences... Les résultats obtenus par ces outils sont eux aussi stockés dans de nouvelles sources de données. Par exemple, la source Pfam[16] (Bateman, et al., 2000) contient des données-résultats d'alignements multiples (algorithmes basés sur des chaînes de Marchov*).

1.2.3. Des données variées

Une des particularités du domaine biologique est son étendue. Ce sujet très vaste est morcelé en de nombreuses sous-disciplines (comme la biochimie, la cytologie, la génétique des populations) en fonction du niveau auquel se situent les observations (respectivement moléculaire, microscopique, et populationnel). Ce découpage a conduit à une grande diversité des données stockées. Ainsi, Les sources biologiques publiques contiennent toutes des informations relatives à des entités biologiques (gènes, protéines, locus*, ...). Ces informations peuvent revêtir différentes formes comme des images (pour les structures 3D, dans wwPDB[17] (Berman, et al., 2007) par exemple) ou des schémas (pour les réseaux de régulation, dans KEGG (Okuda, et al., 2008), et pour les cartes génomiques, dans MapView[18], par exemple). Néanmoins, la grande majorité des informations contenues dans

[12] http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml

[13] http://www.informatics.jax.org/

[14] http://www.ncbi.nlm.nih.gov/geo/

[15] http://www.ebi.ac.uk/arrayexpress/

[16] http://www.sanger.ac.uk/resources/databases/pfam.html

[17] http://www.wwpdb.org/index.html

[18] http://www.ncbi.nlm.nih.gov/mapview/

les sources est sous la forme de texte (séquences ADN, descriptions des structures tridimensionnelles des protéines, informations phénotypiques, publications médicales...).

La variété des contenus a une influence directe sur la variété des contenants (sources de données), comme nous allons voir dans les sections 1.2.4 et 1.2.6. L'information à stocker a souvent dicté les choix de conception ou de solutions logicielles : l'image de la microarray tirée de la base de données SMD[19] (*Stanford Microarray Database*) (Sherlock, et al., 2001) et représentée dans la Figure 1., est en pratique plus facilement enregistrable et manipulable en tant que type binaire dans un serveur Oracle (Ault, et al., 2003) qu'en tant que lien XLink (Wilde and Lowe, 2002) dans un fichier semi-structuré. A l'inverse, un réseau de régulation génétique tel que celui de la Figure 2, aux interactions nombreuses et souvent bidirectionnelles, peut être aussi bien représenté suivant le modèle relationnel qu'à l'aide d'un schéma XML. Les conflits qui surviennent entre les données résultent des différences de perception, de modélisation et d'interprétation des entités du monde réel par les concepteurs des systèmes d'information.

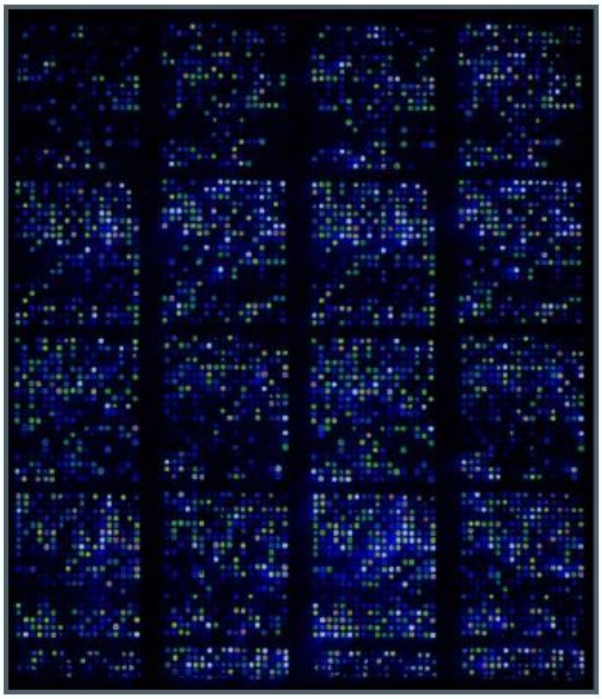

Figure 1. Microarry.

[19] http://smd.stanford.edu/

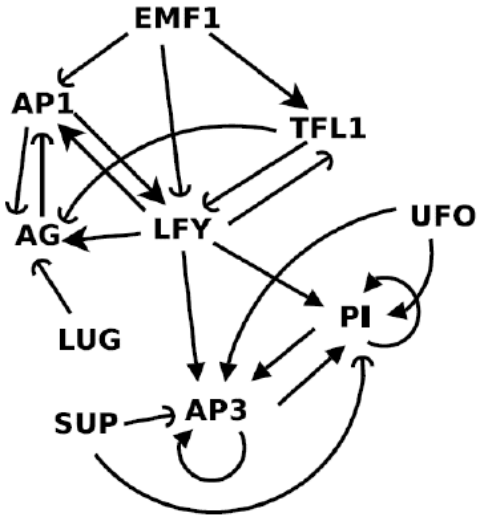

Figure 2. Réseau de régulation génétique. (Gambin, et al., 2006),

1.2.4. Des sources variées

Le grand nombre des sources de données biologiques disponibles actuellement se caractérise aussi par sa diversité que ce soit au niveau du type des sources elles-mêmes, ce qui est en partie la conséquence de la variété des données que nous venant de citer. Ainsi, nous inspirons du (Cohen-Boulakia, 2005) et du (Galperin and Fernández-Suárez, 2012) pour citer quelques types de sources de données :

- ***Les sources regroupant un ensemble d'abstracts*** de publications scientifiques du domaine médical tel que Medline[20], PubMed[21].
- ***Les dépôts de données (ou repository)***. Ils Sont aussi appelés "sources de données primaires" (Goble, 2002). Dans cette catégorie entrent, à titre d'exemple, Les sources GenBank (USA), EMBL (Europe), et DDBJ (Japon) qui sont des dépôts de séquences et ArrayExpress (Europe) et GEO (USA) qui sont des dépôts des données de puces à ADN. Elles ont comme objectif contenir de façon exhaustive l'ensemble des données disponibles (sur les séquences ou les données de puce à ADN). Les données qui y sont présentes sont brutes au sens où elles ne sont pas validées par les propriétaires des sources. L'importance de ces sources se manifeste, entre autres, dans le fait que toute publication scientifique soumise à une revue en biologie au sujet d'un séquençage (ou d'une expérience de puce à ADN) doit être obligatoirement associée à un ou plusieurs numéros d'identification GenBank/EMBL/DDBJ (respectivement GEO/ArrayExpress).
- ***Les sources de données secondaires***. Ces sources se distinguent des précédentes principalement par le fait qu'elles contiennent des informations nettoyées et parfois même validées manuellement par des experts. Leurs objectif est de partir

[20] http://www.proquest.com/en-US/catalogs/databases/detail/medline_ft.shtml

[21] http://www.ncbi.nlm.nih.gov/pubmed

de données issues des sources primaires pour proposer des informations plus synthétiques et le cas échéant, ajouter des informations complémentaires.

Les sources RefSeq[22] (Pruitt, et al., 2012) et UniGene[23] (Pontius, et al., 2003) du NCBI sont deux exemples de sources secondaires des données génomiques, qui proposent de regrouper les fiches GenBank. La première utilise des techniques de regroupement semi-automatique et propose une version non redondante de GenBank, alors que la seconde construit de façon automatique des clusters de séquences.

Pour les données protéomiques, SwissProt (Boeckmann, et al., 2003) est un exemple de source secondaire. Cette source fournit des informations sur les protéines, elle est construite à partir des données de TrEMBL (pour "*Translations of EMBL nucleotide sequences*") qui contient la traduction (les séquences d'acides aminés) des données génomiques (sous forme de séquences ADN) soumises à EMBL. SwissProt contient des données validées, nettoyées et corrélées avec des informations de la littérature et d'autres données expérimentales publiées. Notons que les données de UniProt, TrEMBL et PIR ont le même format et sont aujourd'hui regroupées dans la source UniProt, bien que la distinction des trois sources d'origine soit maintenue.

- ***Les sources de données d'expertises***. Ces sources contiennent essentiellement du texte et proposent des fiches contenant une analyse et une synthèse d'un ensemble d'articles scientifiques. Par exemple, la source OMIM[24] (Hamosh, et al., 2002) fournit un ensemble d'informations sur les maladies humaines sous la forme de fiches dans lesquelles des experts (de l'université Johns Hopkins aux USA) commentent les résultats associés à un gène ou un groupe de gènes décrits dans un ensemble de publications, et associés à un phénotype (une maladie) donné.

- ***Les sources de données-résultats d'outils***. On retrouve beaucoup de ces sources au niveau du recensement des domaines fonctionnels : Pfam[25] (Bateman, et al., 2000) et ProDom[26] (Corpet, et al., 1998). Ces sources ont des contenus générés automatiquement qui résultent de l'utilisation d'une succession précise d'outils bioinformatiques. Elles sont ensuite validées ou non par des experts. Ces sources sont aussi caractérisées par le fait qu'elles offrent des outils de visualisation des résultats, qui permettent de comparer et d'analyser les informations ainsi générées.

[22] http://www.ncbi.nlm.nih.gov/RefSeq/

[23] http://www.ncbi.nlm.nih.gov/unigene

[24] http://www.ncbi.nlm.nih.gov/omim

[25] http://pfam.sanger.ac.uk/

[26] http://prodom.prabi.fr/prodom/current/html/home.php

- ***Les sources qui s'offrent à regrouper et organiser une famille de données bien précise***. Par exemple, la source Enzyme[27] (Bairoch, 2000) est dédiée à la description des protéines dont la fonction est enzymatique, la source FlyBase[28] (Consortium, 2002) est dédiée à la drosophile alors que la source SGD (Saccharomyces Genome Database)[29] (Cherry, et al., 2012) est dédiée à la levure.
- **L*es sources synthétiques qui proposent un ensemble de fiches de synthèse***. Chacune de ces fiches regroupe des informations présentes dans d'autres sources associées à un même gène ou une même protéine. On trouve dans cette catégorie GeneCards[30] (Rebhan, et al., 1997), qui fournit des fiches de synthèse reprenant et proposant des liens hypertextes vers des informations relatives aux gènes humains, qui proviennent d'une vingtaine de sources de données (dont SwissProt, GenBank...). On trouve aussi la source Entrez Gene[31] qui donne accès à partir d'un nom de locus donné, à l'ensemble des informations présentes dans la trentaine de sources du NCBI.

1.2.5. Des sources autonomes mais reliées

La majorité des sources biologiques disponibles sur internet fonctionnent en mode totalement autonome. Autrement dit, les administrateurs et curateurs de ces sources sont tout à fait libres de modifier leur schéma ou de mettre à jour leur contenu (ces sources fonctionnent souvent sur le principe de mises à jour régulières) sans en faire état préalablement aux utilisateurs. Néanmoins, un effort très important est fait pour relier systématiquement les données entre sources par des liens de références croisées. C'est l'une des particularités majeures des sources biologiques. Ainsi, la plupart des sources font référence à des identifiants communs sur lesquels il est possible de s'appuyer afin de rassembler les données. Suivre une référence croisée permet à partir d'une entrée dans une source d'obtenir une autre entrée (dans une autre source ou dans la même) fournissant des informations complémentaires ou plus précises. Ces références croisées permettent de regrouper des données proches : soit parce qu'il s'agit de la même instance décrite différemment à travers des sources, soit parce qu'on accède à une instance d'une entité qui lui est associée (par exemple, on part d'une protéine et on accède à la séquence de son gène).

Les références croisées permettent en quelques sorte de répondre aux attentes des biologistes en ce qui concerne le rassemblement des données et de leur appliquer des méthodes de prédiction dont le but final est de détecter de nouvelles corrélations, de

[27] http://enzyme.expasy.org/

[28] http://flybase.org/

[29] www.yeastgenome.org/

[30] http://www.genecards.org/

[31] http://www.ncbi.nlm.nih.gov/gene/

mettre en évidence des relations biologiques encore inconnues, et d'ouvrir de nouvelles pistes de recherche.

1.2.6. De l'hétérogénéité

Les sources de données biologiques sont conçues pour fonctionner de façon autonome. Dans la plupart des cas, les sources ont été développées indépendamment et sont par conséquent hétérogènes. Cette hétérogénéité a plusieurs aspects. Elle peut provenir du format ou de la structure des sources (sources structurées : bases de données relationnelles, sources semi-structurées : documents XML, ou non structurées : textes), du mode d'accès et de requête ou de l'hétérogénéité sémantique : entre les schémas conceptuels ou ontologies implicites ou explicites sous-jacentes. La littérature a détaillées à de nombreuses reprises cette hétérogénéité (Davidson, et al., 1995; Froidevaux and Cohen-Boulakia, 2002; Hernandez and Kambhampati, 2004; Olken and Jagadish., 2003). En ce qui suit, nous allons reprendre les principales facettes d'hétérogénéité rencontrées dans les bases de données biologiques.

1.2.6.1. Hétérogénéité syntaxique

L'hétérogénéité syntaxique est causée par les différences entre plateformes logicielles, et les formats qu'elles manipulent. Ainsi, chaque sources de données adopte un modèle ou un schéma de données propre qui diffère d'une source à une autre : on trouve le modèle relationnel (ENSEMBL (Flicek, et al., 2012), SGD (Cherry, et al., 2012)), le modèle objet (rencontré très souvent dans le cas d'entrepôts comme Gedaw (Guérin, et al., 2005)), des modèles semi-structurés XML (dans UniProt (Apweiler, et al., 2004)).

Les formats de représentation des données manifestent aussi de l'hétérogénéité syntaxique par le fait qu'on représente souvent le contenu des sources à l'aide de formats variés. L'utilisation de fichiers plats est encore aujourd'hui le standard *de facto*, ce qui nécessite une phase d'extraction de données afin de retrouver la structure des données originelles. Certaines sources fait appel à des notations formelles telles que ASN 1.0 (Abstract Syntax Notation One)[32], Fasta[33], ou plus récemment XML. En effet, le développement du langage XML et des technologies qui y sont liées (notamment autour du langage Java avec par exemple les API* JAXP (Griffth, 2005), JAXB (McLaughlin, 2002))permet de plus en plus de simplifier les échanges de données biologiques (Achard, et al., 2001). D'autres, comme HTML, n'offrent pas de structure.

Il résulte de cette hétérogénéité syntaxique que les sources de données rendent leurs données accessibles par différentes manières. Tandis que certaines sources de données sont accessibles à travers des formulaires d'interrogation, d'autres sont disponibles sous forme de fichiers plats ou à travers des mécanismes FTP. D'autres fournissent un ensemble d'API (Application Programming Interface) pour permettre un accès programmé aux données (KEGG).

[32] http://www.itu.int/ITU-T/asn1/ecn/

[33] http://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml

Les langages d'interrogation des données diffèrent aussi d'une source à une autre. Le langage d'interrogation d'une banque de données (comme Genbank) est souvent une simple combinaison de mots à chercher dans les textes, tandis que les bases de données relationnelles par exemple, peuvent être interrogées en SQL (ENSEMBL). Certains projets d'entrepôt orienté objet (Gedaw) offrent la possibilité de poser des requêtes OQL sur leur schéma.

1.2.6.2. Hétérogénéité sémantique

La résolution de l'hétérogénéité sémantique est une des questions fondamentales à résoudre quand on veut réaliser l'interopérabilité entre plusieurs sources (Sheth and Kashyap, 1992). Bishr dans (Bishr, 1998) a considéré deux niveaux où se manifeste l'hétérogénéité sémantique :

- ***Niveau cognitif*** : Elle révèle les visions conceptuelles différentes des communautés de recherche à propos d'entités du monde réel à modéliser. Ainsi, selon les sources, une même entité biologique est définie différemment. Un des exemples les plus marquants concerne l'entité *gène*. Steffen Schulze-Kremer (Schulze-kremer, 1998) en compare deux définitions : dans la base de données GDB (Letovsky, et al., 1998), "*un gène* est un *brin d'ADN qui peut être transcrit et traduit en protéine*" ; dans GenBank (Benson, et al., 2012) et GSDB (Harger, et al., 2000), "*un gène* est un *fragment d'ADN d'intérêt biologique, portant un nom précis et qui est vecteur d'un trait génétique ou phénotypique particulier*". Cette dernière définition prend donc en compte des morceaux d'ADN qui n'interviennent pas dans la constitution d'une protéine, et il en ressort une contradiction avec la première définition.
- ***Niveau de la nomenclature*** : plusieurs aspects à citer dans ce cas. Nous nous contentons des plus connus pour illuminer l'hétérogénéité à ce niveau.
    - Différents vocabulaires sont utilisés pour annoter les séquences. Certains annotateurs emploient, par exemple, le terme de "*putative*" pour exprimer que l'annotation n'est pas sûre, tandis que d'autres utilisent le terme "*hypothetical*". D'autres encore ne précisent rien.
    - La synonymie et l'homonymie est aussi une cause de conflit sémantique dans les sources de données biologiques. elles engendrent une incompréhension entre sources ayant choisi des taxonomies distinctes. Il existe très souvent plusieurs noms (synonymes) pour un même gène ou pour une même protéine, et ce, à l'intérieur d'une même source mais aussi à travers les sources et les espèces. Il est donc courant qu'un gène ou une protéine ait plusieurs noms. De même, il est possible que deux protéines ou deux gènes différents aient le même nom ou un nom en commun : on est dans ce cas en présence d'homonymie.

- Des entités équivalentes identifiées par ou contenant des valeurs[34] différentes : la Table 1 en montre un exemple, où un même objet peut se trouver associe à des identifiants distincts. A l'inverse, un seul et unique identifiant peut désigner plusieurs entités différentes. Cette situation peut également se produire au sein d'une seule et unique source, introduisant des contradictions qui impactent la qualité de la source et la confiance qui lui est accordée.

Database	Numéro d'accession
dans SwissProt	P29460
EMBL	M65272, M65290 AY008847, AF180563 AF512686
GeneW	HGNC : 5970, IL12B
GeneCard	IL12B
Prosite	PS50853, PS01354 PS50835

Table 1. Références croisées de sources différentes concernant le gène IL12B.

1.2.6.3. Hétérogénéité qualitatif

La différence de qualité entre les sources est une notion subjective, puisque liée à l'intérêt que portent les utilisateurs aux données. Ainsi, le crédit accordé à une source n'est pas le même. Ceci est dû d'une part au fait que chaque source se focalise sur un type d'objet ou une entité biologique. Par exemple, dans GenBank, la séquence nucléotidique est l'entité centrale dans ses entrées. La protéine ne constitue qu'un attribut additionnel. Dans UniProt, les données sont focalisées sur la protéine, qui est l'entité centrale alors que son gène est vu comme un simple attribut. Le focus peut porter aussi sur l'espèce (SGD pour la levure), le domaine fonctionnel des protéines (InterPro), la structure tridimensionnelle d'une protéine (PDB)…

La curation et l'annotation des données est un autre facteur qui influe la qualité d'une source de données suivant que l'information qu'elle contient a été curée manuellement ou de façon automatique[35]. Ceci a un impact direct sur la confiance du biologiste. Ce dérnier

[34] Des tentatives d'uniformisation des identifiants utilisés par les bases de données bioinformatiques voient peu à peu le jour ; le Consensus CDS (Pruitt, K.D*., et al.* (2009) The consensus coding sequence (CCDS) project: Identifying a common protein-coding gene set for the human and mouse genomes, *Genome Research*, **19**, 1316-1323.) s'est constitué autour de l'EBI, du NCBI , du WTSI (http://www.sanger.ac.uk/), et de la base UCSC (Karolchik, D*., et al.* (2003) The UCSC Genome Browser Database, *Nucleic Acids Research*, **31**, 51-54.) afin d'identifier un ensemble de régions codant des protéines dans le génome humain, dont l'annotation serait de grande qualité. Les gènes intégrés à l'ensemble CCDS se voient attribuer un identifiant unique, qui est d´désormais incorporé sur plusieurs bases de données biologiques telles qu'Entrez Gene et Ensembl.

[35] Ensembl est annotée automatiquement Curwen, V*., et al.* (2004) The Ensembl Automatic Gene Annotation System, *Genome Research*, **14**, 942-950. et SwissProt possède une équipe de curateurs O'Donovan, C*., et al.* (2002) High-quality protein knowledge resource : SWISS-PROT and TrEMBL, *Briefings in Bioinformatics*, **3**, 275–284.

privilégie les informations issues de la source en laquelle il a le plus de confiance (notons que cette confiance est variable, puisqu'elle peut dépendre du domaine de recherche, voire de l'expérience qu'a un biologiste de l'utilisation de la source) (Cohen-Boulakia, 2005). Les sources adoptant une curation manuelle ont souvent plus de crédibilité que celles ayant une curation automatique même si la curation manuelle à elle seule n'est pas suffisante (Baumgartner, et al., 2007). La curation manuelle apporte plus de précision et une qualité élevée aux données par rapport à la curation automatique (Davis, et al., 2011).

De même, le niveau de détail varie d'une source à l'autre. Une même donnée n'est pas représentée avec le même niveau de granularité, de détail. Par exemple, UniProt propose des informations sur des protéines issues de différentes espèces. Elles sont précises mais "généralistes" au sens où elles ne sont pas ciblées sur une famille particulière de données. Au contraire, chez SGD on pourra connaître de façon spécifique la fonction de chacune des protéines de la levure.

1.3. Intégration de données : Définition et Objectifs

L'intégration de données est un domaine de l'informatique qui s'occupe de l'uniformisation de l'accès à une grande quantité de données variées et dispersées. Elle regroupe les processus par lesquels les données provenant de différentes sources d'information sont déplacées, combinées et consolidées. Ces processus consistent habituellement à extraire des données de différentes sources (bases de données, fichiers, applications, services Web, emails, etc.), à leur appliquer des transformations (jointures, lookups, déduplication, calculs, etc.), et à envoyer les données résultantes vers leurs sollicitants. Ils s'appuient fortement sur les technologies du Web Sémantique (ontologie, services Web, XML, RDF… etc.)(Voir chapitre 2).

La problématique de l'intégration de données peut être décrite comme suit (Figure 3) : Étant donnés un ensemble de sources d'information autonomes et hétérogènes (sources locales) qui utilisent des schémas possiblement différents, on veut pouvoir interroger les données de ces sources comme si elles constituaient une seule source, avec un schéma unique. Le processus d'intégration de données consiste à prendre en entrée cet ensemble de sources de données (schémas et populations), et à produire en sortie une description unifiée des schémas initiaux (le schéma intégré) et les règles de traduction ou de correspondance (mapping) qui vont permettre l'accès aux données existantes à partir du schéma intégré.

Le but de l'intégration consiste donc à offrir à l'utilisateur un accès uniforme et transparent aux données. Les systèmes d'intégration de données se chargent de la répartition des requêtes aux sources qui participeront à la construction des réponses. Le problème de combinaison des sources de données hétérogènes sous une seule interface de requête n'est pas récent : le développement rapide des bases de données dans les années soixante-dix a naturellement fait surgir le besoin de partager et d'intégrer les données déjà entreposées.

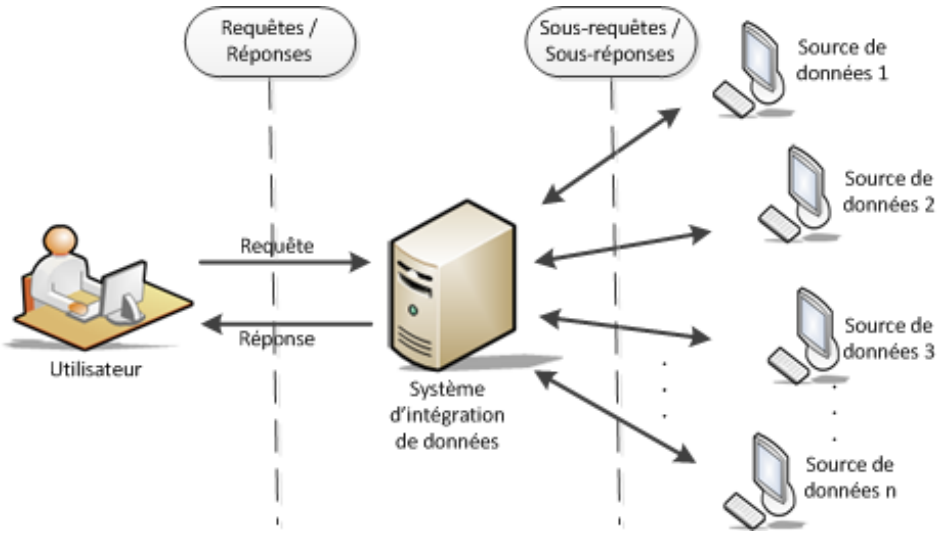

Figure 3. Principe d'intégration de données.

1.4. Classification des systèmes d'intégration de données biologiques

L'intégration de sources de données est un problème complexe. Un nombre considérable d'articles en ont étudié différents aspects : il en résulte une multitude de contributions techniques, de méthodologies et de prototypes. Dans cette section, nous allons discuter les approches proposées dans le domaine de l'intégration de données biologiques avec la citation de quelques systèmes adoptant ces architectures dans le domaine de l'intégration des données biologiques.

1.4.1. Entrepôt de données

Dans cette approche, les données en provenance de sources diverses sont intégrées en fournissant une vue unifiée des données qui sont stockées dans une seule structure appelée *entrepôt de données*. Cette approche est, depuis plusieurs années, très couramment utilisées dans le domaine bioinformatique, pour stocker des informations provenant de sources multiples et hétérogènes. Nous citons principalement GUS (Davidson, et al., 2001), Atlas (Shah, et al., 2005), BioSQL (Lapp, 2006), BioMart[36], BioWarehouse(Lee, et al., 2006), chado (Mungall and Emmert, 2007).

Trois étapes importantes sont impliquées dans la construction et la maintenance d'un entrepôt de données :

- ***modélisation et conception*** : dans l'étape de la conception, les développeurs ont besoin de déterminer les informations qui seront utilisées dans l'entrepôt de données pour chaque source, les vues (requêtes) sur ces sources qui seront matérialisées et le schéma global de l'entrepôt.

[36] http://www.biomart.org/index.html

- **Maintenance** : elle détermine la façon par laquelle l'entrepôt de données est initialement peuplé à partir des sources de données et comment est rafraîchit quand les données des sources sont mises à jour.
- **Opération** : l'opération d'un entrepôt de données implique le traitement des requêtes, et les problèmes de stockage et d'indexation.

Un exemple d'une architecture d'entrepôt de données est donné dans la Figure 4.

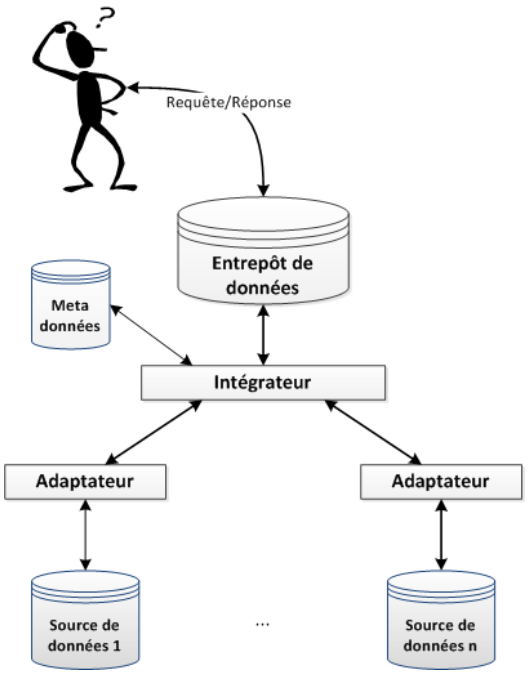

Figure 4. Architecture d'un entrepôt de données.

Deux approches de construction d'entrepôts : procédurale et déclarative (Calí, et al., 2003).

- ***L'approche procédurale*** intègre les données de façon *ad-hoc** sans chercher à construire un schéma intégrateur.
- ***L'approche déclarative*** est plus complexe et est majoritairement suivie. La structuration des données de l'entrepôt se fait grâce à son schéma global, ou schéma intégrateur. Le modèle dans lequel le schéma global est défini détermine le langage de requêtes utilisé pour interroger l'entrepôt.

Deux types d'intégration se dégagent selon le niveau d'intégration des données par le schéma intégrateur : syntaxique et sémantique.

- ***L'intégration syntaxique*** est assurée par les adaptateurs. Ils convertissent l'ensemble des données des sources définit selon un modèle (relationnel, objet, semi-structuré…) dans le modèle choisi pour l'entrepôt. C'est l'union des schémas des sources qui définit le schéma global de l'entrepôt à cette étape.

- ***L'intégration sémantique*** est fondée sur la construction d'un schéma global intégrateur (une DTD, un schéma XML, un schéma relationnel..), et vise à convertir les données des sources en termes de données dans ce schéma global intégrateur. Ce type d'intégration est nécessaire pour supprimer toute redondance au niveau du schéma de l'entrepôt.

Le fait que l'approche entrepôt permet un stockage local des données offre plusieurs avantages. Tout d'abord, elle permet un accès direct, rapide et sécurisé aux données. Elle permet aussi d'effectuer des analyses complexes, trop lourdes pour être lancées à distances. Ceci élimine les temps de réponse trop longs des serveurs, l'engorgement du réseau, et l'éventuelle indisponibilité des sources. La gestion des métadonnées (statistiques sur les données interrogées, information sur leur origine…) constitue un autre avantage non négligeable pour cette approche. En effet, il est facile de stocker des métadonnées dans un entrepôt de données et de permettre aux utilisateurs de les interroger. D'autre part, le stockage local des données permet un grand contrôle de l'information dans la mesure où on est propriétaire des données, ce qui fournit encore la possibilité d'appliquer différentes techniques du Data Mining* (fouille de données). Dans le cadre d'un entrepôt dédié à la biologie, les annotations sur les séquences peuvent notamment être retravaillées.

L'inconvénient majeur de cette approche réside au niveau de la qualité des données stockées dans l'entrepôt. En effet, les réponses aux requêtes peuvent fréquemment être construites à partir de données périmées. La mise à jour de l'entrepôt peut s'avérer coûteuse, sans prendre en considération l'existence des droits de copie de données de certaines sources.

Les difficultés liées à l'architecture se rencontrent d'abord lors de la construction de l'entrepôt puis lors de sa maintenance. En effet, afin de construire le schéma intégrateur de l'entrepôt, une étude des sources doit être menée au préalable pour identifier les informations pertinentes à stocker avant de les extraire. Là, le choix d'un langage adapté à la représentation des informations à stocker est nécessaire. Une série de nettoyage des données visant à supprimer les redondances des données des sources précède souvent leur insertion.

La maintenance de l'entrepôt quant à elle, concerne généralement la mise à jour des données. Ceci impose l'élaboration des mécanismes permettant de détecter quand et comment les données dans les sources changent. Dans le domaine de la biologie, les sources évoluent extrêmement vite, ce qui complique davantage le problème de la mise à jour, car ces sources n'indiquent pas précisément ce qui a été changé, mais listent simplement les fiches d'annotation qui ont été touchées par une mise à jour. Il faut ainsi prévoir des mécanismes spécifiques de répercussion des mises à jour de l'entrepôt pour modifier les annotations ajoutées dans l'entrepôt et relancer les outils de fouille de données en fonction de l'évaluation de données des sources.

1.4.2. La médiation de données

(Wiederhold, 1998) a introduit le concept de médiateur, ou "manager" de requêtes, entre bases de données hétérogènes et distribuées. En relation avec l'intégration de données, l'approche Médiateur consiste à définir un système de médiation entre l'agent (humain ou logiciel) qui pose une requête et l'ensemble des sources accessibles via le Web potentiellement pertinentes pour répondre à la requête. L'objectif est de donner l'impression d'interroger un système centralisé alors que les sources interrogées sont réparties, autonomes et hétérogènes. L'architecture générale commune à tous les systèmes Médiateurs centralisés est illustrée dans la Figure 5.

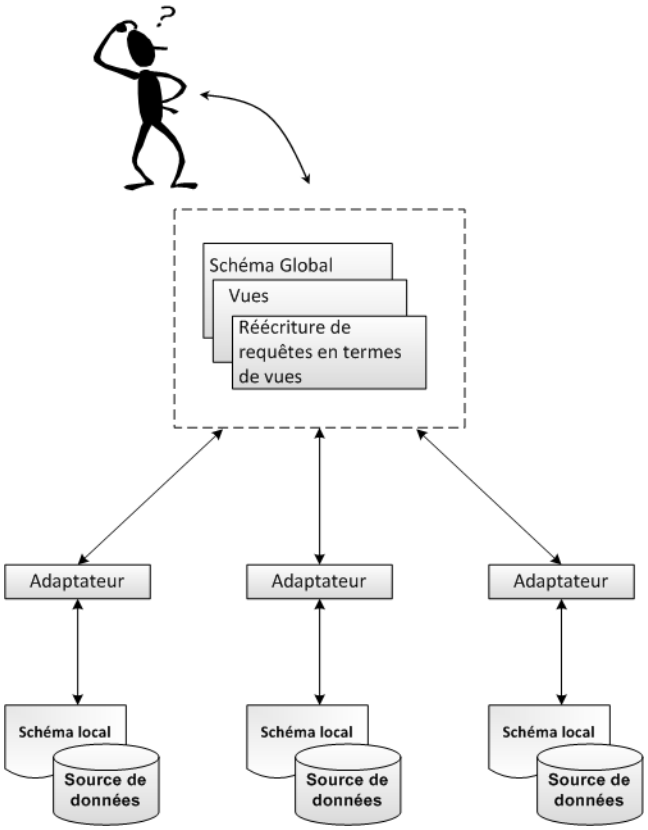

Figure 5. Architecture d'un système Médiateur.

Les composants principaux d'un système de médiation sont le *Médiateur* et un *adaptateur* (Wrapper) pour chaque source de données. Le Médiateur comprend un schéma global (qui peut être une ontologie) dont le rôle est central : c'est un modèle du domaine d'application du système. Le schéma global fournit un vocabulaire structuré servant de support à l'expression des requêtes. Par ailleurs, elle établit une connexion entre les différentes sources accessibles.

L'adaptateur contient des détails techniques et des détails sur le modèle de données de la source. C'est un composant important dans un système d'intégration, mais aussi dans l'architecture d'un entrepôt de données.

L'utilisation d'un Médiateur pour intégrer les sources de données s'appuie sur une transformation des requêtes lors de l'exécution. Un utilisateur pose ses requêtes en fonction du schéma global du Médiateur au moyen du langage de requêtes adopté par le système. Le Médiateur ne peut pas évaluer directement ces requêtes qui lui sont posées, car les données sont stockées de façon distribuée dans les sources indépendantes. Il ne dispose que de vues abstraites des données des sources. Ces vues sont des requêtes prédéfinies exprimées en fonction du schéma global dans un langage particulier, appelé langage de vues. La requête exprimée par l'utilisateur sur le schéma global est décomposée et réécrite en termes de vues, en requêtes locales adressées aux sources de données accédées.

Le Médiateur se charge alors des actions suivantes dans le système :

- reçoit une requête formulée sur le schéma global à partir de l'utilisateur ;
- décompose cette requête en sous-requêtes qui vont chacune interroger une source individuelle. Cette décomposition se base sur les descriptions des sources (vues) ;
- optimise le plan de l'exécution en se basant aussi sur les descriptions des sources ;
- envoie les sous-requêtes aux adaptateurs des sources individuelles, qui vont transformer ces sous-requêtes en requêtes sur les schémas et les modèles locales des sources ;
- reçoit les réponses sur ces sous-requêtes à partir des adaptateurs ;
- combine les réponses reçues en une seule finale ;
- puis envoie la réponse finale à l'utilisateur. Ce dernier reçoit ainsi une réponse à sa requête initiale.

Les différents systèmes d'intégration d'informations à base de Médiateur se distinguent par la façon dont est établie la correspondance entre le schéma global et les schémas des sources de données à intégrer (Levy, 2000). On distingue en effet deux manières principales d'établir la correspondance entre le schéma global et les schémas des sources de données à intégrer (GAV et LAV) et une troisième manière qui combine les deux précédentes (GLAV) introduite beaucoup plus récemment (Baader, et al., 2003). (Voir la section 1.5.1 pour plus d'explications).

L'avantage immédiat d'une architecture Médiateur est que l'utilisateur n'a pas à se soucier du choix des sources ce qui est d'autant plus important qu'il a un grand nombre de sources disponibles sur le Web. D'autre part, l'ajout d'une nouvelle source est simple, surtout dans l'approche LAV puisqu'il suffit de décrire la source à ajouter en termes du schéma du Médiateur. Par rapport à l'architecture entrepôt, un Médiateur évite toute gestion de mises à jour des données puisque les données restent dans les sources. Dans le contexte des données biologiques qui évoluent très rapidement, cet avantage est non négligeable. Cependant, certains problèmes peuvent être rencontrés dans une architecture Médiateur. Ils sont liés au fait que les données ne sont pas accessibles localement. Le premier est celui du cas de panne d'une source. Dans une telle situation, on peut plus

répondre à certaines requêtes. Le second inconvénient est celui du temps de réponse. Les réponses étant construites à la volée et au fur et à mesure de la collecte des informations dans les différentes sources, le temps de réponse à une requête est nettement supérieur à celui qu'on a dans une approche entrepôt où des données centralisées sont interrogées directement.

Globalement, les trois principales difficultés rencontrées dans la construction d'un Médiateur sont :

- le choix du langage utilisé pour exprimer le schéma global (OWL, Logiques de Description…), ainsi que le choix des langages pour exprimer, en fonction de ce schéma, les vues sur les sources à intégrer et les requêtes des utilisateurs (SQL, XQuery…);
- en fonction de ces choix de modélisation, la conception et la mise en œuvre d'algorithmes de réécritures de requêtes à exécuter afin d'obtenir l'ensemble des réponses à une requête globale (voir l'annexe A pour quelques algorithmes de réécriture de requête).
- l'évaluation des plans de requêtes sur les sources.

Lorsque l'on évalue les plans de requêtes sur les sources, on récupère un ensemble d'instances qui peuvent être potentiellement redondantes. On peut alors chercher à faire correspondre les instances entre elles.

Quelques exemples de systèmes de médiation en domaine bioinformatique sont : BioKleisli (Davidson, et al., 1997), TAMBIS (Stevens, et al., 2000), DiscoveryLink (Haas, et al., 2001), BACIIS (Ben Miled, et al., 2004).

1.4.3. La fédération de données

Dans un Système de bases de données fédérées (SBDF), les sources sont indépendantes les unes des autres, et des connections entre toutes les paires de sources que l'on souhaite faire communiquer sont établies afin que chaque source partage partiellement les données avec les autres (Sheth and Larson, 1990). Chaque source dans la fédération peut aussi fonctionner indépendamment des autres et de la fédération.

Il y a des SBDF dits "*fortement couplés*" et d'autres dits "*faiblement couplés*".

- Un SBDF fortement couplé possède un schéma intégrateur (ou plusieurs schémas intégrateurs) qui peut être construit soit semi-automatiquement par des techniques d'intégration de schémas, ou crée manuellement par les développeurs. Pour résoudre l'hétérogénéité logique, un expert du domaine a besoin de déterminer les correspondances entre les schémas des sources. Ce genre de SBDF (fortement couplé) est souvent statique et difficile à évoluer, car les techniques d'intégration de schéma ne permettent pas d'ajouter ou de retirer facilement les composants du système.
- Un SBDF faiblement couplé n'a pas de schéma intégrateur, mais il fournit quelques langages unifiés pour interroger les sources. Dans cette configuration, les sources

ont plus d'autonomie, mais les utilisateurs doivent résoudre toutes les hétérogénéités sémantiques. Chaque composé du système peut décider sur la façon de voir les données accessibles dans la fédération ; et puisqu'il n'y a pas de schéma global, chaque source peut créer son propre "schéma fédéré" pour ses besoins.

La façon la plus naïve pour assurer l'interopérabilité est de faire correspondre le schéma de chaque source aux schémas des autres. Un exemple d'un tel SBDF est donné dans la Figure 6.

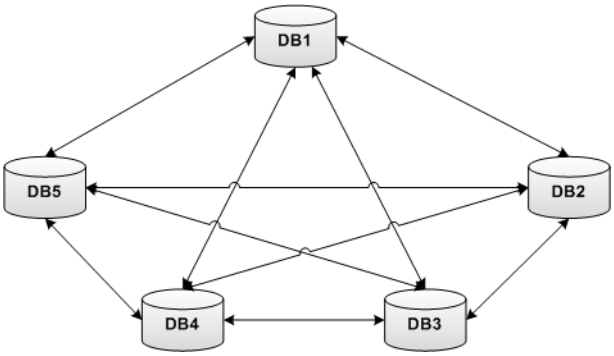

Figure 6. Architecture de bases de données fédérées.

L'architecture fédérée est très appropriée lorsqu'il s'agit d'un nombre de sources autonomes et se veut d'une part conserver leurs "indépendance", permettant ainsi aux utilisateurs de les interroger séparément, et d'autre part leur permettre de collaborer entre elles pour répondre à certaines requêtes. Cette approche est très simple, mais aussi très coûteuse puisque afin de permettre à *n* sources de communiquer chacune avec *(n-1)* sources, implique donc de décrire *n*(n - 1)* traductions de schémas. Ce qui devient fatigant lorsqu'il s'agit d'un grand nombre de composés dans la fédération.

Plusieurs projets d'intégration de données biologiques ont adopté l'approche fédérée : ENQUire (Jamison, et al., 1996), Docking-D (Aberer, 1995), GDB (Fasman, 1994) ou TINet (Eckman, et al., 2001).

1.4.4. L'approche multi-agents

Depuis quelques années les systèmes multi-agents (SMA) ont pris une place importante dans le domaine de l'informatique en général et dans le domaine de l'intelligence artificielle et des systèmes distribués en particulier. Ce sont des systèmes idéaux pour représenter des problèmes de nature hétérogènes et distribués (Jarras and Chaib-draa, 2002).

Dans la littérature, plusieurs définitions d'un système multi-agents sont disponibles. Ainsi, un système multi-agents est défini par Mandiau (Danflous, 2002) comme étant "*un ensemble d'entités qui coordonnent leurs connaissances, buts, expériences et plans pour agir ou résoudre des problèmes, incluant le problème de coordination inter-agents lui-même*". Il est défini d'une façon plus détaillée par Ferber (Ferber, 1995) (Figure 7) comme

étant "*un système composé d'un environnement ; d'un ensemble d'objets passifs pouvant être perçus, crées, modifiés ou détruits par des agents ; d'un ensemble d'agents actifs ; d'un ensemble de relation qui relient les objets entre eux ; d'un ensemble d'opérations ou de compétences offrant la possibilité aux agents de percevoir, produire, consommer, transformer et manipuler des lois de l'environnement et d'un ensemble de lois qui sont des opérateurs chargés de représenter l'application des actions des agents sur le monde et la réaction de ce monde à ces actions*". Une autre définition citée par Fayech (Fayech, 2003), et qui s'inspire de celle de (Geen, et al., 1997), énonce que "*Un système multi-agents est un réseau d'agents (solveurs) faiblement couplés qui coopèrent ensemble pour résoudre des problèmes qui dépassent les capacités ou les connaissances individuelles de chaque agent. Les agents sont autonomes et peuvent être de natures hétérogènes*".

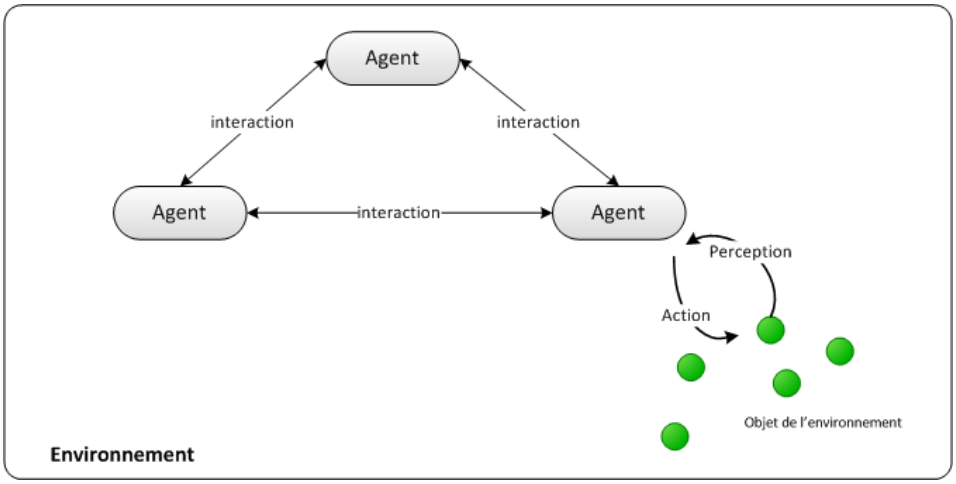

Figure 7. Représentation d'un système multi-agents selon Ferber.

Les agents d'un SMA, n'ayant pas une visibilité globale sur leur environnement, ils ne peuvent avoir qu'un champ d'actions limité sur l'ensemble des objets de cet environnement. De ce fait, pour résoudre un problème global, ces agents sont amenés à coopérer et à communiquer pour échanger des informations et pour mieux coordonner leurs actions individuelles et locales.

Dans le domaine de l'intégration de données, Les agents d'un système multi-agents sont souvent appelés "*agents médiateurs*" ou "*agents de médiation*" de sources de données. Ils sont aussi appelés, selon (Sycara and Zeng, 1996), *des* "*agents d'exécution de tâches*" (*Taskagent*), puisqu'ils vont coordonner la répartition et l'exécution des requêtes sur les agents sources d'informations. (Jarras and Chaib-draa, 2002) et (Shakshuki, et al., 2003) les appellent aussi "*Agents courtier*" ou "*broker agents*", puisqu'ils assurent le courtage d'information entre clients et fournisseurs.

Les agents médiateurs sont des facilitateurs, qui facilitent l'accès à plusieurs sources d'information. Ils sont aussi utilisés entre un utilisateur ou un agent interface et plusieurs agents adaptateurs de sources d'informations. *Les agents adaptateurs* ou *wrappers* sont des agents permettant de récupérer des données dans des sources d'informations

hétérogènes et distribuées sur le réseau et pouvant communiquer avec une source de données à distance pour extraire des informations (Shi, 2002; Wang, et al., 2003) (comme dans le cas des entrepôts de données ou des systèmes médiateur). Ils permettent d'adapter le contenu et le format des informations récupérées sur les sources distantes, de leur format d'origine, au format requis par l'application SMA qui les utilisent. Ce sont des traducteurs qui permettent au système d'information orienté agents de communiquer avec des sources de données hétérogènes ; ils sont capables de traduire les données récupérées en langages SMA et vice versa.

Les agents médiateurs fournissent aux sources de données des mécanismes d'interopérabilité. Ces mécanismes peuvent comprendre en plus du protocole de communication, des ontologies décrivant les données contenues dans ces sources (Roze, 2003). L'architecture globale du système est définit par l'organisation du SMA qui précise aussi pour chaque agent son rôle et ses fonctions par rapport au groupe, et les règles d'interaction à adopter dans son environnement (Adam., 2000).

Les SMA d'intégration de données sont généralement dotés d'agents capables d'offrir, entre autres, les fonctions suivantes:

- ***La découverte des sources*** : trouver la bonne source de données pour l'interroger.
- ***La recherche des informations*** : identifier les informations non structurées et semi-structurées.
- ***Le filtrage des informations*** : analyser les données et éliminer celles qui sont inutiles.
- ***La fusion des informations*** : regrouper les informations d'une manière significative.

Dans le domaine de l'intégration des données biologiques, l'approche multi-agents a été utilisée dans le cadre des projets ISYS et IGD-GIS. Le projet ISYS (Integrated SYStem) (Siepel, et al., 2001) offre une architecture "*plug and play*" orientée agents, grâce à laquelle les biologistes peuvent manipuler les données et les outils de leur choix. Ces outils peuvent avoir été développés et être maintenus indépendamment les uns des autres. Les composants échangent des informations sans avoir connaissance les uns des autres, mais en s'adressant uniquement à des agents médiateurs en charge de diffuser leurs messages. La manipulation des composants enregistrés auprès du système se fait à la discrétion de l'utilisateur ; ISYS se contente uniquement de suggérer les composants offrant les traitements les plus adaptés aux données sélectionnées.

Dans IGD-GIS (Integrated Genomic Database - Genome Information System) (Burger, et al., 1997), l'architecture proposée fait appel à un réseau d'agents communiquant entre eux via Corba (Geib, et al., 1999) et KQML (Finin, et al., 1994). Tous ont une fonction bien précise, tel que l'agent EIA (External Interface Agent) qui gère l'interface utilisateur, ou l'agent SCA (Selector Composer Agent) qui s'occupe de décomposer la requête globale en sous-requêtes destinées aux sources de données locales. C'est une approche très modulaire et facilement extensible.

1.4.5. L'intégration P2P

L'émergence de systèmes pair-à-pair (*Peer-to-Peer* ou *Peer Data Management Systems ou* encore *PDMS*) de partage de fichiers a conduit les chercheurs à considérer l'architecture P2P dans le contexte de l'intégration et le partage de données (Ives, et al., 2011; Milo, et al., 2005; Nejdl, et al., 2002; Rousset, et al., 2006; Yang and Garcia-Molina, 2002). La médiation de données dans ces systèmes est fondée sur une architecture décentralisée constituée d'un ensemble de pairs autonomes et distribués contenant des données qui peuvent être partagées. Les mappings sont établis entre petits ensembles de pairs. Chaque pair est interconnecté avec un certain nombre de pairs du réseau (appelés voisins) à l'aide de formules de coordination (Bernstein, et al., 2002).

Les systèmes P2P sont une généralisation des systèmes de médiation, puisque certains pairs jouent le rôle de médiateurs vis-à-vis des autres. Dans un système pair à pair, chaque participant est à la fois consommateur (client) et producteur (serveur) de ressources et de services. L'objectif principal de tels systèmes est de fournir une interopérabilité sémantique entre plusieurs sources en l'absence de schéma global.

Les besoins pour un PDMS sont à l'intersection de ceux des systèmes distribués et des bases de données. Les problématiques introduites par les PDMS par rapport aux SGBD classiques concernent la localisation des données, la médiation dynamique avec réécriture de requêtes, ainsi que la volatilité des sources et l'évolution de la taille du système.

Dans le domaine bioinformatique nous citons deux exemples de ce type de systèmes : Promethea (Claypool and Madria, 2006) et le SEED (Overbeek, et al., 2004).

1.4.6. L'intégration navigationnelle

L'intégration navigationnelle s'inspire de ce que font habituellement les utilisateurs lors d'une recherche d'information sur le Web, qui implique une recherche de page en page par clic de souris (Davidson, et al., 1995). Elle est fondée sur l'existence de liens entre les données représentées dans différentes sources du Web. Ces liens sont rendus possibles par la présence de références qui permettent à un utilisateur de passer d'une source à une autre et donc d'une donnée à une autre (Figure 8).

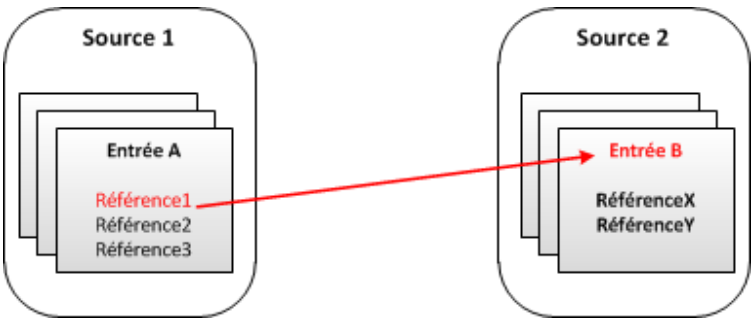

Figure 8. Connexion entre deux sources via une référence.

Dans l'approche d'intégration navigationnelle, les requêtes sont transformées en des expressions de chemin (éventuellement plusieurs) qui peuvent chacune répondre à la requête avec différents niveaux de satisfaction (Mork, et al., 2001). Un chemin correspond à une suite de références permettant d'accéder à l'information finale attendue. Les chemins spécifiques constitue essentiellement des workflows dans lesquels la (ou les) sortie(s) d'une source ou d'un outil est (ou sont) redirigée(s) vers l'entrée de la source suivante jusqu'à ce que l'information sollicitée est atteinte (Buttler, et al., 2002). C'est une approche intéressante puisqu'elle peut permettre d'accéder à des informations uniquement accessibles par le biais d'une navigation au sein de plusieurs (Friedman, et al., 1999). Il faut noter que pour une même requête, plusieurs chemins peuvent accéder à une réponse, chacun ayant son propre niveau de satisfaction à évaluer.

L'approche navigationnelle ne propose pas une modélisation des données elles-mêmes mais plutôt applique une modélisation représentant les sources comme un ensemble de pages avec des interconnexions et des points d'entrée spécifiques, ainsi que des informations complémentaires telles que la spécification du contenu des sources, des éventuelles contraintes de chemins, et des paramètres facultatifs ou obligatoires d'entrée (Calí, et al., 2002; Lenzerini, 2002). (Friedman, et al., 1999) affirment que ce modèle permet en effet la représentation des cas où la page contenant les informations souhaitées est uniquement accessible par un chemin de navigation particulier à travers d'autres pages (Hernandez and Kambhampati, 2004).

Le succès de l'approche navigationnelle en bioinformatique se trouve facilité grâce à un partage très important de références matérialisé par l'existence d'un grand nombre de liens entre les sources de données. Ces liens sont largement utilisés par les chercheurs pour naviguer d'une source à une autre afin de trouver une information. Les liens sont purement syntaxiques, basés sur la présence d'un identifiant commun entre deux sources, comme le montre l'exemple de la Figure 8. Par exemple, un chercheur qui souhaite des informations sur les protéines impliquées dans une voie métabolique donnée de la levure va naturellement traverser plusieurs sources telles que BIOGRID (Stark, et al., 2005), SGD (Cherry, et al., 1998), CYGD-MIPS (Guldener, et al., 2005) et *UniProt*[37] (Consortium, 2012) en suivant les références croisées que proposent les sources.

L'approche navigationnelle a ses particularités en bioinformatique. Ceci se manifeste dans la diversité des liens et des chemins possibles à travers les sources. Ainsi, quant à la nature variée des liens entre les données génomiques, on distingue dans un premier temps les liens qui conduisent à des informations sur une même entité (par exemple, de CYGD-MIPS à SGD) des liens qui apportent des informations sur une autre entité (par exemple, de SGD à BIOGRID). Ensuite, on distingue les *liens internes* permettant d'accéder à des données d'une même source (par exemple, BIOGRID vers BIOGRID), les *liens externes* permettant d'accéder à des données d'une autre source (par exemple SGD vers AmiGO[38]).

[37] http://www.uniprot.org/

[38] http://amigo.geneontology.org/cgi-bin/amigo/go.cgi

Les liens externes sont également qualifiés de références croisées, ou cross-références, ils ne sont pas nécessairement symétriques. Il y a par exemple un grand nombre de sources qui crossréférencent SGD et qui ne sont pas référencées en retour. Enfin, on distingue les liens matérialisés par des *liens hypertextes* qui assurent une navigation d'une source à une autre, des liens matérialisés par des identifiants qui nécessitent alors une connexion manuelle aux sources.

En ce qui concerne la diversité des chemins, pour une requête donnée, il existe un très grand nombre de chemins possibles au travers les sources. En effet, considérons par exemple la requête "*lister toutes les citations de PubMed qui sont reliées à une entrée SGD décrivant une protéine qui figure dans une interaction donnée*". Pour répondre à une telle requête, un biologiste (ou un moteur de requêtes) doit naviguer au sein de plusieurs sources. Il est possible de partir de BIOGRID qui contient des informations sur l'interaction donnée, puis d'utiliser les différentes sources qui y sont reliées par des liens de référence. La Figure 9 illustre un graphe de liens existants entre les différentes sources requises pour répondre à la requête.

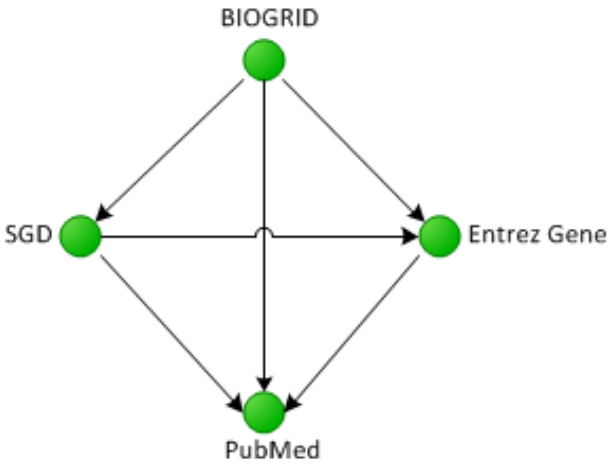

Figure 9. Graphe de liens entre les sources.

En partant de BIOGRID, et en effectuant une recherche par mot clé sur l'interaction donnée, différents chemins sont possibles pour accéder aux citations PubMed. Il est possible d'accéder directement aux citations PubMed à partir de BIOGRID, mais il est également possible d'utiliser des sources intermédiaires, générant ainsi plusieurs chemins. Au total, on trouve quatre chemins entre BIOGRID et PubMed en utilisant le graphe de la Figure 9. Ces chemins sont représentés dans Figure 10.

(C1) BIOGRID ⇨ PubMed
(C2) BIOGRID ⇨ SGD ⇨ PubMed
(C3) BIOGRID ⇨ Entrez Gene ⇨ PubMed
(C4) BIOGRID ⇨ SGD ⇨ Entrez Gene ⇨ PubMed

Figure 10. Les quatre chemins (C1 à C4) depuis BIOGRID jusqu'à PubMed.

Le choix des chemins a un impact sur le résultat, que ce soit sur le plan qualitatif ou quantitatif (Lacroix, et al., 2004). Par exemple, utiliser un chemin passant par la source SGD (chemin C2) peut amener plus de citations qu'un autre chemin passant par la source Entrez Gene (chemin C3). Le résultat va dépendre directement des sources intermédiaires du chemin et donc des entités biologiques correspondantes traversées et du contenu de chaque source. Cependant, ce procédé de navigation devient vite impossible si la recherche doit s'effectuer en masse.

Quelques exemples de systèmes navigationnels bioinformatique sont les suivants : SRS (Etzold, et al., 1996), Entrez (Sayers, et al., 2011) et GeneCards (Michael Rebhan, 1998).

1.4.7. Les accès par les portails et les plateformes logicielles

Dans le domaine de la bioinformatique, d'autres systèmes, qui ne font pas partie des systèmes multi-bases mais des portails tels que Expasy[39] (Artimo, et al., 2012), GenomeNet (Kataoka and Satou, 1999) ou ENSEMBL (Flicek, et al., 2012), se contentent de donner accès à une ou plusieurs ressources bioinformatiques à travers d'interfaces Web. La couche logicielle destinée à les interroger peut être constituée d'une page Web statique ou dynamique ; des restrictions sont imposées aux utilisateurs, notamment pour l'exécution de requêtes qui sont jugées trop coûteuses, comme celles contenant des jointures multiples, ou faisant appel à de nombreux opérateurs.

Une autre forme d'accès aux données peut se faire via des plateformes logicielles telles que Iogma/Genostar (Rechenmann, 2002) ou Imagene (Médigue, et al., 1999), qui fournissent des fonctions d'accès aux données et une panoplie d'outils d'analyse.

1.5. Traitement des requêtes dans un système de médiation

Pour qu'un système de médiation soit capable de traiter une requête, le médiateur doit être capable de la reformuler en sous-requêtes qui seront envoyées vers les sources de données pertinentes à la résolution de la requête. En effet, un utilisateur pose sa requête dans le langage de requêtes et en termes du schéma global du médiateur. Le médiateur ne peut pas évaluer directement les requêtes qui lui sont posées car il ne possède pas de données. Ces dernières sont stockées de façon distribuée dans les sources. Le médiateur ne dispose que de vues abstraites des données stockées dans les sources. Ces vues sont des requêtes prédéfinies exprimées en fonction du schéma global dans un langage particulier, appelé langage de vues. Elles décrivent l'extension de chacune de ces vues est représentée par les données qui peuvent être obtenues par le médiateur en interrogeant les sources par des requêtes spécialisées, selon leur schémas. Puisque les contenues des sources sont représentés par des vues, le problème de la traduction revient alors à trouver une méthode pour répondre à une requête en utilisant un ensemble de vues.

Les différentes étapes du traitement d'une requête se résument comme suit (Figure 11):

[39] http://expasy.org/

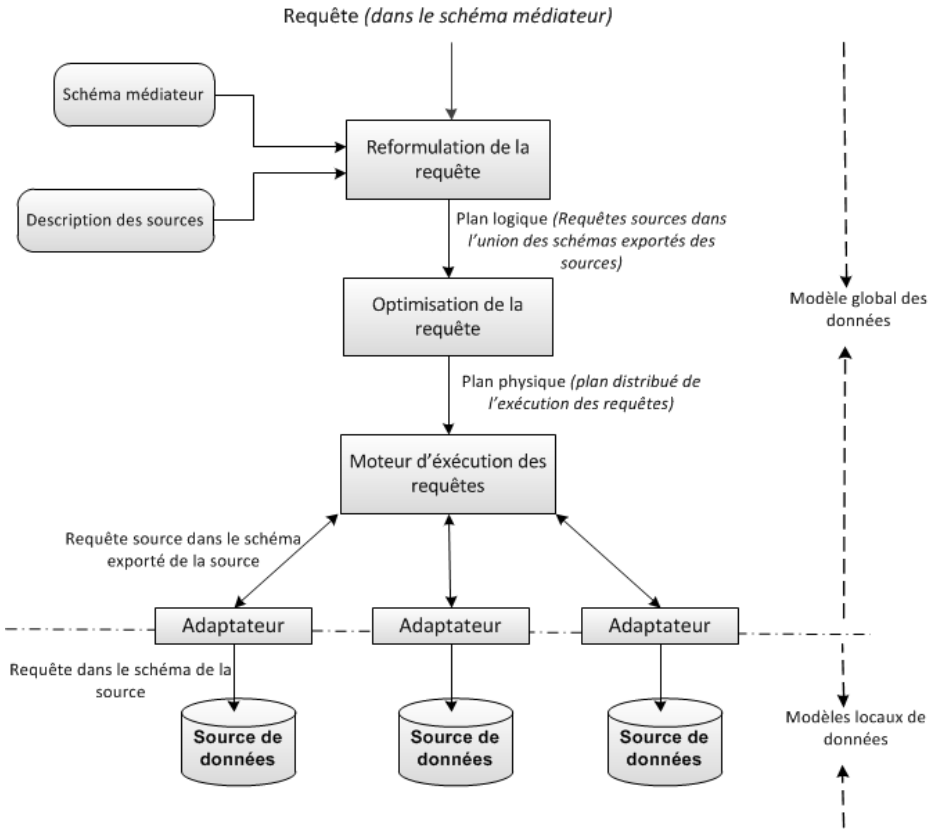

Figure 11. Etapes du traitement des requêtes dans un système médiateur (Levy, 2000).

En utilisant le schéma global et les vues, la requête de l'utilisateur est reformulée en requêtes sources dans les schémas exportés des sources. Un schéma exporté d'une source réfère au schéma de la source traduit dans le modèle global de données. Ces requêtes source fournissent un plan logique à l'optimisateur de requêtes qui produit ensuite un plan physique d'exécution de requêtes. Par la suite, ce plan est exécuté par le biais du moteur d'exécution de requêtes qui se communiquent avec les sources de données à travers leurs adaptateurs. Le moteur d'exécution de requêtes collecte ultérieurement les résultats à partir des sources et les combine pour les présenter à l'utilisateur.

Deux critères importants doivent être pris en considération lors de la reformulation de la requête (Levy, 1999) :

- ***L'exactitude ou la justesse sémantique de la reformulation*** : les réponses obtenues à partir des sources doivent être des réponses correctes à la requête originale.
- ***Minimisation des accès aux sources*** : les sources qui ne peuvent pas contribuer totalement ou partiellement à la réponse de la requête ne devraient pas être accédées. En plus d'éviter l'accès à des sources redondantes, on devrait reformuler les requêtes le plus spécifiquement possible à chacune des sources interrogées pour éviter l'évaluation redondante des requêtes.

1.5.1. Approches de l'établissement des correspondances

Lenzerini formalise un système de médiation comme un triple (*G*, *S*, *M)* (Lenzerini, 2002) où :

- *G* est le schéma global, exprimé dans un langage L_G sur un alphabet A_G. L'alphabet comprend un symbole pour chaque élément de *G* (i.e., une relation si *G* est relationnelle, classe si *G* est orienté objet, etc.)
- *S* est le schéma source, exprimé dans un langage L_S sur un alphabet A_S. L'alphabet A_S comprend un symbole pour chaque élément de la source.
- *M* est le mapping entre *G* et *S*, constitué par un ensemble d'assertions de la forme :

$$q_S \text{ --------> } q_G$$

$$q_G \text{ --------> } q_S$$

où q_S et q_G sont deux requêtes de la même arité, respectivement sur le schéma source *S*, et sur le schéma global *G*. les requêtes q_S sont exprimées dans un langage de requête $L_{M,S}$ sur l'alphabet A_S, et les requêtes q_G sont exprimées dans un langue de requêtes $L_{M,G}$ sur l'alphabet A_G. Intuitivement, l'assertion q_S ------> q_G précise que le concept représenté par la requête q_S sur les sources correspond au concept dans le schéma global représenté par la requête q_G (la même chose pour une assertion de type q_G ------> q_S).

Les deux principales approches utilisées pour établir les correspondances entre le schéma source et le schéma global sont (Lenzerini, 2002) :

- l'approche Global-As-View (GAV), aussi appelée approche centrée-requête (Halevy, 2001), et
- l'approche Local-As-View (LAV) aussi appelée approche centrée-source (Levy, et al., 1995; Ullman, 2000). Ces deux approches se distinguent par la façon de définir les liens sémantiques *M (Lenzerini, 2002; Rousset and Reynaud, 2004)*.

1.5.1.1. Approche centrée-requêtes

L'approche centrée-requêtes ou l'approche GAV (Global As View) a été la première à être proposées pour l'intégration d'informations. Elle consiste à définir le schéma global en fonction des schémas des sources de données à intégrer. Pour cela, les structures du schéma global sont définies à partir des vues sur les structures des schémas des sources à intégrer ; cette approche suppose donc que les sources à intégrer soient connues à l'avance.

D'une manière plus formelle, le mapping *M* associe à chaque élément *g* dans *G* une requête q_s sur *S*. En d'autres termes, le langage de requêtes $L_{M,G}$ permet seulement des expressions constituées par un symbole de l'alphabet A_G. Ainsi, un mapping GAV est un ensemble d'assertions, un pour chaque élément *g* du *G*, de la forme (Lenzerini, 2002):

$$g \text{ -----> } q_s$$

D'un point de vue modélisation, l'approche GAV est basée sur l'idée que le contenu de chaque élément *g* du schéma global devrait être caractérisé en termes d'une vue q_s sur les sources. Dans un certain sens, le mapping montre explicitement au système comment retrouver les données quand on veut évaluer les différents éléments du schéma global.

1.5.1.2. Approche centrée-source

L'approche centrée-source ou l'approche LAV (Local As View) est l'approche duale qui consiste à définir les schémas des sources de données à intégrer, en fonction du schéma global. Le mapping *M* associe à chaque élément *s* du schéma source *S* une requête q_G sur *G*. en d'autres termes, le langage de requêtes $L_{M,S}$ permet seulement des expressions constituées par un symbole de l'alphabet A_s. Ainsi, un mapping LAV est un ensemble d'assertions, une pour chaque élément s du *S*, de la forme (Lenzerini, 2002) :

$$s \text{ -----> } q_G$$

D'un point de vue modélisation, l'approche LAV est basée sur l'idée que le contenu de chaque source *s* devrait être caractérisé en termes d'une vue q_G sur le schéma global.

Les avantages et inconvénients de cette approche sont inversés par rapport à l'approche GAV. Les approches LAV et GAV sont comparées dans (Ullman, 2000) du point de vue traitement de requêtes. D'une manière générale, il est très bien connu que le traitement des requêtes dans l'approche LAV est une tâche difficile. En fait, dans cette approche l'information unique que nous possédons sur les données dans le schéma globale est à travers les vues représentant les sources, et de telles vues fournissent seulement des informations partielles sur les données. Puisque les mappings associent à chaque source une vue sur le schéma globale, il n'est pas immédiat de déduire comment utiliser les sources afin de répondre aux requêtes exprimées au sujet du schéma global. D'autre part, le traitement des requêtes semble être plus facile dans l'approche GAV, où l'on peut profiter du fait que le mapping spécifie directement les requêtes de source qui correspondent aux éléments du schéma global. En effet, dans la plupart des systèmes GAV, la réponse aux requêtes se base sur une stratégie simple.

Du point de vue modélisation du système d'intégration de données, l'approche GAV prévoit un mécanisme de spécification qui a une saveur plus procédurale à l'égard de l'approche LAV. En effet, alors que dans LAV le concepteur peut se concentrer sur la spécification déclarative de contenu de la source en termes du schéma global, en GAV, on est obligé de spécifier la façon d'obtenir les données du schéma global par le biais de requêtes sur les sources. Une analyse des différences/similitudes entre les deux approches du point de vue modélisation est rapportée dans (Calí, et al., 2002; Calí, et al., 2001), où les auteurs abordent le problème de vérifier si un système LAV peut être transformé en GAV, et vice-versa.

Du point de vue maintenance du système d'intégration, l'approche LAV est très flexible par rapport à l'ajout ou la suppression de sources de données à intégrer : cela n'a aucun effet sur le schéma global, seules des vues doivent être ajoutées ou supprimées. En effet, rajouter une source revient à la décrire en fonction du schéma global qui n'est donc

absolument pas modifié. Le prix à payer pour cette flexibilité et cette simplicité de mise à jour est la complexité de la construction des réponses à une requête dans un Médiateur conçu selon l'approche LAV. La réécriture de requêtes en termes de vues est en effet bien plus complexe dans une approche LAV que dans une approche GAV comme déjà évoqué.

1.5.1.3. Approches mixtes

Les avantages des approches LAV et GAV ont été combinés dans des approches mixtes. C'est ainsi qu'ont été proposées l'approche Global-Local-As-View (GLAV) (Friedman, et al., 1999), qui utilise des règles de correspondance ayant plus d'un terme dans leur en-tête (l'en-tête peut être la combinaison d'un nombre quelconque de prédicats, contrairement à l'approche LAV ou GAV), et l'approche Both-as-View (BAV) (McBrien and et Poulovassilis, 2003), qui utilise des transformations incrémentales afin de lier deux schémas. L'approche BAV a été introduite dans le cadre du projet d'intégration AutoMed (Boyd, et al., 2002).

1.6. Conclusion

Avec la prédominance des technologies de réseau et d'Internet, l'accès aux données indépendamment de leurs locations de stockage physique est devenu fortement facilité. Ceci a permis aux biologistes d'accéder à une multitude de sources de données réparties, hétérogènes et autonomes et de combiner les données afin d'analyser leurs résultats. Cependant, la grande hétérogénéité que présentent les sources de données biologiques a rendu le recoure aux systèmes d'intégration de données une exigence. L'intégration de données est un domaine de l'informatique qui s'occupe de l'uniformisation de l'accès à ce type de données variées et dispersées.

Dans ce chapitre, nous avons donné un état de l'art sur les types des systèmes d'intégration qui ont été proposés dans le domaine de la biologie. Le lecteur a surement constaté que nous avons aussi donné une importance exceptionnelle aux systèmes de médiation à travers un survol sur son processus du traitement des requêtes. Ceci est dû au fait que dans la deuxième partie de cette thèse, nous allons parler d'un prototype d'un système de médiation conçu pour l'intégration des données de la levure *Saccharomyces cerevisiae*.

Chapitre 2

Ontologies et service Web au service de l'intégration des données biologiques

Sommaire

2.1. Introduction

L'objectif de l'intégration des données biologiques est de fournir un accès uniforme à travers le Web à un ensemble de sources de données autonomes, hétérogènes et distantes. A partir d'un point de vue technique, la difficulté vient du manque d'interopérabilité entre ces sources de données, qui peuvent utiliser des formats variés, des capacités spécifiques de traitement des requêtes, des protocoles différents. Toutefois, le véritable problème

pour l'intégration des données biologique est logique. Il vient de, ce que nous avons appelé dans le chapitre précédent, l'hétérogénéité sémantique entre les sources de données. Ainsi, la résolution de l'hétérogénéité sémantique est une des questions fondamentales à résoudre lors de la réalisation de l'interopérabilité entre les sources dans un système d'intégration de données. Elle tire avantage généralement de la vision du Web sémantique. L'objectif de ce dernier est d'avoir une architecture distribuée dans le monde entier où les données et les services interagissent facilement. Cette vision n'est pas encore une réalité dans le Web d'aujourd'hui, dans lequel, étant donné un besoin particulier, il est difficile de trouver une ressource Web appropriée. En outre, étant donné une ressource pertinente, il n'est pas facile de comprendre ce qu'elle fournit et comment l'utiliser. Pour résoudre ces limitations, faciliter l'interopérabilité et permettre ainsi la vision du Web sémantique, l'idée principale est de publier également des descriptions sémantiques sur les ressources Web. Ces descriptions reposent sur des annotations sémantiques qui relient les ressources à certains termes dans des ontologies prédéfinies.

Les ontologies constituent un outil principal du Web sémantique servant de base pour la représentation et le partage des connaissances dans, entre autres, un système d'intégration de données. D'autre part, la notion de service Web est une autre technologie permettant à des applications de dialoguer à distance via Internet, et ceci indépendamment des plates-formes et des langages sur lesquelles elles reposent. Vu le recoure que font fréquemment les systèmes d'intégration de données à ces deux technologies, et le fait qu'elles font parties des architectures discutées dans les chapitres 3 et 4, nous allons nous focaliser, dans ce chapitre, sur ces deux technologie. Ainsi le chapitre est structuré comme suit : dans la section 2.2, nous allons introduire le Web sémantique, puis la section qui suit, la section 2.3, sera consacrée aux ontologies avant de détailler la technologie de service Web dans la section 2.4.

2.2. Le Web sémantique

Le Web sémantique *(généralement associé au terme "Web 3.0")* est un projet initié par Tim Berners-Lee en 2001 (Berners-Lee, et al., 2001). Ce projet s'est développé sous l'égide du W3C[40] (World Wide Web Consortium) qui est un organisme de standardisation des formats informatiques utilisés sur internet. A l'origine et depuis lors, son ambition a été de développer un ensemble de technologies visant à décrire et exploiter de manière systématique la sémantique des ressources du Web. Sans remettre en question les fondements technologiques du Web actuel, il a pour objectif une évolution du Web qui permettrait aux données disponibles d'être plus facilement utilisables et interprétables automatiquement, par des agents logiciels**.**

Les technologies du Web sémantique facilitent ceci en permettant la création des données, l'expression des vocabulaires et des règles qui les décrivent, et la construction des systèmes capables de les manipuler dans de bonnes conditions d'interopérabilité. Ceux-ci grâce à un certain nombre de standards et de technologies qui ont été développés par le

[40] http://www.w3.org/

W3C, avec pour objectif de sortir les données des silos fermés que constituent les bases de données en ligne (Figure 12).

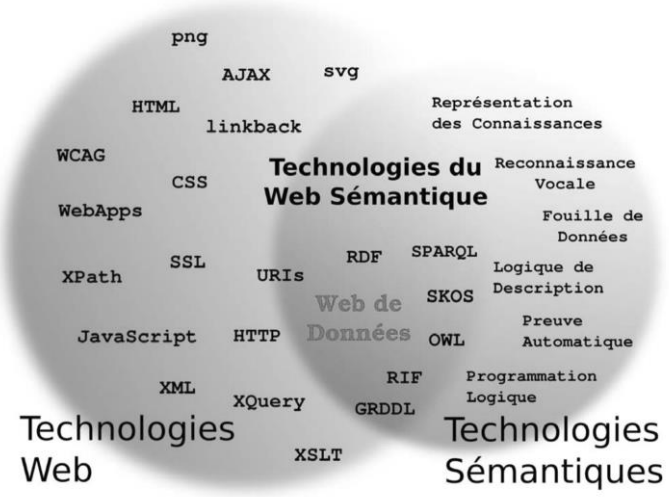

Figure 12. Technologies du Web sémantique (Bertails, et al.,2010)

D'après la vision de Tim Berners-Lee du Web sémantique, l'information pourrait être accessible et compréhensible non seulement par les humains mais aussi par les machines. Dans cette vision, le Web sémantique peut être considéré comme une pile "pièce montée" de langages comme nous le montrons dans la Figure 13 et que l'on peut structurer sur trois niveaux principaux (Berners-Lee, 2004):

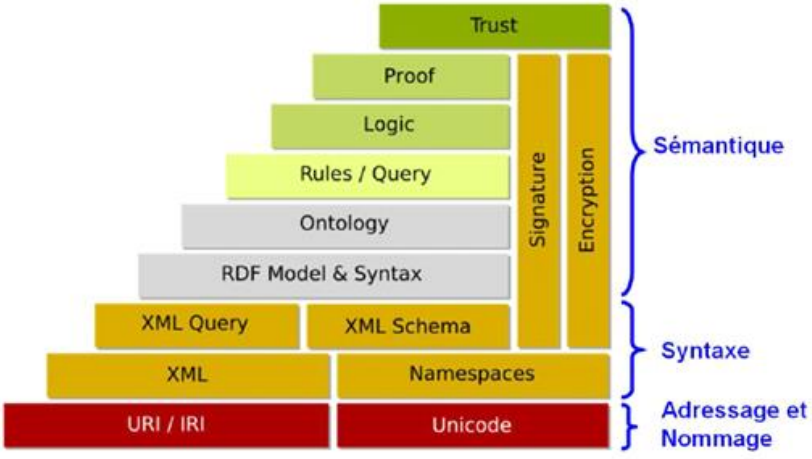

Figure 13. La pièce montée (Layer Cake) du Web sémantique proposé par Tim Berners-Lee

- ***Niveau d'adressage et de nommage***. Ce niveau est représenté par le standard d'adressage des ressources du Web URI et la norme Unicode pour le codage des caractères.
- ***Niveau syntaxique***. Ce niveau syntaxique est représenté par la définition des espaces de nomage qui permettent d'identifier les ressources du Web, le langage XML, XML schéma et le langage de requêtes XML Query.
- ***Niveau Sémantique.*** Ce niveau est représenté d'une part par les langages de représentation d'ontologies RDF/RDFS et OWL, et d'autre part par les langages de règles, de logique, de preuves et de confiance (trust).

2.2.1. Notions de ressource et de sémantique

La compréhension des concepts de base du Web sémantique se base en premier lieu en deux notions : "*ressource*" et "*sémantique*". Nous essayerons dans cette section de donner des définitions simple en s'inspirant de (Arena, 2012)[41].

Le Web repose sur trois technologies fondamentales :

- Le langage HTML permet de décrire la structure d'une page Web ;
- Une URI (Uniform Resource Identifier)* désigne de manière unique un document sur le Web ;
- HTTP est un protocole décrivant les requêtes et réponses échangées entre deux machines (client/serveur).

Ainsi, le Web actuellement contient un ensemble de pages localisées grâce à des adresses virtuelles, les URLs (Uniform Resource Locator)* et d'autres objets, recensés par des URNs (Uniform Resource Names)*. Les URLs et les URNs sont des URIs.

Une ressource, identifiée de manière unique par ces dernières peut renvoyer à une page HTML. A titre d'exemple, l'URL "*http://www.uniprot.org/uniprot/P04745*" renvoie à l'entrée décrivant la protéine "*AMY1A*" dans la source de données biologique UniProt (Apweiler, et al., 2004) et s'affichant comme une simple page HTML dans un navigateur. La limitation du Web actuel naît du fait qu'un ordinateur ayant à manipuler des URLs ne sait pas "rentrer" dans le contenu des ressources associées pour réaliser des traitements informatiques plus fins. Par exemple, on souhaiterait qu'un ordinateur puisse répondre correctement et automatiquement à un autre ordinateur lui posant la question : "*Trouves-moi la fonction moléculaire de la protéine AMY1A*". Pour cela, il faut que les deux programmes possèdent chacun des informations supplémentaires sur les entrées qu'ils ont à disposition. C'est à ce niveau qu'intervient les notions de sémantique[42].

La notion de sémantique peut se caractériser par le triplet (ressource, agent, concept), où une ressource est définie comme précédemment (ex. une page HTML), un agent peut

[41] http://blog.mysciencework.com/2012/06/25/Web-semantique-projet-amener-le-Web-plein-potentie.html

[42] http://www.semantique-gdr.net. Un dictionnaire des notions de sémantique utilisées en linguistique formelle.

être un simple programme s'exécutant sur un ordinateur et un concept est le terme utilisé par le Web sémantique pour désigner l'information associée à une ressource, et qui définit à quelle catégorie cette dernière appartient. La sémantique consiste donc fondamentalement en une relation d'interprétation. En langage courant, une réponse possible à la question : "*Que veut dire pour le programme X le terme UniProt ?*", est : "*UniProt, pour ce programme, est une source de données biologique !*". Ce principe est déjà utilisé en informatique avec les métadonnées, les TAGs… Les métadonnées sont des informations inscrites à l'intérieur d'un fichier, qui le décrivent et n'apparaissent pas lorsqu'on le visualise. Elles sont la base de l'archivage et du catalogage. Le Web sémantique en propose une utilisation systématique, structurée et normalisée.

Ce principe de construction des données prend toute son ampleur et son sens dans le contexte du Web, où les ressources sont entièrement distribuées. Localement les données nécessaires ne sont pas toujours disponibles. Par conséquent, un système doit être capable de mettre en œuvre des procédures automatiques d'exploration et d'interrogation. Deux agents logiciels peuvent par exemple échanger à distance des données, mais sous réserve qu'ils partagent le même système de description et les mêmes conventions d'interprétation sémantique des objets qu'ils manipulent (ils doivent parler le même langage). D'où l'intérêt de la standardisation et le rôle du W3C.

2.3. L'ontologie, outil principal du Web sémantique

Ce système de description basé, comme on l'a vu, sur la notion de concept, consiste en fait en un réseau de concepts désigné par le terme "ontologie". L'ontologie est la base de ce que l'on appelle la représentation des connaissances. Ce domaine est né dans le Web sémantique de la volonté des chercheurs de représenter diverses connaissances du monde actuel, de façon à ce qu'elles soient utilisables par des ordinateurs, pour qu'ils puissent effectuer des raisonnements sur ces connaissances.

Plusieurs définitions du terme ontologie ont été proposées selon les courants et les communautés de pensée.

2.3.1 Définitions d'ontologie

Le concept d'ontologie est employé dans des domaines très différents tels que la philosophie, la linguistique ou l'intelligence artificielle. En mettant la chaîne "*define: ontologie*", le moteur de recherche Google trouve une vingtaine de définitions d'ontologie. Elles sont toutes regroupées autour d'une définition philosophique ou informatique. Nous devrions donc vraiment faire la distinction entre l'ontologie philosophique et l'ontologie informatique et enlever une partie du litige.

Dans l'univers du traitement automatique de l'information, le terme "ontologie" semble avoir été introduit dans les années 1990 avec les travaux de Tom Gruber et son équipe à Stanford (Gruber, 1991; Gruber, 1992; Gruber, 1993). Tom Gruber possède l'une des définitions de l'ontologie les plus largement citées en informatique (Gruber, 1995), bien que les modèles conceptuels de divers types aient été construits dans l'informatique depuis

des décennies. Cette définition l'a expliqué davantage dans le "post" nommé "*What is an Ontology*"[43] en disant:

> *"In the context of knowledge sharing, the term ontology means a specification of a conceptualisation. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set-of-concept-definitions, but more general. And it is certainly a different sense of the word than its use in philosophy."*

Une traduction possible de cette définition est :

> *"Dans le cadre du partage des connaissances, le terme ontologie désigne une spécification d'une conceptualisation. Autrement dit, une ontologie est une description (comme une spécification formelle d'un programme) des concepts et des relations qui peuvent exister pour un agent ou une communauté d'agents. Cette définition est compatible avec l'utilisation de l'ontologie comme un ensemble de définitions de concept, mais plus général. Et il est certainement un sens différent du mot que son utilisation dans la philosophie"*

Gruber affirme que sa définition de l'ontologie n'est pas "ontologie au sens philosophique", mais dans le cadre du partage des connaissances.

Pour mieux éclaircir cette définition, il faut illuminer les mots clés suivant : "Partage", "Conceptualisation" et "Formelle".

Gruber définit "Une conceptualisation" comme suit (Gruber, 1995):

> *"A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose"*

Ce qui peut être traduit comme:

> *"Une conceptualisation est une vue abstraite et simplifiée du monde que l'on veut représenter pour un certain but"*

Cette définition fait référence à un modèle abstrait du monde à représenter, modèle qui identifie les concepts pertinents des phénomènes de ce monde. "Formelle" se réfère au fait qu'une ontologie doit être compréhensible par les machines. "Partage" reflète la notion de connaissance consensuelle décrite par l'ontologie, c'est-à-dire qu'elle n'est pas restreinte au point de vue de certains individus seulement, mais reflète un point de vue plus général, partagé et accepté par d'autres dans un domaine particulier et vers un objectif commun.

Parmi les nombreuses autres définitions contemporaines, citons :

- "Une ontologie est la définition des classes, relations, contraintes et règles d'inférence qu'une base de connaissances peut utiliser" (Welty and Ide, 1999);

[43] http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

- "Une ontologie est un vocabulaire partagé, plus une spécification (en fait une caractérisation) du sens "convenu" de ce vocabulaire" (Guarino, 1999) ;
- "Une ontologie définit les termes utilisés pour décrire et représenter un domaine de la connaissance" (W3C).

A partir de ces définitions, nous déduisons qu'une ontologie informatique définit un vocabulaire commun pour les chercheurs qui ont besoin de partager l'information dans un domaine. Elle code une connaissance en incluant des définitions, des informations exploitables, des concepts élémentaires d'un domaine et de leurs relations. De cette façon, les ontologies permettent la modélisation, le partage et la réutilisation des connaissances.

La modélisation d'un domaine nécessite la définition de concepts, relations, axiomes, fonctions et instances (Corcho and Gómez-Pérez, 2000). Ainsi, en pratique, une ontologie contient une description hiérarchique des concepts importants d'un domaine et décrit les propriétés de chaque concept à travers un mécanisme d'attributs-valeurs. Des relations entre les concepts peuvent être décrites au moyen de phrases logiques ou axiomes. Le processus d'analyse d'un domaine dans le but de spécifier formellement une ontologie requiert plusieurs étapes. Une des plus importantes est l'identification des types d'objets (concepts), la spécification de leurs attributs, des types de relation qu'il peut y avoir. Un concept est défini comme le signifié d'un terme dont on décide de négliger la dimension linguistique.

2.3.2 Types d'ontologies

Les ontologies nous permettent donc de capturer et de structurer les connaissances. Cependant, ces dernieres peuvent être appréhendées de différentes façons, c'est à dire différentes manières de représenter les connaissances. Il existe différents types d'ontologies qui, chacun, permet de décrire les connaissances du domaine selon un point de vue particulier. Heijst dit dans (Heijst, et al., 1997) que Les ontologies peuvent être classées selon deux dimensions : (i) la quantité et le type de structure de la conceptualisation ; et (ii) l'objet de la conceptualisation.

Trois catégories sont distinguées dans la première dimension :

- ***Ontologies terminologiques*** : tels que les lexiques, elles spécifient les termes qui sont utilisés pour représenter la connaissance dans le domaine du discours. Des exemples de telles ontologies dans le domaine médical est ICD[44] (International Classification of Diseases) et le réseau sémantique UMLS[45] (Unified Medical Language Système) (Bodenreider, 2004). Quant au domaine de la bioinformatique, il est très riche d'ontologies de cette classe. Il suffit de visiter le site Web de "The OBO Foundry"[46] pour une liste de tel type d'ontologies. A titre d'exemple, nous

[44] http://www.who.int/classifications/icd/en/

[45] http://www.nlm.nih.gov/research/umls/

[46] http://www.obofoundry.org/

citons plus particulièrement GO[47] (Gene Ontology) (Ashburner, et al., 2000) et CHEBI[48] (Chemical Entities of Biological Interest) (Degtyarenko, et al., 2008).

- ***Ontologies d'information*** : elles précisent la structure des enregistrements dans une base de données. Les schémas des Base de données sont un exemple de cette classe d'ontologies. Un exemple typique d'une telle ontologie dans le domaine médical est *Level 1* du modèle PEN&PAD (Rector, et al., 1993) qui propose un cadre de modélisation des enregistrements médicaux des patients
- ***Ontologies de modélisation des connaissances*** : Elles spécifient les conceptualisations des connaissances. Comparées aux ontologies d'information, les ontologies de modélisation des connaissances ont généralement une structure interne plus riche. En outre, ces ontologies sont souvent accordées à une utilisation particulière des connaissances qu'elles décrivent. Dans le cadre du développement des Systèmes de bases de connaissances, les ontologies de modélisation des connaissances sont les ontologies qui nous intéressent le plus. Un exemple dans le domaine bioinformatique est celui de l'ontologie TaO (Baker, et al., 1999) du système TAMBIS (Stevens, et al., 2000).

En se basant sur l'objet de la conceptualisation, on distingue les classes d'ontologies suivantes :

- ***Ontologie d'application*** : elle contient toutes les définitions nécessaires pour modéliser les connaissances requises pour une application particulière. En règle générale, les ontologies d'application sont un mélange de concepts qui sont prises à partir des ontologies de domaine et des ontologies génériques (qui sont décrites ci-dessous). Une ontologie de ce type est non réutilisable puisqu'elle est adaptée à une application.

  Une ontologie d'application permet aux experts du domaine d'utiliser le même langage que celui de l'application. Elle réalise trois objectifs :

  - faciliter le processus d'acquisition des connaissances de l'application ;
  - permettre l'intégration avec les serveurs existants dans l'environnement de l'application ;
  - sélectionner les structures de données appropriées au modèle computationnel.
- ***Ontologie de domaine*** : elle exprime des conceptualisations qui sont spécifiques à un domaine particulier. L'ontologie de domaine est constituée de la terminologie du domaine d'application et d'un ensemble de relations et de types de base comme la relation *classe/super-classe*. Une ontologie de domaine et alors constituée de :

[47] www.**geneontology**.org/

[48] www.ebi.ac.uk/**chebi**/

- une description en extension du vocabulaire du domaine,
- une typologie,
- une hiérarchie ou un treillis de classes.

Il faut noter que les méthodes d'ingénierie des connaissances établissent une distinction explicite entre les ontologies de domaine et les connaissances de domaine. Alors que les connaissances de domaine décrit des situations de fait dans un certain domaine, l'ontologie de domaine impose des contraintes sur la structure et le contenu des connaissances (Heijst, et al., 1997). Les concepts d'une ontologie de domaine sont souvent définis comme une spécialisation des concepts des ontologies génériques.

- ***Ontologie génériques*** : est similaire aux ontologies de domaine, mais les concepts qu'elle définit sont considérés comme génériques dans de nombreux domaines. En règle générale, les ontologies génériques définissent des concepts tels que l'état, événement, processus, action, composant, etc. Les concepts des ontologies de domaine sont souvent définis comme des spécialisations des concepts liés aux ontologies génériques. Elles sont prévues pour être utilisées dans des situations diverses, et pour servir une large communauté d'utilisateurs.
- ***Ontologie de représentation*** : elle spécifie un formalisme de description qui fournit une structure de représentation et des primitives pour décrire les concepts des ontologies de domaine et des ontologies génériques. Autrement dit, elle explique les conceptualisations qui sous-tendent les formalismes de représentation des connaissances (Davis, et al., 1993).

2.3.3 Rôle et utilisation d'ontologies

Développer une ontologie juste par plaisir n'a pas de sens en soit et c'est pourquoi cette démarche s'inscrit dans l'optique de répondre à un besoin spécifique (ou à un ensemble). Les auteurs de (Noy and Deborah, 2001) listent un ensemble de raisons qui nécessite le développement d'une ontologie :

- ***Partager la compréhension commune de la structure de l'information entre les personnes ou les fabricants de logiciels*** : c'est une des raisons les plus courantes qui conduit à développer des ontologies (Gruber, 1993; Musen, 1992). Par exemple, Les agents informatiques seront capables d'extraire et d'intégrer des informations d'un certain nombre de sites Web contenants des informations médicales, si ces sites partagent et publient tous la même ontologie qui est à la base des termes qu'ils utilisent. Les agents peuvent utiliser l'information intégrée pour pouvoir répondre aux interrogations des utilisateurs ou comme données d'entrée pour d'autres applications.
- ***Permettre la réutilisation du savoir du domaine*** : c'était une des raisons majeures qui ont poussé la recherche sur les ontologies ces dernières années. Lorsqu'un groupe de chercheurs développe une ontologie, les autres groupes peuvent la

réutiliser pour leurs propres domaines. Ainsi, s'il y a besoin de construire une ontologie plus large, il serait possible d'intégrer plusieurs ontologies existantes décrivant des portions d'un domaine. On peut, également, réutiliser une ontologie générale et l'étendre pour permettre de décrire un domaine d'intérêt spécifique. Un exemple dans le domaine médical est celui de l'ontologie OntoDPN. les auteurs de (Dhombres, et al., 2010) décrivent une modélisation ontologique pour le domaine du diagnostic prénatal qui a donné lieu à cette ontologie. Elle a pour premier objectif de représenter la sémiologie de l'imagerie prénatale. Cette sémiologie étant pour une large part morphologique (description de structures anatomiques du fœtus), Les auteurs ont donc choisi de réutiliser l'ontologie FMA (Foundational Model of Anatomy) (Rosse, et al., 1995) qui est une ontologie de référence de l'anatomie humaine.

- ***Rendre explicites les postulats d'un domaine*** sous-jacents à une implémentation rend possible la modification de ces spécifications, en cas d'évolution du savoir sur le domaine. Les postulats implicites sur un domaine exprimés en langage codé de programmation deviennent non seulement difficiles à deviner et comprendre mais, également difficiles à modifier, en particulier pour des personnes non expertes en programmation. Les spécifications explicites du savoir sur un domaine sont, de surcroît, utiles pour les nouveaux utilisateurs qui doivent apprendre la signification des termes du domaine.

- ***Distinguer le savoir sur un domaine du savoir opérationnel*** est une autre des finalités courantes des ontologies. Nous pouvons décrire la tâche de configuration d'un produit à partir de ses constituants, en respectant les spécifications requises et implémenter un programme qui réalisera cette configuration indépendamment des produits et de leurs composants (McGuinness and Wright, 1998).

- ***Analyser le savoir sur un domaine*** est possible dès que la spécification des termes du domaine est faite. L'analyse formelle des termes est extrêmement précieuse aussi bien quand on veut réutiliser les ontologies existantes, que quand on veut les étendre (McGuiness, et al., 2000). Souvent une ontologie de domaine n'est pas toujours un but en soi. Développer une ontologie s'apparente à définir un ensemble de données et leur structure pour qu'elles soient utilisées par d'autres programmes. Les ontologies et les bases de connaissances élaborées à partir des ontologies sont utilisées comme données par les méthodes de solutions de problèmes, les applications indépendantes des domaines et les fabricants de logiciels.

De façon plus générale, pour (Uschold and Grüninger, 1996), les avantages à tirer d'une ontologie peuvent être réparties en 3 catégories :

- ***Communication*** : en fixant un vocabulaire et la grammaire, elles permettent de répondre au besoin de communication entre personnes, personnes et systèmes et systèmes et systèmes ;

- ***Interopérabilité*** : en offrant un accès commun à l'information et une compréhension partagée des concepts, elles permettent de faciliter l'interopérabilité des systèmes et la réutilisation des sources de connaissances ;
- ***Amélioration logicielle*** : au niveau de la spécification (compréhension partagée des problèmes), fiabilité (tests de consistance [semi-]automatique) et de la réutilisabilité (fixe la structure des connaissances).

2.3.4 Langages d'ontologies

Une ontologie nécessite un formalisme pour la représenter. Ainsi plusieurs langages de représentation d'ontologies existent. Le langage utilisé pour décrire les termes d'une ontologie a un impact direct sur le niveau de formalisme de l'ontologie. Ainsi, on peut distinguer les ontologies informelles, qui utilisent le langage naturel et qui peuvent coïncider avec les terminologies*, les ontologies semi-formelles, qui fournissent une faible axiomatisation (ou formalisation), comme les taxonomies*, et enfin les ontologies formelles, qui font appel à un engagement sémantique plus fort en définissant la sémantique des termes par une axiomatisation complète et rigoureuse.

Différents langages de spécification d'ontologies issus des formalismes sont apparus à partir des années 1990. Ces langages sont classés par (Corcho and Gómez-Pérez, 2000) en deux grandes catégories en vue d'une comparaison de leur expressivité : *langages d'ontologie traditionnels* et *langages Web d'ontologies* (Figure 14). Dans la première catégorie on trouve Ontolingua (Gruber, 1992), OKBC (Chaudhri, et al., 1997), OCML (Motta, 1999), F-Logic (Kifer, et al., 1995) et LOOM (MacGregor, 1991). Pour une évaluation détaillée de ces langages, voir (Corcho and Gómez-Pérez, 2000; Su and Ilebrekke, 2002). Dans cette section, nous allons faire une revue des langages Web de représentation d'ontologies les plus utilisés vu leur orientation Web sémantique et leur recommandation par le W3C.

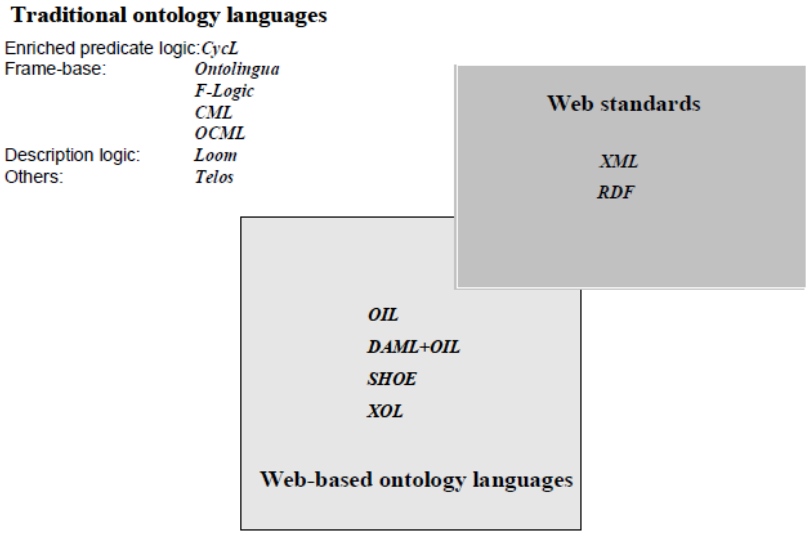

Figure 14. Classification des langages d'ontologie

2.3.4.1. RDF et RDFs

RDF (Resource Description Framework)[49] (Lassila and Swick, 1999) est une recommandation du W3C pour décrire des ressources. C'est un modèle de graphe qui permet d'encoder, d'échanger et de réutiliser les données et les métadonnées en permettant leur traitement automatisé. A l'origine, il a été défini pour décrire des ressources du Web telles que les pages Web, mais peut aussi gérer les métadonnées des documents XML et être utilisé pour des ontologies.

L'élément de base d'un document RDF est la ressource, correspondant à la représentation conceptuelle d'une entité qui peut être toute chose ayant une identité (objet physique, concept abstrait, etc.). L'identification d'une ressource se fait par un URI. Un document structuré en RDF est un ensemble de triplets qui se formalise sous forme d'une assertion ayant la forme <sujet S, prédicat P, objet O> qui signifie "*le sujet S a comme valeur pour le prédicat P l'objet O*". Par exemple, le sujet peut être un gène donné, le prédicat, une propriété de ce gène comme le nom, et l'objet, la valeur de cette propriété comme par exemple "TOP3". La Figure 15 donne la représentation graphique de l'exemple sous forme d'un triplet RDF.

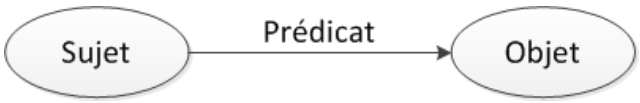

Figure 15. Triplet RDF.

RDF décrit alors les ressources en exprimant leurs propriétés et en leur attribuant des valeurs. Il utilise pour cela le vocabulaire défini par *RDF-Schema*[50] (Brickley and Guha, 2004). RDF-Schema fournit un ensemble de primitives simples, mais puissantes, pour la structuration des connaissances d'un domaine en classes et sous-classes, propriétés et sous-propriétés avec la possibilité de restreindre leur domaine d'origine et leur domaine d'arrivée (respectivement par le biais des expressions *rdf:domain* et *rdf:range*). Un exemple simple d'un RDF-Schema est présenté graphiquement dans la Figure 16.

Le pouvoir sémantique de ces deux langages est limité car les axiomes ne peuvent pas être directement décrits. Ainsi, RDF-Schema présente assez rapidement des limites lorsqu'il s'agit de son utilisation comme langage de représentation d'ontologies ayant de fortes contraintes (McBride, 2004).

[49] http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[50] http://www.w3.org/TR/rdf-schema/

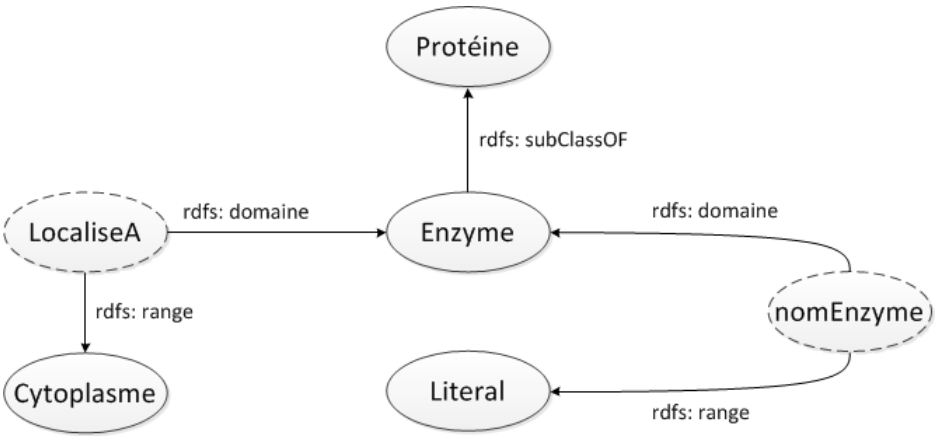

Figure 16. Une représentation graphique d'un exemple simple d'un RDF-Schema.

2.3.4.2. OIL

OIL (Ontology Inference Layer) (Fensel, et al., 2000) est à la fois un langage de représentation et d'échange d'ontologies. Il a été conçu pour fournir la plupart des primitives de modélisation couramment utilisés dans les ontologies qui se basent sur les Logiques de Description et les frames, avec une sémantique simple, propre et bien définie, et un support au raisonnement automatisé. Pour être utilisé sur le Web, il repose sur les standards RDF-Schema et XML. Dans OIL, une ontologie est une structure composée de plusieurs composants, organisés en trois couches: la couche objet (instances concrètes), la couche de premier méta-niveau (définition de l'ontologie) et la couche de second méta-niveau (définition des caractéristiques de l'ontologie). Concepts, relations, fonctions et axiomes peuvent être définies en utilisant des définitions ontologiques OIL. Les relations sont considérées comme des classes et peuvent être organisées hiérarchiquement.

2.3.4.3. DAML+OIL

Le langage DAML+OIL (Connolly, et al., 2001; McGuinness, et al., 2002)[51] est un langage conçu dans le but de dépasser la simple "présentation" d'informations sur le Web pour aller vers l'interopérabilité, la compréhension et le raisonnement sur ces informations. DAML+OIL est le résultat de la fusion de OIL et de DAML[52]. Il repose sur RDF et RDF schéma et fournit en plus des primitives plus riches issues de la logique de description. Les frames définis dans OIL ont été pour la plupart supprimées et remplacées par des assertions faites à l'aide d'un ensemble limité d'axiomes. Le résultat est que le langage est mieux adapté que RDF à l'utilisation et la maintenance d'ontologies mais présente des limites quant à la construction d'ontologie (Bechhofer, et al., 2001).

[51] http://www.w3.org/TR/daml+oil-reference

[52] http://www.daml.org/about.html

2.3.4.4. OWL

Comme nous avons cité précédemment, RDF(S) ne permet pas d'exprimer certaines notions que l'on voudrait décrire avec les ontologies. Dans le but d'étendre l'expressivité de RDF(S), OWL (Ontologie Web Language)[53] (McGuinness and Harmelen, 2004) a été proposé par W3C comme un standard pour représenter les ontologies. Il a été créé pour être utilisé par les applications cherchant à traiter le contenu de l'information et non plus uniquement à présenter l'information. OWL est une révision de DAML+OIL qui se veut plus représentatif du contenu du Web que XML, RDF et RDF-Schéma en apportant un nouveau vocabulaire avec une sémantique formelle pour décrire les propriétés et les classes, comme par exemple la disjonction des classes, la cardinalité, l'égalité, les types de propriétés plus riches, les caractéristiques de propriété (symétrie, transitivité, …) et les classes énumérées.

Le langage OWL a une expressivité très élevée et une complexité de raisonnement très grande. Afin d'avoir un compromis entre l'expressivité et la complexité de ce langage, OWL a été conçu comme une famille de trois sous-langages : OWL-Lite, OWL-DL et OWL-Full ayant une expressivité et une complexité croissantes :

- ***OWL-Lite*** : Il est fait pour des besoins préliminaires permettant de représenter des classifications sous forme hiérarchique et d'exprimer des contraintes simples (contraintes de cardinalité de type 0 ou 1). La disjonction de classes, la définition de classes à partir d'une union de classes, et bien d'autres possibilités offertes par OWL-DL, ne sont pas autorisées. Ce langage est particulièrement indiqué pour représenter des taxonomies et des thésaurus.

- ***OWL-DL*** : Il est appelé ainsi (OWL Description Logics) à cause de son origine, le formalisme de représentation des connaissances appelé Logiques de Description (Baader, et al., 2003). Cette logique appartient à un domaine de recherche qui a pour but d'aider au raisonnement sur une base de connaissances. Ainsi, OWL-DL en se basant aussi sur OWL-Lite offre une plus grande expressivité et décidabilité aux systèmes de raisonnement.

  Une ontologie exprimée en OWL-DL peut être vue comme une *TBox*[54] dans les Logiques de description, avec une relation hiérarchique décrivant le domaine en termes de classes (correspondant aux concepts), et des propriétés (correspondant aux relations). Chaque élément de l'ontologie appartient à une seule catégorie, c'est-à-dire qu'un même objet ne peut être défini à la fois comme une classe et comme une propriété. OWL-DL a l'avantage de fournir un support pour les inférences, mais la compatibilité totale avec RDF/RDFS est perdue.

[53] http://www.w3.org/TR/owl-features/

[54] Dans les logiques de description, Une base de connaissances K est une paire (T, A), où T est un ensemble d'axiomes «terminologiques" (appelé TBox) et A un ensemble d'axiomes "assertionnels" (appelé ABox).

- ***OWL-Full*** : se base aussi sur OWL-Lite. Il utilise tous les éléments disponibles en OWL pour permettre un maximum d'expressivité. Il permet aussi de combiner de façon quelconque ces éléments avec RDF et RDFS ce qui permet donc d'augmenter le sens du vocabulaire prédéfini (en OWL ou RDF). OWL-Full est totalement compatible avec RDF; ainsi tout document RDF valide est aussi un document OWL-Full valide. En levant les contraintes imposées par OWL-DL, il rend certaines valeurs disponibles et utilisables dans des bases de données ou de connaissances, mais il ne supporte pas les raisonnements liés à la logique de description. Il n'impose pas de séparation entre classe, propriété, individu et valeur des données. Cependant, son inconvénient principal vient de son pouvoir d'expression élevé, qui le rend indécidable.

Il y a une stricte compatibilité ascendante de ces trois langages : toute ontologie OWL-Lite valide est une ontologie OWL-DL valide, et toute ontologie OWL-DL valide est une ontologie OWL-Full valide. Ainsi, toute conclusion d'une ontologie OWL-Lite est une conclusion valide de OWL-DL, et toute conclusion OWL-DL est une conclusion valide de OWL-Full (Figure 17).

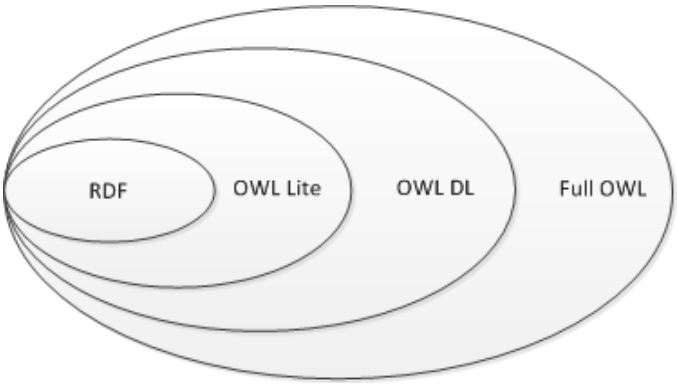

Figure 17. Hiérarchie de langage OWL.

2.4. Service Web, outil de communication du Web Sémantique

Il s'agit d'une technologie permettant à des applications de dialoguer à distance via Internet, et ceci indépendamment des plates-formes et des langages sur lesquelles elles reposent. Pour ce faire, les services Web s'appuient sur un ensemble de protocoles Internet très répandus (XML, HTTP), afin de communiquer. Cette communication est basée sur le principe de requête et réponses, effectuées avec des messages XML.

W3C définit un service Web comme suit (Haas and Brown, 2004) :

> "*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its*

description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

La notion de "service Web" désigne essentiellement une application (un programme) mise à disposition sur Internet par un fournisseur de service, et accessible par les clients à travers des protocoles Internet standards (Casati and Shan, 2001; Fensel, et al., 2002). Il existe probablement plusieurs définitions des services Web mais presque toutes ont ceci en commun :

- les services Web proposent aux utilisateurs du Web des fonctionnalités pratiques grâce à un protocole Web standard.
- les services Web offrent un moyen de décrire leurs interfaces suffisamment en détail pour permettre à un utilisateur de créer une application cliente capable de converser avec eux.
- les services Web sont inscrits afin que les utilisateurs potentiels puissent les trouver facilement.

Un service Web est donc une application ou un composant logiciel qui vérifie les propriétés suivantes :

- Il est identifié par un URI ;
- Ses interfaces et ses liens (binding) peuvent être décrits en XML ;
- Sa définition peut être découverte par d'autres services Web;
- Il peut interagir directement avec d'autres services Web à travers le langage XML et en utilisant des protocoles Internet.

Techniquement, un service Web peut être perçu comme étant une interface décrivant une collection d'opérations accessibles via le réseau à travers des messages XML standardisés. La description d'un service Web inclut tous les détails nécessaires à l'interaction avec le service comme, par exemple, le format des messages, les signatures des opérations, le protocole de transport et la localisation du service. Les services Web s'appuient sur des mécanismes et des protocoles standards et sont donc indépendants des langages de programmation (Java, J#, C++, Perl, C#, etc.), du modèle objet (COM, EJB, etc.) ainsi que des plates-formes d'implémentation (J2EE, .NET, etc.).

L'objectif ultime de l'approche des services Web est de transformer le Web en un dispositif distribué de calcul où les programmes (services) peuvent interagir de manière intelligente en étant capables de se découvrir automatiquement, de négocier entre eux et de se composer en des services plus complexes (Fensel, 2002; McIlraith, et al., 2001). En d'autres termes, l'idée poursuivie avec les services Web, est de mieux exploiter les technologies de l'Internet en substituant, autant que possible, les humains qui réalisent actuellement un certain nombre de services (ou tâches), par des machines en vue de permettre une découverte et/ou une composition automatique de services sur l'Internet.

2.4.1. Architecture de référence d'un service Web

L'architecture de référence des services Web (Figure 18) s'articule autour des trois rôles suivants :

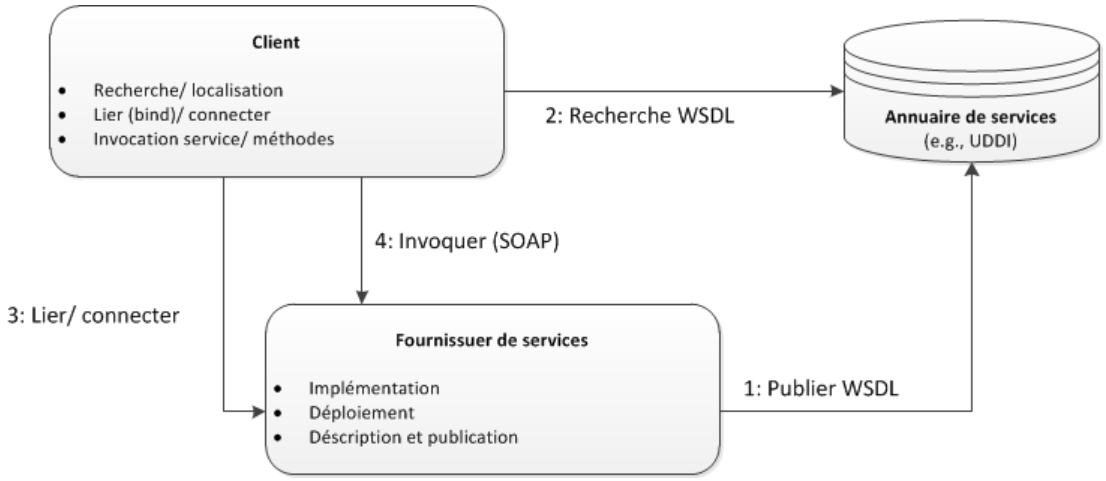

Figure 18. Architecture des services Web.

- ***Le fournisseur de service*** : correspond au propriétaire du service. D'un point de vue technique, il est constitué par la plate-forme d'accueil du service.
- ***Le client*** : correspond au demandeur de service. D'un point de vue technique, il est constitué par l'application qui va rechercher et invoquer un service. L'application cliente peut être elle-même un service Web.
- ***L'annuaire des services*** : correspond à un registre de descriptions des services offrant des facilités de publication de services à l'intention des fournisseurs ainsi que des facilités de recherche de services à l'intention des clients.

Les interactions de base entre ces trois rôles incluent les opérations de publication, de recherche et de liens (bind) d'opérations. Ainsi, le fournisseur de services définit la description de son service et la publie dans un annuaire de service. Le client utilise les facilités de recherche disponibles au niveau de l'annuaire pour retrouver et sélectionner un service donné. Il examine ensuite la description du service sélectionné pour récupérer les informations nécessaires lui permettant de se connecter au fournisseur du service et d'interagir avec l'implémentation du service considéré.

Pour garantir l'interopérabilité des trois opérations précédentes (publication, recherche et lien), des propositions de standards ont été élaborées pour chaque type d'interactions. Nous citons, notamment les standards suivants :

- ***SOAP*** définit un protocole de transmission de messages basé sur XML.
- ***WSDL*** introduit une grammaire commune pour la description des services.

- ***UDDI*** fournit l'infrastructure de base pour la publication et la découverte des services.

Les services Web sont décrits par des documents WSDL, qui précisent les méthodes pouvant être invoquées, leurs signatures et les points d'accès du service (URL, port). Les services Web sont accessibles via SOAP, la requête et les réponses sont des messages XML transportés sur HTTP.

L'infrastructure de base autour de ces standards répond aux problèmes d'intégration technique des applications. En effet, contrairement aux approches d'intégration classiques qui ne sont pas exemptes d'inconvénients (Stal, 2002), les services Web proposent une approche flexible et "universelle" pour l'intégration de systèmes hétérogènes en s'appuyant sur un modèle d'intégration basé sur un couplage faible des composants (peer-to-peer) et en exploitant de manière intensive les standards du Web. Ceci a pour effet de permettre une intégration des applications plus rapide et moins coûteuse et avec des perspectives d'évolution et de réutilisation réelles.

2.4.2. SOAP

SOAP (Simple Object Access Protocol)[55] est un protocole de RPC (Remote Procedure Call)[56] (Birrell and Nelson, 1984) permettant d'invoquer des méthodes d'objets distantes. C'est un protocole standardisé permettant à une application d'envoyer et de recevoir des messages, de types requêtes-réponses, via l'Internet (Gudgin, et al., 2007). Il est comparable à DCOM (Distributed Component Object Model)[57] mais contrairement à celui-ci, il s'appuie sur des standards très connus. Il utilise XML pour définir les fonctions et les définitions disponibles. Il prend en charge divers protocoles de transport, tels que HTTP (HyperText Transfer Protocol)[58] et SMTP (Simple Mail Transfer Protocol)[59], ainsi que différents formats comme MIME (Multipurpose Internet Mail Extensions)[60]. Ces derniers sont très répandus sur de multiples plates-formes, ce qui donne à SOAP une grande portabilité et interopérabilité.

SOAP est une spécification non propriétaire. Il n'est pas lié à un protocole particulier. Il n'est pas non plus lié à un système d'exploitation ni à un langage de programmation. SOAP étant un protocole d'échange d'informations entre diverses machines sur un réseau, elle

[55] http://www.w3.org/TR/soap12-part1/

[56] C'est un protocole réseau permettant de faire des appels de procédures sur un ordinateur distant à l'aide d'un serveur d'applications. Ce protocole est utilisé dans le modèle client-serveur pour assurer la communication entre le client, le serveur et des éventuels intermédiaires.

[57] C'est une technique propriétaire de Microsoft qui permet la communication entre des composants logiciels distribués au sein d'un réseau informatique.

[58] C'est un protocole de communication client-serveur développé pour le World Wide Web.

[59] C'est un protocole de communication utilisé pour transférer le courrier électronique (courriel) vers les serveurs de messagerie électronique.

[60] C'est un standard internet qui étend le format de données des courriels pour supporter des textes en différents codage de caractères autres que l'ASCII, des contenus non textuels, des contenus multiples, et des informations d'en-tête en d'autres codages que l'ASCII.

nécessite un format pour transporter les données. Pour cela elle utilise des messages SOAP qui sont en fait des documents XML.

SOAP est un ensemble de conventions qui spécifient un format de message avec un ensemble de règles qui régissent le traitement du message. Les conventions décrivent comment un message est assemblé et quelles interactions peuvent accomplir les nœuds SOAP traitant les messages.

Un message SOAP est l'unité de base de la communication entre les nœuds SOAP. Les messages SOAP sont structurés en un document XML qui comporte une enveloppe et un corps. Il se compose d'une enveloppe qui contient zéro ou plusieurs en-têtes SOAP. Les en-têtes SOAP sont destinés à un récepteur de message. L'enveloppe contient également un corps qui contient la charge utile du message. Un corps SOAP peut contenir, par exemple, une demande du service et des données d'entrée exigées pour le traitement du service. Lors du traitement d'un message SOAP, un nœud SOAP pourrait générer un état d'erreur (ou de faute). Si cela se produit, le nœud renvoie un message contenant une erreur SOAP. Ces éléments imbriqués sont illustrées graphiquement dans la Figure 19 et un exemple simple où on y retrouve bien ces éléments et donné dans la (Figure 20).

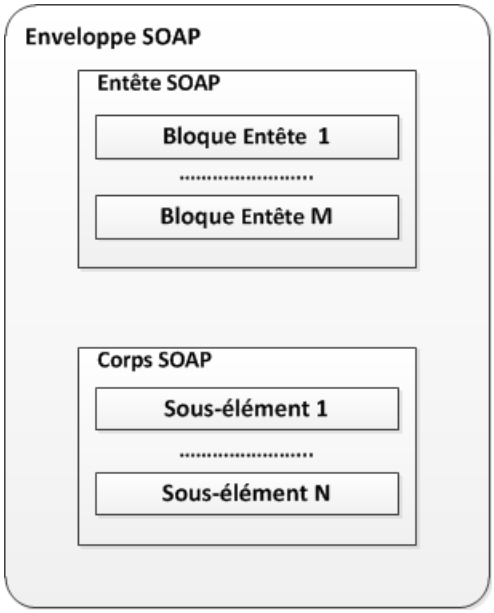

Figure 19. Représentation graphique des éléments d'un message SOAP.

Un nœud SOAP est une implémentation des règles du traitement décrit dans la spécification SOAP qui peut transmettre, recevoir, traiter ou retransmettre un message. Bien que le nœud SOAP implémente le modèle SOAP de traitement, il peut également accéder à tout service que pourraient fournir les protocoles sous-jacent de réseau. Il le fait à travers une liaison SOAP qui spécifie les règles pour le transport de message SOAP sur certains autres protocoles sous-jacents du réseau. Ces protocoles de transport peuvent être

d'autres standards, tels que les protocoles HTTP, SMTP ou TCP (Transmission Control Protocol)[61].

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <pns:qualityOfService xmlns:pns="http://example.org/qos">
      <pns:priority>3</pns:priority>
      <pns:timestamp>2004-02-25T01:00:00-00:00</pns:timestamp>
      <pns:persist>true</pns:persist>
    </pns:appHeaderBlock>
  </env:Header>
  <env:Body>
    <bmns:businessPO xmlns:env="http://example.org/po">
      <bmns:description>Widgets</bmns:description>
      <bmns:quantity>100</bmns:quantity>
      <bmns:price>20.5</bmns:price>
    </bmns:businessPO>
  </env:Body>
</env:Envelope>
```

Figure 20. Un exemple simple d'un message SOAP.

Les nœuds SOAP peuvent envoyer et recevoir des messages SOAP. Si un nœud SOAP transmet un message, il est appelé un émetteur SOAP ; Si elle reçoit un message, il est appelé un receveur SOAP. Certains nœuds SOAP pourraient à la fois recevoir et transmettre des messages. Dans ce cas, ils sont appelés intermédiaires SOAP. L'émetteur SOAP qui crée tout d'abord le message SOAP est appelé l'émetteur SOAP initial. La destination finale du message est appelée receveur SOAP final. Ce nœud SOAP est responsable du traitement de la charge utile du message qui est contenue dans le corps SOAP.

2.4.3. WSDL

WSDL (Web Services Description Language) (Christensen, et al., 2001) est une grammaire XML utilisé pour décrire les services Web. Elle définit comment un service Web est accessible ainsi que les opérations qu'il exécute, il montre comment les messages sont transmis et structurés. Bien qu'il n'est pas nécessaire que WSDL travaille avec SOAP, il est une partie intégrante de la WS-I Basic Profile[62] de l'organisation de l'interopérabilité des services Web[63], et il fait travailler avec SOAP beaucoup plus facile.

La version initial du WSDL (la version 1.1) a été publié par le W3C en Mars 2001 comme une Note (Christensen, et al., 2001). Son successeur (la version 2.0)[64] est une recommandation W3C depuis 2007 (Chinnici, et al., 2007). WSDL spécifie ce que doit contenir un message SOAP et l'apparence du message de réponse dans une notation sans

[61] TCP est un protocole orienté connexion, c'est-à-dire qu'il permet à deux machines qui communiquent de contrôler l'état de la transmission entre elles.

[62] http://ws-i.org/Profiles/BasicProfile-1.2-2010-11-09.html

[63] http://www.ws-i.org/

[64] http://www.w3.org/TR/wsdl20/

ambiguïté. Le fait que la notation utilisée est basée sur la norme du schéma XML, signifie que WSDL est à la fois neutre par rapport au langage de programmation et à la plateforme.

Outre la description du contenu des messages, WSDL définit l'endroit où le service est disponible et le protocole de communication utilisé pour converser avec le service. Cela signifie que le fichier WSDL définit tout ce qui est nécessaire pour écrire un programme fonctionnant avec un service Web.

Les documents WSDL définissent les services comme des collections de points finaux du réseau ou des ports. Dans WSDL, la définition abstraite des points finaux et des messages est séparée de l'installation concrète du réseau ou des liaisons du format de données. Ceci permet de réutiliser les définitions abstraites: les messages qui sont des descriptions abstraites des données échangées et des types de ports, qui sont des collections abstraites d'opérations. Les spécifications concrètes du protocole et du format de données pour un type de port particulier constituent une liaison réutilisable. Un port est défini par l'association d'une adresse de réseau et d'une liaison réutilisable, et une collection de ports définit un service. Pour cette raison, un document WSDL utilise les éléments suivants dans la définition des services Web :

- ***Types*** : conteneur de définitions du type de données qu'utilise un schéma de types (par exemple XSD).
- ***Message***: définition abstraite et écrite des données qui se communiquent.
- ***Operation*** : description abstraite d'une action admise par le service.
- ***Port Type*** : ensemble abstrait des opérations admises par un ou plusieurs points finaux.
- ***Binding*** : spécification du protocole et du format de données pour un type de port déterminé.
- ***Port*** : point final unique qui se définit comme la combinaison d'une liaison et d'une adresse réseau.
- ***service*** : Une collection de points finaux connexes.

En résumé WSDL peut être considéré comme un contrat entre un client et un serveur qui fait état :

- des spécifications d'interfaces qui décrivent toutes les méthodes publiques,
- des spécifications relatives aux types de donnée de messages mis en œuvre dans les questions-réponses,
- des informations liées au protocole de transport utilisé,
- des informations d'adresse permettant de localiser le service décrit.

En un mot, WSDL définit le contrat existant entre un client et un serveur sans dépendance particulière pour une plateforme ou un langage. La (Figure 21) montre le

contenu du fichier WSDL utilisé par *YeastMed* (voir chapitre 4) pour convoquer le service Web de la source de données SBD.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
   targetNamespace="http://150.214.214.1/SGD/services/SGD"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://150.214.214.1/SGD/services/SGD"
   xmlns:intf="http://150.214.214.1/SGD/services/SGD"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->

   <wsdl:message name="getDataProvenanceRequest"> [1 line]
   <wsdl:message name="getSchemaResponse"> [2 lines]
   <wsdl:message name="getDataProvenanceResponse"> [2 lines]
   <wsdl:message name="QueryRequest">
      <wsdl:part name="XQuery" type="xsd:string"/>
   </wsdl:message>
   <wsdl:message name="QueryResponse">
      <wsdl:part name="QueryReturn" type="xsd:string"/>
   </wsdl:message>
   <wsdl:message name="getSchemaRequest"> [1 line]
   <wsdl:portType name="Service">
      <wsdl:operation name="getSchema"> [3 lines]
      <wsdl:operation name="Query" parameterOrder="XQuery">
         <wsdl:input message="impl:QueryRequest" name="QueryRequest"/>
         <wsdl:output message="impl:QueryResponse" name="QueryResponse"/>
      </wsdl:operation>
      <wsdl:operation name="getDataProvenance"> [3 lines]
   </wsdl:portType>
   <wsdl:binding name="SGDSoapBinding" type="impl:Service">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getSchema"> [8 lines]
      <wsdl:operation name="Query">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="QueryRequest">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                           namespace="http://sgdPackage" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="QueryResponse">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                           namespace="http://150.214.214.1/SGD/services/SGD" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getDataProvenance"> [8 lines]
   </wsdl:binding>
   <wsdl:service name="ServiceService">
      <wsdl:port binding="impl:SGDSoapBinding" name="SGD">
         <wsdlsoap:address location="http://150.214.214.1/SGD/services/SGD"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Figure 21. Le contenu du fichier WSDL utilisé par *YeastMed* pour convoquer le service Web de la source de données SGD.

2.4.4. UDDI

Un service Web peut être publié d'une manière à ce que les clients intéressés puissent facilement le découvrir et l'utiliser dans leurs applications. Il existe déjà un mécanisme de découverte qui répond à ces exigences: UDDI (Universal Description Discovery and Integration), une initiative de l'industrie pour rendre compatible la découverte des services Web avec toutes sortes de technologies et plates-formes.

UDDI est un registre public destiné à stocker des informations d'une manière structurée sur les fournisseurs et les services Web qu'ils offrent. Il contient des informations sur les interfaces techniques des services d'un fournisseur et sert d'infrastructure pour une collection de logiciels basés sur des services Web.

Pourquoi UDDI? Pourquoi s'avère nécessaire un registre de ce type?

Étant donné qu'il existe une collection de logiciels de milliers (voire des millions) de services Web, les utilisateurs de ces services sont confrontés à plusieurs questions difficiles:

- Comment trouver les services Web sollicités?
- Comment se catégorisent les informations sur les services Web de façon qui rend leurs exploitations plus facile?
- Comment cela peut affecter le processus de la localisation des services?
- Comment assurer l'interopérabilité des mécanismes de la découverte?
- Comment interagir en temps d'exécution avec le mécanisme de la découverte quand une application dépend d'un service Web?

L'initiative UDDI surgi comme réponse à ces questions. Plusieurs entreprises, dont Microsoft, *IBM*, *Sun*, *Oracle*, *Compaq*, *Hewlett Packard*, *Intel*, *SAP* et quelque trois cents de plus ont uni leurs forces pour développer une spécification basée sur des standards libres et des technologies no propriétaires qui permettrait de régler les problèmes ci-dessus. Le résultat, dont la version bêta a été lancé en Décembre 2000 et a été en production en Mai 2001, était un registre global hébergé par des nœuds multiples d'opérateurs où les utilisateurs peuvent rechercher et publier des services Web sans frais.

Depuis la création de cette infrastructure pour les services Web, les données sur ces services peuvent être trouvées d'une forme systématique, fiable et totalement indépendante des fournisseurs. Il est possible d'effectuer des recherches catégoriques précises à l'aide des systèmes d'identification et taxonomiques extensibles. L'intégration de UDDI en temps d'exécution peut être incorporées dans des applications. En conséquence, il s'est encouragé le développement d'un environnement pour les services Web.

UDDI est relativement léger, est conçu comme un registre, et non comme un dépôt. La différence, bien que subtile, est essentielle. Un registre redirige l'utilisateur à des ressources, tandis que les dépôts stockent seulement des informations.

UDDI se base sur des identificateurs globaux uniques (GUID) pour assurer la capacité de rechercher et d'identifier l'emplacement des ressources. En fin de compte, les requêtes à

UDDI conduisent à une interface (un archive .WSDL, .XSD, .DTD, etc.) ou à une implémentation (comme un fichier .ASMX ou .ASP) situé sur un autre serveur. Par conséquent, UDDI peut répondre à ce type de questions:

- "Quelles sont les interfaces des services Web basées sur WSDL qui sont publiés et mises en place pour un secteur particulier ?"
- "Quels sont les services Web, classés en quelque sorte, qui sont actuellement offerts ?"
- "Quels sont les services Web fournit par un fournisseur de services Web particulier?"
- "Avec qui doit communiquer l'utilisateur afin d'utiliser un service Web particulier ?"
- "Quels sont les détails de l'implémentation d'un service Web particulier ?"

Le but fonctionnel d'un registre UDDI est la représentation des données et des métadonnées sur les services Web. Un registre, que ce soit pour une utilisation sur un réseau public ou dans une infrastructure interne d'une organisation, offre un mécanisme basé sur des standard pour classer, cataloguer et gérer des services Web, de sorte qu'ils puissent être découverts et consommé par d'autres applications.

En conséquence, le standard spécifie les protocoles d'accès à un registre de services Web, les méthodes de contrôle de l'accès à la base de registre, et un mécanisme pour distribuer ou déléguer des enregistrements à d'autres registres. En d'autres termes, le standard fournit un moyen pour localiser un service logiciel, invoquer ce service, et gérer les métadonnées relatives à ce service.

Pour atteindre ces objectifs, les principaux concepts fonctionnels pour travailler avec UDDI comprennent:

- **Le modèle de données UDDI.** La spécification UDDI définit les types de données de base qui comprennent une description de la fonction de l'activité du service, des informations sur l'éditeur du service, les détails techniques du service et de l'API, et d'autres métadonnées. Ces types de données sont définis dans plusieurs schémas XML, qui forment ensemble un modèle d'information de base et un cadre d'interaction des registres UDDI. Ils incluent:
    - Une description de la fonction de l'activité d'un service (appelée *businessService*)
    - Informations sur l'organisation qui a publié le service (*businessEntity*)
    - Détails techniques du service (*bindingTemplate*), y compris une référence à l'interface de programmation du service ou API, et
    - Divers autres attributs ou métadonnées telles que la taxonomie, les transports, les signatures numériques, etc. (*tModels*)
    - Les relations entre les entités dans la base de registre (publisherAssertion)

- Demandes permanents pour suivre l'évolution de la liste des entités (abonnement)

- **Définir les nœuds UDDI et les registres.** UDDI inclut une définition spécifique de la relation hiérarchique entre une instance d'une application UDDI et d'autres auxquels elle est liée. Techniquement, il existe trois grandes catégories de serveurs UDDI:
  - Un noeud est un serveur UDDI qui prend en charge au moins un ensemble minimum de fonctionnalités définies dans la spécification. Il peut effectuer une ou plusieurs fonctions sur les données UDDI auxquelles il a accès. Il est un membre d'exactement un registre UDDI.
  - Un registre est constitué d'un ou plusieurs noeuds. Un registre effectue l'ensemble complet de fonctionnalités telles que définies dans la spécification.
  - Les registres affiliés sont des registres UDDI individuels qui implémentent le partage de l'information entre eux. Les registres affiliées partagent un espace de nommage commun pour les clés UDDI qui identifient d'une manière unique les enregistrements de données.

- **Interfaces de programmation essentielles.** Un registre UDDI fournit plusieurs fonctions clés, notamment:
  - La publication d'informations sur un service à un registre
  - Recherche dans un registre UDDI pour des informations sur un service

Ces fonctions d'enquête et de publication représentent les principaux outils de gestion des données d'un registre UDDI. En outre, UDDI décrit comment plusieurs registres peuvent former un groupe, connu sous le nom d'une affiliation, afin de permettre l'élaboration de la copie des structures de données de base parmi eux. Quelques-uns des concepts les plus importants que soutient l'interaction des registres sont les suivants:

- Réplication et transfert de la sauvegarde des données sur un service.
- La gestion et la génération de la clé d'enregistrement.
- Ensemble d'API de souscription d'enregistrement.
- La sécurité et l'autorisation.

La spécification UDDI divise ces fonctions en "*Node API Sets*" qui sont pris en charge par un serveur UDDI et "*Client API Sets*" qui sont pris en charge (naturellement) par un client UDDI.

Bien entendu, on peut proposer un service Web sans l'inscrire, mais si on souhaite toucher un public plus important, UDDI permet aux clients éventuels de trouver le service.

UDDI (dans sa version 3)[65] est un standard OASIS depuis 2005 (Bellwood, et al., 2005) .

2.5. Conclusion

L'ambition du Web sémantique, développé sous l'égide du W3C, est de développer un ensemble de technologies visant à décrire et exploiter de manière systématique la sémantique des ressources du Web. Un grand nombre de standards et de technologies ont été développés par le W3C, avec pour objectif de sortir les données des silos fermés que constituent les bases de données en ligne. Ces technologies permettent la création des données, l'expression des vocabulaires et des règles qui les décrivent, et la construction des systèmes capables de les manipuler dans de bonnes conditions d'interopérabilité.

Les systèmes d'intégration de données tirent avantage des technologies du Web sémantique pour résoudre le problème de la diversité et l'hétérogénéité que présentent les sources de données distantes lors de leur exploitation. Ces systèmes s'occupent de l'uniformisation de l'accès à une grande quantité de données variées et dispersées. Dans le chapitre suivant, nous allons parler des caractéristiques des sources de données biologiques qui rendent l'intégration de données une exigence. Nous allons détailler aussi les processus de l'intégration de données ainsi que les différents types des systèmes proposées.

[65] http://uddi.org/pubs/uddi_v3.htm

Deuxième Partie

INTÉGRATION DES DONNÉES BIOLOGIQUES DE LA LEVURE

Chapitre 3

KOMF : UNE ARCHITECTURE DE MEDIATION SEMANTIQUE

Sommaire

3.1. Introduction

Après un état de l'art, présenté dans la première partie de ce mémoire, sur les systèmes d'intégration de données et les technologies du Web Sémantique, la deuxième partie sera consacrée à la contribution de cette thèse : la description d'un prototype d'un système d'intégration de données de la levure *Saccharomyces cerevisiae*.

Dans ce chapitre, nous allons détailler l'architecture de l'infrastructure de médiation à base d'ontologie sur laquelle est fondé le système *YeastMed* qui constituera le sujet du chapitre suivant. Cette infrastructure se nomme KOMF (Khaos Ontology-based Mediation

Framework). Elle a pour objectif la fourniture d'un accès aux données en utilisant un modèle de données et un langage de requête communs. Cette architecture fournit une représentation d'un modèle sémantiquement cohérent des données combinées à partir des sources de données intégrées et un accès transparent à travers des vues sur les données. Deux composants principales dans cette architecture : un Répertoire Sémantique et un médiateur appelé SB-KOM. Le Répertoire Sémantique se charge principalement du stockage et de la gestion des sémantiques et des métadonnées des ressources de données manipulées par *YeastMed*. SB-KOM est un médiateur qui se charge du traitement des requêtes soumises au système en tirant avantage des sémantiques du répertoire Sémantique.

Ce chapitre sera organisé comme suit : dans la section suivante, nous allons décrire le Répertoire Sémantique et le rôle joué par chacun de ses composants. Puis, les fonctionnalités du médiateur SB-KOM seront détaillées dans la section 3.3.

3.2. Répertoire Sémantique

Les applications du Web sémantique font appel souvent à des ontologies pour introduire de la sémantique et annoter des ressources de données. Des composants se chargeant de la gestion de ces ontologies, des métadonnées des ressources et des relations entre ces derniers et les ontologies, est une nécessité pour la plupart des applications du Web sémantique. L'architecture du KOMF discutée dans ce chapitre se base en premier lieu sur une infrastructure jouant le rôle d'un middleware prenant en charge cette fonctionnalité. Elle est appelée *répertoire sémantique*. Il s'agit d'une infrastructure où les ressources Web sont enregistrées et leurs sémantiques publiées. Faire appel à ce composant pour publier les sémantiques des ressources permet leurs interopérabilités avec d'autres applications que ça soit dans le même domaine ou dans un autre. Le Répertoire Sémantique présente un certain nombre de caractéristiques :

- Il permet de fournir un ensemble minimum de composants nécessaire pour le développement d'applications dans le domaine du Web sémantique.
- En se basant sur cette infrastructure comme une structure sémantique commune dans le développement d'applications du Web sémantique, rend possible leurs interconnexion et leurs interopérabilité.
- Le Répertoire Sémantique est extensible; c'est à dire elle donne la possibilité de greffer sur lui d'autres composants ajoutant de nouvelles fonctionnalités. Ceci conduit à la construction d'applications plus complexes.
- Le fait que les sémantiques du répertoire sont explicitement et publiquement disponibles à travers trois interfaces, donne la possibilité aux applications du Web sémantique d'interopérer avec elles.

Le Répertoire Sémantique (**Erreur ! Source du renvoi introuvable.**) est formé d'un répertoire de données appelé *SD-Data* (Semantic Directory Data) et d'un noyau appelé *SD-Core* (Semantic Directory Core). SD-Data permet la description des sémantiques internes du

Répertoire Sémantique à travers deux ontologies alors que SD-Core fournit des composants utiles pour enregistrer et gérer des ontologies ainsi que leurs relations avec les sources de données. Il permet aussi l'enregistrement des métadonnées sur les sources annotées par les ontologies enregistrées.

3.2.1. SD-Data

SD-Data a pour but la manipulation des sémantiques des ressources de données. Il inclut deux ontologies inter-reliées : *OMV* (Ontology Metadata Vocabulary) (Hartmann, et al.) dans sa version 0.9.1 et *SDMO* (Semantic Directory Metadata Ontology). Ces ontologies décrivent les sémantiques dans le Répertoire Sémantique. L'utilisation des ontologies pour représenter les métadonnées tire son avantage du fait que celles-ci peuvent être gérer par des outils allants des parseurs simples d'ontologie à des raisonneurs d'ontologie plus complexes.

OMV est utilisée pour enregistrer des informations sur les ontologies dans le but de permettre aux utilisateurs de les localiser et de les utiliser facilement. Ce système des métadonnées contient assez d'éléments pour décrire les différents aspects concernant la création, la gestion et l'utilisation d'une ontologie (Figure 22).

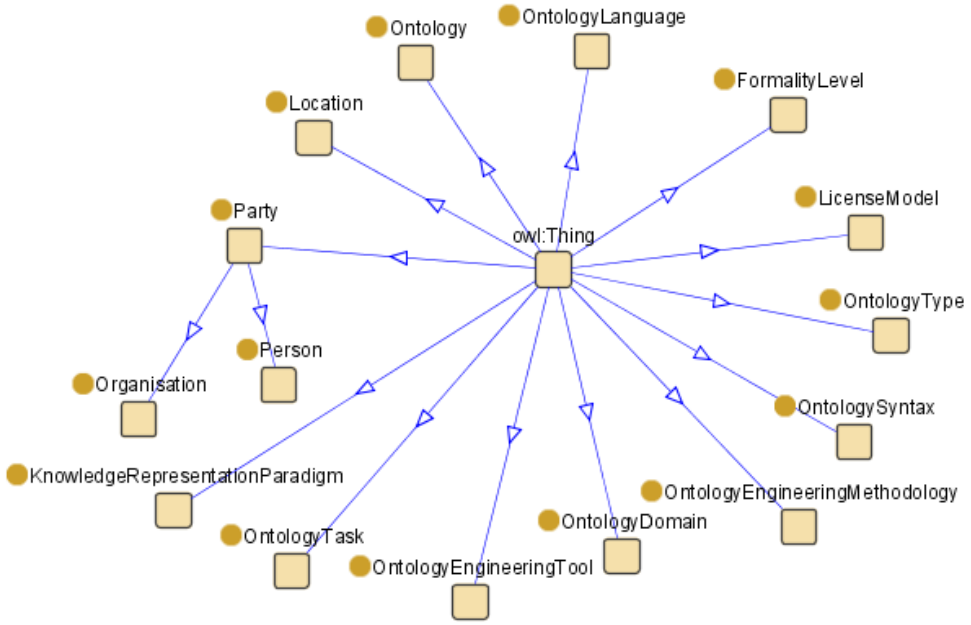

Figure 22. Ontologie OMV (version 2.4).

La version de l'ontologie OMV utilisée par SD-Data est la version 0.9.1. Elle contient une classe appelée *OntologyImplémentation*. C'est cette classe qui joue le plus grand rôle dans le stockage des métadonnées des ontologies enregistrées dans le répertoire sémantique. OntologyImplémentation a plusieurs propriétés pour la description des ontologies :

- *implementationName* : elle définit le nom de l'ontologie. Sa valeur est retrouvée à partir des paramètres de la méthode d'enregistrement. Deux ontologies différentes peuvent avoir le même nom.

- *ontologyURL* : cette propriété permet de donner une URL à l'ontologie. Cette information est délivrée durant le processus de l'enregistrement et sera utilisée pour distinguer deux ontologies différentes.
- *numIndividuals* : elle définit le nombre des instances que l'ontologie possède dans sa définition. Cette information est obtenue directement de l'ontologie.
- *implementationAcronym* : cette propriété permet de définir un acronyme à l'ontologie.
- *Imports* : c'est une propriété qui permet d'établir des références à d'autres ontologies. Cette information est obtenue lors du processus d'enregistrement et permet de trouver des relations explicites entre différentes ontologies.
- *Rdfs :comment* : elle stocke des informations supplémentaire qui ne peuvent pas être classées en utilisant les autres propriétés.

SDMO est l'ontologie qui se charge de l'enregistrement des informations sur les ressources de données et les relations entre elles et des ontologies enregistrées dans le répertoire sémantique. SDMO et OMV sont reliées par une classe (ou concept) appartenant à la première et qui permet de relier des ressources (SDMO instances) à des ontologies (OMV instances). Dans sa version utilisée par SD-Data, SDMO est composée de cinq classes (Figure 23):

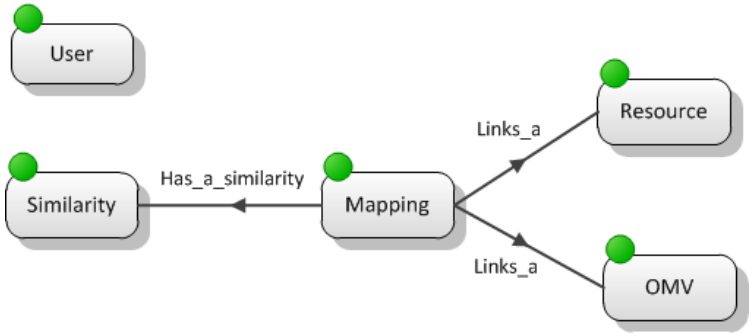

Figure 23. Semantic Directory Metadata Ontology (SDMO).

- *OMV* : c'est un concept utilisé pour relier des ressources à des ontologies (définis comme des instances de l'ontologie OMV). Il contient le nom et l'URL de l'ontologie. L'enregistrement d'une ontologie implique la création d'une instance de cette classe.
- *Resource* : cette classe est utilisée pour stocker des informations (nom, URI, schéma de la source, interface d'interrogation et les capacités d'interrogation) sur les ressources.
- *Similarity* : cette classe contient 3 propriétés (*concept1*, *concept2* et *similarityValue*) pour représenter une relation de similitude entre une classe d'ontologie et un élément du schéma d'une ressource.

- *Mapping* : elle permet de spécifier les relations entre les ressources et les ontologies enregistrées. Chaque mapping est reliée à une instance de type *Similarity* qui définit les similitudes entre les concepts d'ontologies et les éléments des schémas des ressources.
- *User* : cette classe est ajoutée pour mener avec les informations concernant les utilisateurs.

3.2.2. SD-Core

SD-Core est le composant du Répertoire Sémantique qui prend en charge la gestion des métadonnées des ontologies et des ressources ainsi que les relations entre elles. Il est disponible comme un serveur d'enregistrement des sémantiques fournissant des services pour interroger et parcourir les sémantiques enregistrées.

SD-Core est composé de trois interfaces (Figure 29) : 1) l'*Interface du Répertoire des Métadonnées d'Ontologies* (*IRMO*); 2) l'*Interface du Registre Sémantique* (*IRS*) et 3) l'*Interface du Répertoire des Métadonnées des Ressources* (*IRMR*).

IRMO (Figure 24) est une interface qui facilite l'enregistrement de nouvelles ontologies et la récupération d'informations sur d'autres déjà enregistrées. Elle offre différents types d'accès à ces informations par le biais des méthodes suivantes :

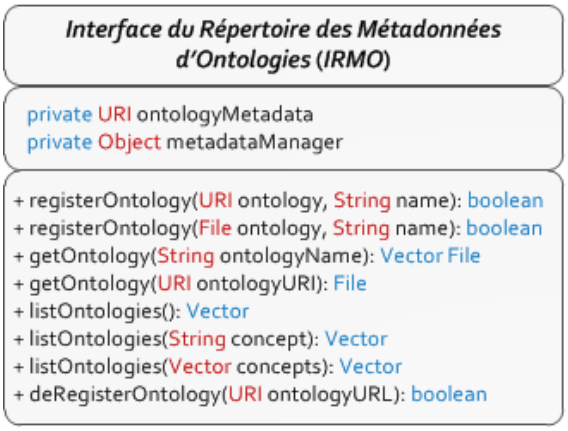

Figure 24. Interface du Répertoire des Métadonnées d'Ontologies.

- *registerOntology* : c'est une méthode surchargée (donnant lieu à plusieurs méthodes avec un même nom et arguments différents) permettant d'enregistrer des informations sur des ontologies dans le Répertoire Sémantique selon les paramètres fournis par ces méthodes. Ces informations créeront une instance de l'ontologie OMV.
- *getOntology* : c'est une méthode surchargée fournissant une manière pour retrouver des ontologies enregistrées dans le répertoire sémantique.

- *listOntologies* : ces méthodes permettent de retrouver une liste de toutes les ontologies enregistrées où quelques-unes d'entre elles dépendant du paramètre fourni à la méthode.

IRS (Figure 25) est l'interface qui permet d'enregistrer les ressources de données et de les relier à une ou plusieurs ontologies déjà enregistrées dans le répertoire sémantique. Lors de l'enregistrement d'une ressource, les implémentations de cette interface généreront une instance de la classe *Resource* dans l'ontologie SDMO. Cette instance contient des informations sur la ressource. L'interface du registre sémantique fournie principalement la méthode suivante :

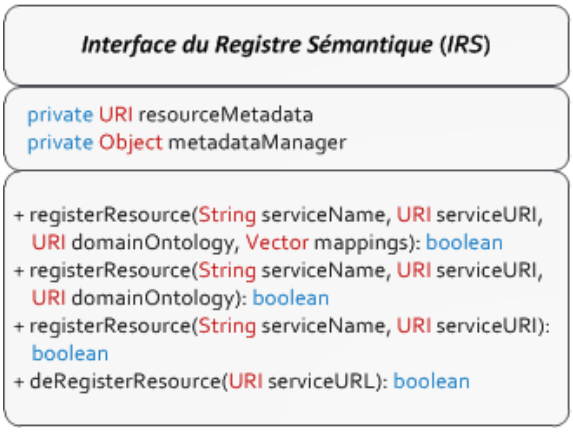

Figure 25. Interface du Registre Sémantique.

- *registerResource* : c'est une méthode surchargée permettant l'enregistrement des ressources de données et de les relier aux ontologies précédemment enregistrées.

IRMR (Figure 26) est une interface permettant l'accès aux informations des ressources. Elle fournit des méthodes permettant la récupération des métadonnées et des sémantiques liées à des ressources en se basant sur leurs URLs, noms, relations avec des ontologies, etc. elle contient les méthodes suivantes :

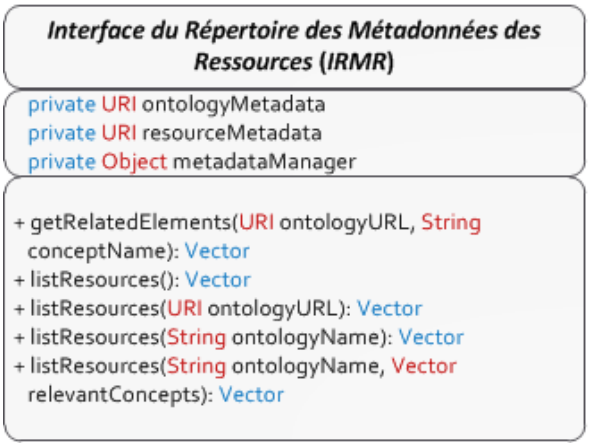

Figure 26. Interface du Répertoire des Métadonnées des Ressources.

- *getRelatedElements* : cette méthode liste les éléments reliés à un concept spécifique d'une ontologie enregistrée dans le répertoire.
- *listResources* : c'est un ensemble de méthodes permettant de lister toutes les ressources ou celles reliées à une ontologie spécifique.

3.3. SB-KOM

KOMF est une infrastructure destinée à faciliter un accès transparent aux sources de données hétérogènes en exploitant leurs sémantiques mises explicites et accessibles par le répertoire sémantique. Afin d'atteindre cet objectif, l'infrastructure se dispose d'un composant permettant de traiter les requêtes en termes d'une ontologie enregistrée dans le répertoire sémantique. Ce composant a toutes les caractéristiques d'un médiateur jouant le rôle fondamental et inévitable de la réécriture et le détournement de la requête initiale en termes de l'ontologie en sous-requêtes en termes des schémas des sources intégrées. Ce composant est appelé SB-KOM (Navas-Delgado and Aldana-Montes, 2009). Il se dispose principalement de trois modules qui seront décrit dans ce qui suit.

3.3.1. Le Contrôleur

Le Contrôleur assure la coordination entre les différents composants du SB-KOM en jouant le rôle d'un middleware. Il reçoit et crée des threads pour les requêtes de l'utilisateur avant de procéder à leur évaluation en faisant appel aux autres composants. Une fois reçue, la requête est transférée au Planificateur de requêtes qui génère un ou plusieurs plans d'exécution. Un plan d'exécution décrit l'ordre de l'exécution des sous-requêtes générées par le Planificateur. Le Contrôleur choisit un plan d'exécution, dans le cas où il y a plusieurs, et l'envoie à l'Evaluateur-Intégrateur qui se charge de l'évaluation des sous-requêtes sur les sources et la génération d'une instance d'ontologie qui sera retournée au client comme une réponse à sa requête initiale.

3.3.2. Le Planificateur de requêtes

Le Planificateur de requête est le module du SB-KOM qui assure la réécriture des requêtes en provenance des clients en plusieurs sous-requêtes. Ces dernières sont à destination des sources de données qui vont participer partiellement ou totalement dans la construction d'une réponse à la requête initiale. La réécriture d'une requête est accompagnée par l'élaboration d'un ou plusieurs plans d'exécution fournissant l'ordre et les données nécessaires pour l'exécution postérieure des sous-requêtes, à savoir, les sous-requêtes exprimées en XQuery, la localisation (ou les URLs) des composants (que nous appelons services de données) qui vont recevoir et exécuter ces sous-requêtes sur les sources de données, les règles de mappings correspondantes et l'ordre dans lequel seront interrogées les sources de données.

3.3.2.1. Requête Client

Les clients d'une application du Web Sémantique adoptant l'architecture KOMF, expriment leurs requêtes en termes d'une ontologie déjà enregistrée dans le répertoire sémantique.

Le Planificateur reçoit ces requêtes sous forme de requêtes conjonctives[66] (Cantor, 1990). C'est-à-dire une conjonction de prédicats (ou atomes) issus de l'ontologie et qui sont de trois types : *classe*, *propriété de type de données* et *propriété d'objet*. Un prédicat de type classe représente une instance d'un concept de l'ontologie tandis qu'un prédicat de type propriété d'objet représente une relation entre deux instances appartenant à une ou deux classes différentes de l'ontologie. Quant au prédicat de type propriété du type de données, il représente une relation entre une instance d'une classe de l'ontologie et une valeur littérale (de type string, integer…). Tous les prédicats sont connectés par des variables en commun. Un exemple de requête conjonctive exprimée en termes de l'ontologie du *YeastMed* est donné ci-après:

```
Ans(P, E, W) :- Protein(P), hasAASeqLength(P, "<320"), composes(P, E),
Enzyme(E), involvedIn(P, W), Pathway(W);
```

Cette requête se forme de six prédicats reliés par trois variables (P, E et W). Trois prédicats sont du type classe (*Protein(P)*, *Enzyme(E)* et *Pathway(W)*), deux sont du type propriété d'objet (*composes(P, E)* et *involvedIn(P, W)*) et un prédicat du type propriété de type de données (*hasAASeqLength(P, "<320")*). Cette requête doit retourner un ensemble de triplet dont chacun est formé du nom d'une protéine *P*, du nom d'une enzyme *E* et du nom d'une voie métabolique *W*. La protéine *P* doit avoir une longueur de séquence en acides aminées strictement inférieure à 320, fait partie de la constitution de l'enzyme *E* et impliquée dans la voie métabolique *W*.

3.3.2.2. Algorithme de réécriture de requêtes et de planification

Comme nous l'avons déjà évoqué dans la section 3.2, le Répertoire Sémantique permet d'enregistrer les relations, exprimées sous forme de règles de correspondance ou encore mappings, qui existent entre les termes d'une ontologie et les éléments des schémas des sources de données que cette ontologie réconcilie. Selon un algorithme simple se basant sur ces règles de correspondance, le Planificateur est capable de réécrire la requête conjonctive initiale en plusieurs sous-requêtes exprimées en XQuery à destination des sources de données et de générer un plan d'exécution (Figure 27).

L'algorithme reçoit en entrée une requête conjonctive exprimée en termes d'une ontologie, et retourne en sortie un ensemble de sous-requêtes structurées dans un arbre selon leurs ordres d'exécution. Les arêtes de l'arbre sont formées à partir des prédicats de type objet, alors que les nœuds sont des instances des prédicats de type classe (représentés par les variables de la requête conjonctive). Les sous-requêtes se génèrent au fur et à mesure que l'arbre se forme et se répartissent le long des sommets et des arêtes de l'arbre. A Chaque nœud correspondent une sous-requête exprimée en XQuery, les termes de l'ontologie correspondant aux prédicats de la sous-requête et les éléments du schéma de la source reliés par le mapping (ou les mappings) utilisé dans la génération de la sous-requête,

[66] http://en.wikipedia.org/wiki/Conjunctive_query

l'emplacement de l'ontologie et du service de données qui va recevoir la sous-requête pour interroger la source de données.

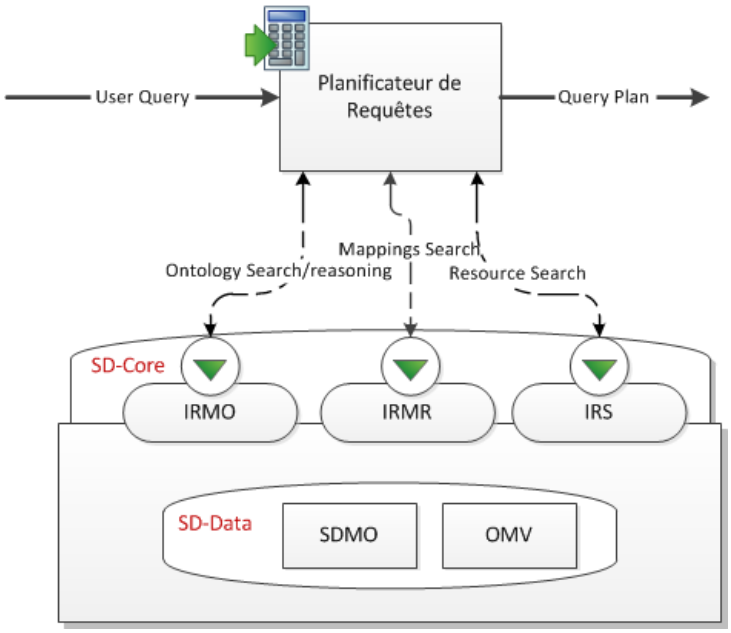

Figure 27. Le mécanisme du Planificateur de requêtes.

Les étapes de l'algorithme sont décrites ci-après.

1. Obtenir tous les prédicats de la requête conjonctive.
2. Distribuer les prédicats obtenus en deux groupes en se basant sur le nombre des arguments : le groupe G_1 contiendra les prédicats ayant un seul argument (prédicats du type de classe) et G_2 contiendra les prédicats ayant 2 arguments (prédicats de type propriété).
3. Construire des combinaisons entre les éléments des deux groupes en se basant sur la présence d'arguments en commun. Ces combinaisons formeront le groupe G_c et faciliteront la réécriture des requêtes dans le cas où un prédicat dans une situation seul n'a pas de mapping dans le Répertoire Sémantique que lorsqu'il se trouve associé avec un autre prédicat de type propriété.
4. Ajouter les prédicats obtenus dans l'étape 1 à l'ensemble G_c et éliminer les éléments répétés.
5. Filtrer les éléments résultants en gardant que les éléments qui ont au moins un mapping enregistré dans le Répertoire Sémantique.
6. Dans une liste *L*, mettre toutes les variables instanciées.
7. Pour chaque variable élaborer un plan d'exécution comme suit :
    a. La variable instanciée constitue la racine de l'arbre.

b. Les éléments qui contiennent un prédicat spécifiant une valeur à la variable instanciée et les éléments contenant seulement cette variable (sans autres variables) seront passés au nœud actuel et éliminés de G'_c.

c. Les éléments contenants, en plus de la variable instanciée, une autre variable constitueront les arêtes sortantes du nœud actuel vers de nouveaux nœuds et éliminés du G'_c. Chaque nouveau nœud sera représenté par l'autre variable de l'élément et deviendra la nouvelle variable instanciée.

d. S'ils restent encore des éléments dans G'_c, pour chaque nouvelle variable instanciée, continuer à partir de l'étape 7.b.

Un formalisme de l'algorithme est le suivant :

Algorithme Planifier

/* Reçoit une requête en termes d'une ontologie et retourne une liste d'arbre d'exécution de la requête */

0. List<Tree> *trees* = new LinkedList<Tree>() ;
1. List<String> *listeI* = tous les prédicats qui constituent la requête.
2. List<String> *listeGroupeI* = prédicats qui ont un seul argument, extraits de *listeI*.
3. List<String> *listeGroupeII* = prédicats qui ont plus d'un seul argument, extraits de *listeI*.
4. TreeSet<String> *combinaisons* = conjonctions issues de la réunion des prédicats obtenus dans *listeGroupeI* et *listeGroupeII* qui ont un argument en commun.
5. *combinaisons* = *combinaisons*.add(*listeI*).
6. *combinaisons* = ce qui restent de *combinaisons* après élimination des conjonctions qui n'ont pas des règles de correspondance dans le répertoire sémantique.
7. List<String> *instanciées* = toutes les variables instanciées.
8. For(int i=0 ; i<*instanciées*.size() ; i++){
    a. String *instanciée* = *instanciées*.get(i);
    b. Tree *arbre* = ElaborerArbre(*instanciée*); // la procédure ElaborerArbre sera détaillée par la suite.
    c. *trees*.add(*arbre*) ;

}

9. return *trees* ;

La procédure ElaborerArbre(String instanciée)

List *predicats* ; // liste de tous les prédicats.

for(int i=0 ; i<*predicats*.size(); i++){

```
String prédicat = predicats.get(i);
If(prédicat.containsArguments(instanciée)){
        //construire un nouveau nœud
        //construire une nouvelle arête
        Instanciée = prédicat.TheOtherArgument() ;
        ElaborerArbre(instanciée) ;
    }
}
```

Un exemple réel de l'application de l'algorithme est donné dans la section 4.7 (cas d'utilisation) du chapitre 4.

3.3.2.3. Conception et implémentation

Le module de réécriture de requêtes a été implémenté à travers plusieurs classes qui interagissent entre elles. Le diagramme de (Figure 28) donne une représentation abstraite dans laquelle les différents packages, interfaces, classes ainsi que les relations entre eux sont exhibées.

Le module nécessite une connexion au Répertoire Sémantique qui fournit toute les informations nécessaires sur les ressources de données, les ontologies, les mappings des sources aux ontologies enregistrées, etc. L'interface *IConnection* a été conçue pour faciliter cette connexion. Elle définit des méthodes qui sont implémentées par deux classes qui fournissent un accès correct au Répertoire Sémantique : *ConnectionDS* et *ConnectionDSWS*. La première classe traite le Répertoire Sémantique comme une librairie faisant partie intégrante des librairies du projet où le Planificateur de requêtes a été intégré (sous forme d'un .jar) alors que la deuxième utilise le Répertoire Sémantique comme un service Web. La différence entre un accès à travers *ConnectionDS* et un accès faisant appel à *ConnectionDSWS* est que le premier nécessite une installation locale du Répertoire Sémantique, alors que le deuxième ne la nécessite pas. Cependant, l'installation locale offert un accès plus rapide puisque qu'il évite l'accès à une machine distante.

Le traitement des requêtes en termes d'ontologies passe à travers l'interface *IQuery* qui définit les méthodes nécessaires pour gérer les prédicats et les arguments des requêtes, comparer les requêtes et les relier, etc. Une seule classe implémente cette interface. C'est la classe *Query*. D'autre part la classe *XQuery* définit les méthodes nécessaires pour l'élaboration des requêtes XQuery à partir de requêtes en termes d'ontologie.

Les classes *Ontology* et *Resource* fournissent des méthodes pour la gestion des informations nécessaires pour la présentation d'une ontologie ou d'une ressource de données tel que le nom, l'URL, etc., tandis que la classe *Relation* se charge de la représentation des relations entre les concepts de l'ontologie et leurs équivalents dans les sources de données. Quant à La classe *Tree*, son rôle est d'élaborer le plan de l'exécution des requêtes en faisant intervenir plusieurs autres classes auxiliaires. Cependant, les deux

principales classes du module sont nommées *Planner* y *Planificador*. Ces deux classes font usage des autres classes pour accomplir la tâche de la réécriture et l'exécution des requêtes.

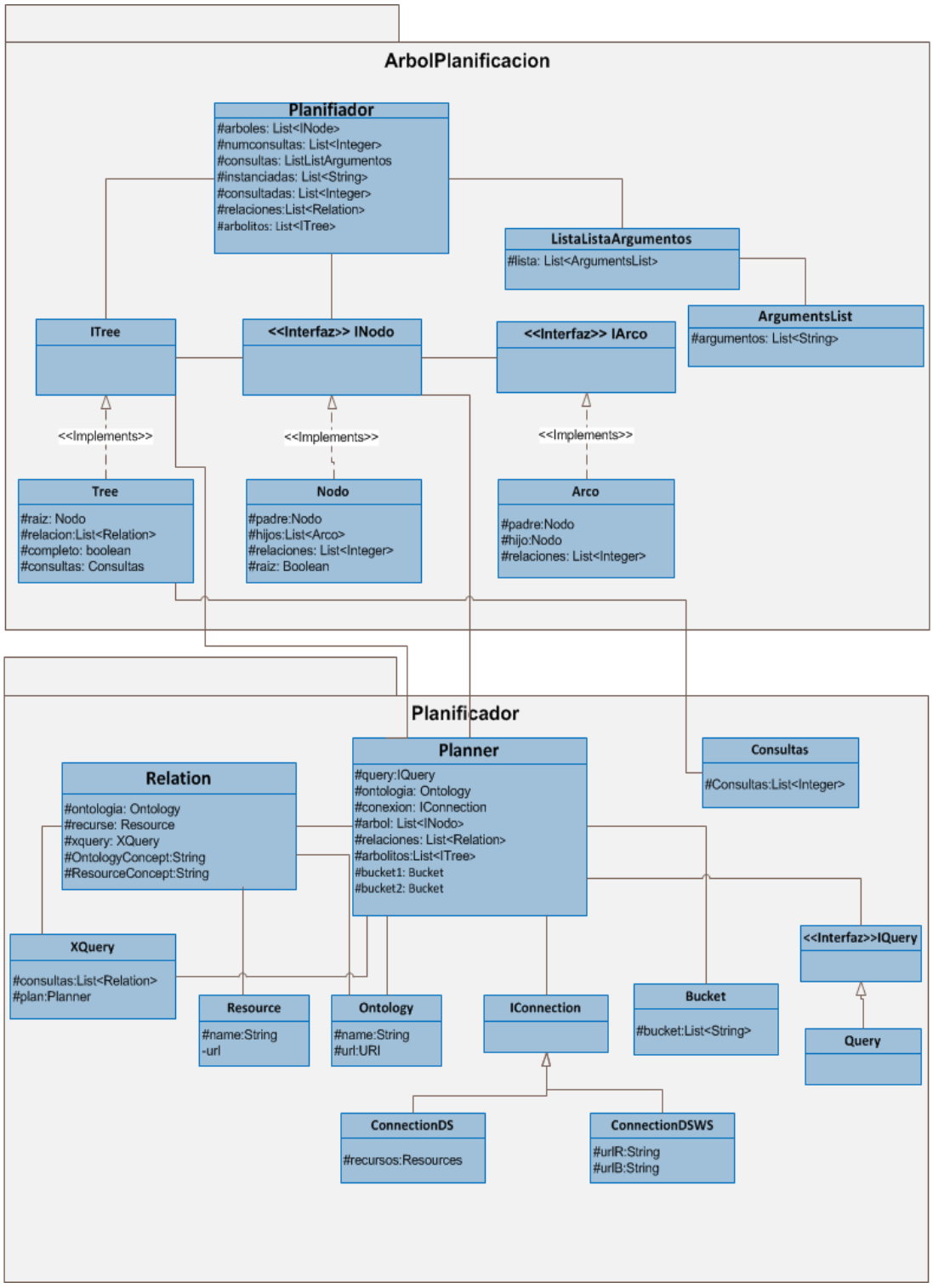

Figure 28. Diagramme de classes du module de réécriture de requêtes.

La classe *Planner* est celle qui se charge du traitement de la requête reçue par le Planificateur de requêtes. Elle fragmente la requête en sous-requêtes, cherche et procure les mappings en se connectant au Répertoire Sémantique et prépare un objet de la classe *ArgumentsList* qui contient tous les arguments des tuples qui s'obtiennent de la cinquième étape de l'algorithme de réécriture de requêtes décrit précédemment. La classe *Planificador* reçoit cette liste d'arguments et se charge de l'élaboration de tous les arbres d'exécution possibles en utilisant la méthode *Planificar*().

Le diagramme de séquence de la Figure29 reprend quelques des interactions qui se déroulent, selon un ordre chronologique, entre les principales classes du Planificateur.

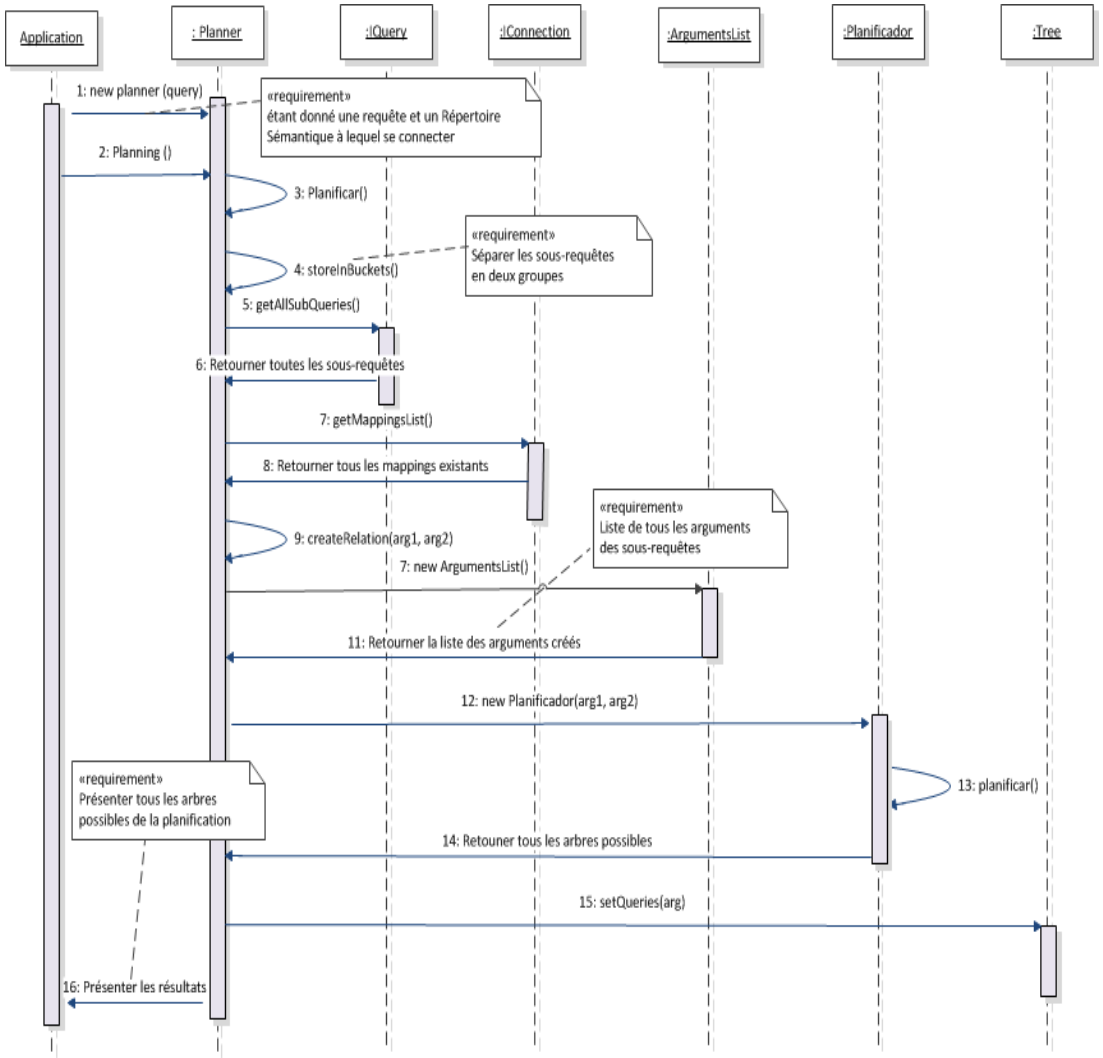

Figure29. Diagramme de séquence de la planification de requêtes.

3.3.3. L'Evaluateur-Intégrateur

Un plan d'exécution décrit l'ordre de l'exécution des sous-requêtes générées par le Planificateur. Chaque nœud de l'arbre contient, comme décrit précédemment, toutes les informations exigées par l'Evaluateur-Intégrateur pour exécuter la sous-requête qui lui correspond ; ces informations incluent principalement l'URI de la ressource à interroger et la sous-requête exprimée en XQuery. L'Evaluateur-Intégrateur exécute les sous-requêtes en faisant appel aux services de données dans l'ordre déterminé dans le plan d'exécution. Au fur et à mesure de l'exécution, il génère des instances pour les variables des sous-requêtes à exécuter à partir des résultats des sous-requêtes déjà exécutées. Pour minimiser le temps d'exécution, les sous-requêtes qui ne dépendent d'aucune autre, sont exécutées en même temps. Une requête Q_j est dite dépendante d'une autre Q_i, si Q_j est une requête paramétrée et dont les paramètres prennent valeur à partir du résultat du Q_i.

Les résultats retournés par les services de données sont des fichiers XML dont la structure est prédéfinie par un schéma XML. L'Evaluateur-Intégrateur a un processus qui se charge de la traduction des fichiers XML en une instance de l'ontologie exprimée en RDF. Ce processus de traduction est effectué par une API appelée *JXML2RDF* dont la conception et le fonctionnement seront détaillés par la suite dans la section 3.3.3.2.

3.3.3.1. Processus d'évaluation

Dans cette section, nous allons détailler le processus d'évaluation d'un plan d'exécution possible (Figure 30) correspondant à la requête exemple donnée dans la section 3.3.2.1, à partir de l'appel d'un service de données jusqu'à l'obtention d'un ensemble de triples qui satisfait tous les prédicats de la requête.

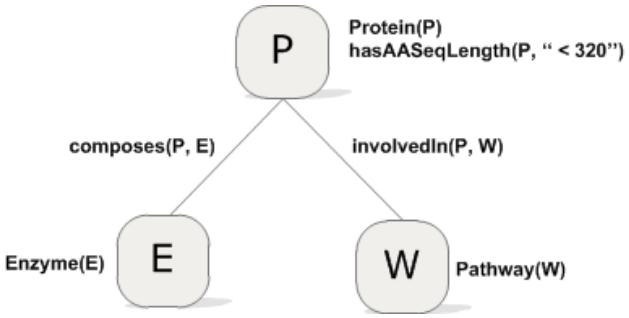

Figure 30. Exemple d'arbre de planification.

L'évaluation d'un arbre d'exécution commence d'habitude par la racine et s'étend vers les feuilles. La sous-requête d'un nœud dont toutes les variables sont instanciées se traduit par le composant de réécriture de requêtes en une requête XQuery qui s'exécute directement par le composant d'évaluation. Cependant, quand un service de données (DS2) dépend (dans l'arbre d'exécution) d'un autre (DS1), dans la plupart des cas il est nécessaire de réaliser plusieurs sous-requêtes différentes au service DS2. Le fait que les sous-requêtes soient différentes est dû à ce que la sous-requête XQuery est paramétrée par la déclaration

de variables qui seront instanciées automatiquement, à l'aide de l'API de Saxon[67], au moment de l'appel au service de données DS2. Les valeurs que prennent les variables proviennent du résultat XML du service DS1. Comme Saxon permet d'interroger un fichier XML en utilisant XPath (Arena, 2012) (puisqu'une requête XPath est une requête XQuery valide), l'évaluateur utilise l'expression XPath, dans le fichier XML de sortie du DS1, correspondant à la variable en commun entre le groupe des sous-requêtes correspondant au service DS2 et le groupe qui établit la relation entre les deux services.

Eclaircissons ce qui précède par un exemple : pour pouvoir réaliser une requête au service de données correspondant au nœud "*E*" de l'arbre d'exécution de la Figure 38, il est nécessaire d'obtenir une liste des *Protéines* à partir du résultat XML élaboré par le service de données correspondant au nœud "*P*". Cette liste est obtenue en sélectionnant les éléments dont le chemin XPath est celui associé au prédicat de la variable "*P*" dans les règles de mappings. Avec chaque élément de la liste, s'instancie la variable de la sous-requête XQuery associée avant d'appeler le service de données correspondant. Ce mécanisme qui s'applique au nœud "*E*", se reproduit aussi au niveau du nœud "*W*" de la même façon.

L'étape suivante du processus de l'évaluation est l'obtention des instances de l'ontologie à partir des résultats d'exécution des services de données. Dans cette étape, il se fait usage de l'API *JXML2RDF* dont les caractéristiques seront détaillées par la suite dans la section 3.3.3.2.

Quand l'exécution du service de données correspondant au nœud racine est terminée, et après la génération des instances des variables à partir du XML résultant, chaque appel d'un service de données correspondant à un nœud sera réalisé pour chacune des instances générées par le nœud père. Pour obtenir le modèle global final afin de pouvoir évaluer la requête globale initiale, les triplets RDF de chaque paire de services dépendants doivent être reliés. La liaison se fait à travers d'une propriété d'objet (l'arrête reliant les deux nœuds correspondants à l'appel des deux services de données dépendants).

Comme le montre la Figure 31, l'établissement des relations commence d'une manière ascendante quand l'exécution atteint une feuille dans l'arbre ; chaque ensemble de triplets d'un nœud *N* se relie avec le triplet du nœud parent *Np* pour lequel l'appel du service de données correspondant au nœud *N* s'est réalisé. Cette liaison se fait par le biais de la propriété d'objet correspondante à l'arrête reliant les deux nœuds *N* et *Np*, suivi de l'union des triplets des deux nœuds et ainsi successivement jusqu'à arriver au nœud racine. Pour l'exemple de la Figure 39, les triplets *D* se relient avec les triplets *B_R* sélectionnés à travers la propriété d'objet *3* et s'unissent à l'ensemble des triplets *B_R*. A partir de l'ensemble des triplets résultant, seulement les trois triplets rouges du nœud *B* se relient avec le triplet rouge du nœud *A* par la propriété *d'objet 1* et l'ensemble entier des triplets s'unissent à l'ensemble des triplets du nœud *A*. Ce processus se répète successivement pour l'ensemble des triplets générés. Etant donné que les sujets résultants de la traduction du résultat XML

[67] http://sourceforge.net/projects/saxon/

d'un service de données ne sont pas du même type, ils se relient seulement les sujets dont les types (ou bien classes) sont connectés dans l'ontologie à travers la propriété d'objet en question.

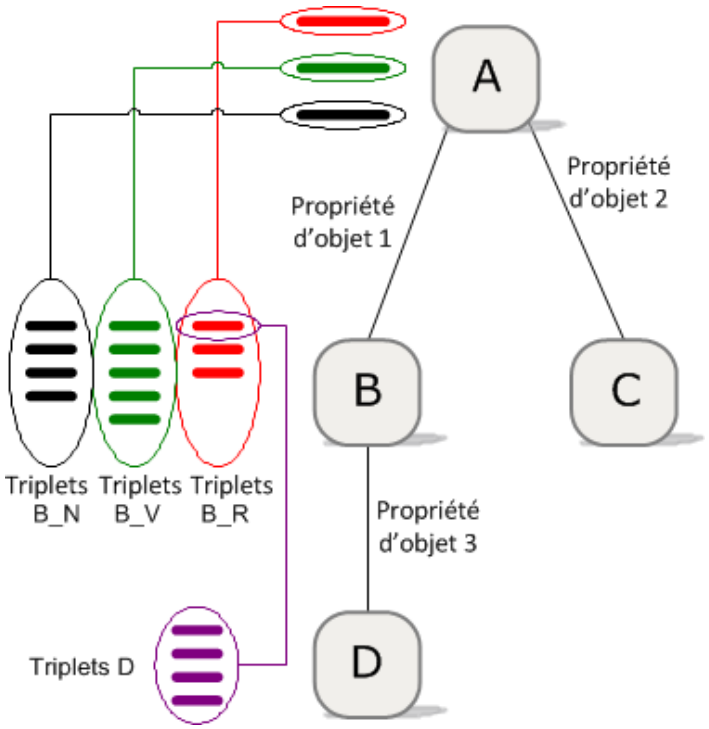

Figure 31. Relation entre les triplets en vue d'obtenir le modèle global.

3.3.3.2. Conception et implémentation de l'API JXML2RDF

La transformation des documents XML retournés par les services de données, lors de l'exécution du plan des sous-requêtes issues de la réécriture de la requête initiale, en instances d'ontologie se fait par l'API *JXML2RDF*. Ce processus se base sur les règles de mappings (ou associations) entre les éléments du schéma XML des sources de données, et les éléments de l'ontologie aux termes de laquelle, la requête initiale a été formulée. Vu que les prédicats d'une requête se forment à partir des classes et des propriétés d'une ontologie, trois types de règles de mappings sont utilisés (nous reviendrons sur ceci avec des exemples dans la section 4.6.3 du chapitre 4) :

- Mapping de classe : il associe un élément du schéma XML à un concept de l'ontologie.
- Mapping de propriété du type de données : il associe un élément du schéma XML à une propriété du type de données.
- Mapping de propriété d'objet : il associe deux mappings de classe à une propriété d'objet.

Les ressources de l'ontologie (classes et propriétés) dans le mapping sont représentées par des références URI, alors que les éléments des schémas XML des sources de données sont représentés par des expressions XPath. L'utilisation des expressions XPath permet de distinguer entre les éléments XML portant le même nom mais ayant des ancêtres différents ce qui permet de les identifier et de les associer aux ressources correspondantes de l'ontologie d'une manière précise.

Les mappings de classes sont définies par une paire constituée de la référence URI de la classe de l'ontologie et l'expression XPath qui identifie l'élément XML. Ceci pour dire qu'une instance de la classe associée est créée pour chaque élément XML correspondant à l'expression XPath. Les associations de propriété de type de données sont définies à travers la référence URI de la propriété, le mapping utilisé pour la classe du domaine de la propriété et l'expression XPath qui identifie l'élément XML dont la valeur est le rang de la propriété. Quant aux associations de propriété d'objet, elles sont définies par la référence URI de la propriété et les mappings de la classe du domaine et du rang de la propriété.

Le processus de la traduction nécessite un chargement préalable de l'ontologie et des schémas XML avant la spécification des règles de mappings entre les éléments XML et les éléments de l'ontologie. Ceci est exhibé dans La Figure 32 qui donne un aperçu schématique du processus. Le point d'accès à l'API est une classe appelée *XML2RDF* à travers la méthode statique *createMappingProject()* (Figure 33). L'API *JXML2RDF* est composée d'un ensemble d'interfaces permettant la définition de méthode pour le traitement des Mappings, d'ontologies ou des fichiers XML. Ainsi, Les mappings sont manipulés par cinq interfaces dont la principale est l'interface *XML2RDFProject* (Figure 34).

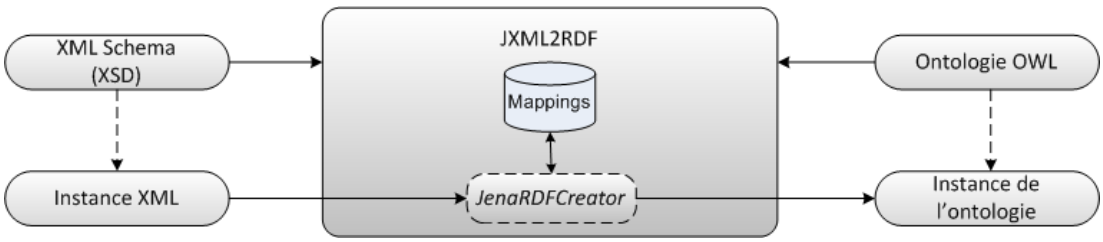

Figure 32. Schéma du processus de la traduction réalisé par l'API JXML2RDF.

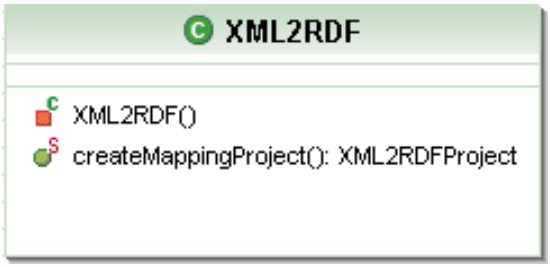

Figure 33. La classe d'entrée à l'API JXML2RDF.

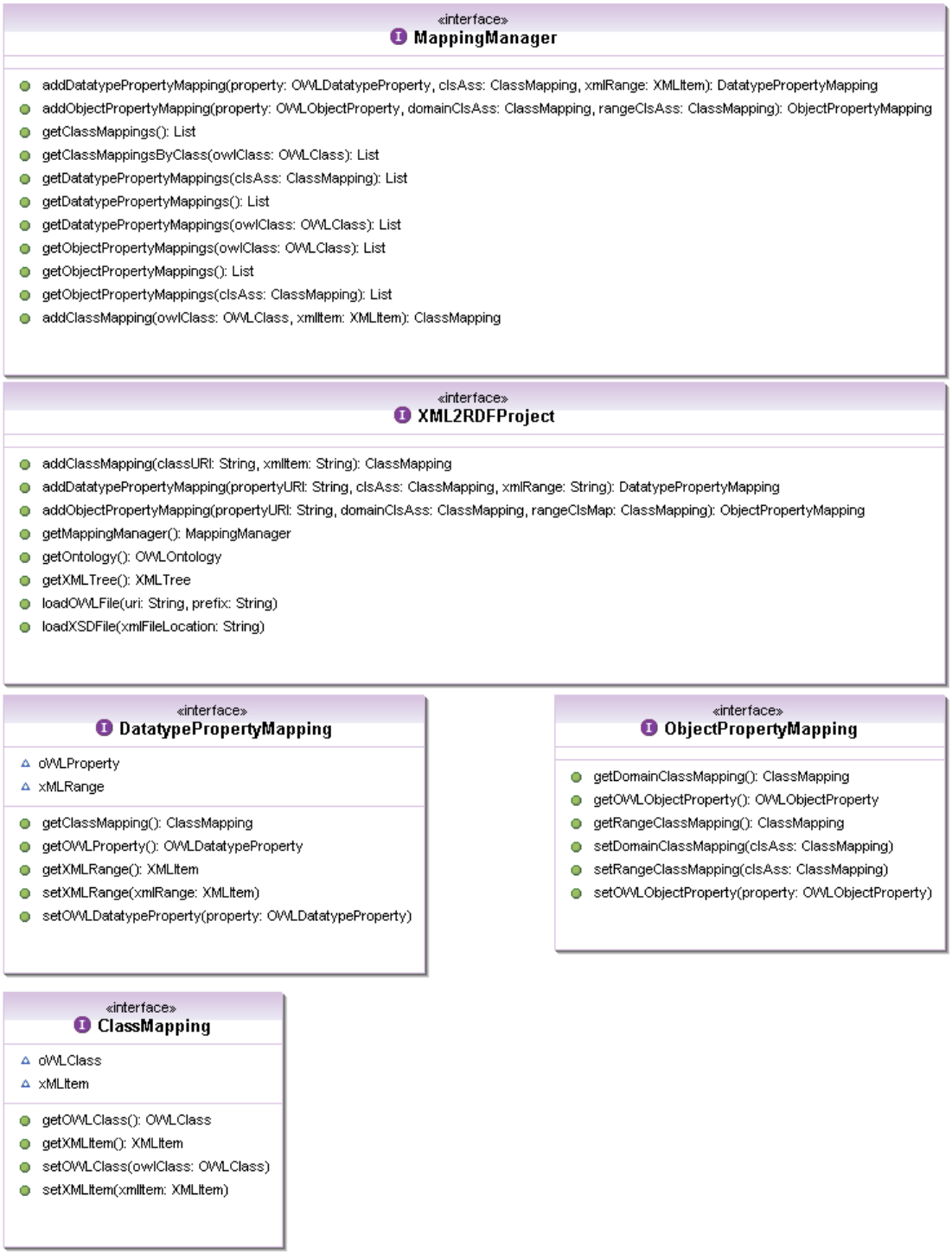

Figure 34. Interfaces de définition des mappings.

Cette dernière fournie des méthodes pour charger les schémas XML et les ontologies. Elle permet aussi de créer des associations entre les classes et les propriétés de l'ontologie d'une part et les éléments XML de l'autre part.

L'interface *ClassMapping* représente une association entre une classe de l'ontologie (représentée dans l'API par *OWLClass*) et un élément XML (représenté dans l'API par

XMLItem), tandis que L'interface *DatatypePropertyMapping* représente une association entre une propriété de type de données de l'ontologie (représenté dans l'API par *OWLDatatypeProperty*) et un élément XML dont la valeur représentera le rang de la propriété pour chaque instance. Pour cette association, il faut tenir en compte lequel des mappings déclarés pour la classe du domaine de la propriété se veut être utilisé, puisque l'API permet des associations un-à-un, un-à-plusieurs et plusieurs-à-plusieurs. L'interface *ObjectPropertyMapping* représente une association entre deux classes de l'ontologie. Pour spécifier l'association, il faut tenir en compte lequel des deux mappings déclarés pour les classes du domaine et du rang de la propriété se veut être utilisé puisqu'elles sont permis les relations plusieurs-à-plusieurs. Finalement l'interface *MappingManager* gère les associations entre les ressources de l'ontologie et les éléments XML. Elle fournit des méthodes pour ajouter des associations. Des classes implémentant ces interfaces ont été créées dont la plus importante est la classe *MappingManagerImpl* qui gère tous les mappings à travers des associations du type *classMappings*, *datatypePropertyMappings* et *objectPropertyMappings*.

Les ressources d'ontologies sont définit par un package contenant 6 interfaces (Figure 35). OWLResource est l'interface basique de toutes les ressources d'ontologies. Les classes et les propriétés sont traitées respectivement par les interfaces OWLClass et OWLProperty. Les propriétés du type de données sont manipulées par l'interface OWLDatatypeProperty alors que les propriétés d'objet sont manipulées par l'interface OWLObjectProperty. Ces deux dernières interfaces héritent de la classe OWLProperty qui, à son tour avec l'interface OWLClass, hérite de l'interface OWLResource. Toutes les ressources de l'ontologie s'identifient avec un nom et un préfix (représentés successivement par localname et prefix dans l'interface OWLResource) dont le produit de la concaténation est l'URI de la ressource. Finalement, l'interface OWLOntology représente le modèle de l'ontologie qui s'importe et fournie des méthodes qui permettent de manipuler les concepts et les propriétés. Dans cette interface, une ontologie s'identifie par son URI et il est possible de l'associer un préfix ; en plus, il a souvent un ensemble d'associations entre préfixes et espaces de nommage. Plusieurs classes implémentant ces interfaces ont été créées.

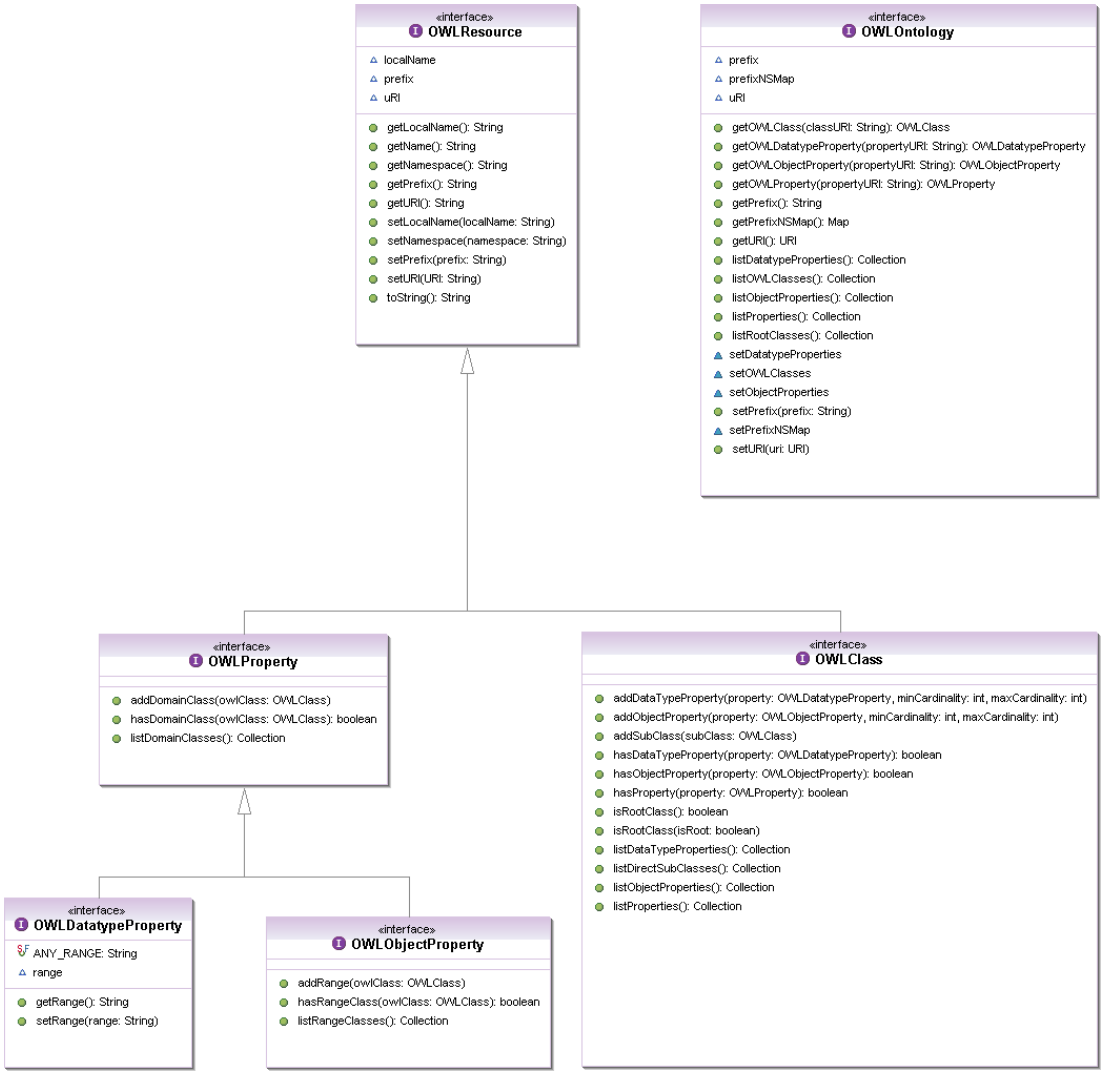

Figure 35. Interfaces de définition des ressources ontologiques.

Pour mener avec les ressources XML, deux interfaces ont été créées : L'interface *XMLItem* qui représente un élément XML ou un attribut qui se caractérise par un chemin XPath absolu et par un nom ; et l'interface *XMLTree* qui représente une structure en arbre des éléments et des attributs XML (*XMLItems*). Chacune de ces interfaces a été implémentée par une classe (Figure 36).

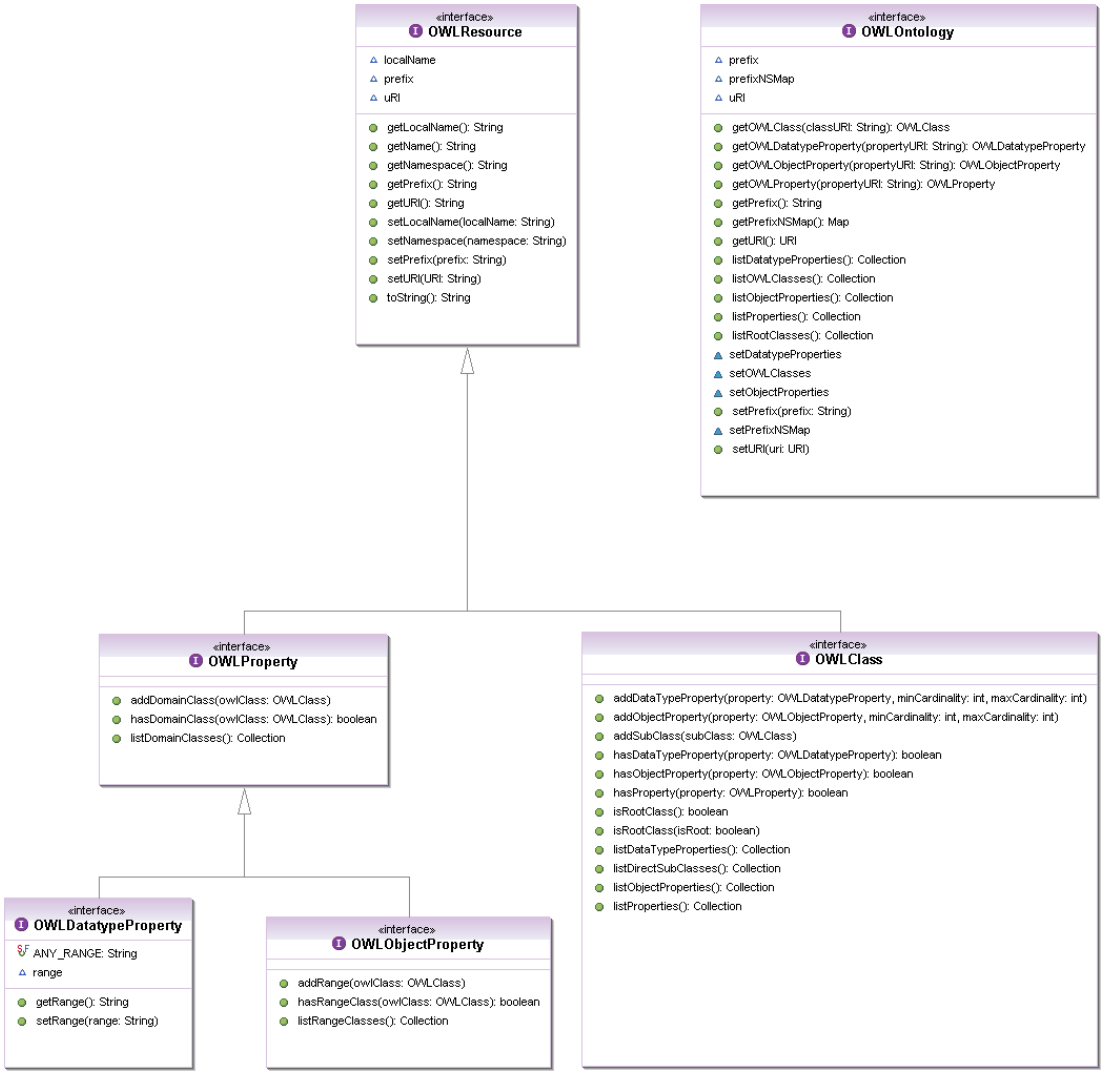

Figure 36. Interfaces et classes de définition des éléments XML.

Pour pouvoir parser un fichier OWL et construire un modèle d'ontologie, une classe abstraite appelée *AbstractOWLFactory* (Figure 37) a été créée. Une seule fabrique, nommée *ProtegeOWLFactory* implémente *AbstractOWLFactory*. Elle utilise l'API Protégé-OWL[68], mais elle peut implémenter autres fabriques par exemple en se basant sur le projet Jena[69].

[68] http://protege.stanford.edu/plugins/owl/api/

[69] http://jena.apache.org/

Figure 37. Fabrique pour parser les documents OWL.

Le package *es.uma.khaos.jxml2rdf.xmlLoader* offre la classe abstraite *XMLTreeFactory* (Figure 38) qui permet d'instancier une fabrique pour parser les fichiers des schémas XML et construire un arbre *XMLTree* avec les éléments XML. Une seule fabrique basée sur le projet XSOM[70] a été implémentée. La classe *XMLInstanceLoader* offre une méthode statique qui charge en mémoire le fichier XML et le traduit en un objet DOM.

[70] http://xsom.java.net/

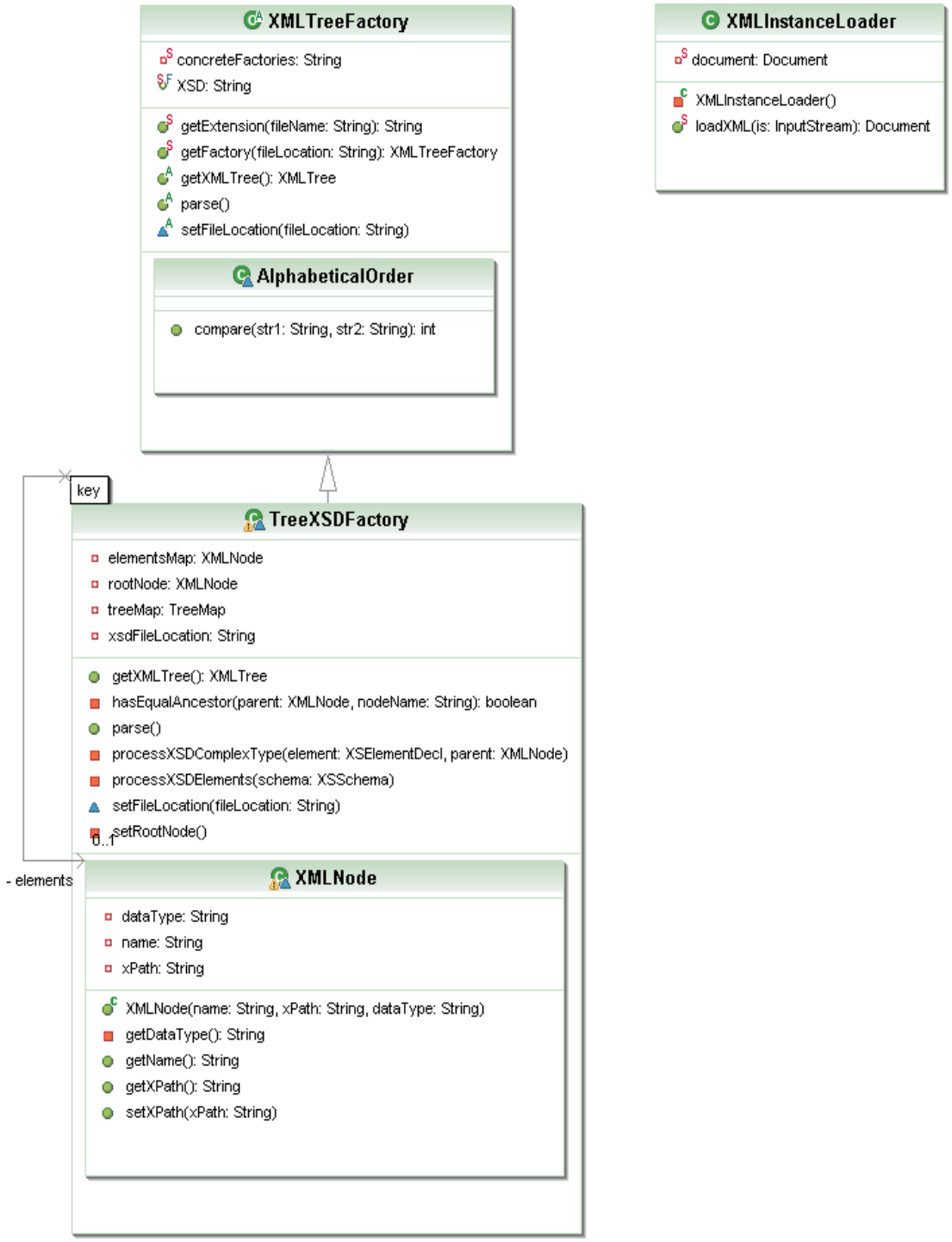

Figure 38. Fabrique pour parser les documents XSD et classe pour charger un document XML en mémoire dans un objet DOM.

La classe abstraite *AbstractRDFCreator* (Figure 39) est une classe qui se charge de l'instanciation d'une fabrique pour la création d'un modèle RDF ou d'un fichier RDF/XML en se basant sur les règles de mappings entre les éléments XML et les ressources de

l'ontologie. Une seule fabrique, *JenaRDFCreator*, a été implémentée en se basant sur le projet Jena.

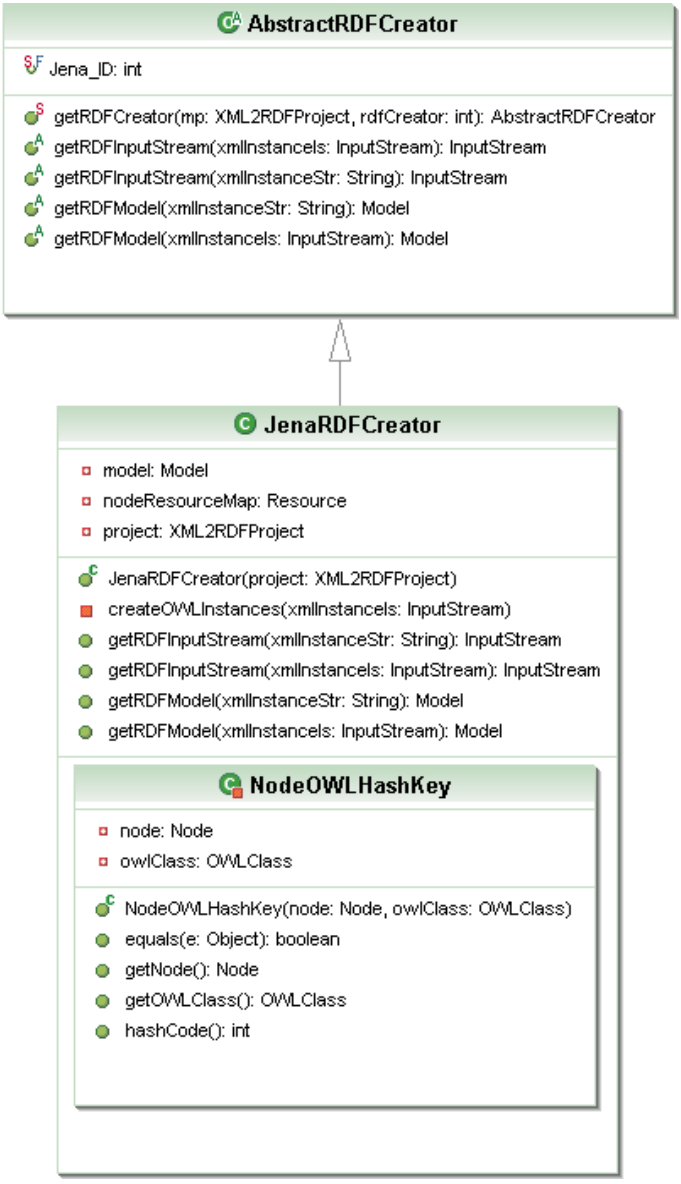

Figure 39. Fabrique pour créer les instances de l'ontologie.

Le diagramme de séquence de la Figure 40 illustre la séquence principale du fonctionnement de l'API.

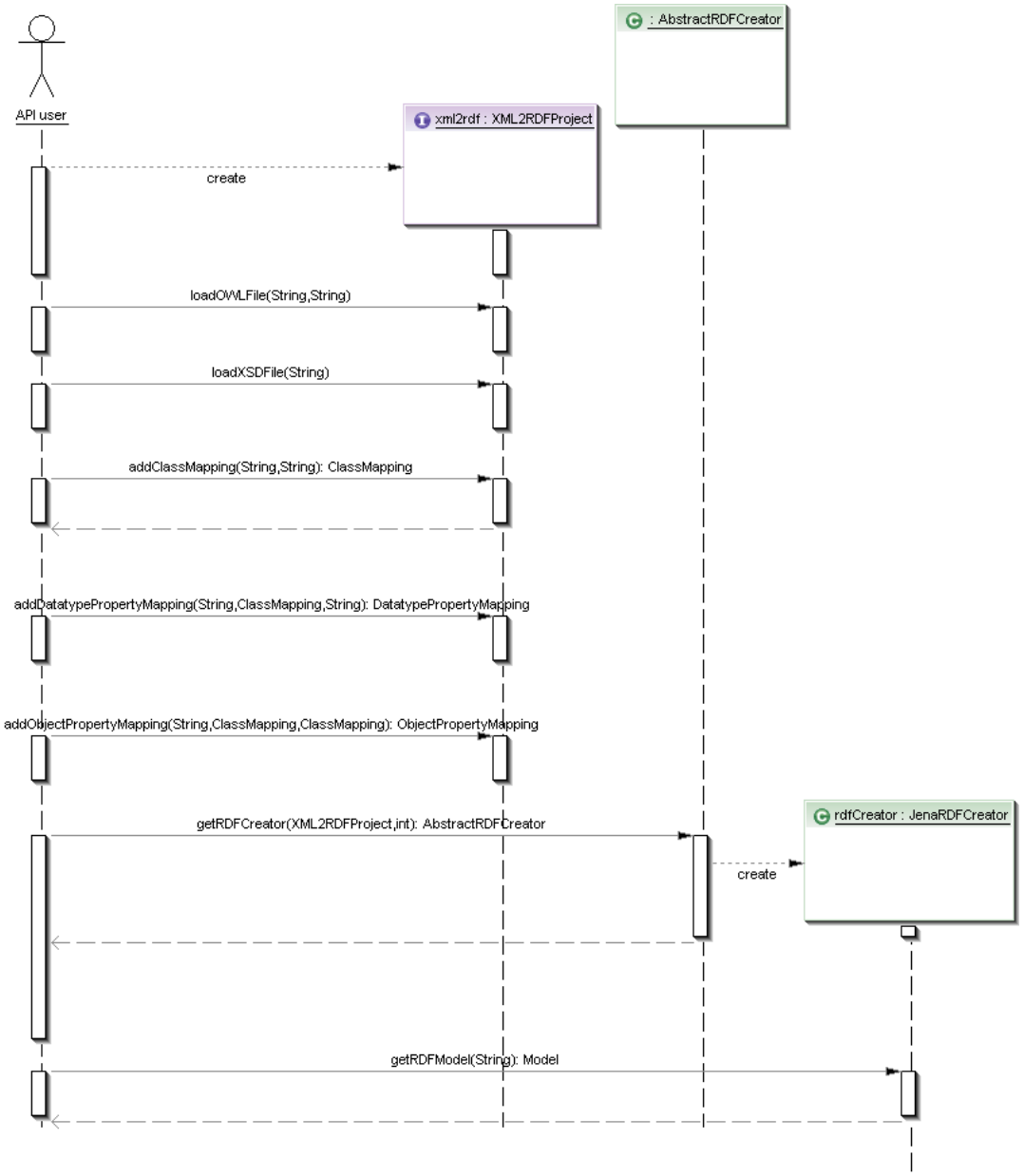

Figure 40. Diagramme de séquence principal de l'API JXML2RDF.

3.4. Conclusion

Dans ce chapitre, nous avons présenté KOMF : une infrastructure de médiation à base d'ontologie qui permet de faciliter un accès transparent à des sources de données. KOMF rend explicites et utilisables les sémantiques des sources de données à travers leurs publication dans le Répertoire Sémantique. Ce dernier se dispose d'une part du SD-Data : un composant qui stocke les métadonnées des sources et d'ontologies. Il fait usage de deux ontologies (OMV et SDMO) qui décrivent les sémantiques dans le Répertoire Sémantique. D'autre part, le Répertoire Sémantique se dispose du SD-Core qui se forme de trois interfaces prenant en charge la gestion des métadonnées des ontologies et des ressources ainsi que les relations entre elles. Il est disponible comme un serveur d'enregistrement des

sémantiques fournissant des services pour interroger et parcourir les sémantiques enregistrées. Ainsi, *IRMO* est l'interface qui facilite l'enregistrement de nouvelles ontologies et la récupération d'informations sur d'autres déjà enregistrées ; *IRS* est l'interface qui permet d'enregistrer les ressources de données et de les relier à une ou plusieurs ontologies déjà enregistrées dans le répertoire sémantique ; puis finalement *IRMR* est l'interface permettant l'accès aux informations des ressources.

Le traitement des requêtes dans KOMF se fait par un médiateur qui se nomme SB-KOM. Il se compose de trois modules : Le Contrôleur, le Planificateur de requêtes et l'Evaluateur-Intégrateur. Le Contrôleur assure la coordination entre les différents composants du SB-KOM en jouant le rôle d'un middleware ; Le Planificateur de requête est le module qui assure la réécriture des requêtes en provenance des clients en plusieurs sous-requêtes tandis que l'Evaluateur-Intégrateur exécute les sous-requêtes en faisant appel aux services de données dans l'ordre déterminé dans le plan d'exécution généré par le Planificateur.

Dans le chapitre suivant, nous allons voir comment *YeastMed* : le système d'intégration des sources de données de la levure *Saccharomyces cerevisiae*, fait usage du KOMF pour accomplir sa fonction.

Chapitre 4

YeastMed : Un Prototype de systeme d'integration de donnees de la levure Saccharomyces cerevisiae

Sommaire

4.1. Introduction

La levure *Saccharomyces cerevisiae* fut le premier eucaryote à entrer au panthéon des organismes entièrement séquencés. La puissance de sa génétique et l'importance de la communauté scientifique internationale qui s'y consacre ont fait de cet organisme un modèle pour les études fonctionnelles post-séquençage aussi bien qu'une plate-forme expérimentale pour développer et valider de nouvelles approches et stratégies génomiques. Cette importance dans les recherches biologiques et l'abondance des données issues des expérimentations biologiques, ou des analyses computationnelles des laboratoires bioinformatiques ont donné lieu à plusieurs sources et bases de données. Un grand nombre de ces sources sont actuellement publiquement disponibles et accessibles via internet. Elles regroupent, organisent et présentent des données variées et utiles aux biologistes souhaitant trouver des informations sur une séquence, un gène ou une protéine de la levure. . Le manque de systèmes qui peuvent assurer un accès transparent a ces sources rend leur interrogation un travail très durs et fatigant pour les biologistes ; ce qui minimise les avantages et les bénéfices qui peuvent être tirés de ces sources. Cette place importante qu'occupe cet organisme dans les recherches biologiques, ainsi que la grande diversité et l'hétérogénéité de ses sources de données ont fait de L'intégration des données de *Saccharomyces cerevisiae* une nécessité.

Dans ce chapitre, nous allons présenter *YeastMed* : le système d'intégration des données de la levure *Saccharomyces cerevisiae*. Dans sa version actuelle. *YeastMed* intègre cinq sources de données de la levure. Il se base sur l'architecture KOMF, présentée dans le chapitre précèdent.

Ce chapitre sera organisé comme suit : dans la section qui suit, nous introduisons la levure et ses sources de données qui seront manipulées par le système. Dans la section d'après, nous donnons quelques facettes de la modélisation UML que nous avons menée pour le système. Puis, l'architecture générale du *YeastMed* sera détaillée dans la section

4.5. La section 4.6 sera consacrée pour parler de l'implémentation et la mise en œuvre des différents composants du système. Un cas d'utilisation montrant le processus du traitement d'une requête soumise à *YeastMed* sera donné dans la section 4.7. Et puis et avant de conclure, la section 4.8 donne les résultats de deux études menées sur *YeastMed* qui renseignent sur son utilisabilité et ses performances.

4.2. La levure

La levure est un champignon unicellulaire (certaines levures sont cependant capables d'arborer un aspect pseudo pluricellulaire par la formation, par ex., de pseudomycélium*) apte à provoquer la fermentation des matières organiques animales ou végétales. Les levures sont employées pour la fabrication du vin, de la bière, des spiritueux, des alcools industriels, du pain et d'antibiotiques. Ces micro-organismes, de forme variable selon l'espèce (sphérique, ovoïde, en bouteille, triangulaire ou apiculée, c'est-à-dire renflée à chaque bout comme un citron) mais le plus souvent ovales, d'environ 6 à 10 microns et jusqu'à 50 microns, se multiplient par bourgeonnement ou par fission (scissiparité). Ils sont fréquemment capables d'accomplir une sporulation soit dans un but de dormance en milieu défavorable, soit dans un but de dispersion.

L'expression levure découle de l'observation des fermentations et tout spécifiquement celle qui a lieu durant la fabrication du pain : on dit couramment et depuis longtemps que le pain lève. Ce n'est pas, à proprement parler, une appellation scientifique actuelle. Mais l'importance des levures dans le domaine des fermentations conduit à conserver ce terme générique qui continue à être correctement perçu.

Le terme courant de levure sert à désigner le plus souvent le genre *Saccharomyces cerevisiae* (Figure 41). Ce dernier est une levure spécifique parmi l'ensemble des ferments, levains, levures, etc. utilisés depuis l'aube de l'humanité dans l'élaboration du pain, du kéfir, des yaourts, du vin et de la bière de fermentation haute. Elle a été découverte, isolée et identifiée au milieu du XIX^{ème} siècle par des brasseurs hollandais à la demande de la corporation des boulangers parisiens qui commençaient à industrialiser leur production et cherchaient, pour faire leur pain, un procédé de fermentation plus fiable et plus rapide que leur levain sauvage respectant les traditions. Ainsi, dans ces domaines, elle est nommée "levure de boulanger" ou "levure de bière". Enfin, dû à son mode de reproduction, elle est aussi appelée "levure à bourgeon" (ou "levure bourgeonnante", budding yeast en anglais).

Saccharomyces cerevisiae peut produire l'énergie indispensable à sa survie ainsi qu'à sa reproduction de deux manières différentes, selon le milieu ambiant. Ces deux modes de production d'énergie sont :

- la respiration, transformation du glucose en gaz carbonique avec l'oxygène (aérobie),
- la fermentation alcoolique du glucose (anaérobie).

Le premier est utilisé en cuisine, alors que le second est privilégié pour la production d'alcool tel que le vin ou la bière.

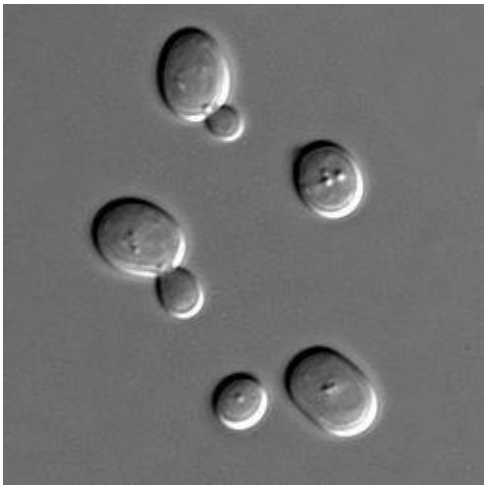

Figure 41. *Saccharomyces cerevisiae* (image prise du site Wikipedia).

4.2.1. Cycle de vie

La levure *Saccharomyces cerevisiae* a un cycle biologique particulier (Figure 42). Elle est capable de se multiplier sous deux formes : une forme diploïde (2n = 32 chromosomes) et une forme haploïde (1n = 16 chromosomes).

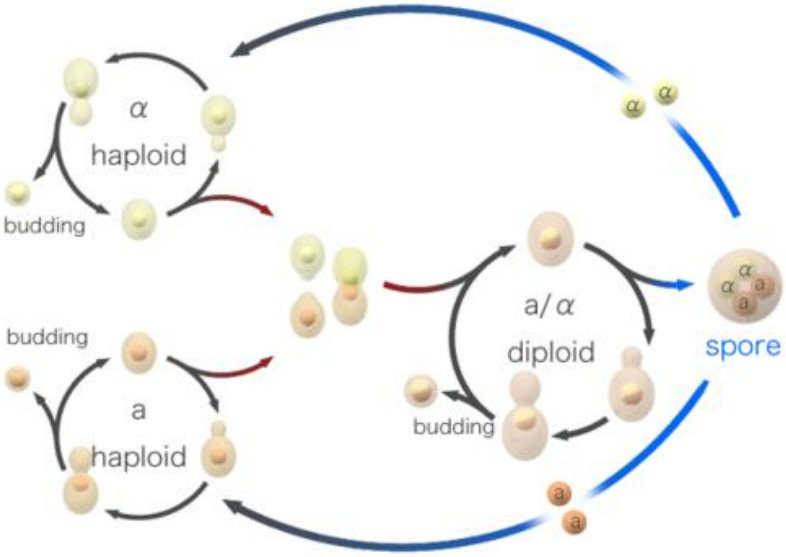

Figure 42. Cycle de vie de la levure *Saccharomyces cerevisiae* (image prise du site Wikipedia).

Les cellules haploïdes se multiplient en bourgeonnant : la cellule mère bourgeonne une cellule fille plus petite (mitose), mais possédant la même information génétique. Il existe des cellules haploïdes "a" et des cellules haploïdes "α" qui correspondent à des signes sexuels distincts. Ces deux types de cellules ne se distinguent pas morphologiquement mais par la phéromone qu'elles produisent : *MATα* ou *MATα*. Les phéromones libérées permettent l'amorce du processus de fécondation en se liant à un récepteur spécifique.

Ensuite c'est la fusion entre une cellule "a" et une cellule "α" qui donne naissance à une cellule diploïde "a/α". Tant que l'environnement est favorable, le diploïde se multiplie par bourgeonnement. Si les nutriments viennent à manquer, la cellule repasse en phase haploïde par un processus de méiose. On obtient finalement quatre noyaux haploïdes qui sont inclus dans les spores (ascospores) contenues dans un sac appelé asque. L'enveloppe de l'asque se rompt à maturité et libère alors deux cellules "a" et deux cellules "α" qui peuvent recommencer le cycle.

Les souches industrielles sont souvent polyploïdes (3, 4, 5n Chromosomes) et donc possèdent plusieurs gènes pour un même caractère. Elles sont donc plus stables génétiquement car difficiles à faire muter. La plupart de ces souches sont incapables de sporuler dans les conditions de culture industrielle et se reproduisent par bourgeonnement.

4.2.2. Un système modèle

La levure *Saccharomyces cerevisiae* (que nous nous contentons parfois de l'appeler seulement *levure* dans ce mémoire) est maintenant reconnue comme un système modèle représentant un eucaryote simple dont le génome peut être facilement manipulé. Cette levure a seulement une complexité génétique légèrement plus élevée que les bactéries et ils partagent de nombreux avantages techniques qui ont permis un progrès rapide dans la génétique moléculaire des procaryotes et leurs virus. Certaines propriétés qui rendent la levure particulièrement convenable pour les études biologiques sont : la croissance rapide, la dispersion des cellules, la facilité de l'isolement des mutants, un système génétique bien défini, et plus important encore, un système de transformation d'ADN polyvalent (Daniel Gietz and Woods, 2002). Etant non pathogène, la levure peut être traitée avec peu de précautions.

Des souches de *Saccharomyces cerevisiae*, contrairement à la plupart des autres micro-organismes, ont un état, haploïde et diploïde, stable et viable avec un grand nombre de marqueurs. Ainsi, des mutations récessives sont convenablement manifestées chez des souches haploïdes, tandis que des tests de complémentarité peuvent être effectués avec des souches diploïdes (Sherman, 2002). Le développement de la transformation de l'ADN a rendu la levure particulièrement accessible aux techniques du génie génétique et du clonage génétique. Les gènes structuraux correspondants à n'importe quel trait génétique peuvent être identifiés par complémentation à partir de bibliothèques de plasmides. Les plasmides peuvent être introduits dans les cellules de la levure en tant que molécules de réplication ou par intégration dans le génome. Contrairement à la plupart des autres organismes, une recombinaison intégrative de l'ADN transformant dans la levure se produit exclusivement par une recombinaison homologue. Les séquences clonées de levure, accompagnées de séquences étrangères sur des plasmides, peuvent donc être orientées à volonté à des endroits précis dans le génome.

En outre, la recombinaison homologue, couplée avec des niveaux élevés des conversions génique a conduit à l'élaboration de techniques pour le remplacement direct des séquences d'ADN génétiquement modifiées dans leurs emplacements chromosomiques normales. Ainsi, des gènes sauvages normaux, même ceux n'ayant aucune mutation connue

précédemment, peuvent être convenablement remplacés par des allèles altérés. Les phénotypes survenant après l'altération des gènes de la levure ont grandement contribué à la compréhension de la fonction de certaines protéines *in vivo*. Les gènes peuvent être directement remplacés à haute efficacité dans la levure. Aussi spécifique à la levure, la transformation peut être procédée directement avec des oligonucléotides synthétiques monocaténaires courts, permettant les productions convenables de plusieurs formes altérées de protéines. Ces techniques ont été largement exploitées dans l'analyse de la régulation génétique, les relations structure-fonction des protéines, la structure des chromosomes et autres questions d'ordre générales en biologie cellulaire (Sherman, 2002).

Saccharomyces cerevisiae était le premier eucaryote dont le génome a été complètement séquencé (Goffeau, et al., 1996). Par la suite, la levure est devenu l'un des organismes clés pour la recherche en génomique (Kumar and Snyder, 2001), y compris une vaste utilisation des puces à ADN pour étudier le transcriptome (Gasch, 2002; Marc, et al., 2001) ainsi que l'analyse à l'échelle génomique des fonctions de gènes par altération génique (Oliver, 1996), d'analyse séquentielle de l'expression des gènes (SAGE) (Velculescu, et al., 1997), des cartes 2D des protéines (Blomberg, 2002; Garrels, et al., 1997), la localisation des protéines (Ross-Macdonald, et al., 1999), des activités enzymatiques (Hizicky, et al., 2002; Martzen, et al., 1999), des interactions protéine-protéine par l'analyse à double hybride (Ito, et al., 2000; Uetz, et al., 2000), et de la létalité synthétique de l'analyse fonctionnelle (Tong, et al., 2001). Par ailleurs, les séquences génomiques des espèces apparentées ont amélioré l'affectation des gènes et des motifs régulateurs dans *Saccharomyces cerevisiae* (Cliften, et al., 2003; Kellis, et al., 2003).

Les vertus impérieuses de la levure sont illustrées par le fait que des gènes chez les mammifères sont régulièrement introduits dans la levure pour des analyses systématiques des fonctions des produits des gènes correspondants. De nombreux gènes humains associés à des maladies ont des orthologues chez la levure (Ploger, et al., 2000), et la conservation élevée des mécanismes métaboliques et régulateurs a contribué à l'utilisation répandue de la levure comme un modèle eucaryotique pour des études biologiques diversifiées. En outre, la capacité de la levure de répliquer des chromosomes artificiels circulaires et linéaires a permis des études détaillées des télomères, des centromères, des dépendances de longueur et des origines de la réplication. L'ADN mitochondrial peut être modifié de manières définies par transformation (Bonnefoy and Fox, 2002). La facilité avec laquelle le génome de la levure peut être manipulé est véritablement sans précédent pour les autres eucaryotes.

4.2.3. Génome

Saccharomyces cerevisiae contient un ensemble haploïde de 16 chromosomes bien caractérisés, allant de 200 à 2 200 Kb. La séquence totale d'ADN chromosomique, constituant 12 052 Kb, a sorti en 1996 (Goffeau, et al., 1996), d'où un total de 6 183 ORFs de plus de 100 acides aminés ont été signalés. Par la suite, des comparions aux séquences d'espèces reliées à *Saccharomyces* a indiqué que *Saccharomyces cerevisiae* contient 5 773 (Cliften, et al., 2003) ou 5 726 (Kellis, et al., 2003) des gènes codant pour des protéines. A la

différence des génomes des organismes multicellulaires, le génome de la levure est très compact, avec des gènes qui représentent 72 % de la séquence totale. La taille moyenne des gènes de la levure est 1.45 kb, soit 483 codons, avec une plage de 40 à 4 910 codons. Un total de 3,8% des ORFs contiennent des introns (Sherman, 2002).

Les ARN ribosomaux sont codés par environ 120 exemplaires d'un tandem unique sur le chromosome XII. La séquence d'ADN a révélé que la levure contient 262 gènes d'ARNt, dont 80 ont des introns. En outre, les chromosomes contiennent des éléments d'ADN mobiles, rétrotransposons, qui varient en nombre et position dans différentes souches du *Saccharomyces cerevisiae*, sachant que la plupart des souches de laboratoire ayant environ 30.

L'ADN mitochondrial encode les composants de la machinerie de la traduction mitochondriale et environ 15 % des protéines mitochondriales. Les mutants ρ^o de la levure manquent complètement de l'ADN mitochondrial et sont déficient en polypeptides respiratoires synthétisées sur les ribosomes mitochondriaux, c'est-à-dire, du cytochrome b et les sous-unités du cytochrome oxydase et les complexes de l'ATPase. Même si les mutants ρ^o sont déficients du coté respiratoire, ils sont viables et conservent des mitochondries, bien que les mitochondries sont morphologiquement anormaux.

Les plasmides, d'un cercle de 2 μm, présents dans la plupart des souches de *Saccharomyces cerevisiae*, apparemment fonctionnent uniquement pour leur propre réplication. Généralement les souches *cir^o*, qui sont dépourvus d'ADN de 2 μm, n'ont généralement aucun phénotype observable. Toutefois, une certaine mutation chromosomique, *nib1*, provoque une réduction de croissance des souches *cir$^+$*, en raison d'un nombre anormalement élevé de copies de l'ADN de 2 μm (Holm, 1982; Sweeney and Zakian, 1989).

De même, presque toutes les souches de *Saccharomyces cerevisiae* contiennent des virus à dsRNA intracellulaire qui constitue environ 0,1 % du total des acides nucléiques. Les virus à ARN incluent trois familles avec dsRNA génomes, *L-A*, *L-BC* et *M*. Deux autres familles de dsRNA, *T* et *W*, se répliquent dans la levure, mais pour l'instant pas avérés être virale. Le dsRNA *M* encode une toxine, et L-*A* encode la protéine majeure de la capside et les composants requis pour la réplication virale et l'entretien de *M*. Les deux dsRNA, *M* et *L-A*, sont emballés séparément avec la protéine commune de la capside codée par *L-A*, résultant en des particules de type viral transmises cytoplasmiquement pendant la croissance végétative et la conjugaison. *L-B* et *L-C* (collectivement désignées par les numéros *L-BC*), similaire à *L-A*, ont une ARN-polymérase ARN-dépendante et sont présents dans les particules intracellulaires. Les Mutants *KIL-o*, dépourvus du dsRNA *M* et, par conséquent la toxine, sont facilement induite par la croissance à des températures élevées et des agents chimiques et physiques.

La levure contient également un ARN monocaténaire circulaire 20S qui apparait pour encoder une ARN-polymérase ARN-dépendante, qui agit comme un réplicon indépendant, et qui est hérité comme un élément génétique non-mendélien.

Seules les mutations des gènes chromosomiques présentent la ségrégation mendélienne 2:2 en tétrades après la sporulation des diploïdes hétérozygotes ; Cette propriété est dépendante de la disjonction des centromères chromosomiques. En revanche, l'hérédité non-mendélienne est observée pour les phénotypes associés à l'absence ou l'altération d'autres acides nucléiques.

4.3. Sources de données de la levure

Dans le cadre du travail décrit dans ce mémoire de thèse, nous allons faire un survol, dans cette section, sur quelques sources de données de la levure. Nous allons exclusivement parler des sources de données manipulées par *YeastMed*. Dans sa version actuelle. *YeastMed* intègre cinq sources de données de la levure. Elles ont été sélectionnées pour le fait qu'elles présentent des données complémentaires l'une à l'autre fournissant ainsi des potentialités adéquates pour étudier *Saccharomyces cerevisiae* que ça soit sur le plan génomique, protéomique, métabolomique ou interactomique.

4.3.1. SGD

SGD (Saccharomyces Genome Database[71]) (Cherry, et al., 2012) est la plus importante et la plus riche des bases de données spécialisées des levures. C'est une ressource de communauté consacrée à l'espèce *Saccharomyces cerevisiae*. Elle fournit des informations de haute qualité issues de la curation manuelle de la littérature évaluée par les pairs (peer-review literature[72]). Les données dans SGD sont organisées au tour des séquences du génome et ses gènes. SGD a comme but principale la présentation d'information sur les séquences d'ADN et ses composants individuels, ARNs, protéines codées et les structures et les fonctions biologiques de n'importe quel produit connu d'un gène.

4.3.1.1. Contenu

SGD a été choisi par la communauté de la levure afin de maintenir une approche cohérente de nomenclature pour *Saccharomyces cerevisiae*. Elle maintient une liste complète de tous les noms de gènes de *Saccharomyces cerevisiae*, appelé *le Registre de Nom de Gene*. Cette liste comprend tous les noms standards de locus dans la base de données, les noms "non-standards" (c-à-d. alias) et les noms "réservés" de locus.

SGD fournit ses informations à travers des entrées structurées sur plusieurs pages HTML selon un focus bien précis. Chaque entrée décrit un locus. Ainsi, les informations sur chaque ORF et/ou gène sont présentées directement sur une page appelée *Page de Locus* ou par le biais de liens hypertextes vers d'autres pages qui peuvent être atteintes via des onglets placés sur une barre d'onglets en haut de la *Page de Locus* (Figure 43).

[71] http://www.yeastgenome.org

[72] http://en.wikipedia.org/wiki/Peer_review

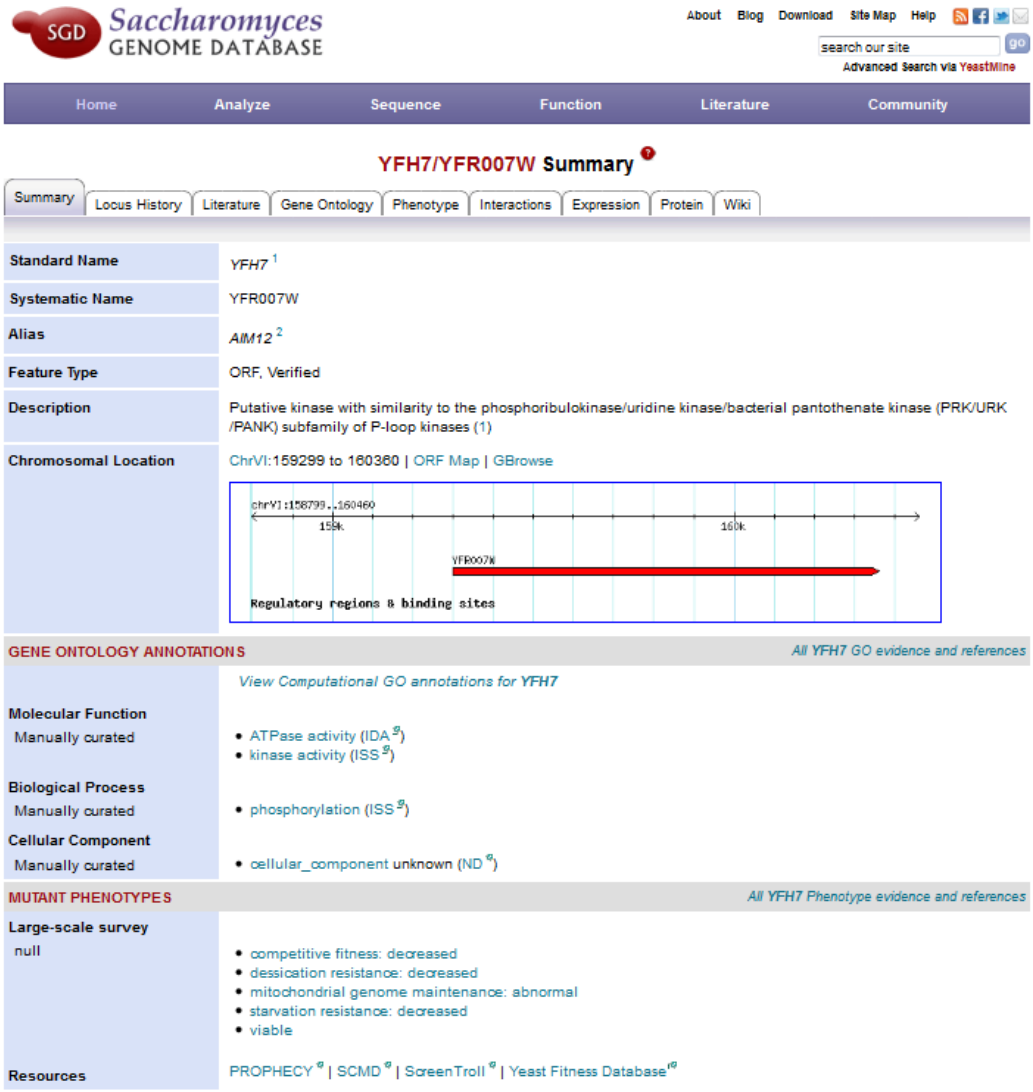

Figure 43. Exemple d'une entrée de la base de données SGD.

En plus de la curation manuelle à partir de la littérature et qui fournit une représentation de haute qualité des résultats expérimentaux publiés, SGD utilise des vocabulaires contrôlés (ou Ontologies) pour annoter davantage ces résultats. Dans ce contexte, l'annotation en utilisant des termes GO[73] permet à SGD de capturer des informations biologiques concernant des produits de gènes spécifiques sous une forme consultable et calculable qui peut être facilement comparée entre organismes (Hong, et al., 2008).

Des descriptions complètes de phénotypes de mutants sont également représentées à l'aide d'un vocabulaire contrôlé connu sous le nom *Ontologie de Phénotype d'Ascomycètes*

[73] www.geneontology.org

(*APO*[74]). Chaque annotation phénotype en SGD est représentée comme un observable, qui décrit l'entité ou le processus, couplé avec un qualificatif qui décrit le changement de cette entité ou processus. Les composés chimiques, qui sont souvent utilisés pour obtenir des phénotypes, sont spécifiés en utilisant un vocabulaire contrôlé appelé Ontologie ChEBI (Degtyarenko, et al., 2008).

SGD fournit aussi des informations sur les voies biochimiques de la levure. Elles sont aussi manuellement annotées et sont rendues disponible grâce au navigateur *Pathway Tools* (Karp, et al., 2010). L'interface du navigateur fournit une description complète de chaque voie avec des structures moléculaires, les numéros E.C. (Numéro de classifications d'enzyme) et une liste de références complètes. Le navigateur de voies biochimiques est accessible à partir de la section *Pathway* de la *Page de Locus* en suivant un lien hypertexte.

4.3.1.2. Accès aux données.

Toutes les annotations, les séquences et les ensembles de données à haut débit contenues dans SGD sont accessibles par les utilisateurs via des points d'accès et des méthodes multiples. Les données sont intégrées pour affichage sur des pages spécifiques de locus et peuvent être analysées à l'aide d'outils Web, ou téléchargées pour une utilisation personnalisée.

Les dernières améliorations apportées aux pages de locus permettent maintenant le téléchargement des informations détaillées. Par exemple, les données de phénotypes sont disponible en sélectionnant le bouton '*Download Data*' accessible depuis l'onglet '*Phenotype*' d'une '*Page de Locus*'. Un fichier tabulaire (tab-delimited file) de toutes les données d'interactions d'un gène peut être obtenu en cliquant sur le lien intitulé '*Download Unfiltered Data*' ou via le lien '*Download options*' qui passe la liste des gènes à *YeastMine*[75] (Balakrishnan, et al., 2012), où les chercheurs peuvent explorer la riche collection des données associées de SGD, y compris les interactions génétiques et physiques. YeastMine est un outil de recherche et de récupération multiforme qui fournit un accès à des types différents de données. Les recherches peuvent être entreprises avec une liste de gènes, une liste de termes de GO, ou des résultats de recherches multiples d'une variété de types de données. A chaque étape, les résultats peuvent être combinés pour une analyse plus approfondie. La recherche et ses résultats peuvent être sauvegardés ou téléchargés dans des formats de fichier personnalisable. YeastMine peut actuellement interroger les types de données suivants: les annotations GO, les caractéristiques chromosomiques, les coordonnées, les annotations de phénotypes, les données des interactions, les orthologues, l'expression de gènes et la littérature. YeastMine a beaucoup d'autres fonctionnalités telles que des services Web et des widgets personnalisés qui permettent l'affichage et la récupération des données.

[74] Disponible à partir de la Fonderie OBO, http://www.obofoundry.org

[75] http://yeastmine.yeastgenome.org

L'outil de recherche Textpresso[76] (Muller, et al., 2004) est disponible pour la recherche d'environ 50 000 documents recueillis par le projet SGD. Textpresso donne accès au texte intégral et aux résumés de documents qui ont été extraites pour une curation potentielle. Bien que des documents n'ont pas suffisamment d'informations à inclure dans SGD, ces documents peuvent encore être généralement utile et sont donc inclus dans le catalogue Textpresso. Alors que la recherche plein texte est disponible à partir de nombreuses autres sources, Textpresso fournit la capacité unique de découvrir rapidement et efficacement une collection de littérature à l'aide de requêtes très spécifiques et complexes, comme par exemple trouver toutes les phrases contenant une description de deux gènes en interaction.

La base de données SGD fournit aussi un site spécifique de téléchargement[77] où il est possible de télécharger toutes les données qu'elle contient sous des formats standards (texte, GFF, tabulaire, …etc.).

4.3.2. YEASTRACT

Yeastract[78] (YEAst Search for Transcriptional Regulators And Consensus Tracking) (Teixeira, et al.) est une base de données publique qui fournit des informations actualisées sur les associations de régulation entre les facteurs de transcription et les gènes de la levure *Saccharomyces cerevisiae*. Elle rend aussi possible la comparaison entre des motifs d'ADN, tels que ceux trouvés surreprésentées dans les régions promotrices de gènes co-régulés, et les sites de liaison de facteurs de transcription décrits dans la littérature. Le système fournit également un mécanisme utile pour le regroupement d'une liste de gènes (par exemple un ensemble de gènes des mêmes profils d'expression, tel que révélées par une analyse de biopuces) en se basant sur Leurs associations de régulations avec des facteurs de transcription connues (Figure 44).

4.3.2.1. Contenu

Après sa dernière mise à jour (Abdulrehman, et al., 2011), Yeastract est devenue un référentiel de plus de 48200 associations de régulation entre les facteurs de transcription et ses gènes cibles, basées sur des preuves expérimentales qui se sont répandues dans plus de 1200 références bibliographiques. Toutes les associations sont annotées manuellement après examen des références pertinentes. En outre, Yeastract comprend 298 sites de liaison spécifiques pour plus d'une centaine de facteurs de transcription caractérisés. Des informations supplémentaires sur chaque gène inclus dans la base de données ont été extraites à partir de la base de données SGD, des outils d'analyse de séquences de régulations et du consortium de l'ontologie de gène (GO).

Etant donné que les trois principales entités impliquées dans la régulation des gènes sont les concepts gène, protéine et site de liaison (ou consensus), la base de données est

[76] http://textpresso.yeastgenome.org

[77] http://downloads.yeastgenome.org

organisée sur ces trois concepts. Ces concepts sont reliés par des relations de régulation qui sont de deux types : documentées et potentielles.

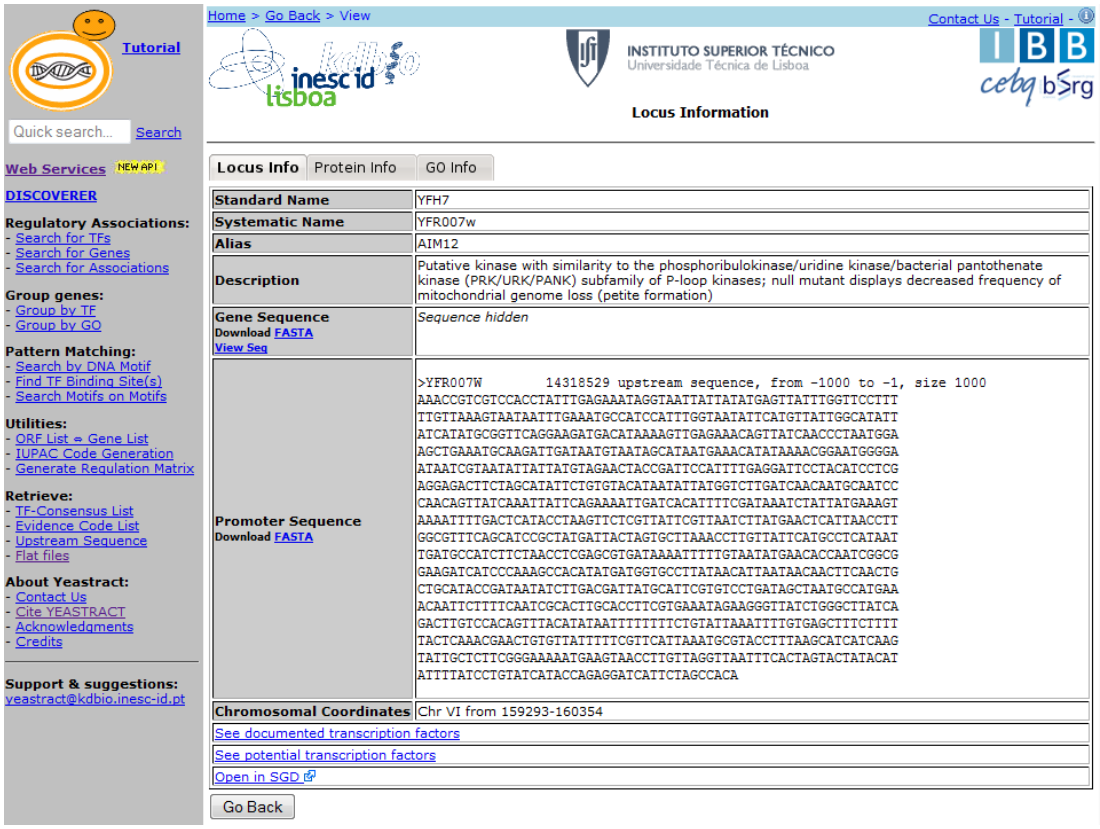

Figure 44. Exemple d'entrée de la base de données Yeastract.

Une association documentée entre un facteur de transcription et un gène cible est supportée par des données publiées issues de preuves expérimentales. Chaque association documentée est annotée par la référence (ou les références) qui fournit l'information avec un lien hypertexte à PubMed[79].

Une association potentielle entre un facteur de transcription et un gène cible est basée sur l'occurrence de site de liaison du facteur de transcription dans le promoteur du gène cible. Les sites de liaison qui ont été considérés pour chaque facteur de transcription sont supportés par des expériences de footprinting (Galas and Schmitz, 1978) ou ChIP (Chromatin Immunoprecipitation) (Collas, 2010).

4.3.2.2. Accès aux données.

Yeastract rend disponibles les informations stockées dans la base de données à travers un ensemble de requêtes et d'outils. Les principales requêtes sont :

---

[78] http://www.yeastract.com

[79] http://www.ncbi.nlm.nih.gov/pubmed/

- chercher des facteurs de transcription par des gènes régulés.
- chercher des gènes régulés par des facteurs de transcription spécifiques.
- grouper les gènes par facteurs de transcription.
- chercher par motif d'ADN.
- chercher des associations de régulations entre des facteurs de transcriptions et des gènes.

Dans sa dernière version (Abdulrehman, et al., 2011), Yeastract a rendu aussi possible l'accès programmatique à ses données via un ensemble de services Web[80]. Ainsi, les utilisateurs peuvent développer un code, exprimant une vue (ou des vues) sur les données, pour interroger, retrouver et exploiter les données selon des besoins spécifiques.

Finalement, il est possible de télécharger quelques fichiers plats de données à partir de la page de téléchargement disponible sur le site Web[81]. Ces fichiers concernent principalement les séquences des promoteurs, séquences de gènes, séquences protéiques, les régulations documentées et la liste des sites de liaisons présentes dans la base de données. D'autres fichiers peuvent être téléchargés sous demande.

4.3.3. MIPS-CYGD

La base de données CYGD[82] (Comprehensive Yeast Genome Database) (Guldener, et al., 2005) maintenue par MIPS[83] (Munich Information Center for Protein Sequences) (Mewes, et al., 2011) est une ressource d'information sur les fonctions cellulaires de la levure *Saccharomyces cerevisiae* et des espèces reliées choisis comme organismes eucaryotes modèles. La base de données sert comme une ressource commune développée et maintenue par un groupe de bases de données européens et laboratoires de levures formant un réseau décentralisé d'expertise dans le but de fournir des informations détaillées sur des séquences codantes pour protéines ainsi que autres éléments génétiques.

4.3.3.1. Contenu

Les informations présentées par la base de données CYGD se focalisent sur une protéine de la levure *Saccharomyces cerevisiae*. Chaque protéine annotée a une entrée dans la base de données (Figure 45). CYGD utilise un ensemble de catalogues dans l'annotation des protéines. Un catalogue est une classification hiérarchique, flexible et évolutive de vocabulaire contrôlé bien précis. Les catalogues qui ont été développés concernent principalement les fonctions (FunCat), les localisations, les classes, les phénotypes et les complexes protéiques (Pu, et al., 2009); en plus des nomenclatures EC (Enzyme

[80] http://www.yeastract.com/services/

[81] http://www.yeastract.com/download.php

[82] http://mips.helmholtz-muenchen.de/genre/proj/yeast/

[83] http://www.helmholtz-muenchen.de/en/ibis

Classification) et des transporteurs/protéines membranaires. Le catalogue fonctionnel (FunCat (Ruepp, et al., 2004)) permet une description fonctionnelle des protéines ; il a été développé et utilisé pour la première fois pour annoter le génome de la levure (Mewes, et al., 1997). Le catalogue PPI (Protein-Protein Interaction), le catalogue des complexes protéiques et le catalogue de la localisation des protéines permettent d'obtenir des informations reliées à la proximité des protéines dans *Saccharomyces cerevisiae*. Plus de 15600 interactions protéine-protéine ont été manuellement compilées à partir de la littérature en plus d'un millier de complexes manuellement extraits ou dérivés des expérimentations à large échelle. La plus grande majorité est documentée par des références bibliographiques issues du PubMed.

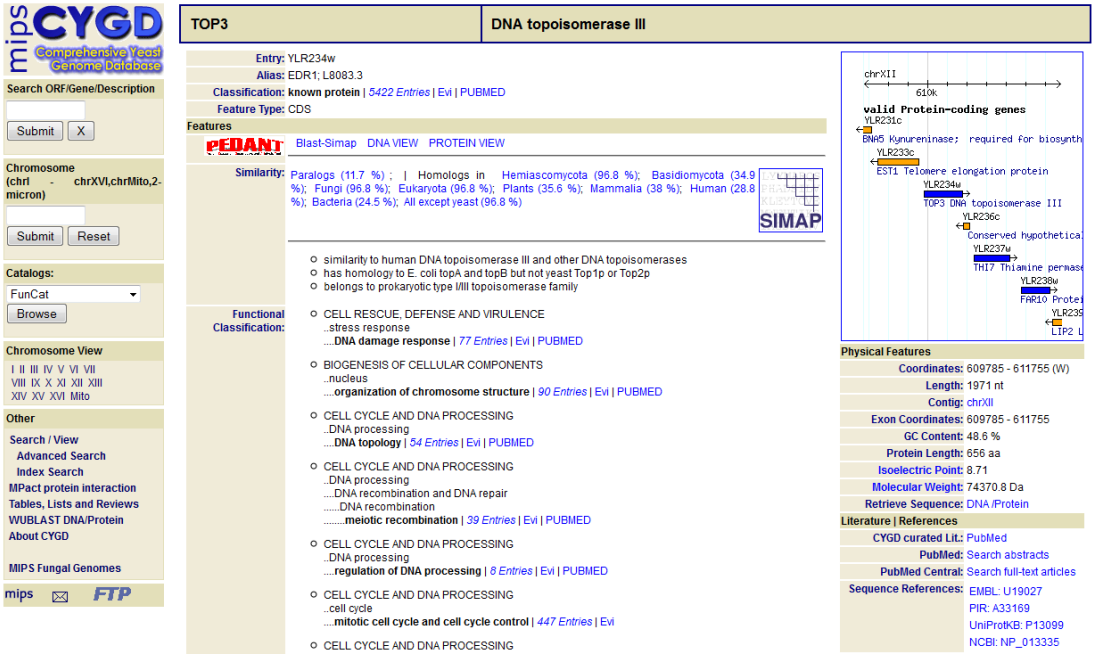

Figure 45. Exemple d'entrée de CYGD.

Pour des informations sur les protéines de transport membranaires, CYGD intègre la base de données YTPDB[84] (Yeast Transport Protein DB) (Brohee, et al., 2010) donnant ainsi accès à 298 protéines de levure classées comme transporteurs membranaires établis ou prédits. Pour les transporteurs reconnus sur une base expérimentale et des critères de séquence, la littérature a été scannée pour retrouver deux types d'information : le composant (ou les composants) chimique reconnu par la protéine, et la localisation cellulaire de la protéine. Pour les deux types d'informations, des vocabulaires contrôlés ont été utilisés pour définir des listes de termes organisés sous forme d'arbres et liés à des tables de synonymes. La classification phylogénétique des transporteurs et d'autres

[84] http://homes.esat.kuleuven.be/~sbrohee/ytpdb/index.php/Main_Page

protéines membranaires est aussi intégrée dans CYGD comme catalogue (De Hertogh, et al., 2002).

La base de données contient aussi une révision d'introns[85] (Bon, et al., 2003) du *Saccharomyces cerevisiae*. Elle a aussi analysé les structures des séquences des extrémités 3'. Par conséquent les données de plus de 1540 sites ploy(A) pour plus de 920 gènes ont été incorporées dans des pages d'entrée individuelles.

D'autre part, la collection de facteurs de transcription, leurs gènes cibles et sites de liaisons dans CYGD est structurellement basée sur la base de données TRANSFAC® (Matys, et al., 2003). Ainsi, elle contient non seulement des informations pertinentes sur les facteurs de transcription, leurs gènes cibles et les sites de liaisons de régulation, mais aussi elle a en plus un tableau avec des matrices de poids de position dérivées à partir des collections des sites de liaisons pour des facteurs donnés. D'autre part, des informations sur des voies et processus cellulaires dans *Saccharomyces cerevisiae* sont fournis à travers un lien vers l'interface Web de la base de données aMAZE (Lemer, et al., 2004). Des informations sur des voies de transduction de signal sont aussi fournies. Ces informations sont organisées d'une façon similaire aux voies métaboliques à l'exception que toutes les interactions sont modélisées comme des transformations spécialisées comme *Expression* (de gènes), *Assembly* (des entités biochimiques), *Translocation* (des entités biochimiques entre des localisations biochimiques) et *Reaction* (principalement modifiant les entités biochimiques).

4.3.3.2. Accès aux données

L'accès et la recherche de données dans CYGD se fait à travers son interface Web par plusieurs manières. Ainsi, la méthode la plus simple est à travers un formulaire de recherche qui permet de soumettre rapidement une requête pour chercher des informations sur une protéine dans la base de données en se basant sur des mots clés simples (nom de la protéine ou du gène…). Ce formulaire permet aussi l'obtention d'un listing d'éléments génétiques qui se trouvent sur un chromosome donné ou de naviguer dans les différents catalogues pour chercher des informations spécifiques. La base de données dispose aussi d'un autre formulaire de recherche plus avancé qui permet de réaliser des requêtes plus complexes et plus raffinées.

En plus des formulaires de recherches, Les données des interactions protéine-protéine sont particulièrement accédées à travers de l'interface MPact (Guldener, et al., 2006) qui support la recherche et la récupération des données dans le format standardisé PSI-MI (Hermjakob, et al., 2004). MPact fournit à l'utilisateur des formulaires de requêtes intuitifs pour récupérer rapidement les interactions d'intérêt et aussi des représentations graphiques permettant une navigation facile à travers les réseaux d'interactions protéiques.

[85] http://mips.helmholtz-muenchen.de/proj/yeast/reviews/intron/

CYGD donne aussi la possibilité de télécharger ses données à travers son site de téléchargement[86] sous différentes formats.

4.3.4. BioGRID

BioGRID (Biological General Repository for Interaction Datasets) (Stark, et al., 2011) est une base de données publique qui archive et diffuse des informations sur les interactions génétiques et physiques de plusieurs organismes modèles y compris la levure *Saccharomyces cerevisiae*. Les interactions de BioGRID sont annotées à partir des ensembles de données des études individuels ou à haut débit. Dans sa version 3.1.89 du juin 2012, BioGRID contient 302013 interactions dont 113476 sont des interactions physiques et 188537 sont des interactions génétiques annotées respectivement à partir de 6474 et 6569 publications. Ces ensembles de données sont disponible à partir du site Web du BioGRID[87] que ça soit pour recherche ou téléchargement.

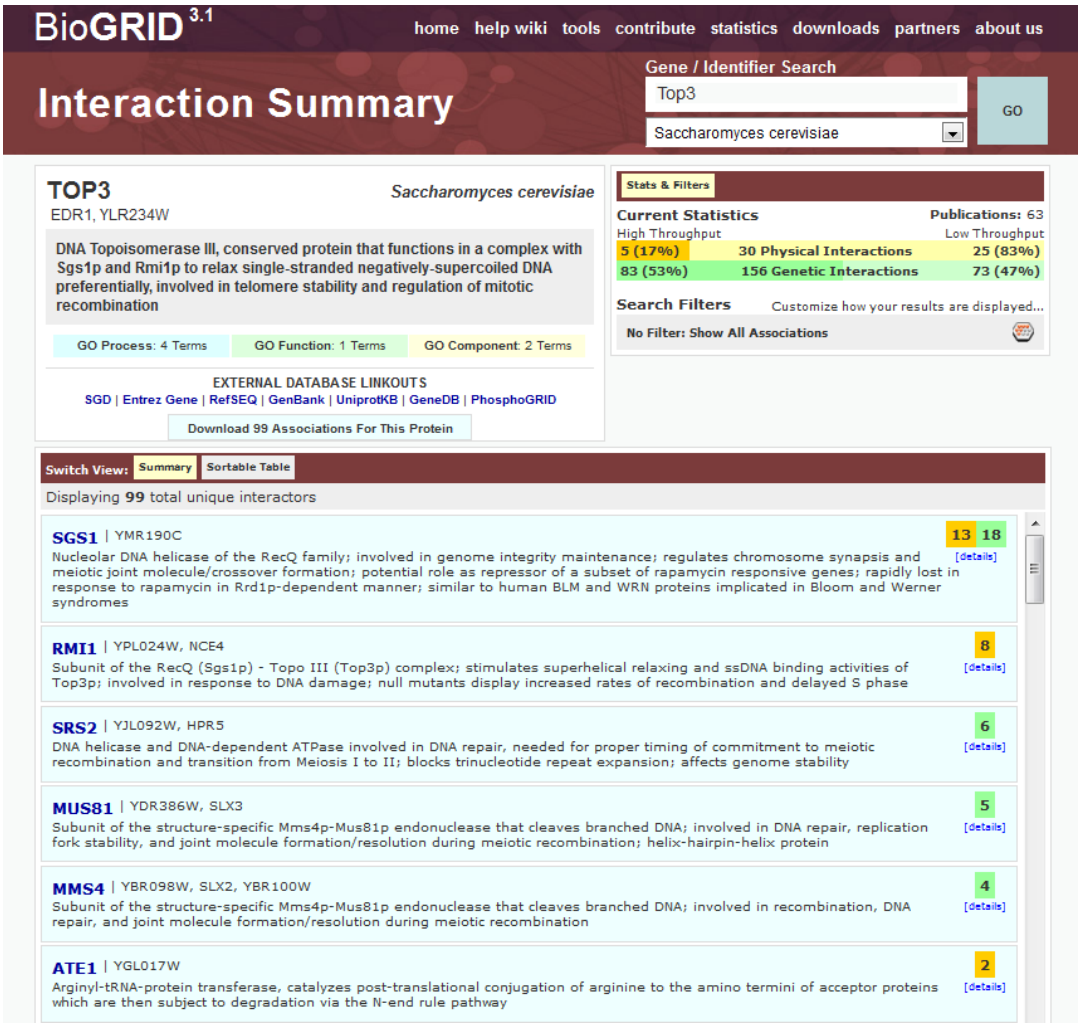

Figure 46. Exemple d'entrée de la base de données BioGRID.

[86] ftp://ftpmips.gsf.de/yeast/

[87] http://www.thebiogrid.org

Un formulaire de recherche simple est disponible sur le site Web de la base de données pour la recherche des interactions d'une protéine (Figure 46). La recherche peut être faite en introduisant un identifiant d'un gène/protéine, un identifiant PubMed ou des mots clés d'une publication. Tous les enregistrements d'interactions de BioGRID sont librement téléchargeables à partir de son site de téléchargement[88] sous forme d'une variété de formats standards incluant les formats PSI-MI et tabulaire.

4.3.5. PhosphoGRID

PhosphoGrid[89] (Stark, et al., 2010) est une base de données publique présentant des informations sur les sites de phosphorylation expérimentalement vérifiés *in vivo* et qui sont annotés à partir de la littérature primaire de la levure *Saccharomyces cerevisiae*. La base de données enregistre les positions de plus de 6430 sites de phosphorylation sur plus de 1500 produits de gènes. Les sites de phosphorylation in vivo sont documentés par une hiérarchie de codes d'évidences expérimentales.

La base de données, quand il est disponible pour des sites spécifiques, fournit aussi des informations sur les enzymes kinases et/ou phosphatases pertinentes à la protéine, les conditions spécifiques sous lesquelles la phosphorylation se produit et les effets que la phosphorylation a sur la fonction de la protéine (Figure 47). L'accès aux informations de phosphorylation d'un produit de gène se fait à travers d'un formulaire de recherche simple disponible sur le site Web de la base de données. La recherche se fait en fournissant un nom de protéine ou de gène. La page d'une protéine dans PhosphoGRID fournit sa séquence en acides aminées avec tous les sites de phosphorylation documentés et expérimentalement vérifiés en rouge. Une fois la sourie survole sur un site de phosphorylation, une fenêtre pop-up apparaît et fournit un résumé sur l'évidence du site de phosphorylation et les conditions sous lesquelles il se produit avec la conséquence fonctionnelle si elle est connue. Les séquences consensus d'un nombre limité de kinases de protéines avec une spécificité bien définit, qui se chevauchent avec les sites de phosphorylation vérifiés, sont indiquées par un texte bleu dans la séquence en acides aminées. Des tables en dessous de la séquence de protéine fournissent plus de détails sur chaque site de phosphorylation expérimentalement identifié, incluant l'évidence expérimentale, les conséquences fonctionnelles et l'identité des protéines kinases et/ou phosphatases apparentées et, le cas échéant, les sous-unités de régulation spécifiques. Pour les protéines kinases et phosphatases elles-mêmes, et leurs sous-unités de régulations correspondantes, une table additionnelle montre les sites de phosphorylation/déphosphorylation pour les substrats protéiques connus et inclut un résumé des évidences de l'implication dans ces réactions, ainsi que des liens vers les pages correspondantes des substrats.

[88] http://thebiogrid.org/download.php

[89] http://www.phosphogrid.org

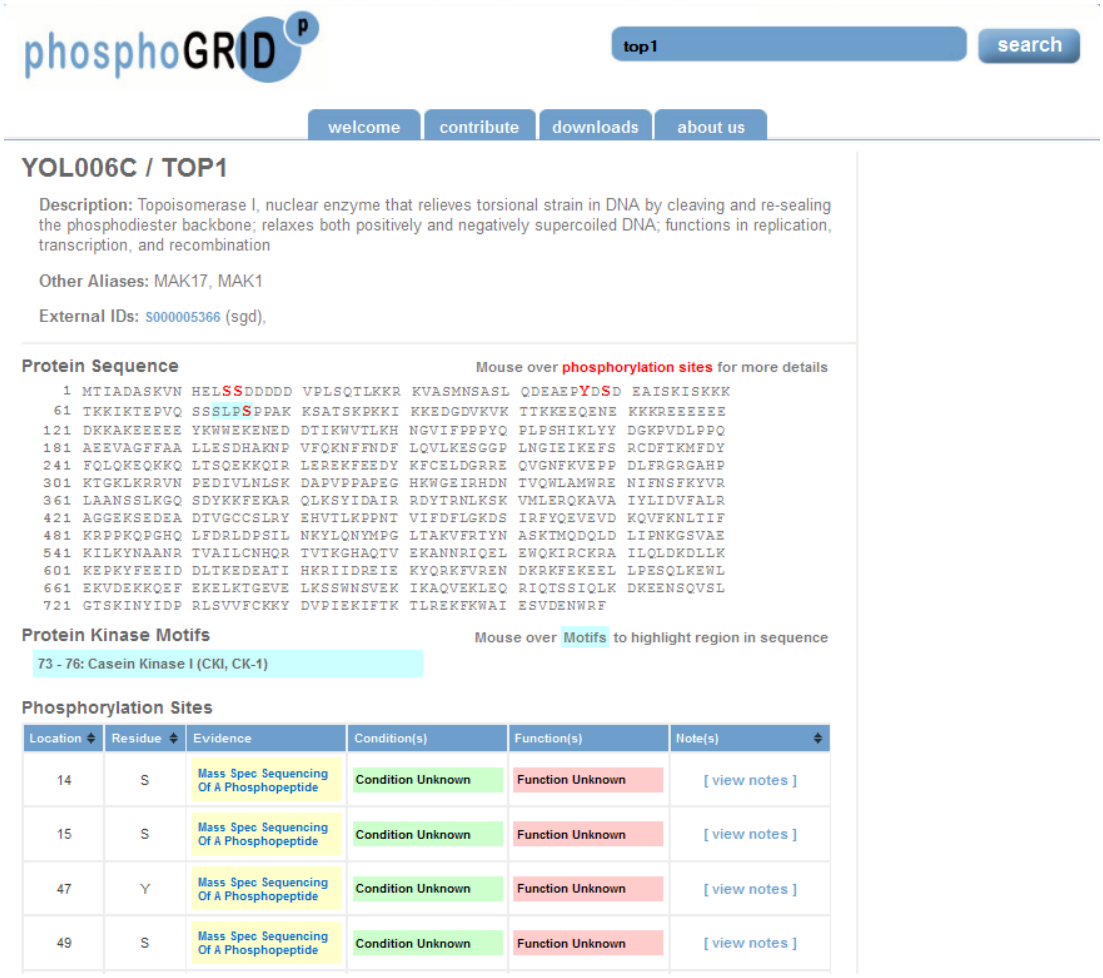

Figure 47. Exemple d'entrée de la base de données PhosphoGRID.

Toutes les données de la base de données PhosphoGRID sont librement téléchargeables en format fichier texte à partir de la page de téléchargement[90].

4.4. Modélisation et conception générale du Système YeastMed

La modélisation est le pilier de toute activité qui conduit au déploiement de logiciels de qualité. Les modèles sont construits pour spécifier la structure et le comportement attendu d'un système. Ils permettent de visualiser et de contrôler son architecture, ce qui permet de mieux comprendre l'organisation afin de gérer les risques encourus. Dans cette optique, plusieurs modèles ont été construits pour modéliser les différentes vues (fonctionnelles, statiques et dynamiques) du *YeastMed*. La modélisation a été réalisée grâce à UML (Unified Modeling Language)[91] (Booch, et al., 2005).

[90] http://www.phosphogrid.org/downloads.php

[91] http://www.omg.org/spec/UML/

4.4.1. UML

UML est un langage graphique qui permet de modéliser des applications au cours de leur phase de conception. Il aide le concepteur à exprimer ses besoins et à construire l'architecture de son application. Il est très bien adapté à la modélisation des applications bioinformatiques distribuées basées sur le Web.

UML n'est pas une méthode (*i.e.* une description normative des étapes de la modélisation) : ses auteurs ont en effet estimé qu'il n'était pas opportun de définir une méthode en raison de la diversité des cas particuliers. Ils ont préféré se borner à définir un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système d'information. Pour plus d'informations sur les diagrammes UML, voir l'annexe C.

Dans ce qui suit nous allons donner quelques aspects de modélisation du système *YeastMed* en utilisant quelques diagrammes UML.

4.4.2. Diagramme des cas d'utilisation du système

Les diagrammes de cas d'utilisation que propose UML permettent de recueillir, d'analyser et d'organiser les besoins, et de recenser les grandes fonctionnalités d'un système. Il s'agit donc de la première étape UML d'analyse d'un système. Il capture le comportement d'un système, d'un sous-système, d'une classe ou d'un composant tel qu'un utilisateur extérieur le voit. Il scinde la fonctionnalité du système en unités cohérentes, les cas d'utilisation, ayant un sens pour les acteurs. Les cas d'utilisation permettent aussi d'exprimer le besoin des utilisateurs d'un système, ils sont donc une vision orientée utilisateur de ce besoin au contraire d'une vision informatique. Cette première étape est nécessaire pour produire un logiciel conforme aux attentes des utilisateurs.

4.4.2.1. Définition des acteurs et des scénarios du YeastMed

Un acteur représente un rôle joué par une entité externe (utilisateur humain, dispositif matériel ou autre système) qui interagit directement avec le système. Un acteur peut consulter et/ou modifier directement l'état du système, en émettant et/ou en recevant des messages susceptibles d'être porteurs de données.

YeastMed est un système qui permettra à ses utilisateurs de trouver des données à partir des bases de données distantes. *YeastMed* interagira donc avec des utilisateurs et des bases de données externes. Les deux sont des acteurs et interagissent avec le système de manière différente.

Un acteur *Utilisateur* peut être n'importe quelle personne (Généticien, Bioinformaticien, Biologiste…) s'intéressant aux données biologiques du *Saccharomyces cerevisiae* ou application qui cherche à interroger et extraire des données à travers *YeastMed*. Ils peuvent formuler une requête dans le but de la soumettre au système, recevoir des réponses et visualiser, extraire et/ou sauvegarder leurs données.

Un acteur *Source de Données* est une des bases de données intégrées par le système. Il interagit principalement avec *YeastMed* en répondant à une requête générée par le système.

Comme tout système, *YeastMed* nécessite une maintenance régulière qui permet de gérer les problèmes qui peuvent émerger lors de l'utilisation du système. Vue aussi que le système vise à intégrer d'autres sources de données dans le future, des mise à jours lors de l'ajout d'une source seront obligatoires. Ces scénarios définissent un troisième acteur qui est l'*Administrateur* du système. L'acteur *Administrateur* est aussi une spécialisation de l'acteur *Utilisateur* puisqu'il peut aussi avoir les même cas d'utilisation d'un *Utilisateur*. La Figure 48 résume ceci dans un diagramme des cas d'utilisation.

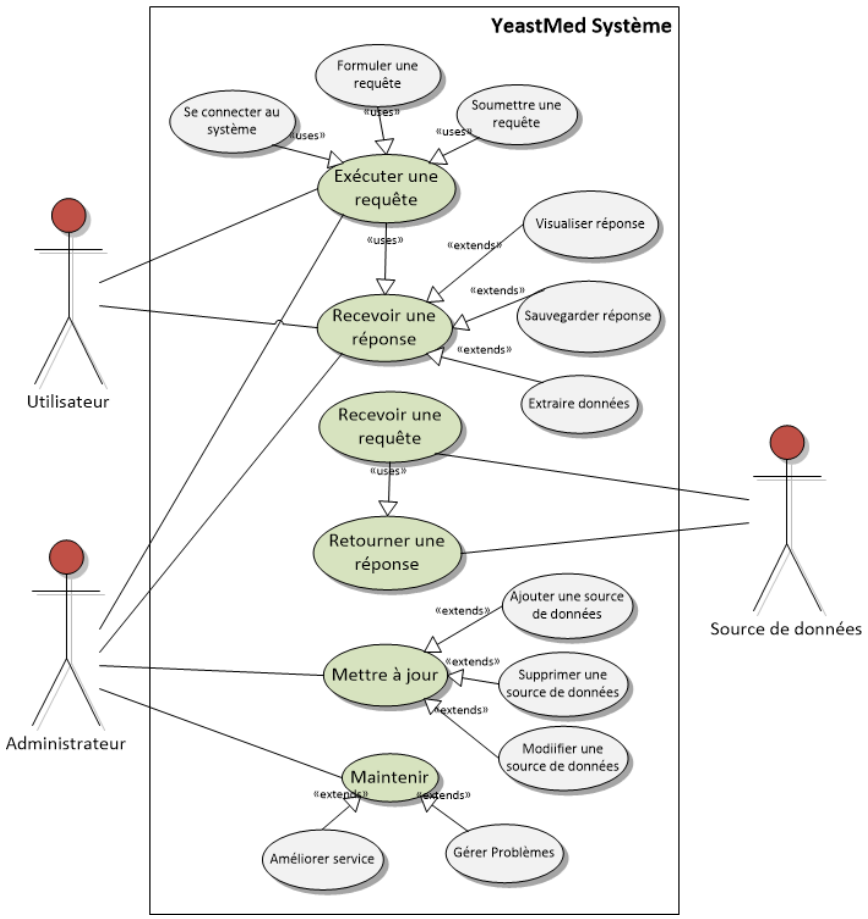

Figure 48. Les principaux cas d'utilisation de YeasMed.

4.4.3. Diagrammes de séquence

Pour donner plus d'explications sur les cas d'utilisation de l'acteur *Utilisateur*, nous avons fait recoure à un diagramme de séquence pour montrer comment les enchaînements des cas d'utilisation se succèdent. Un diagramme de séquence est une représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique. Dans un diagramme de séquence, on représente l'acteur principal à gauche du diagramme (l'acteur *Utilisateur* dans ce cas), et les acteurs secondaires à droite du système (ici l'acteur *Source de données*). Le but étant de décrire comment se déroulent les actions entre ces deux acteurs. La dimension verticale du diagramme représente le temps, permettant de

visualiser l'enchaînement des actions dans le temps. Les périodes d'activités des acteurs sont symbolisées par des rectangles verticaux, et ces acteurs dialoguent par le biais de messages.

Pour formuler une requête à *YeastMed*, l'utilisateur doit d'abord se connecter au système à travers un ordinateur (client). L'ordinateur client, dans ce cas, joue le rôle de l'acteur *Utilisateur*. Une fois la requête est formulée, l'utilisateur la soumet à *YeastMed* qui doit faire appel à un module de réécriture de requêtes pour générer un ensemble des sous-requêtes à destination des sources de données. Ces dernières reçoivent les sous-requêtes et chacune d'elles engendre une réponse à la sous-requête qu'elle avait reçue. Les sources de données retournent ces réponses à *YeastMed* qui va les combiner en une réponse à destination de l'utilisateur. L'utilisateur ensuite peut visualiser, stocker ou naviguer à travers les données reçues. La Figure 49 suivante résume ceci dans un diagramme de séquence.

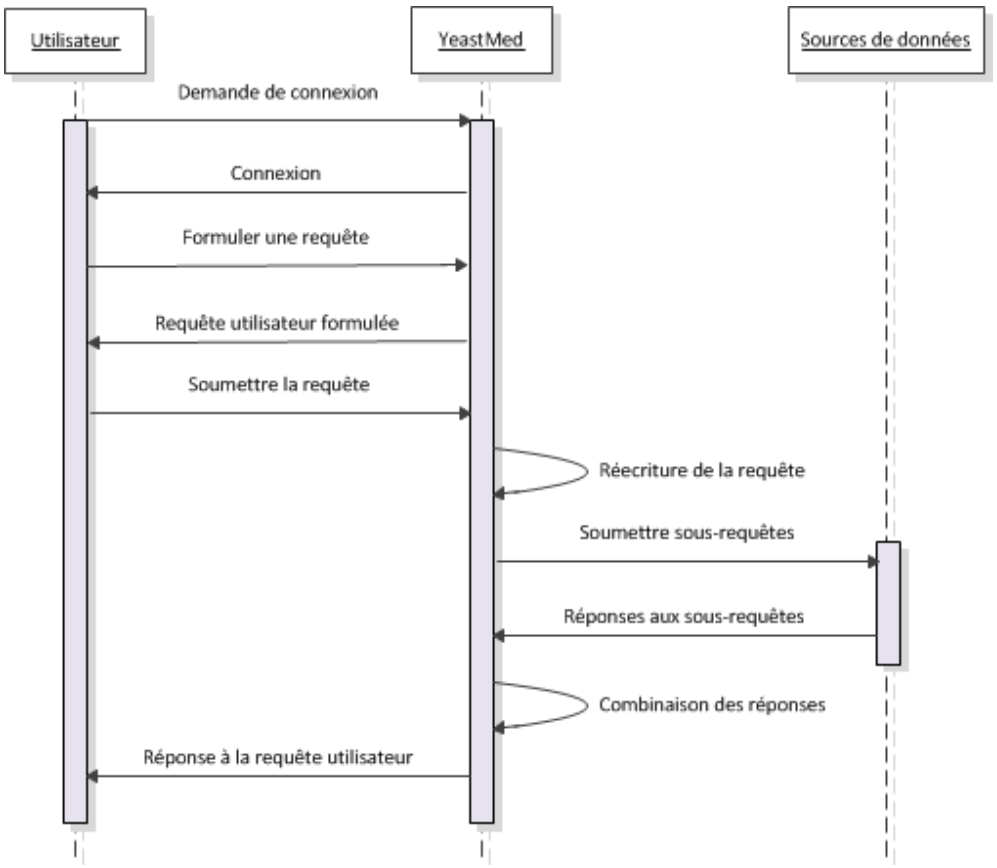

Figure 49. Diagramme de séquence de l'acteur Utilisateur.

Pour aller plus dans les détails, nous avons analysé les tâches et scénarios que *YeastMed* doit intérieurement accomplir. Ceci nous a permis de définir par la suite une liste d'objets (qui seront convertit en des éventuels composants du système) qui échangent des messages entre eux pour assurer le déroulement de l'enchainement des actions.

Le scénario général du fonctionnement attendu du *YeastMed* peut être résumé comme suit : Une fois connecté, l'utilisateur interagit avec le système à travers son *Interface Web*. Il formule une requête en termes d'un schéma global qui fournit un ensemble de vocabulaires contrôlés modélisant les sources intégrées et leurs Données. L'utilisateur par la suite soumet la requête qui va être capturée par un processeur de requête. Le rôle de ce dernier est de décomposer la requête de l'utilisateur en plusieurs sous-requêtes adaptées et à destination des sources de données cibles qualifiées comme susceptibles de répondre partiellement ou complètement à la requête de l'utilisateur. Pour rester dans le cadre d'un système basé sur l'approche de médiation, les sources de données seront accessibles à travers des structures qui permettent l'accès et l'interrogation des sources de données et l'extraction de données en jouant, entre autres, le rôle des adaptateurs. Les réponses retournées seront destinées à un module d'intégration qui va assurer la combinaison des réponses partielles des sources de données en une réponse intégrale qui sera considérée comme une réponse finale à la requête de l'utilisateur.

L'analyse du ce scénario nous a permis de définir certains objets (ou composants) qui interagissent et échangent des messages entre eux dans le système *YeastMed*:

- Interface Web : elle permet à l'utilisateur, une fois connecté, de formuler et de soumettre une requête.
- Processeur de requêtes : il permet de décomposer une requête reçue en sous-requêtes et de les envoyer aux sources de données convenables.
- Intégrateur : il est responsable de l'intégration des réponses des sous-requêtes en une seule réponse intégrale qui sera considérée comme une réponse finale à la requête reçue par le processeur de requêtes.
- Adaptateurs : ce sont les structures qui permettent d'interroger les sources de données et d'extraire des données spécifiques à partir de leurs réponses.

Les messages qui peuvent être échangés entre ces composants sont présentés dans la table 2 suivant :

Nom du message	**Emetteur**	**Receveur**
Soumettre Requête	Interface Web	Processeur de requêtes
Décomposer et réécrire la requête	Processeur de requêtes	Processeur de requêtes
Identifier les sources à interroger	Processeur de requêtes	Processeur de requêtes
Exécuter les Sous-requêtes	Processeur de requêtes	Adaptateurs
Adapter les requêtes aux sources	Adaptateurs	Adaptateurs
Interroger les sources de données	Adaptateurs	Adaptateurs
Extraire les données sollicitées	Adaptateurs	Adaptateurs
Créer les réponses partielles adaptées	Adaptateurs	Adaptateurs

Répondre aux sous-requêtes	Adaptateurs	Intégrateur
Combiner les réponses partielles	Intégrateur	Intégrateur
Adapter la réponse finale	Intégrateur	Intégrateur
Retourner la réponse finale	Intégrateur	Interface Web

Table 2. Messages qui peuvent être échangés entre les composants spécifiés du système.

La Figure 50 suivante montre le diagramme de séquence qui modélise ce que nous venons d'analyser.

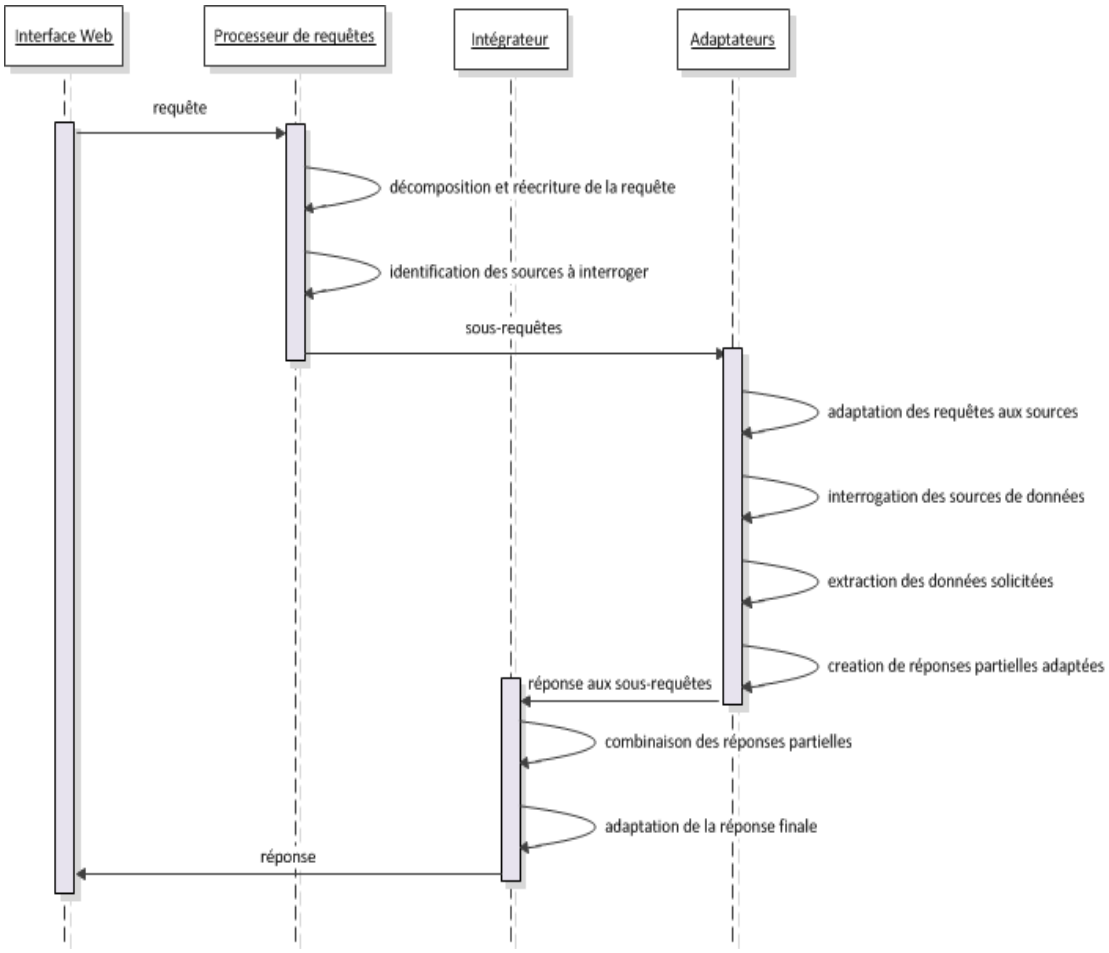

Figure 50. Diagramme de séquences générale du système YeastMed.

4.5. Architecture générale du YeastMed

Après l'analyse et la modélisation des différentes facettes et besoins du bon fonctionnement du *YeastMed*, nous avons pu définir l'architecture générale du système. *YeastMed* se veut être un système d'intégration de données qui a pour but de fournir un accès transparent à des sources de données distantes et hétérogènes de la fameuse levure *Saccharomyces cerevisiae* à travers une interface unique. C'est un système qui adopte une architecture médiateur basée sur KOMF. *YeastMed* est formé d'un ensemble de composants qui ont été développés indépendamment et contribuent au processus de

l'intégration de données de manières différentes. La Figure 51 montre l'architecture générale du système avec les éléments échangés entre ses composants.

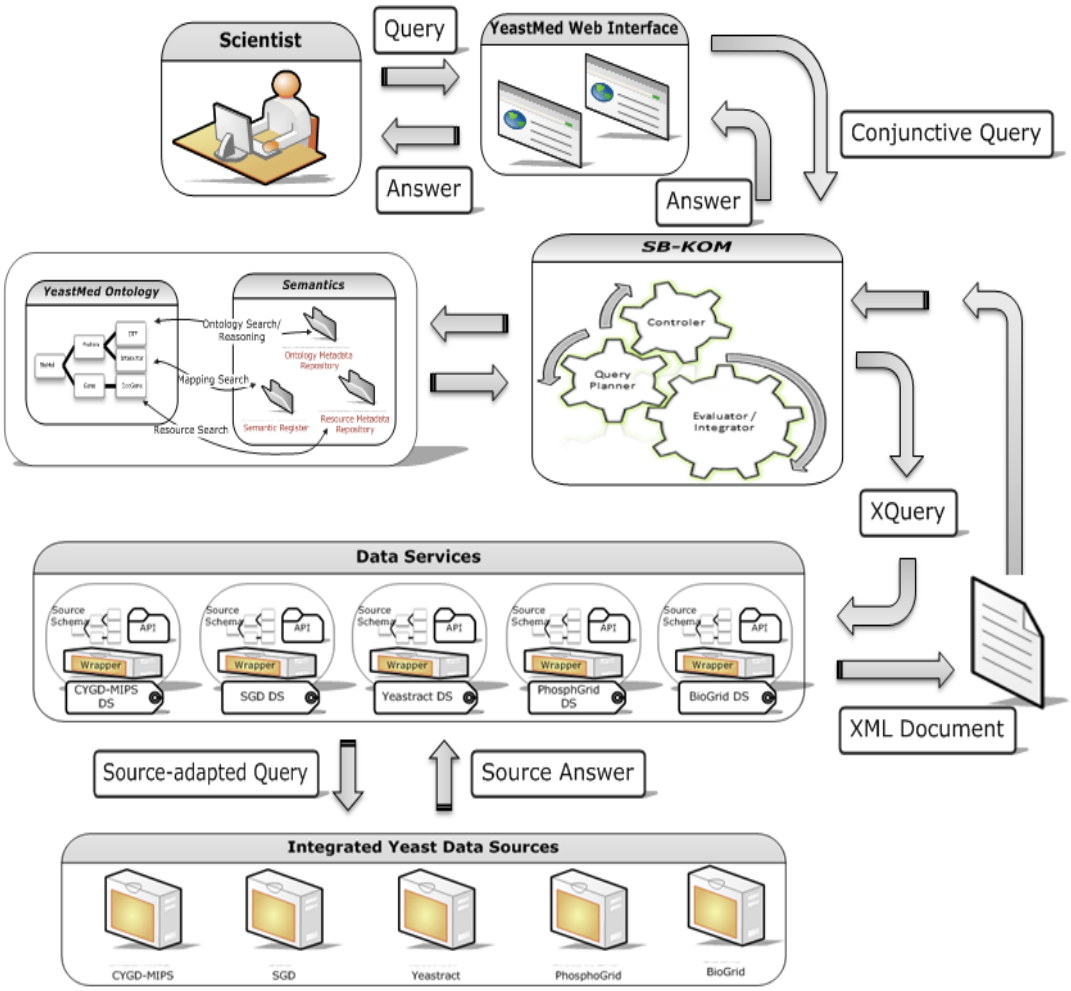

Figure 51. General Architecture of YeastMed System

Le point d'accès à *YeastMed* est une interface Web qui fournit deux types de formulaires de recherche : un formulaire pour une recherche rapide et un autre pour une recherche avancée. Le formulaire de recherche rapide permet aux utilisateurs du système de soumettre rapidement une requête en se basant sur des mots clés tels que les noms d'un gène ou d'une protéine, un terme de l'ontologie du gène (GO ou Gene Ontology) (Ashburner, et al., 2000) ou n'importe quel mot qui peut apparaitre dans les champs de recherche des sources de données. Ce type de recherche ne fait pas intervenir le médiateur pour trouver une réponse ; il fait appel directement aux services de données du système pour retrouver l'information à partir des sources de données intégrées dans le système.

Le formulaire de recherche avancée fait usage de l'ontologie du domaine du *YeastMed* pour permettre aux utilisateurs d'exprimer leurs requêtes. Les termes de l'ontologie sont présentés en langage naturel dans les champs du formulaire, et l'utilisateur fait des

combinaisons entre eux pour formuler une requête qui sera soumise. Ce type de recherche fait appel au médiateur pour traiter les requêtes.

YeastMed se base sur SB-KOM pour réaliser la réécriture de requêtes en temps d'exécution. Une fois une requête est construite puis soumise à partir de l'interface Web, *YeastMed* génère une requête conjonctive en termes de l'ontologie du système. SB-KOM se charge ensuite de sa décomposition et réécriture en sous-requêtes destinées aux sources de données en se basant sur un ensemble de règles de mapping. Les sous-requêtes sont exprimées en XQuery, car les sources sont accessibles via des services Web qui acceptent seulement ce type de requêtes. Pour chaque source de données, un service Web (appelé dans notre cas aussi service de données) a été développé. Les services Web reçoivent des requêtes XQuery du SB-KOM et retournent des documents XML. Le rôle des services de données et de permettre au système l'utilisation des fonctionnalités des adaptateurs (Levy, 1999) pour trouver et extraire les informations sollicitées à partir des sources de données à travers leurs pages Web ou des mécanismes FTP. Les réponses, matérialisées par des documents XML, aux requêtes XQuery sont renvoyées au médiateur SB-KOM qui leur combine sous forme d'une instance d'ontologie du système exprimée en RDF. Le résultat final est fourni à l'utilisateur sous forme HTML.

4.6. Implémentation et mise en œuvre

YeastMed est formé d'un ensemble de modules qui dépendent fortement des technologies XML et du Web sémantique. Pour assurer une intégration syntactique et sémantique des données biologiques, nous avons commencé par une étude exhaustive des données des sources afin de construire des schémas XML des sources et aussi sur les capacités d'interrogation de chacune de ces sources. Ceci nous a permis par la suite d'envisager comment se fera l'accès aux données. La réconciliation de ces schémas a permis la création d'un schéma globale au système sous forme d'une ontologie de domaine. C'est une classification hiérarchique de concepts (ou classes) reliés par des relations sémantiques (ou propriétés). Les concepts de l'ontologie sont des abstractions des données réelles dans les sources. Et pour assurer le fonctionnement du module de réécriture de requêtes, un ensemble de règles de mappings (ou de correspondances) reliant les éléments de l'ontologie aux éléments des schémas des sources a été défini et publié dans le Répertoire Sémantique du système. Comme nous avons mentionné dans l'architecture générale, l'accès aux données se fait à travers un ensemble de services Web. Ainsi, nous avons développé un service Web pour chaque source de données.

4.6.1. Schémas des sources

Chaque source a été modélisée et représentée par un modèle abstrait des données qu'elle fournit. L'objectif est de représenter les données sous une forme qui permet de faciliter leur localisation lors du processus de la réécriture des requêtes des utilisateurs par le médiateur, et de définir une structure sous laquelle les réponses partielles des sources de données seront façonnées. Le modèle définit n'est qu'un schéma XML. Il permet de définir les données de la source sous forme d'éléments structurés, et de leur attribuer les types de

valeurs qu'ils peuvent avoir dans la source de données (string, integer ...). La Figure 52 montre un fragment du schéma de la source SGD.

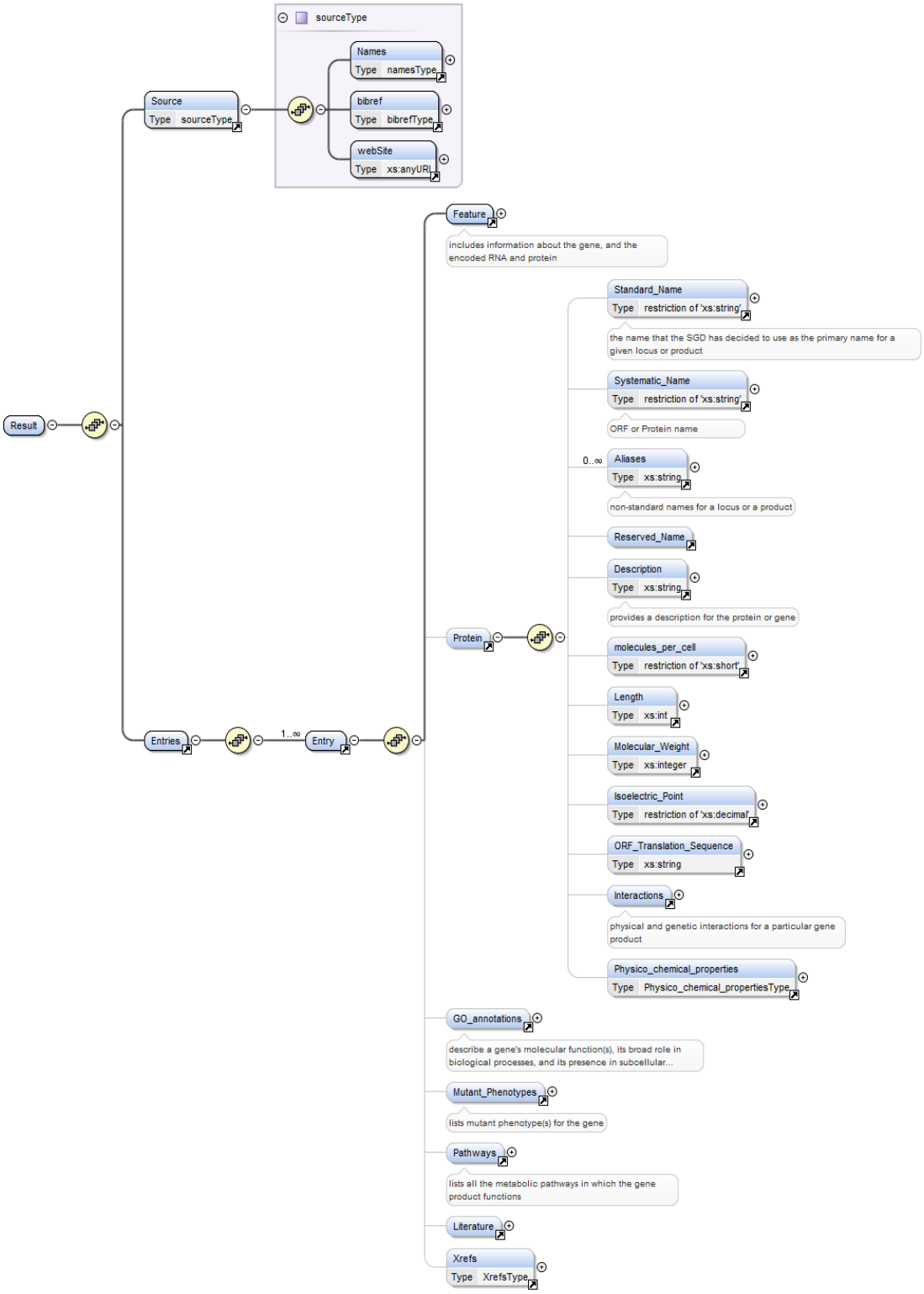

Figure 52. Un fragment du Schéma XML de la source SGD.

Comme nous avons dit précédemment, un schéma de sources dans *YeastMed* joue le rôle d'un modèle de référence lors de la construction de la réponse XML qui se retourne du service de données de la source vers SB-KOM. Chaque Schéma de source commence par une racine qui s'appelle *Result* (qui tire son nom du résultat d'interrogation de la source) d'où se ramifient deux éléments fils : *Source* et *Entries*. L'élément *Source* imbrique des éléments qui modélisent les données définissant la source de données (les noms de la source de données, une référence bibliographique et l'adresse Web). Cet élément joue un rôle primordiale lors de la traçabilité des réponses : grâce à cet élément et à la méthode *getProvenance()* fournie par l'API du service Web sous-jacent la source de données (voir section 4.6.4), il est possible de savoir l'origine d'une information dans la réponse finale.

De l'élément *Entries* va l'élément *Entry* avec une multiplicité supérieur ou égal à 1. La réponse d'une source, pour qu'elle soit construite par le service Web, exige que le résultat reçu par le service Web suite à l'interrogation de la source doive au moins contenir une entrée. Une entrée dans une base de données biologique est l'ensemble des informations relatives à une instance de l'entité du focus de cette base de données (un locus chromosomique, une protéine…). L'élément *Entry* donne ensuite naissance à plusieurs éléments qui décrivent et organisent les informations d'une entrée (description d'un gène et, si c'est le cas, de la protéine qu'il code, annotations GO, littérature, références croisées …).

A chaque élément du schéma a été définit le type de valeur qu'il peut avoir dans le document XML, la présence (obligatoire ou optionnelle), et l'ordre d'apparition.

4.6.2. Schéma global

Comme mentionné précédemment, le but du système *YeastMed* est d'aider ses utilisateurs à retrouver des informations à partir de sources de données multiples et hétérogènes en fournissant un point d'accès unique. Pour cette fin, nous avons équipé le système par une ontologie de domaine, appelée *YeastMedOnto*[92], qui joue le rôle du schéma global. L'objectif principal de cette ontologie est de supporter les requêtes des utilisateurs. Les requêtes sont formulées en termes de l'ontologie et *YeastMed* les convertit en requêtes XQuery à destination des sources appropriées et susceptibles de participer à l'élaboration d'une réponse à la requête initiale.

L'ontologie du *YeastMed* a été construite à partir de zéro en réconciliant les différents schémas des sources de données. Elle assure l'encapsulation des sources de données en définissant une hiérarchie de concepts. C'est une classification structurale de toutes les entités biologiques manipulées par le système. Il représente un modèle capturant les connaissances biologiques dans une structure conceptuelle hiérarchique contrainte par des relations du type parent-fils (Figure 53) : une classe enfant est un sous-ensemble d'éléments du parent ; chaque classe enfant hérite toutes les propriétés du parent et en plus possède des propriétés propres plus spécialisées.

[92] http://www.khaos.uma.es/yeastmed/download/YeastMedOntology.owl

Figure 53. Un fragment de l'ontologie du YeastMed.

Les concepts du *YeastMedOnto* peuvent être classés selon deux catégories : les concepts purement biologiques et les concepts reliés aux sources de données.

- La catégorie des concepts purement biologique est l'union de toutes les classes modélisant les entités biologiques présentes dans les sources intégrées. Comme exemple de cette catégorie nous citons le concept *ChromosomalFeature* : C'est une superclasse de 38 classes représentant les différents types d'éléments chromosomiques caractérisés (gènes, CDS, intron, régions répétées, etc.).

- L'ensemble des concepts relative aux sources des données est représenté par des concepts (ou classes) qui réfèrent aux sources de données elles-mêmes. Par exemple le concept *Source* représente les cinq sources de données intégrées et le concept *Entry* réfère aux entrées des sources. L'ajout de cet ensemble dans l'ontologie a comme objectif de permettre aux utilisateurs d'exprimer leurs préférences aux sources de données. Ainsi, l'utilisateur a la possibilité de choisir et de préciser la source d'où obtenir des entrées au lieu que le système fait son choix. Par exemple, la requête suivante retourne toutes les entrées de la base de données SGD ayant dans leurs descriptions le mot clé "*Topoisomerase*".

  Ans(E) :- SGDEntry(E), hasDescription(E, "Topoisomerase") ;

  alors que la requête:

  Ans(E) :- Entry(E), hasDescription(E, "Topoisomerase") ;

  retourne les entrées qui existent dans toutes les sources de données ayant dans la description "Topoisomerase".

Pour représenter plus d'informations sémantiques sur les concepts, l'ontologie définit deux types de propriétés. Le premier est définit par un ensemble de propriétés d'objet qui modélisent les relations qui existent entre deux individus appartenant à une ou deux classes différentes de l'ontologie. Le deuxième type concerne les propriétés du Type de Données. Ce sont les relations qui relient des individus à des données littérales.

Pour donner des exemples et illustrer davantage le rôle des propriétés dans l'enrichissement de la sémantique dans *YeastMedOnto*, nous détaillons l'exemple réel suivant : *SWI4*, ayant le nom systématique *YER111c*, est un gène qui code pour un composant de liaison d'ADN du complexe SBF (Swi4-Swi6). C'est un facteur de transcription (un activateur plus précisement) qui, conjointement avec MBF (Mbp1-Swi6), régule la transcription tardive spécifique du G1 des cibles incluant des cyclines et des gènes nécessaires pour la synthèse et la réparation d'ADN (Nasmyth and Dirick, 1991), un exemple est l'enzyme Topoisomerase I (codée par le gène TOP1 ayant le nom systématique YOL006c)(French, et al., 2011).

A partir de cette description, nous pouvons déclarer les assertions suivantes :

- *SWI4* et *TOP1* sont deux gènes ayant respectivement les noms systématiques YER111c et YOL006c ;
- SWI4 et TOP1 codent respectivement pour un facteur de transcription et une enzyme.
- SWI4 régule l'activité transcriptionnelle du TOP1 ;
- SWI4 et TOP1 codent pour des protéines (ayant les mêmes noms standards que leurs gènes).

Ces assertions nous permettent de définir dans l'ontologie:

- Quatre concepts : *Gene*, *Protein*, *TranscriptionFactor* et *Ezyme* ;
- Quatre propriétés Object : *codesFor* et sa propriété inverse *codedBy* liant *Gene* à *Protein*, et la propriété *regulates* et sa propriété inverse *regulatedBy* liant *TranscriptionFactor* à Gene ;
- Deux propriétés du type de données : *hasSystematicName* et *hasStandardName* liant *TranscriptionFactor* et *Enzyme* à des valeurs du type String (chaine de caractères. *SWI4* et *TOP1* pour le premier et *YER111c* et *YOL006c* pour le dénier) ;
- Enzyme et *TranscriptionFactor* comme des concepts fils du concept *Protein*.

La Figure 54 suivante illustre la partie du *YeastMedOnto* qui modélise ce que nous venons de citer.

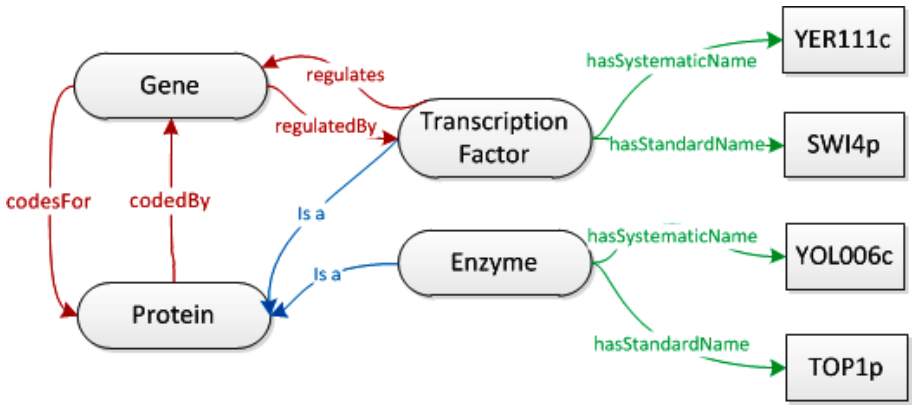

Figure 54. Représentation graphique de la partie de l'ontologie concernant l'exemple.

Dans *YeastMed*, nous avons choisi OWL (Hitzler, et al., 2009) comme un langage standard d'ontologie pour exprimer YeastMedOnto. Le choix s'est articulé sur le fait que OWL est comme RDF (Brickley and Guha, 2004), tire son avantage de l'universalité syntactique du XML. En se basant sur la syntaxe RDF/XML, OWL fournit une manière d'écrire les ontologies Web. Il est différent du couple RDF/RDFS dans le sens où il est juste un langage d'ontologies : si RDF et RDFS procure à l'utilisateur la possibilité de décrire des classes (avec des constructeurs) et des propriétés, OWL incorpore, en plus, des outils de comparaison des propriétés et des classes : identité, équivalence, contraire, cardinalité, symétrie, transitivité, disjonction, etc. Ainsi, OWL offre aux machines une capacité plus importante pour l'interprétation du contenu Web que RDF et RDFS, avec un vocabulaire plus large et une sémantique formelle réelle. Parce que nous avons décidé dès le début d'équiper *YeastMed* avec une ontologie montrant une hiérarchie et des contraintes simples, nous avons fait utilisation seulement du OWL-Lite (un sous-langage de OWL).

4.6.3. Règles de correspondance

Avoir une ontologie du domaine facilite la formulation des requêtes au système. Les utilisateurs soumissent leurs requêtes en termes de l'ontologie au lieu de les exprimer directement en termes des schémas de sources. Même si il est très pratique et effectif du point de vue de la transparence à l'utilisateur, il procure cependant le problème de réécrire la requête de l'utilisateur en une ou plusieurs requêtes en termes des schémas de sources. Dans *YeastMed*, ce problème est surmonté grâce à l'utilisation du SB-KOM comme déjà mentionné. Cependant, en plus de la modélisation de l'ontologie et des sources, il nous a fallu établir des associations entre les concepts de l'ontologie et les éléments appropriés dans les schémas de sources afin de permettre le bon fonctionnement SB-KOM. Ces associations sont matérialisées dans *YeastMed* par des règles de correspondances ou de mappings.

SB-KOM est conçue pour décomposer les requêtes en se basant sur des mappings exprimés selon l'approche GAV (Levy, 1999). Cela veut dire que chaque concept (aussi propriété dans notre cas) de l'ontologie est une vue définit en termes des éléments des schémas des sources. Cette vue définit comment obtenir des instances des classes du

schéma globale du système à partir des sources. Dans ce contexte, les règles de mapping que nous avons établi sont définit comme des paires (P, Q). *P* est un ou un couple d'expression de chemin dans un schéma de sources exprimé en XPath, et *Q* une requête conjonctive exprimée en termes de l'ontologie. Trois types de règles de mappings ont été définit :

- Mapping de classe : il fait correspondre les classes de l'ontologie aux schémas des sources. Il a la forme suivante :

XPath-Element-Location, Ontology-Class-Name, Corrrespondence-index

XPath-Element-Location est l'emplacement d'un élément dans le schéma de la source ; il est exprimé en XPath. *Ontology-Class-Name* est le nom de la classe de l'ontologie correspondante à l'élément dans l'emplacement *XPath-Element-Location*. *Corrrespondence-index* est un entier exprimant l'exactitude du mapping. Dans *YeastMed*, *Corrrespondence-index* est toujours égale à 100 parce que tous les mappings ont été construit manuellement et non automatiquement, donc nous considérons que les mappings sont correctes 100%.

Un exemple qui fait correspondre la classe *Protein* à un élément du schéma du SGD est le suivant :

Result/Entries/Entry/Protein, Protein, 100

- Mapping des propriétés de type de données : il fait correspondre une propriété du type de données à un élément dans un schéma de source. Il s'écrit sous la forme suivant :

XPath-Domain-Location; XPath-Value-Location, Ontology-Domain-Name; Property-Name, Correspondence-index

XPath-Domain-Location est le chemin dans le schéma de source à l'élément correspondant à la classe domaine de la propriété du type de données en sujet. *XPath-Value-Location* est le chemin dans le schéma de source à l'élément d'où la propriété prend la valeur du son rang. *Ontology-Domain-Name* et *Property-Name* sont les noms respectivement du domaine et de la propriété en jeux.

L'exemple suivant concerne le mapping de la propriété *hasName* :

Result/Entries/Entry/Protein/; Result/Entries/Entry/Protein/SysName, TranscriptionFactor ; hasName, 100

- Mapping des propriétés d'objet : il fait correspondre une propriété d'objet de l'ontologie à un élément d'un schéma de source. Il se présente sous la forme suivant :

XPath-Domain-Location; XPath-Range-Location, Ontology-Domain-Name; Ontology-Range-Name; Property-Name, Correspondence-index

XPath-Range-Location est le chemin dans le schéma de source à l'élément qui correspond au rang de la propriété d'objet en question. *Ontology-Range-Name* est le nom du rang de la propriété.

L'exemple suivant montre comment la propriété d'objet *hasBibRef* est mappée au schéma de la source SGD :

Result/Entries/Entry/Protein; Result/Entries/Entry/Literature, Protein; BibRef ; hasBibRef, 100

4.6.4. Services Web du YeastMed

YeastMed fait usage d'un ensemble de services Web pour accéder aux sources de données. Nous avons implémenté un service Web pour chaque source de données intégrée. Ces composants cachent au médiateur les détails techniques et le modèle de données des sources. Ils reçoivent des requêtes XQuery du SB-KOM et retournent des documents XML structurant un résultat d'interrogation de la source et d'autres métadonnées.

Le rôle des services Web du *YeastMed* est double :

- Ils permettent à *YeastMed* d'utiliser les fonctionnalités d'un adaptateur pour retrouver et extraire des informations sollicitées à partir des sources de données en utilisant des protocoles HTML ou des mécanismes FTP. Cela veut dire donner l'habilité de résoudre des requêtes XQuery et retourner des résultats en format XML.
- Ils exportent des informations sémantiques sur les schémas de sources et la provenance des données. Cela permet surtout à *YeastMed* de maintenir la traçabilité des informations retournées lors de leur combinaison et de savoir quelle source a été interrogée.

Il est communément connu qu'un adaptateur est une interface de communication avec une source de données et qui traduit les données dans un modèle de données commun utilisé par un médiateur (Levy, 1999). Parce que le but du *YeastMed* est l'intégration de bases de données accessible via des protocoles Web, il est complétement normale qu'un adaptateur se considère comme le composant le plus important dans l'architecture des services de données du *YeastMed* (Figure 55). C'est une interface qui reçoit des requêtes XQuery générées par SB-KOM, accède la source de données adjacente, extrait des données et les structure selon le schéma XML de la source.

En plus d'un adaptateur, chaque service Web contient une API (Application Programming Interface). C'est le point d'accès du SB-KOM aux fonctionnalités du service Web. Cette API publie trois méthodes : *getQuery(Q)* qui passe à l'adaptateur la requête *Q* et retourne une réponse sous forme XML. La structure XML de cette réponse doit être validée par le schéma XML de la source. Les deux autres méthodes, *getSchema()* et *getProvenance()*, fournissent un accès aux métadonnées que le service Web cache. La première renvoie le schéma XML tandis que *getProvenance()* fournie des informations descriptives de la source de données en jeux. Afin d'utiliser correctement ces méthodes, SB-KOM trouve toute les informations nécessaires dans un document WSDL (Web Service Description Language).

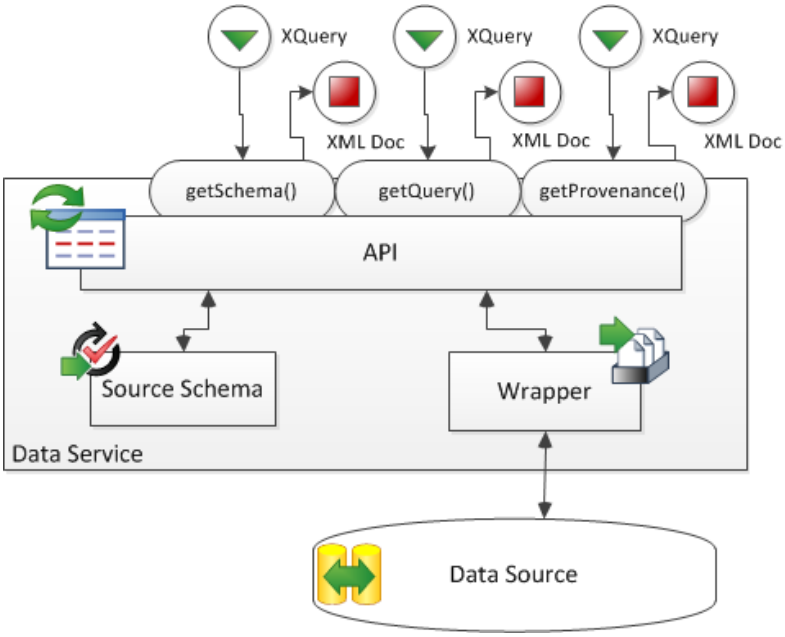

Figure 55. Architecture des services Web du YeastMed.

YeastMed est capable de refléter la provenance des données en appelant la méthode *getProvenance()* qui renvoie des informations sur la source de données ou à travers le document XML retourné par les services Web. Le document XML contient par défaut, comme nous avons vu dans la section 4.6.1, une description de la source interrogée. Ainsi, les instances des données intégrées peuvent être annotées par leur provenance. De cette manière, l'interface utilisateur peut montrer la provenance de chaque partie du résultat.

4.6.4.1. Fonctionnement et développement des services Web

Les services Web sont implémentés en Java. Ils reçoivent des requêtes XQuery du SB-KOM via la méthode *getQuery()* de l'API qui les passe à l'adaptateur. Ce fonctionnement est assuré par un ensemble de modules défini par des classes Java. La requête reçue est analysée par Le module *Query Adapter* afin d'identifier précisément quelles sont les informations sollicitées de la source de données. Ce module génère ensuite une requête adaptée à la source en faisant recoure aux capacités d'interrogation déjà étudiées et spécifiées. Par la suite, il établit une connexion à la source via des protocoles HTML ou FTP. Dans le cas d'un protocole HTML, la base de données est interrogée à travers son interface Web en utilisant sa propre machine d'interrogation. La réponse est constituée d'une ou de plusieurs pages Web qui sont parsées à la volée par un parseur de pages Web qui extrait les informations sollicitées. Un module appelé *DOM Tree Generator* organise par la suite ces informations comme une instance du schéma XML de la source de données avant de l'envoyer à SB-KOM sous forme d'un document XML (Figure 56).

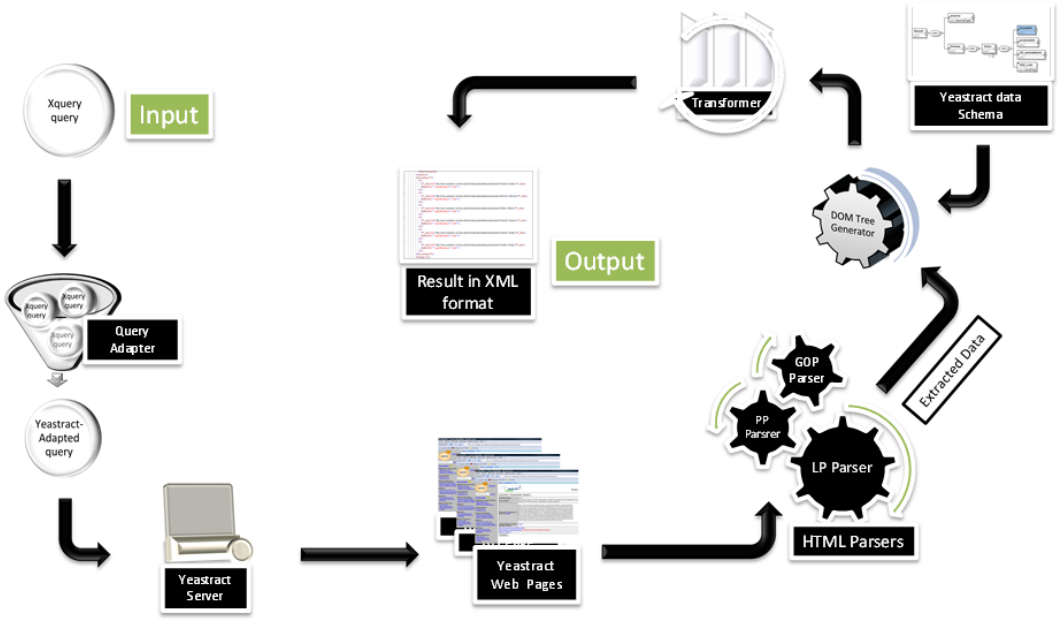

Figure 56. Fonctionnement du service Web de la source de données Yeastract.

Apache Tomcat[93] a été utilisé comme serveur d'application pour publier les services Web et Axis[94] comme une plateforme pour les présenter. Un descripteur de déploiement (un fichier nommé Web.xml) qui contient tous les caractéristiques et les paramètres du service Web est définit pour chaque service Web.

4.6.5. Interface Web

Le fait que les biologistes sont familiarisés avec les formulaires HTML lors de l'interrogation des bases de données biologiques, et dans le but de rendre *YeastMed* facilement accessible et utilisable, nous avons adopté la même stratégie que la plupart des bases de données utilisent pour permettre la réception de requêtes : une interface Web simple basée sur des formulaires HTML a été développée permettant l'expression des requêtes en langage naturel. C'est une interface basée sur YeastMedOnto. Les utilisateurs formulent leurs requêtes en sélectionnant des éléments à partir des champs du formulaire. Ces éléments ont leurs équivalents dans l'ontologie du système (concepts et propriétés) et sont écrits en langage naturel. Par exemple le concept *BibRef* de l'ontologie est traduit dans les champs du formulaire comme *Bibliographic Reference* et la propriété du type de données *hasProductDesc* comme *having Product Description*.

Nous sommes convaincu qu'il est plus facile pour des utilisateurs (comme des biologistes) non familiarisés aux langages d'interrogation de bases de données (tel que SQL, XQuery…) d'exprimer en langage naturel leurs requêtes en utilisant implicitement la conjonction de structures en triplets. Un triplet est composé de trois éléments élus de trois

[93] http://tomcat.apache.org/

[94] http://ws.apache.org/axis/overview.html

champs adjacents différents et du même niveau du formulaire d'interrogation de l'interface Web du *YeastMed*. Chaque niveau permet trois champs au maximum et chaque élément d'un champ a un équivalent dans l'ontologie. Le premier champ prend ses éléments à partir des classes qualifiées comme domaines dans YeastMedOnto. Le champ du milieu reflète les propriétés de l'ontologie, alors que le troisième est conçu pour refléter les rangs des propriétés d'objet ou de recevoir une valeur pour une propriété du type de données.

Un exemple simple pour donner plus d'explications sur l'expression d'une requête dans *YeastMed* est le suivant : supposons qu'un biologiste est intéressé à l'ensemble de gènes régulés par le facteur de transcription ayant le nom *Adr1*. Il peut exprimer ceci en utilisant les deux assertions suivantes :

- Gènes régulés par un facteur de transcription ; et
- ce facteur de transcription a le nom systématique "*Adr1*".

Le formulaire d'interrogation du *YeastMed* permet de capturer ce genre d'expressions. Dans chaque linge, il propose trois champs représentant successivement un domaine, une propriété et un rang. Le rang peut être soit un concept de l'ontologie à choisir à partir des éléments du troisième champ ou une valeur à introduire dans un champ spécifique qui apparaît sous le deuxième si une propriété de type de données y est choisie. L'exemple ci-dessus peut être capturé par *YeastMed* en utilisant deux lignes de champs comme suit (Figure 57):

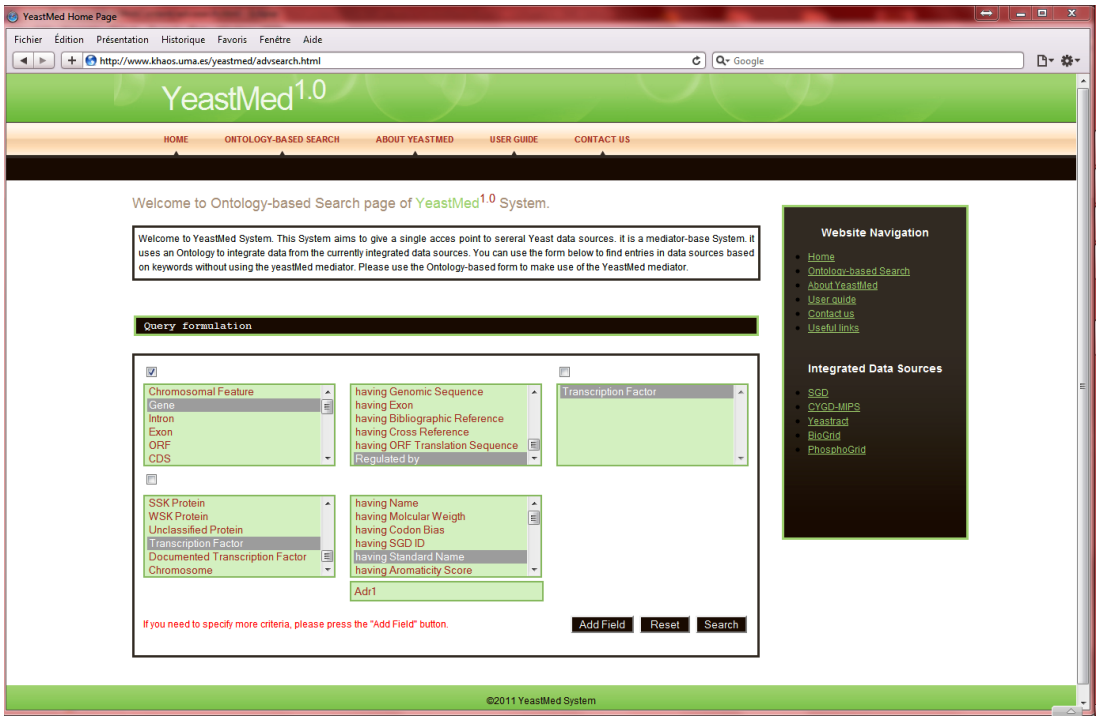

Figure 57. Interface Web d'interrogation du YeastMed en se basant sur l'ontologie du système.

- "Gene", "regulated by", "Transcription Factor";
- "*Transcription Factor*", "*having Standard Name*" "*Adr1*"

Une fois soumise, le système traduit cette requête en une requête conjonctive en faisant appel aux équivalents dans l'ontologie des éléments constituant ces expressions.

> Ans(G) := Gene(G), regulatedBy(G, TF), TranscriptionFactor(TF), hasStandardName(TF, "Adr1");

L'interface Web du système donne aussi la possibilité aux utilisateurs d'utiliser un formulaire de recherche rapide permettant d'interroger les cinq bases de données intégrées de manière indépendante sans passer par le médiateur. Les utilisateurs soumettent des mots clé et choisissent les sources de données à interroger. Le système ensuite utilise de manière indépendante les services Web des sources élus pour rechercher des entrées contenant les mots clé spécifiés.

4.7. Cas d'utilisation

Dans cette section nous allons montrer comment une requête utilisateur est traitée par *YeastMed* et comment ses différents composants participent dans ce processus.

4.7.1. Scénario

Un biologiste utilise *YeastMed* pour trouver des informations sur deux types de protéines. La première est nommée "*DNA Topoisomerase III*", et la deuxième représente "*les facteurs de transcription*" régulant l'expression de la première. Le biologiste est intéressé aussi par les sites de phosphorylation qui se trouvent dans les séquences des facteurs de transcription dont le gène est localisé sur le "*choromosome 16*". En plus, le biologiste veut obtenir toute la littérature sur "*DNA Topoisomerase III*".

4.7.2. Processus du traitement

Comme déjà indiqué, *YeastMed* fournit une interface Web qui permet aux biologistes d'exprimer leurs requêtes en termes du YeastMedOnto. Son formulaire de recherche capture des assertions en langage naturel représentant des triplets formés d'un domaine, d'une propriété et d'un rang provenant du YeastMedOnto.

Le scénario précédent permet de définir les assertions suivantes :

- "Protein", "having Description", "DNA Topoisomerase III";
- "Protein", "having Bibliographic Reference", "Bibliographic Reference";
- "Protein", "Regulated By", "Transcription Factor";
- "Transcription Factor", "Belongs To", "Chromosome";
- "Chromosome", "having Name", "16";
- "Transcription Factor", "having Phosphorylation Site", "Phosphorylation Site".

Chacune de ces assertions peut être exprimée dans une ligne dans le formulaire de recherche (Figure 58). Et Pour spécifier au système quelles sont les entités sur lesquelles les informations sont sollicitées, l'utilisateur doit cocher la case à cocher qui se situe sur le champ de l'entité dont il souhaite retrouver des informations. Dans notre cas, les cases à cocher se situant sur les champs où "*Bibliographic Reference*" et "*Phosphorylation Site*" ont été choisis avant de soumettre.

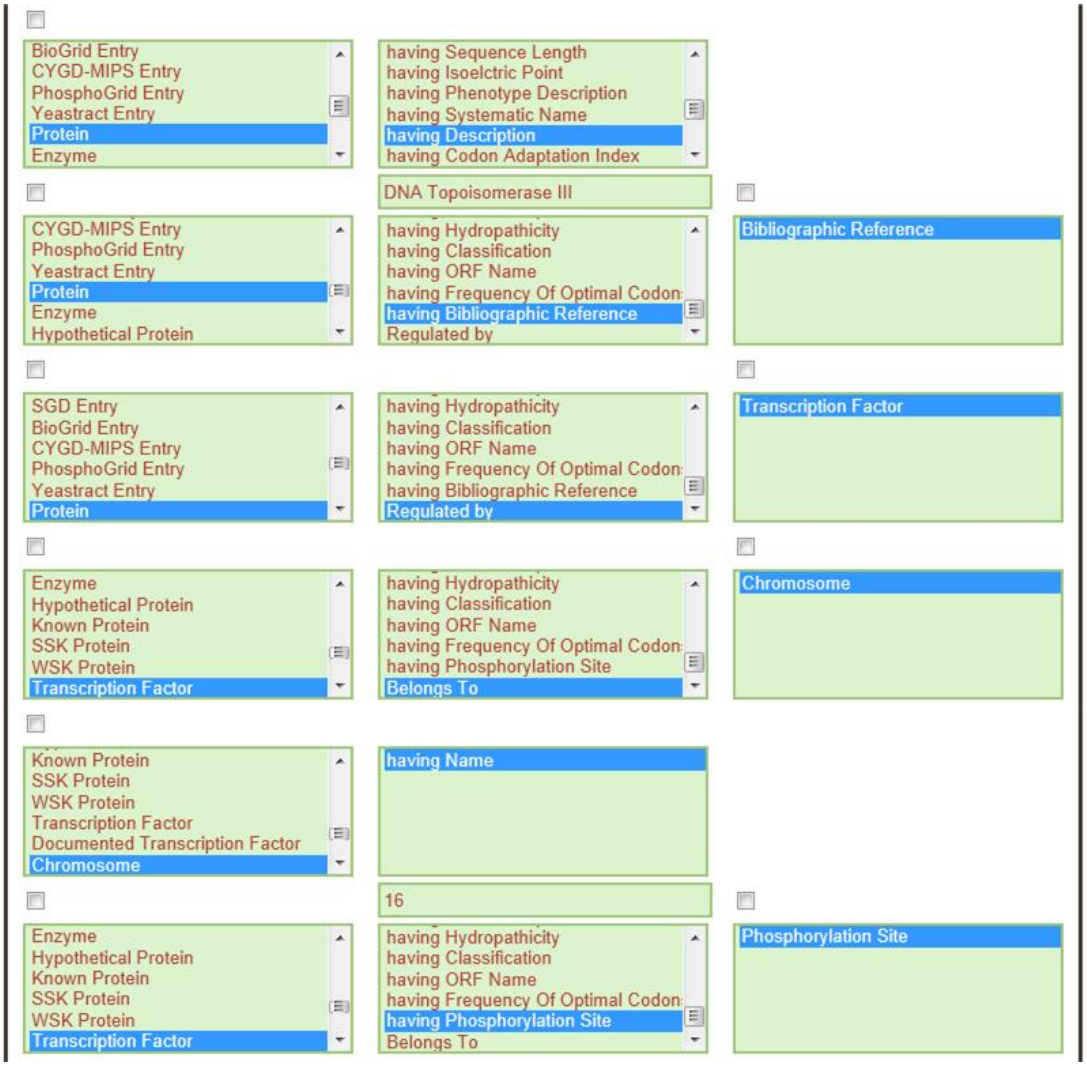

Figure 58. Expression des assertions dans l'interface Web du YeastMed.

Le fragment de l'ontologie enregistrée dans le Répertoire Sémantique qui est impliqué directement dans le processus de la formulation de cette requête est donné dans la Figure 59. A partir de ce fragment, la requête conjonctive suivante est automatiquement générée :

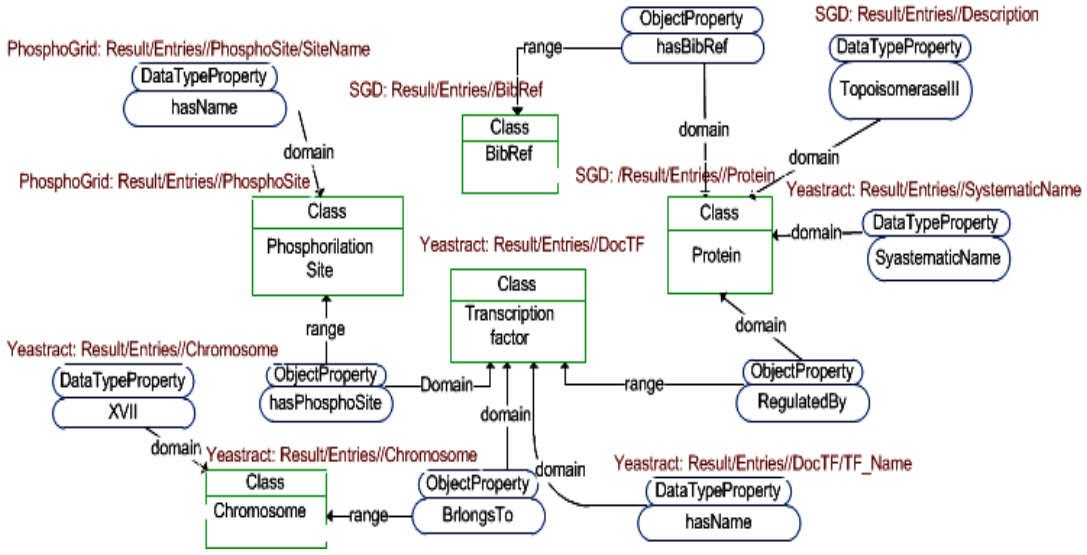

Figure 59. Le fragment de l'ontologie invoqué dans la formulation de la requête du cas d'utilisation. Les classes sont colorées en vert et les propriétés en bleu. Le mapping entre les éléments de l'ontologie et les schémas des sources sont données sur les éléments de l'ontologie et colorés en rouge.

Ans(BR,Ph) := Protein(P), hasDescription(P, "DNA Topoisomerase III"), BibRef(BR), hasBibRef(P, BR), hasSystematicName(P, SN), regulatedBy(P, TF), hasName(TF, Nt), TranscriptionFactor(TF), Chromosome(C), hasName(C,"16"), BelongsTo(TF,C), PhosphoSite(Ph), hasPhosphoSite(TF, Ph);

Cette requête conjonctive inclut comme prédicats cinq classes d'ontologie (*Protein*, *BibRef*, *TranscriptionFactor*, *Chromosome* et *Phosphosite*), trois propriétés de type de données (*hasDescription*, *hasSystematicName* et *hasName*) et quatre propriétés d'objet (*hasBibRef*, *regulatedBy*, *belongsTo* et *hasPhosphoSite*). Cette requête doit retourner des instances de Phosphosite et BibRef qui satisfont ses contraintes.

Comme une étape suivante, la requête conjonctive sera envoyée à SB-KOM. Elle sera reçue par le Contrôleur qui la passe au Planificateur de requêtes. Ce composant, en se basant sur les prédicats de la requête et les règles de correspondance du Répertoire Sémantique, générerait un ensemble de sous-requêtes et aussi un plan pour leur exécution.

Les prédicats de la requête conjonctive sont de deux types : un type qui contient un seul argument (classes) et un autre qui contient deux arguments (propriétés). Selon l'algorithme de la réécriture de requêtes discuté dans la section 3.3.2.2, les prédicats des deux types qui ont des arguments en commun sont regroupés dans des groupes représentés par la combinaison de deux prédicat ou plus. Les groupes qui ne sont pas représentés dans les mappings du Répertoire Sémantique sont écartés. Le reste est ajouté au premier groupe en permettant une seule occurrence d'un groupe. La table 3 liste tous les groupes issus de la requête.

A partir de cet ensemble, le Planificateur de requêtes essaie de construire des plans d'exécution potentiels. Il sélectionne les groupes qui ont des variables instanciées dans le but de définir une racine à un arbre. L'ordre de l'exécution du plan dépend des variables instanciées : le groupe contenant une variable instanciée est exécuté premièrement, puis les groupes qui sont reliés à ces variables, et ainsi de suite jusqu'à ce que tous les groupes soient exécutés. Dans notre cas, G2 et G8 sont sélectionnés. G8 ne peut pas servir de racine, car il n'existe aucun autre groupe qui dépend de son variable instanciée, et cela laisse les autres groupes sans exécution. Ceci n'est pas le cas pour G2 qui sert de racine pour l'arbre donné dans la Figure 60. Il est le premier à être exécuter. Il retourne la protéine qui a la description "*DNA Topoisomerase III*". Ensuite G9 et G10 sont exécutés en parallèle parce qu'ils dépendent de la variable instanciée du G2. A partir de ces exécutions simultanées, tous les objets qui sont reliés à *Protein* par le biais des relations *regulatedBy* et *hasBibRef* seront déterminés. Une fois ces objets sont obtenus, il sera vérifié si ils satisfont les groupes G14 et G15 : cela veut dire s'assurer si les objets obtenu à partir des groupes G9 et G10 sont respectivement des instances de *TranscriptionFactor* et *BibRef*. En se basant sur les résultats du G9, les groupes G11 et G12 sont exécutés mais pas d'une manière simultanée. SB-KOM a un module d'optimisation du plan d'exécution qui peut privilégier l'exécution d'un groupe sur un autre tel que le cas ici : vue que G8 a une variable instanciée (valeur "16") et il est relié à G14 via G11, ce dernier est exécuté avant G12 et le résultat est utilisé par le groupe G12 lors de son exécution.

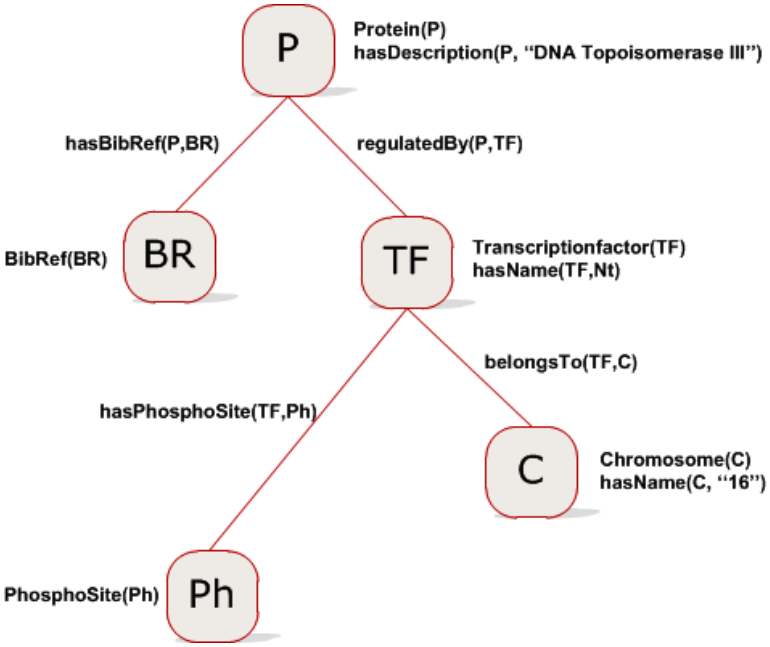

Figure 60. L'arbre du plan d'exécution généré à partir de la requête conjonctive du cas d'utilisation. C'est un arbre binaire où les nœuds sont représentés par les variables des arguments des prédicats de la requête conjonctive et les arcs sont les prédicats contenant les deux variables des nœuds qu'ils relient.

Groupe	Requête	Source concernée
G1	Protein(P), hasBibRef(P, BR)	SGD
G2	Protein(P), hasDescription(P, "DNA Topoisomerase III")	SGD
G3	Protein(P), hasSystematicName(P, SN)	Yeastract
G4	Protein(P), RegulatedBy(P, TF)	Yeastract
G5	TranscriptionFactor(TF), hasName(TF, Nt)	Yeastract
G6	TranscriptionFactor(TF), belongsTo(TF, C)	Yeastract
G7	TranscriptionFactor(TF), hasPhosphoSite(TF,Ph)	PhosphGrid
G8	Chromosome(C), hasName(C, "16")	Yeastract
G9	regulatedBy(P, TF)	Yeastract
G10	hasBibRef(P, BR)	SGD
G11	belongsTo(TF, C)	Yeastract
G12	hasPhosphoSite(TF, Ph)	PhosphGrid
G13	Protein(P)	SGD; Yeastract; PhosphGrid
G14	TranscriptionFactor(TF)	Yeastract ; PhosphGrid
G15	BibRef(BR)	SGD
G16	Chromosome(C)	Yeastract
G17	PhosphoSite(Ph)	PhosphGrid

Table 3. Les groupes générés par l'algorithme de réécriture de requêtes du Planificateur.

Les arcs des arbres de planification générés par le Planificateur représentent les propriétés d'objets, tandis que les nœuds sont des classes d'ontologie ou des instances de celles-ci. Chaque nœud et arc contiennent toutes les informations exigées par l'Evaluateur/Intégrateur pour exécuter les sous-requêtes. C'est-à-dire la requête XQuery (élaborée à partir du mapping) correspondant à la sous-requête du nœud ou de l'arc, les noms et les URLs des services Web d'intérêt. Un exemple est donné dans la Figure 61.

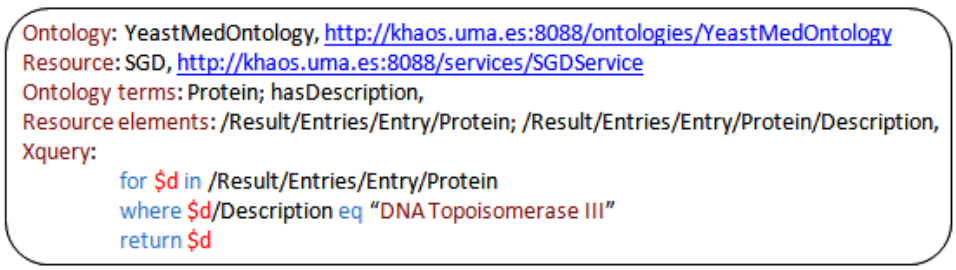

Figure 61. Les informations présentées par le nœud P du l'arbre Plan du cas d'utilisation. Le nœud P contient l'emplacement de YeastMedOnto et du service web à appeler, en plus des ressources du mapping et la requête XQuery à envoyer au service web.

Les services Web du *YeastMed* sont appelés par l'Evaluateur/Intégrateur dans l'ordre du plan, après optimisation, généré par le Planificateur. Dans notre cas, le service Web du SGD reçoit la première sous-requête, car la propriété d'objet *hasDescription* est mappée au schéma du SGD. "*TOP3*" est retourné comme résultat de cette sous-requête et ensuite utilisé par la sous-requête *RegulatedBy* pour trouver des instances du *TranscriptionFactor*. Le service Web du Yeastract est invoqué cette fois-ci car la propriété en jeux est mappée

avec le schéma de la source de données Yeastract. Trois instances de la classe *TranscriptionFactor* sont retournées : "*Fhl1p*", "*Hsf1p*" et "*Swi4p*". Pour chacune de ces instances, le service de données Yeastract est appelé de nouveau. Il reçoit cette fois-ci la sous-requête représentée par la propriété *belongsTo* qui contient les deux arguments instanciés : le premier est un des trois instances retournées par la requête précédente, et le deuxième argument est instancié par le nom du chromosome *16*. Cette sous-requête vérifie si le facteur de transcription a son gène sur le chromosome 16. Seulement l'instance "*Fhl1p*" est maintenue. Finalement la sous-requête *hasPhophoSite* est exécutée sur le service Web de la source de données PhosphoGrid et retourne toutes les instances du *PhosphoSite* du facteur de transcription *Fhl1p*. Dans chaque exécution, l'Evaluateur/Intégrateur reçoit les résultats au format XML à partir du service de données cible.

Les résultats des services Web sont des instances des schémas XML des sources de données adjacentes. En se basant sur les règles de mappings entre les éléments des schémas des sources et les éléments de l'ontologie, ces instances sont traduites en instances d'ontologie qui ne sont pas interconnectées parce qu'elles ont été produites par différents services Web. Pour les associer, l'Evaluateur/Intégrateur utilise les instances des classes des domaines et des rangs des propriétés d'objets. Le résultat final est une instance du YeastMedOnto qui inclut toutes les données extraites à partir des sources de données interrogées. C'est-à-dire toutes les instances des concepts *BibRef* de la protéine "*TOP3*" et tous les objets *PhosphoSite* du facteur de transcription "*Fhl1p*".

4.8. Evaluation du YeastMed

Nous avons mené une estimation d'utilisabilité dans le but de graduer la vitesse parr laquelle les biologistes peuvent apprendre à utiliser *YeastMed*, et pour savoir jusqu'à quel point sont satisfait du système. Nous avons aussi réalisé une étude de performance du système pour révéler comment les temps d'exécution se comportent envers l'augmentation du nombre des sources de données impliquées dans les requêtes. Dans cette section, nous présentons les résultats obtenus à partir de ces deux études.

4.8.1. Utilisabilité du Système

Une variété de méthodes d'estimation de l'utilisabilité des systèmes interactifs ont été signalées dans la littérature. Nous citons particulièrement QUIS (Chin, et al., 1988), SUS (Brooke, 1996), CSUQ (Lewis, 1995) et Microsoft's Product Reaction Cards (Benedek and Miner, 2002). Tullis et Stetson (Tullis and Stetson, 2004) ont mené une étude qui compare ces méthodes et ont montré que la précision de l'analyse croit au fur et à mesure que le nombre des participants devient plus grand (pour un échantillon de 6 à 14 individus) et que la précision du SUS croît plus rapidement que les autres. Pour cela, nous avons choisi de suivre cette méthode dans notre étude.

Le questionnaire de SUS consiste en 10 éléments auxquels les participants manifestent leur niveau d'agrément. Les éléments dont le numéro est impair (1, 3, 5, 7 et 9) sont négativement récompensés et le reste des éléments, dont le numéro est pair, sont

positivement récompensés. Une échelle de 5 points d'agrément numéroté de 1 (fortement désagréer) à 5 (fortement agréer) est utilisée pour chacun d'eux. Chaque élément obtiendra un score élémentaire allant de 0 à 4. Pour les éléments de la numérotation impaire, le score élémentaire égale au numéro de la position active de l'échelle moins 1. Le reste des éléments ont un score élémentaire qui égale à 5 mois le numéro de la position active de l'échelle. Pour avoir le score final, qui est l'indicateur de l'utilisabilité, la somme des scores élémentaire est multipliée par 2.5. Le score du SUS oscille entre 0 et 100 avec 100 représentant un indicateur d'utilisabilité parfaite. La Figure 62 suivante illustre un exemple.

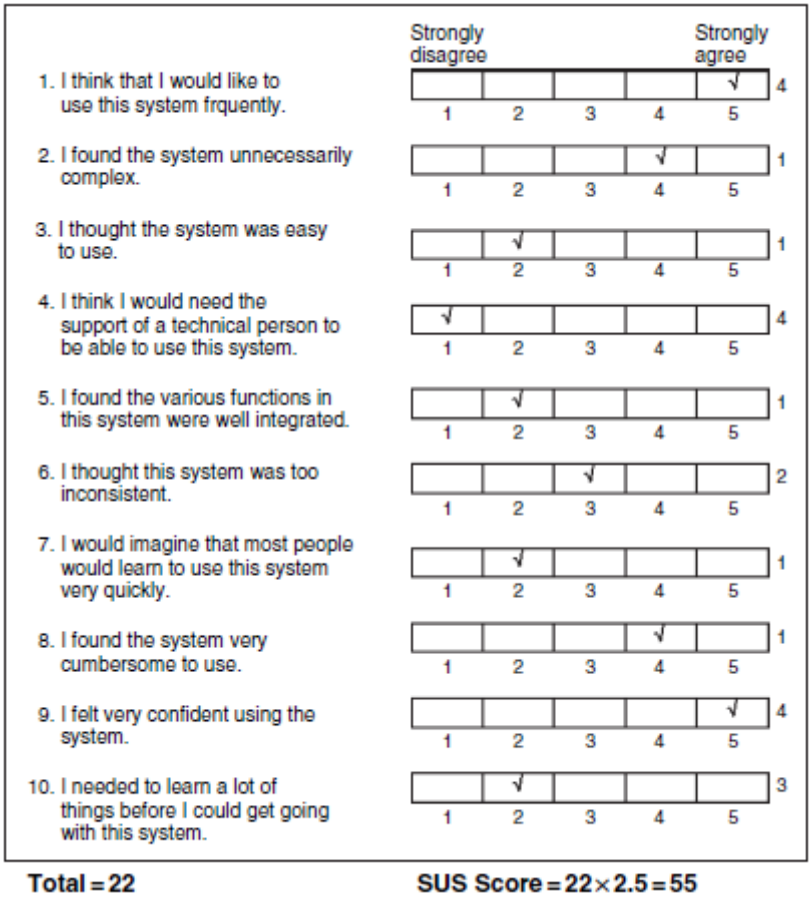

Figure 62. Les éléments du questionnaire de la méthode SUS. Ils sont au nombre de 10. Chaque élément est estimé sur une échelle allant de 1 à 5 avec 5 représentant le niveau d'agrément le plus élevé.

L'étude de l'utilisabilité que nous avons menée avait deux objectifs:

- Avoir un indicateur général de l'utilisabilité du *YeastMed* ;
- Estimer l'évolution de l'utilisabilité du système avec le niveau de la familiarisation aux bases de données biologiques. Ceci est représenté dans notre cas par la fréquence d'utilisation des bases de données biologiques par les participants.

Ces objectifs nous ont permis d'avoir une idée sur les difficultés d'apprentissage et d'utilisation du système par les biologistes, et jusqu'à quel point nous avons réussi à fournir un système facile à utiliser par les biologistes qui sont familiarisés avec les interfaces de recherche des bases de données biologiques.

39 participants ont participé dans cette étude. Chacun a testé *YeastMed* avant d'évaluer les éléments du formulaire de SUS. Tous les participants sont des biologistes répartis sur cinq groupes distingués par leurs niveaux de familiarisation aux bases de données biologiques. Chaque groupe contient entre 7 et 9 participants et sont nommés selon la fréquence de l'utilisation des bases de données biologiques par les participants. Ainsi les cinq groupes sont nommés : *Never*, *Rarely*, *Sometimes*, *Usually* et *Always*. Ce qui veut dire par exemple le groupe *Never* contient des participants qui n'ont jamais utilisé des bases de données biologiques, et le groupe *Always* contient des participants qui en utilisent très souvent. Pour chaque groupe, nous avons calculé le score SUS individuel de chaque participant avant de déduire le score moyen du groupe.

Comme illustré dans la Figure 63, le score de l'utilisabilité du *YeastMed* croit avec la familiarisation aux bases de données biologiques : le score moyen de SUS passe de 60.71 pour les biologistes qui n'ont jamais consulté des bases de données biologiques à 78.75 pour ceux qui en sont à une utilisation fréquente. Le score général est de 71.54. Avec ces scores nous pouvons dire que *YeastMed*, avec son interface simple à base de formulaire HTML, fournit un service facile à utiliser pour les biologistes familiarisés avec les interfaces des bases de données biologiques avec une utilisabilité relativement basse pour les biologistes ayant une faible familiarisation.

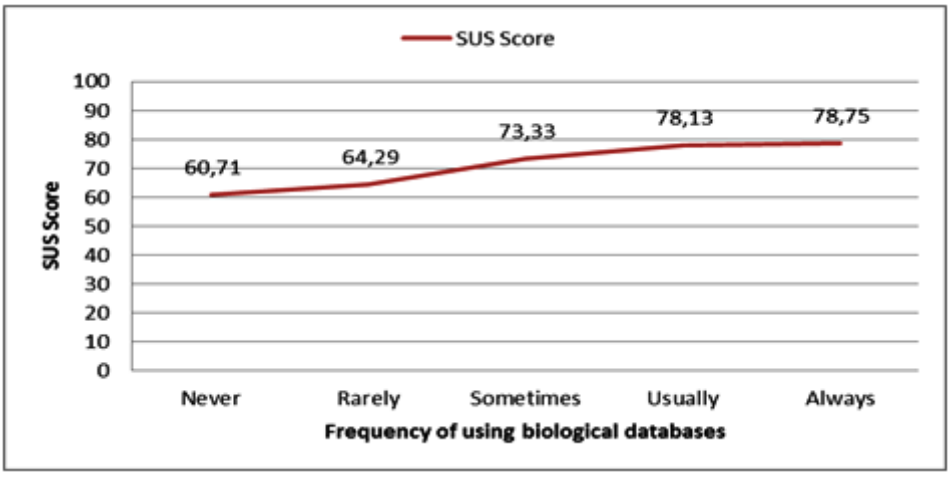

Figure 63. L'évolution des scores SUS du YeastMed en fonction de la familiarisation aux bases de données biologiques.

4.8.2. Performance du Système

Pour illustrer et donner une idée sur les performances du *YeastMed*, nous présentons dans cette section les résultats de l'étude conduite sur les temps d'exécution des trois

principales étapes que *YeastMed* accomplisse lors du traitement d'une requête : Planification, Exécution et Intégration.

L'étude a concerné 50 requêtes réparties sur 5 groupes selon le nombre de sources de données participants dans la réponse à la requête (de 1 à 5). Toutes les requêtes ont été exécutées sur une machine Pentium 4 dotée d'un processeur Dual-Core d'une fréquence de 2.33GHz avec une mémoire de 4Go. L'objectif ne consiste pas à fournir une analyse de performances approfondie, mais seulement montrer comment les temps d'exécution se comportent envers une augmentation du nombre des sources de données impliquées. Chaque requête a été exécutée en trois instances avant de calculer la moyenne des temps d'exécution. Les sources de données impliquées dans les requêtes multi-sources sont appelées exactement une seule fois pour chaque requête. Ceci a comme objectif de fournir une certaine uniformité à l'étude.

La Figure 64 illustre les résultats obtenus. Il ne montre pas grande différence dans les temps de planification quand le nombre des sources impliquées augmente. Le temps de planification passe de 1.149 seconds pour les requêtes impliquant une source à 1.252 seconds pour celles appelant 5 sources. Cependant, le temps d'exécution de la requête montre un comportement différent. Il croît avec le nombre de sources impliquées. Ceci était attendu parce que l'exécution des sous-requêtes dans *YeastMed* fait usage des services de données qui ne sont pas appelés simultanément mais séquentiellement due au fait que l'appel d'un pourrait dépendre du résultat retourné par un autre. En ce qui concerne le temps d'intégration, il croît aussi mais légèrement si l'on compare à celui d'exécution. Il passe de 1.149 seconds pour les requêtes basées sur une seule source à 5.589 seconds pour les requêtes faisant intervenir cinq sources.

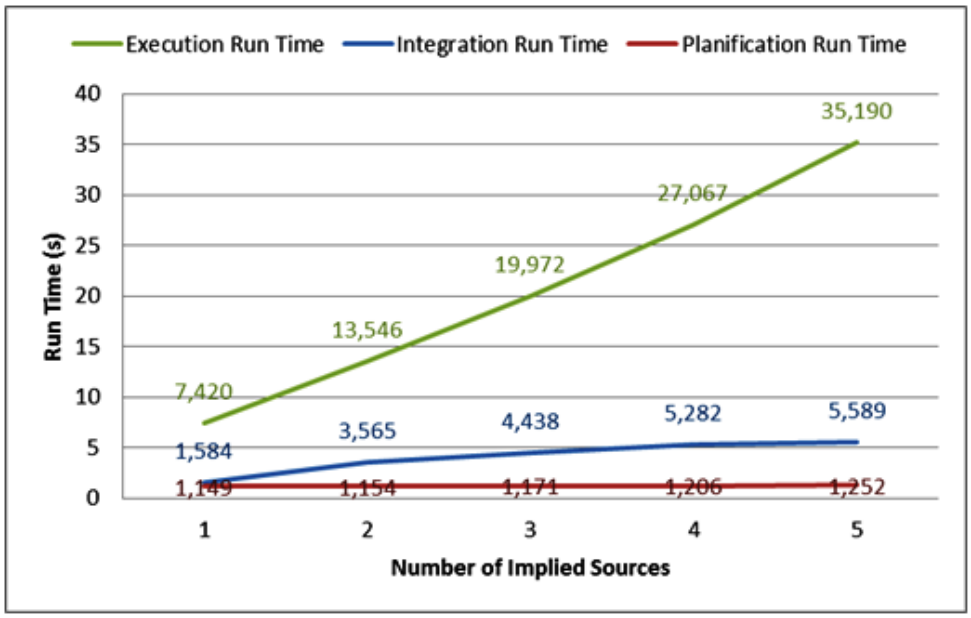

Figure 64. L'évolution des temps d'exécution relatives aux phases de planification, exécution et intégration du processus du traitement de requêtes dans YeastMed en fonction du nombre de sources de données mises en jeu.

Dans *YeastMed*, la phase d'intégration est sollicitée même si une seule source est mise en jeu. Ceci est dû au fait qu'en plus de processus d'intégration de données, Elle réalise la transformation du résultat XML retourné par les services de données en une instance RDF de l'ontologie du *YeastMed*.

4.9. Conclusion

Saccharomyces cerevisiae est une levure ayant une place très importante dans le domaine de la recherche en biologie. Ses gènes sont presque tous caractérisés et sert de référence pour étudier d'autres gènes orthologues. Plusieurs sources de données stockant et organisant les données de cette espèce sont apparues avec le développement des technologies d'internet. Les biologistes font souvent recoure a ces sources pour analyser les résultats de leurs expérimentations. Vue l'hétérogénéité et la diversité de ces sources, leurs exploitation est devenu un travail difficile et fatiguant. Les biologistes n'ont pas, dans la plupart des cas, des connaissances solides en informatique qui leurs permettent d'interroger convenablement chaque source de données et tirer le maximum d'avantages. Les systèmes d'intégration de données apportent alors une solution aux biologistes en leurs permettant d'interroger un ensemble de sources de données d'une manière simultanée et transparente. *YeastMed* fait partie de ces systèmes.

Dans ce chapitre, nous avons parlé du *YeastMed*. Le système qui permet d'interroger un ensemble de sources de données de l'espèce *Saccharomyces cerevisiae*. Nous avons présenté, avant tout, quelques caractéristiques de cette fameuse espèce. Ainsi, nous avons parlé de son cycle de vie et de ses modes de multiplication. Nous avons aussi cité les caractéristiques qui ont été derrière l'importance qu'elle a eu dans les recherches en biologie et par conséquent devenir un organisme modèle. Son génome a été aussi décrit afin que le lecteur ait une idée sur ses composants.

L'importance de la levure se manifeste aussi par le grand nombre des sources et bases de données qui stockent, organisent et présentent des données sur les éléments génomiques, protéiques, métabolique, etc. Nous avons décrit cinq des plus importantes sources de données de la levure et qui sont manipulées par le système *YeastMed*.

Le focus a été fait, cependant, sur l'architecture et le fonctionnement du système. Nous avons vu comment *YeastMed* fournit une interface unique permettant à ses utilisateurs de spécifier des requêtes à travers un formulaire HTML simple. *YeastMed* fait usage d'une ontologie de domaine, qui sert de modèle pour les données des sources qu'il manipule, pour supporter les requêtes des utilisateurs. Il se base sur le médiateur SB-KOM pour traiter les requêtes qu'il reçoit et fait appel à un ensemble de services Web pour interroger les sources de données.

Discussion et conclusion générale

Les outils génétiques qui ont été développés au sein de la levure *Saccharomyces cerevisiae* ont fait de cette espèce l'un des modèles cellulaires et eucaryotiques les plus importants et adéquats pour la recherche en biologie moléculaire. Cette importance a été accompagnée par la création de plusieurs sources et bases de données, publiques ou privées, qui se sont constituées autour d'une thématique biologique bien précise, afin de satisfaire des besoins spécifiques. Ainsi, un volume large de données biologiques de cette fameuse espèce est rendu publiquement disponible à travers le Web. Les types des données sont divers, et les ressources sont très nombreuses. Cependant, les données en provenance de ces différentes ressources présentent une hétérogénéité sémantique et syntaxique très importante. La grande diversité de ces données, l'hétérogénéité de leurs représentations, l'autonomie des sources les unes par rapport aux autres, les interfaces différentes et les capacités d'interrogation inégales rendent difficile, voire impossible leur utilisation combinée par les biologistes.

Tout au long de ce travail de thèse, nous avons eu pour objectif la proposition d'une architecture et d'un système qui pourrait aider les biologistes à interroger simultanément des sources de données biologiques de *Saccharomyces cerevisiae*. Le travail entre dans le cadre d'une collaboration entre notre laboratoire de recherche LABIPHABE et le groupe KHAOS de l'université de Malaga.

Nos travaux se placent dans le contexte général de l'intégration de données hétérogènes. Les contributions de ce travail étaient:

- Le développement d'un système de médiation nommé *YeastMed* qui permet aux utilisateurs, intéressés à *Saccharomyces cerevisiae*, d'accéder, via une interface unique, à des sources de données diverses, hétérogènes et réparties sur le Web.
- La résolution du problème de choix de sources. En effet, il est connu que les systèmes d'intégration de données sont eux qui gèrent le choix des sources pour répondre aux requêtes des utilisateurs. Nous avons voulu proposer un système qui donne aux utilisateurs l'avantage de choisir les sources de données à partir desquelles le système doit construire une réponse.

L'un des grands défis actuellement de la bioinformatique est de permettre aux biologistes d'accéder efficacement à plusieurs sources de données ayant chacune un schéma de données spécifique via des procédures automatiques. Cette automatisation devrait aboutir à une véritable coopération entre le biologiste et la machine (ordinateur), pour une recherche plus efficace des informations via le Web et une meilleure exploitation des résultats. Dans cette optique, il a été largement démontré qu'une approche intégrative pour la recherche des données est adéquate pour découvrir des relations entre les données. De nombreux efforts ont été menés pour la mise à disposition et le partage des données génomiques et biologiques pour leur utilisation dans des projets d'intégration.

Les solutions apportées à l'intégration d'informations proposées dans le cadre du Web sémantique tirent parti des recherches déjà effectuées dans le domaine. Généralement trois grandes approches d'intégration des sources d'information ont alors été proposées : les approches appelées, consécutivement, bases de données fédérées, entrepôt de données et médiateur.

Dans l'approche bases de données fédérées, les sources sont indépendantes les unes des autres et des connections entre toutes les paires de sources que l'on souhaite faire communiquer sont établies. Cette approche est très simple mais aussi très coûteuse puisqu'elle permettent à n sources de communiquer chacune avec n-1 sources, ce qui implique donc d'écrire n(n-1) ensembles de connections entre les sources afin de supporter convenablement les requêtes.

Dans une approche entrepôt de données, les données en provenance de sources diverses sont intégrées en fournissant une vue unifiée des données qui sont stockées dans une seule structure appelée entrepôt de données. Ce dernier est, depuis plusieurs années, très couramment utilisé dans le domaine bioinformatique, pour stocker des informations en provenance de sources multiples et hétérogènes.

L'approche à base de médiateur consiste à définir une interface entre l'agent (humain ou logiciel) qui pose une requête et l'ensemble des sources accessibles via le Web, potentiellement pertinentes pour répondre à la requête. L'objectif est de donner l'impression d'interroger un système centralisé et homogène alors que les sources interrogées sont distantes, hétérogènes et autonomes. Cette approche présente l'intérêt de pouvoir construire un système d'interrogation des sources de données sans toucher aux données qui restent stockées dans leurs sources d'origine. Ainsi, le médiateur ne peut pas évaluer directement les requêtes qui lui sont posées puisqu'il ne contient pas de données, ces dernières sont stockées de façon distribuée dans des sources indépendantes.

L'intégration de sources de données biologiques, hétérogènes, autonomes et évolutives pose des problèmes à la fois structurels et sémantiques. Les travaux concernant l'hétérogénéité structurelle sont relativement anciens et ont abouti à diverses approches permettant de la traiter dans le contexte des fédérations de bases de données. L'hétérogénéité sémantique, par contre, reste la plus importante difficulté. Plusieurs systèmes récents d'intégration de sources de données hétérogènes utilisent des ontologies afin de résoudre les conflits sémantiques.

1) Exemples de systèmes d'intégration de données biologiques

Plusieurs prototypes de systèmes d'intégration de données ont été proposés dans le domaine de la biologie. Certains, ont été cités comme exemples dans la section 1.4, d'autres vont être introduits dans cette section. Nous allons nous focaliser sur les systèmes d'intégration virtuelle de données afin de les comparer au système *YeastMed*. Ainsi, nous allons parler de deux catégories de systèmes : (a) les systèmes médiateurs et fédérés, et (b) les systèmes basés sur XML.

a) Les systèmes médiateurs et fédérés

Dans cette catégorie nous citons :

- ***Kleisli*** (Davidson, et al., 2001) : c'est un système médiateur englobant un modèle de données relationnel, un langage de requête de haut niveau, et un optimiseur de requêtes puissant. Il tourne au-dessus d'un grand nombre d'adaptateur légers pour accéder à diverses sources de données. Le système Kleisli est hautement extensible. Il peut être utilisé pour prendre en charge plusieurs langages de requêtes de haut niveau en remplaçant son module de langage de requête. Kleisli prend en charge le langage de programmation de Collection (CPL) (Buneman, et al., 1995) et une version relationnelle de SQL. Cependant, contrairement à *YeastMed*, Kleisli n'utilise aucun schéma global (ou ontologie) sur lequel (ou laquelle) un utilisateur peut formuler ses requêtes. En effet, le système se base sur le fait qu'un attribut de requête est liée à un attribut correspondant dans une seule source. Par conséquent, il n'y a pas une intégration de données à travers les différentes sources.

- ***DiscoveryLink*** (Haas, et al., 2001) : c'est un système bioinformatique d'intégration orienté adaptateur. Il repose sur la technologie du projet *Garlic* (Haas, et al., 1999). Il sert comme un middleware entre des applications et un ensemble d'adaptateurs. Les applications se connectent à DiscoveryLink et soumettent une requête SQL vers son schéma global. Les adaptateurs fournissent des informations, spécifiques aux sources, sur les capacités d'interrogation qui aident l'optimiseur à déterminer quelles parties d'une requête peuvent être soumises à chaque source. L'optimiseur de requête prend en considération la vitesse des diverses sources, leurs connexions réseau et la taille de leurs données pour prédire les coûts des différents plans. Cependant, DiscoveryLink ne peut pas traiter les bases de données complexes contenant des données imbriquées. La plupart des données biologiques, malheureusement, sont très imbriquées. Par conséquent, il y a une quantité importante de disparité (mismatch) entre la plupart des sources de données et DiscoveryLink. En outre, il est difficile d'ajouter de nouvelles sources de données ou outils d'analyse. D'autre part, il exige SQL comme langage de requête, ce qui n'est pas facile de manipuler par un biologiste.

  TAMBIS (Stevens, et al., 2000) : c'est un système d'intégration de données à base de médiateur et d'ontologie. Il dispose de trois couches: le modèle conceptuel, le

modèle de mapping et le modèle physique. Dans TAMBIS, la formulation des requêtes se fait à travers une interface graphique où l'utilisateur peut naviguer à travers les différents concepts définis dans le schéma global et choisir ceux appropriés pour sa requête. Comme première étape, le système exprime la requête en *GRAIL* (Rector, et al., 1997). Ensuite, la requête est traduite en une forme interne de requête (QIF), qui est à son tour traduite en un plan d'exécution de requête dépendant d'une source et exprimé en CPL. L'ontologie globale est une représentation du niveau conceptuel unifié des ressources des composants enregistrés. Il fournit un schéma global ainsi qu'un cadre abstrait pour relier, réconcilier et coordonner les concepts dans les sources. Le modèle de mapping convertit une requête formulée en termes de la couche conceptuelle en plans exécutables en termes de chaque source. Le modèle physique soumet les plans exécutables aux différentes sources et renvoie les résultats. Bien que TAMBIS est une solution de niveau supérieur que les autres systèmes, mais son interface graphique est très complexe et nécessite que l'utilisateur comprenne le langage de requête.

- ***BioMediator*** (Cadag, et al., 2007) : c'est un système fédéré d'intégration de données se basant sur XML. Il utilise un schéma de médiation qui permet une grande souplesse de modélisation des données. L'élément central du système BioMediator est sa base de connaissances, qui se compose de descriptions des différentes sources de données, les mappings des sources avec le schéma de médiation, et le schéma de médiation lui-même. Le système comprend également des adaptateurs qui effectuent les traductions syntaxiques en traduisant les résultats retournés dans un document XML, un méta-adaptateur qui mène des traductions sémantiques en faisant la correspondance entre ce document XML et le schéma de médiation, et un processeur de requête qui interroge (en utilisant le langage XQuery) le schéma de médiation. BioMediator est donc dédié aux utilisateurs qui connaissent le langage XQuery et n'est pas prêt à être utilisé par des groupes de recherche extérieurs.

b) Les systèmes basés sur XML

A part la possibilité d'utiliser des approches standard pour l'intégration de données (Halevy, et al., 2006), les approches spécifiques fondées sur l'emploi de XML en bioinformatique ont été aussi proposées. De cette catégorie, nous citons :

- ***AutoMed*** (Zamboulis, et al., 2007) : c'est un système de transformation et d'intégration de données hétérogènes qui offre la possibilité de gérer la transformation/intégration de données virtuelles, matérialisées et hybrides à travers de multiples modèles de données. AutoMed utilise *XML DataSource Schema* (XMLDSS) comme langage de représentation et un type de schéma supportant les annotations de chaque source par des ontologies appropriées. Un schéma XMLDSS peut être extrait automatiquement à partir d'un document XML ou dérivé automatiquement d'un DTD/schéma XML s'il est disponible.

L'approche du système est basé sur: (i) XML comme un format de représentation commun, (ii) XMLDSS comme un type de schéma pour les documents XML de l'entrée et la sortie des services, (iii) la correspondance aux ontologies disponibles; c'est à dire les entrées/sorties des services sont annotées avec des correspondances entre le schéma XMLDSS et quelques ontologies existantes, et (iv) AutoMed toolkit pour transformer automatiquement le schéma XMLDSS de la sortie d'un service donné en un schéma XMLDSS d'entrée d'un autre service.

- ***SWAMI*** (Rifaieh, et al., 2007) définit une architecture middleware riche qui intègrent des bases de données, des formats et des ressources computationnelles différents. La conception de son architecture comprend une *couche de présentation* qui reçoit les requêtes des utilisateurs, les transmet au *Core workbench Application*, et renvoie les résultats d'application à l'utilisateur par le même chemin. Le *Core workbench Application* se compose de quatre éléments principaux: *Le module utilisateur* qui reçoit les données et les instructions de la couche de présentation. *Le module courtier* qui interagit avec les autres modules via des APIs et sert de coordinateur en utilisant un service de registre qui gère les informations sur tous les services et bases de données. Ensuite, *les modules d'outils et de données*, qui sont conceptuellement identiques, abstraient respectivement les applications et les bases de données, et réalisent leurs fonctions en orchestrant une série de services. XML est utilisé pour la spécification déclarative de services.

2) Ce que propose YeastMed en contrepartie

YeastMed est le premier système bioinformatique d'intégration de données qui adopte l'approche de médiation pour intégrer des sources de données spécifiques de la levure *Saccharomyces cerevisiae*. Il se base sur une ontologie de domaine (nommé YeastMedOnto) qui joue le rôle de schéma global et prend en charge les requêtes des utilisateurs. Contrairement aux systèmes cités dans la section précédente, *YeastMed* a une interface graphique facile à utiliser. Cette interface fait usage du YeastMedOnto pour permettre aux utilisateurs d'exprimer leurs requêtes en un langage naturel simple. Ainsi, les utilisateurs n'ont pas besoin de connaître un langage de requête spécifique pour utiliser le système. En outre, grâce à sa conception modulaire, *YeastMed* fournit la possibilité d'ajouter facilement de nouvelles sources de données ou des outils d'analyse.

L'intégration dynamique est un problème très important pour les systèmes traditionnels basés sur l'approche médiateur. Ils sont généralement développés comme des systèmes monolithiques et leur architecture basée sur des adaptateurs implique un haut degré de couplage entre les composants du système. En général, ils ne fournissent pas de solutions évolutives et réutilisables. Par sa conception modulaire et le découplage de tous les composants de *YeastMed*, nous avons cherché de sortir de l'architecture traditionnelle de médiation et de fournir une plateforme flexible pour l'intégration des sources de données de la levure. Ainsi, le système peut être facilement mis à jour et étendu par d'autres sources. Dans la plupart des cas, ce qui est nécessaire pour intégrer une nouvelle source est l'ajout de ses composants, à savoir la publication d'un service Web qui sous-tend cette

source, l'ajout d'une vue sémantique de la source à l'ontologie et la définition des règles de correspondance. Le reste des composants du système ne sont pas modifiés, ce qui réduit les coûts de maintenance du système.

En outre, l'utilisation des services Web au lieu des adaptateurs classiques offre la possibilité de les réutiliser par d'autres médiateurs ou toutes autres applications permettant l'accès aux données. C'est le cas du service de recherche rapide que nous proposons dans le site Web de *YeastMed*. Ce service est une valeur ajoutée de l'architecture du système. Il permet d'utiliser indépendamment les services Web pour rechercher des entrées et accéder directement aux sources de données intégrées sans passer par le système médiateur.

Alors que certains systèmes à base de médiateur ont besoin d'un langage de requête spécifique ou proposent des interfaces utilisateur graphiques complexes, *YeastMed* reçoit des requêtes conjonctives exprimées en termes de son ontologie. Même si nous estimons qu'il n'est pas très compliqué pour les biologistes d'exprimer leurs besoins en requêtes conjonctives, nous avons proposé une interface simple où les requêtes sont exprimées en un langage naturel à travers un formulaire simple. Toutes les traductions aux requêtes conjonctives ne sont pas visibles aux utilisateurs.

D'autre part, il est connu que les biologistes ont leurs propres préférences vis-à-vis les sources de données. Dans *YeastMed* nous avons pris en compte ceci en donnant aux utilisateurs la possibilité de préciser à partir de quelle source préfèrent obtenir des réponses. L'ontologie du *YeastMed* contient certains concepts concernant les sources qui permettent à l'utilisateur d'exprimer ses préférences sur les sources de données. Par exemple, un utilisateur peut spécifier SGD en tant que source de données à partir de laquelle il veut obtenir des entrées. Cela est faisable en sélectionnant "SGD Entry" dans le formulaire de requêtes. La spécification d'une source de données ne signifie pas réduire les contraintes à être seulement appliquées sur les données de cette source. Les utilisateurs peuvent spécifier une source à partir de laquelle il veut obtenir des entrées et appliquer des contraintes sur les données en provenance d'autres sources. Par exemple, un utilisateur peut exiger des entrées à partir de SGD décrivant une caractéristique chromosomique régulée par un facteur de transcription ayant le nom standard "*Rtg1*". Cela se traduit par la requête conjonctive suivante:

```
Ans(E) := SGDEntry(E), describes(E,F), ChromosomalFeature(F),
RegulatedBy(F,R), TranscriptionFactor(R), hasName(R, "Rtg1");
```

Dans cette requête conjonctive, les entrées de SGD sont sollicitées, mais toutes les contraintes sont mises sur les données résidant dans Yeastract (données relatives au facteur de transcription). Si l'entrée de SGD n'a pas été spécifiée, les entrées du résultat seront retournées par défaut de Yeastract. *YeastMed* est capable de trouver l'équivalent de ces entrées en SGD.

3) Perspectives

La récente expansion des sources de données biologiques sur le Web les a mis à disposition d'un nombre sans cesse croissant de chercheurs, ouvrant ainsi de très nombreuses perspectives d'innovation. La biologie a ainsi pris une nouvelle dimension : anciennement divisée en plusieurs disciplines, elle est devenue intégrative et offre désormais de belles perspectives d'appréhension de la complexité du monde vivant. L'intégration de données vise à combler le fossé qui existe entre producteurs et consommateurs de données, particulièrement dans ce domaine. Dans le cadre de cette thèse, nous avons orienté nos recherches afin de rapprocher ces différents acteurs. Ainsi, De nombreuses perspectives importantes peuvent être envisagées. Nous présentons succinctement celles qui nous paraissent être les plus intéressantes.

- Intégrer plus de sources de données pour enrichir le système *YeastMed* et donner à l'utilisateur la possibilité d'interroger plus de données.
- Améliorer le service du système en intégrant des outils bioinformatiques qui facilite la visualisation et l'analyse de données.
- Développer des outils et des applications permettant l'accès aux données publiées dans le nuage "Linked Data".

Annexe A : Réécriture de requêtes dans l'approche LAV

Soit (*G*, *S*, *M*) un système de médiation. Considérons une requête *Q* sur le schéma *G*. Le médiateur doit être capable de calculer les réponses de *Q* à partir des extensions des vues. Ce problème est connu sous le nom de répondre à une requête en utilisant les vues (query answering using views.) (Abiteboul and Duschka, 1998; Goasdoué and Rousset, 2004; Halevy, 2001). Pour préciser ce problème, supposons que les différentes sources de données forment une seule base de données, appelée base de données globale, notée *D*. Le médiateur qui est chargé du traitement de la requête, ne connaît pas le contenu réel de *D*. Il n'a en réalité qu'une vision partielle du contenu de *D* à travers les extensions des vues.

A.1 Requête conjonctive

Dans un système d'intégration, l'interrogation s'effectue généralement en utilisant des requêtes conjonctives à base de règles définies à l'aide du vocabulaire du schéma global qui exprime les vues sur les différentes sources. On parle alors d'interrogation basée sur les vues (Levy, et al., 1995). Une requête conjonctive a la forme d'une règle se présentant comme suit (Ullman., 1989):

$$q(\bar{X}) :- r_1(\bar{X}_1),\ \ldots,\ r_n(\bar{X}_n)$$

où q, et $r_1, \ldots, r_n$ sont des noms de prédicats. Les noms des prédicats $r_1, \ldots, r_n$ réfèrent aux relations de la base de données. Les prédicats $r_1(\bar{X}_1), \ldots, r_n(\bar{X}_n)$ sont les *subgoals* du corps de la requête. L'atome $q(\bar{X})$ est appelé la tête de la requête. Les tuples $\bar{X}, \bar{X}_1, \ldots, \bar{X}_n$ contiennent soit des variables, soit des constantes. Les variables apparaissant dans la tête de la requête sont les variables distinguées, et toutes les autres sont les variables existentielles. Pour garantir que la requête soit saine, chaque variable apparaissant dans la tête doit obligatoirement apparaître quelque part dans le corps. Les prédicats sous forme de comparaison arithmétique $<, \leq, =, \neq$..., peuvent contribuer à la construction d'une requête. Dans ce cas, si une variable X apparaît dans un prédicat de comparaison, elle doit également apparaître dans un subgoal ordinaire.

Exemple : Soit une base de données biologique adoptant un modèle relationnel contenant les relations suivantes :

Genes(g_name, sp_id, p_name);

Products(p_name, p_seq, p_long, p_type);

Species(sp_id, sp_name);

La requête SQL suivante a pour but d'interroger la base de données pour tous les gènes dont le produit est une protéine qui ne dépasse pas 300 acides aminés en longueur dans sa séquence protéique, elle renvoie le nom du gène, la séquence de son produit et le nom de son espèce :

Select g_name, p_seq, sp_name

From Genes, Products, Species

Where Genes.p_name = Products.p_name and Genes.sp_id = Species.sp_id and Products.p_long ≤ 300, and

Products.p_type = "Protein";

Dans la notation des requêtes conjonctives, la requête ci-dessus serait exprimée comme suit :

Q(g_name, p_seq, sp_name) :- Genes(g_name, sp_id, p_name),

Products(p_name, p_seq, p_long ≤ 300, p_type = "Protein"),

Species(sp_id, sp_name);

Dans cette notation, les prédicats de jointure de SQL sont exprimés par les occurrences multiples de la même variable dans différents subgoals du corps (e.g. *g_name* et *sp_id*).

A.2 Inclusion et équivalence

Les notions d'inclusion et d'équivalence de requêtes (Halevy, 2001), permettent la comparaison entre les différentes reformulations de requêtes. Elles seront utilisées lors du test de la justesse d'une réécriture d'une requête en termes d'un ensemble de vues. Dans les définitions qui suivent, nous supposons que les réponses aux requêtes sont des ensembles de tuples.

Définition : Inclusion et équivalence de requêtes :

Une requête Q_1 est dite incluse dans une requête Q_2, dénotée par $Q_1 \sqsubseteq Q_2$, si pour une base de données D, l'ensemble des tuples calculé pour Q_1 est un sous ensemble de l'ensemble des tuples calculé pour Q_2. Les deux requêtes sont dites équivalentes si $Q_1 \sqsubseteq Q_2$ et $Q_2 \sqsubseteq Q_1$.

Le problème d'inclusion et d'équivalence de requêtes a été étudié extensivement dans la littérature, nous citons en particulier l'inclusion des requêtes conjonctives et d'unions (Chandra and Merlin, 1977; Sagiv and Yannakakis, 1980) et des requêtes conjonctive avec des prédicats de comparaison (Klug, 1988; Kolaitis, et al., 1998; Levy and Sagiv, 1993; Zhang and Ozsoyoglu, 1993).

A.3 Réécriture de requêtes en termes de vues

Etant données une requête Q et un ensemble de définitions de vues $V_1, \ldots, V_m$, une réécriture de la requête en utilisant des vues est une expression de requête Q' qui réfère uniquement aux vues $V_1, \ldots, V_m$. Une réécriture qui réfère uniquement aux vues a été appelée réécriture complète dans (Levy, et al., 1995).

Deux types de réécritures à distinguer : Les réécritures équivalentes et les réécritures contenues au maximum.

Définition : réécriture équivalente

Soient Q une requête et $\mathcal{V} = V_1, \ldots, V_m$ un ensemble de définitions de vues. La requête Q' est une réécriture équivalente de Q en utilisant $\mathcal{V}$ si :

- Q réfère seulement aux vues dans $\mathcal{V}$, et
- Q' est équivalente à Q.

Définition : réécriture contenue au maximum

Soit Q une requête, $\mathcal{V} = V_1, \ldots, V_m$ un ensemble de définitions de vues, et $\mathcal{L}$ un langage de requêtes. La requête Q' est une réécriture contenue au maximum en utilisant $\mathcal{V}$ modulo $\mathcal{L}$ si :

- est une requête en $\mathcal{L}$ qui réfère uniquement aux vues de $\mathcal{V}$,
- est contenue dans Q, et
- Il n'y a pas d'écriture $Q_1 \in \mathcal{L}$, tel que $Q' \subseteq Q_1 \subseteq Q$ et Q_1 n'est pas équivalente à Q'.

Les réécritures équivalentes d'une requête sont exigées quand les vues matérialisées sont utilisées pour l'optimisation de requêtes et l'indépendance physique de données. Dans le contexte de l'intégration de données, On a besoin habituellement de se contenter des réécritures contenues au maximum, car les sources ne contiennent pas nécessairement toutes les informations exigées pour fournir une réponse équivalente à la requête (Halevy, 2001).

i) État complet et complexité pour trouver les réécritures de requêtes.

La première question qu'on peut poser à propos d'un algorithme de réécriture de requêtes en utilisant de vues peut concerner sa complétude : étant données une requêtes Q et un ensemble de vues $\mathcal{V}$, l'algorithme sera-t-il capable de trouver une réécriture de Q en termes de $\mathcal{V}$ si une existe ? La réponse à cette question dépend entre autre du langage de requête utilisé pour exprimer les réécritures. Parfois le caractère expressif limité du langage peut empêcher l'algorithme de trouver une réécriture malgré l'existence d'une. Dans le cas de l'inexistence d'une réécriture équivalente, nous essayons de trouver une réécriture contenue au maximum.

La complexité des algorithmes de réécritures de requêtes peut être discutée sous différents langages et suppositions de modélisations. En général, ils sont en NP. Une discussion détaillée pour des cas spécifiques est donnée dans (Halevy, 2001).

ii) Algorithmes de reformulation de requêtes

Vue l'importance de la réécriture de requêtes dans un système de médiation, nous allons décrire, dans ce qui suit, quelques algorithmes de réécritures de requêtes par utilisation de vues. Nous allons détailler trois algorithmes à travers d'exemples: l'algorithme *bucket*, l'algorithme *inverse-rules* et l'algorithme *MiniCon*, puis nous allons faire une petite comparaison entre eux.

a) L'algorithme Bucket

L'algorithme Bucket a été développé dans le cadre du système Information Manifold (Levy, et al., 1996). Son idée principale, est la suivante : le nombre de réécritures qui ont besoin d'être considérées, peut être extrêmement réduit si nous considérons chaque subgoal en isolation, et nous déterminons quelles vues peuvent être pertinentes pour chaque subgoal.

Etant donnée une requête Q, l'algorithme agit en deux étapes. Dans la première, il crée un bucket ("seau") pour chaque subgoal de Q qui ne figure pas dans les subgoals des prédicats de comparaison. Ce bucket contient les vues qui sont pertinentes pour répondre au subgoal particulier. D'une façon générale, une vue V est mise dans un bucket d'un subgoal g de la requête si la définition de V contient un subgoal g1 tel que :

- g et g1 peuvent être unifiés, et
- après l'application de l'unification à la requête et aux variables de la vue qui apparaissent dans la tête, les prédicats dans Q et dans V sont mutuellement satisfiables.

Un subgoal g peut s'unifier avec plus d'un seul subgoal dans une vue V, et dans ce cas, le bucket de g contiendra des occurrences multiples de V.

Dans la seconde étape, l'algorithme considère des réécritures de la requête qui sont des requêtes conjonctive. Chacune consiste en un conjoint d'un seau. Spécifiquement, pour chaque choix possible d'un élément à partir de chaque seau, l'algorithme vérifie si la conjonction résultante est contenue dans la requête Q. Sinon, l'algorithme vérifie si l'addition d'atomes des prédicats de comparaison peut mettre la conjonction résultante contenue dans la requête. Si oui, la réécriture est additionnée à la réponse. Ainsi, le résultat de l'algorithme bucket est une union de réécritures conjonctives.

Dans le cas où les prédicats de comparaison arithmétique sont absents dans la requête (mais dans les définitions des vues ne le seraient pas) l'algorithme bucket garantit de retourner une réécriture contenue au maximum de la requête par utilisation de vues.

Exemple : Supposons un système d'intégration de données biologiques incluant les relations suivantes dans son schéma médiateur :

Gene(g_name, g_sequence, species);

Protein(p_name, p_sequence);

codesFor(g_name, p_name);

hasFunction(p_name, function) ;

Soit les définitions de vues suivantes :

V_1(gn, pn, sp) :- gene(gn, gseq, sp), CodesFor(gn, pn),
hasFunction(pn, func= "Phosphorylation");

V_2(gseq, pseq, sp) :- Gene(gn, gseq, sp), Protein(pn, pseq),
CodesFor(gn, pn) ;

V3(gn, pn, func) :- Gene(gn, gseq, sp), CodesFor(gn, pn),
hasFunction(pn, func) ;

V4(pn, pseq) :- Protein(pn, pseq), Gene(gn, gseq,
sp = "Saccaromyces cervisae"), CodesFor(gn, pn) ;

et la requête suivante interrogeant le système pour l'ensemble des gènes de l'espèce Schwanniomyces occidentalis qui codent pour une protéine ayant une séquence et une fonction connues.

Q(g_name, p_seq, function) :- Gene(g_name, g_sequence,
species = "Schwanniomyces occidentalis"),
Protein(p_name, p_sequence),
codesFor(g_name, p_name),
hasFunction(p_name, function) ;

La première étape de l'algorithme crée un bucket pour chaque subgoal de la requête. La table 4 suivante montre le contenu résultant de chaque bucket. Nous utiliserons les lettres initiales en majuscule pour faciliter la représentation des variables des subgoals de la requête. Les variables marquées se trouvant dans les têtes des vues sont celles qui ne figurent pas dans le domaine du mapping unifiant un subgoal de la requête et un autre de la vue.

Gene(GN, GS, SP)	Protein(PN, PS)	CodesFor(GN, PN)	hasFunction(PN, F)
V_1(gn, pn', sp)	V_2(gseq',pseq,sp')	V_1(gn, pn, sp')	V_1(gn', pn, sp')
V_2(gseq, pseq',sp)	~~V_4(pn, pseq)~~	V_2(gseq',pseq',sp')	V_3(gn', pn, func)
V_3(gn, pn', func')		V_3(gn', pn, func)	
~~V_4(pn', pseq')~~		~~V_4(pn, pseq')~~	

Table 4. Buckets générés par l'algorithme Bucket.

Le bucket de Gene(GN, GS, SP) contient les trois vues V_1, V_2 et V_3 à cause du mapping entre l'atome Gene de la requête et l'atome correspondante dans les trois vues :

(g_name ----> gn, g_seq ----> gseq, species ----> sp)

La vue V_4 ne peut pas être incluse dans ce bucket ni dans les autres à cause des prédicats sp = "Saccharomycess cervisea" et sp = "Schwanniomycess occidentalis" qui ne sont pas réciproquement consistants.

Le bucket de Protein(PN, PS) contient une seule vue, c'est la vue V_2 à cause du mapping :

(p_name ----> pn, p_seq ----> pseq)

Le bucket de CodesFor(GN, PN) contient les vues V_1, V_2 et V_3 selon le mapping :

(g_name ----> gn, p_name ----> pn)

Et finalement le bucket de hasFunction(PN, F) contient les vues V_1 et V_3 à cause du mapping suivant :

(p_name ----> pn, function ----> func)

Dans une deuxième étape, l'algorithme combine les éléments à partir des buckets. Dans notre exemple, nous commençons par une réécriture incluant le premier élément de chaque bucket, c'est-à-dire :

Q'(g_name, p_seq, function) :- V_1(g_name, p_name', species),
V_2(g_seq', p_seq, species'),
V_1(g_name, p_name, species'),
V_1(g_name', p_name, species');

qui peut être réécrite en :

Q'(g_name, p_seq, function) :- V_1(g_name, p_name, species),
V_2(g_seq', p_seq, species);

Comme il peut être vérifié, cette écriture n'est pas contenue dans la requête originale. Cependant, avec l'ajout du prédicat sp = "Schwanniomyces occidentalis" nous pouvons obtenir une réécriture contenue dans la requête. De cette façon, l'algorithme dégage un ensemble de réécritures contenues dans la requête originale, et le résultat sera l'union des tuples qu'elles renvoient.

b) L'algorithme inverse-rules

Comme l'algorithme seau, l'algorithme des règles inversées a été proposé dans le contexte d'un système d'intégration de données (Duschka and Genesereth, 1997). Il se base sur la construction d'un ensemble de règles qui inversent les définitions des vues. C'est-à-dire des règles qui montrent comment calculer les tuples des relations du schéma global à partir des vues (Halevy, 2001). Ce processus donne l'impression d'obtenir des définitions GAV à partir des définitions LAV. Autrement dit, ce n'est pas la requête qui est réécrite, mais les définitions des vues de telle façon qu'on peut facilement obtenir une réponse pour la requête originale à partir des règles réécrites.

Pour chaque vue, et pour chaque subgoal intervenant dans la construction du corps de la vue, une règle inversée est construite. Les variables existentielles qui apparaissent dans

les définitions des vues sont remplacées par un symbole d'une fonction qui prend tous les attributs de la tête de la vue comme entrée. Ceci constitue un mapping qui assure que les équivalences de valeurs entre les variables ne sont pas perdues.

La réécriture de la requête Q par utilisation de l'ensemble de vues $\mathcal{V}$ est le programme datalog* qui inclut les règles inversées de $\mathcal{V}$ et la requête Q.

Exemple : Soit la définition de la vue V suivante :

V(gn, gseq, pn, sp) :- Gene(gn, gseq, sp), Codesfor(gn, pn) ;

L'algorithme inverse cette définition en écrivant une "règle inversée" (*inverse rule*) pour chaque subgoal de la définition de la manière suivante :

Gene(f_1(gn, gseq, pn, sp), gseq, sp) :- V(gn, gseq, pn, sp) ;

CodesFor(f_1(gn, gseq, pn, sp), pn) :- V(gn, gseq, pn, sp) ;

Les règles inversées ont la signification suivante. Un tuple de la forme (gn, gseq, pn, sp) dans d'extension de la vue V est un témoin des tuples des relations Gene et CodesFor, de telle façon qu'il nous dit deux choses :

- la relation *Gene* contient un tuple de la forme *(X, gseq, sp)* ;
- la relation *CodesFor* contient un tuple de la forme *(X, pn)*, pour la même valeur de *X*.

Dans le but d'exprimer l'information qui fait que la valeur inconnue de X est la même dans les deux atomes, nous faisons référence à elle en utilisant le terme fonctionnel f_1(gn, gseq, pn, sp) qui prend en entrée tous les attributs de *V*. f_1 assure un mapping des deux atomes à une même valeur.

Soit la requête *Q* suivante :

Q(gn, pn) :- Gene(gn, gseq, "Schwanniomyces occidentalis"), CodesFor(gn, pn)

Supposons que la vue *V* inclut les tuples :

V = {

("AMY1", "AMY_Seq", "Alpha-amylase 1", "Sch. occ.");

("Pgal", "Pgal_Seq", "Galactose Permease", "Ecoli");

("HXK", "HXK_Seq", "Hexokinase", "Sch. occ.");

("holC", "holC_Seq", "DNA polymerase III", "Pseud. aerug.");

("CYC1", "CYC1_Seq", "Cytochrome C", "Sch. occ.");

}

L'algorithme calculerait les tuples suivants :

Gene(f_1("AMY1", "AMY_Seq", "Alpha-amylase 1", "Sch. occ. "), "AMY_Seq",

"Sch. occ. ");

Gene(f_1("Pgal", "Pgal_Seq", "Galactose Permease", "E. coli"), "Pgal_Seq", "Ecoli");

Gene(f_1("HXK", "HXK_Seq", "Hexokinase", "Sch. occ."), "HXK_Seq", "Sch. occ.");

Gene(f_1("holC", "holC_Seq", "DNA polymerase III", "Pseud. aerug."), "holC_Seq", "Pseud. aerug.");

Gene(f_1("CYC1", "CYC1_Seq", "Cytochrome C", "Sch. occ."), "CYC1_Seq", "Sch. occ.");

CodesFor(f_1("AMY1", "AMY_Seq", "Alpha-amylase 1", "Sch. occ."), "Alpha-amylase 1");

CodesFor(f_1("Pgal", "Pgal_Seq", "Galactose Permease", "E. coli"), "Galactose Permease");

CodesFor(f_1("HXK","HXK_Seq", "Hexokinase", "Sch. occ."), "Hexokinase");

CodesFor(f_1("holC", "holC_Seq", "DNA polymerase III", "Pseud. aerug."), "DNA polymerase III");

CodesFor(f_1("CYC1", "CYC1_Seq", "Cytochrome C", "Sch. occ."), "Cytochrome C");

L'évaluation de la requête *Q* sur ces tuples donne le résultat suivant :

(("AMY1", "Alpha-amylase 1"), ("HKX", "Hexokinase"), ("CYC1", "Cytochrome C"));

Cette algorithme garantit le renvoie d'une réécriture contenue au maximum.

c) L'algorithme MiniCon

L'algorithme MiniCon est une version améliorée de l'algorithme bucket (Pottinger and Halevy, 2001). Il s'applique en deux étapes : il commence de la même manière que l'algorithme Bucket. Il calcule des buckets, un pour chaque subgoal de la requête. Cependant une fois l'algorithme trouve un mapping partiel entre un subgoal *g* de la requête et un subgoal g_1 d'une vue V, il change de perspective et s'intéresse aux variables de la requête. L'algorithme considère les prédicats de jointure de la requête et trouve l'ensemble additionnel minimal de subgoals qui ont besoin d'être mappés avec les subgoals de V, étant donné que *g* sera mappé avec g_1.

Cet ensemble de subgoals et de l'information du mapping est appelé une description MiniCon (MCD), et peut être vue comme une généralisation des buckets.

Dans la deuxième phase, l'algorithme combine les MCDs pour produire des réécritures. Il est important de noter que grâce à la méthode de construction de ces MCDs, l'algorithme

MiniCon n'exige pas de tests d'inclusion (containment) dans la deuxième phase, ce qui donne une accélération additionnelle par rapport à l'algorithme bucket.

Un MCD d'une requête Q sur une vue V est un tuple de la forme $(h_C, V(Y)_C, \varphi_C, G_C)$ avec :

- h_C est un homomorphisme tête sur V. C'est un mapping h à partir de l'ensemble des variables de V ($Vars(V)$) vers lui-même, c'est l'identité sur les variables existentielles, mais peut égaler les variables distinguées.
- $V(\bar{Y})_C$ est le résultat d'application du h_C à V, c'est $\bar{Y} = h_C(\bar{A})$, où $\bar{A}$ sont les variables de la tête de V,
- φ_C est un mapping partiel de l'ensemble des variables de la requête $Vars(Q)$ vers $h_C(Vars(V))$,
- G_C est un sous ensemble de subgoals dans Q qui sont couvert par quelques subgoals dans $h_C(V)$ et le mapping φ_C. Un subgoal *g* de la requête est couvert par un subgoal g_1 d'une vue si étant donné un mapping τ à partir du $Vars(Q)$ à $Vars(V)$, $\tau(g) = g_1$.

La construction des MCDs est basée sur les observations suivantes faites sur les propriétés des réécritures de la requête.

Soit *C* un MCD de Q sur V. *C* peut être utilisé dans une réécriture non redondante de Q seulement si :

- <u>*Clause 1*</u> : pour chaque variable *x* de Q qui est dans le domaine de φ_C, $\varphi_C(x)$ est un variable tête dans $h_C(V)$.
- <u>*Clause 2*</u> : si $\varphi_C(x)$ est une variable existentielle dans $h_C(V)$, alors pour chaque subgoal *g* de la requête qui inclut *x*, (1) toutes les variables de *g* sont dans le domaine de φ_C, et (2) $\varphi_C(g) \in h_C(V)$

La clause 1 est la même que l'algorithme bucket. La clause 2 illustre le fait que pour chaque variable *x* appartenant à un prédicat de jointure qui n'est pas forcé par la vue, doit être dans la tête de celle-ci pour que le prédicat de jointure puisse être appliqué par un autre subgoal dans la réécriture.

Exemple : Considérons les définitions des vues et la requête suivantes :

V_1(pn, func) :- Protein(pn), hasFunction(pn, func);

V_2(gn) :- Gene(gn), Protein(pn), CodesFor(gn, pn);

V_3(gn, pn) :- Gene(gn), CodesFor(gn, pn);

V_4(gn, func) :- Gene(gn), CodesFor(gn, pn), hasFunction(pn, func);

Q(g_name) :- Gene(g_name), Protein(p_name), CodesFor(g_name,
p_name), hasFunction(p_name, function);

L'algorithme commence par créer des buckets pour les différents subgoals de la requête. La table 5 suivant présente le résultat obtenu.

Gene(GN)	Protein(PN)	CodesFor(GN, PN)	hasFunction(PN, F)
	V_1(pn, func).		V_1(pn, func)
~~V_2(gn)~~	~~V_2(gn)~~	~~V_2(gn)~~	
V_3(gn, pn)		V_3(gn, pn)	
V_4(gn, func)		V_4(gn, func)	V_4(gn, func)

Table 5. Buckets générés par l'algorithme MiniCon.

Selon la clause 2, pour que V_2 soit utilisable dans la réécriture de *Q*, nous devons être capable de joindre *Protein* avec *CodesFor* et *hasFunction* sur *p_name*. Puisque ce dernier ne figure pas dans la tête de V_2, et V_2 n'inclut pas *hasFunction*, la jointure avec ce subgoal ne peut pas être établit, et par conséquent l'algorithme élimine V_2.

Les MCDs qui pourraient être créés par l'algorithme sont présentés dans la table 6 qui suit. Les subgoals de la requête ont été numérotés de 1 à 4 pour faciliter la représentation.

$V(\bar{Y})$	h	φ	G
V1(pn, func)	pn ----> pn, func ----> func	P_name ----> pn, function ----> func	2, 4
V3(gn, pn)	gn ----> gn, pn ----> pn	G_name ----> gn, p_name ----> pn	1, 3
V4(gn, func)	gn ----> gn, func ----> func	G_name ----> gn, function ----> func	1, 3, 4

Table 6. MCDs créés par l'algorithme Minicon.

Dans la deuxième phase, la phase de combinaison des MCDs, l'algorithme considère uniquement les MCDs disjoints de telle façon que étant donnée une requête Q, un ensemble de vues $\mathcal{V}$ et un ensemble de MCDs de la requête Q sur les MCDs, les seules combinaisons des MCDs qui peuvent engendrer des réécritures non redondantes de Q sont de la forme $C_1, \ldots, C_l$, où $G_{C_1} \cup \ldots \cup G_{C_l} = Subgoals(Q)$ et pour chaque $i \neq j$, $G_{C_i} \cap G_{C_j} = \emptyset$. $Subgoals(Q)$ est l'ensemble des subgoals de la requête Q.

L'application de cette phase sur notre exemple engendrait uniquement la réécriture suivante :

Q'(g_name) :- V1(p_name, function), V3(g_name, p_name)

d) Comparaison d'algorithmes

L'algorithme Bucket et l'algorithme MiniCon partagent le même esprit en ce qui concerne la construction des buckets et la production des réécritures qui se base sur la combinaison des différents subgoals des buckets. Cependant la déficience de l'algorithme bucket réside dans le fait que les buckets construits peuvent contenir des éléments qui ne sont pas utiles pour réécrire la requête originale. Ces éléments provoquent l'augmentation des nombres des

combinaisons à calculer et à tester pendant la deuxième phase. L'algorithme MiniCon utilise une approche très rigoureuse pour prévenir cette déficience.

L'algorithme Inverse-Rules a l'avantage d'être indépendant de la requête. C'est-à-dire que les règles sont calculées une seule fois et puis elles sont appliquées à n'importe quelle requête plus tard. Aussi les règles sont facilement étendues par des contraintes additionnelles qui peuvent être ajoutées au système. D'autre part, les réécritures obtenues par cet algorithme peuvent contenir des vues qui ne sont pas convenables à la requête car il ignore les prédicats qui imposent des contraintes sur les variables.

Annexe C : Diagrammes UML

UML, dans sa version 2.3, comporte treize types de diagrammes représentant autant de vues distinctes pour représenter des concepts particuliers du système d'information. Ils se répartissent en deux grands groupes :

1. Diagrammes structurels ou diagrammes statiques :
    - diagramme de classes : il présente les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci.
    - diagramme d'objets : il sert à représenter les instances de classes (objets) utilisées dans le système.
    - diagramme de composants : il permet de montrer les composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...).
    - diagramme de déploiement : il sert à représenter les éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage...) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent entre eux.
    - diagramme de paquetages : un paquetage étant un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML, le Diagramme de paquetage sert à représenter les dépendances entre paquetages, c'est-à-dire les dépendances entre ensembles de définitions.
    - diagramme de structures composites : depuis UML 2.x, permet de décrire sous forme de boîte blanche les relations entre composants d'une classe.
2. Diagrammes comportementaux ou diagrammes dynamiques :
    - diagramme de cas d'utilisation : il permet d'identifier les possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système), c'est-à-dire toutes les fonctionnalités que doit fournir le système.
    - diagramme d'activités : permet de décrire sous forme de flux ou d'enchaînement d'activités le comportement du système ou de ses composants.
    - diagramme d'états-transitions : permet de décrire sous forme de machine à états finis le comportement du système ou de ses composants.
    - Diagrammes d'interaction :

- diagramme de séquence : représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.
- diagramme de communication : depuis UML 2.x, représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets.
- diagramme global d'interaction : depuis UML 2.x, permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences (variante du diagramme d'activité).
- diagramme de temps : depuis UML 2.3, permet de décrire les variations d'une donnée au cours du temps.

Ces diagrammes, d'une utilité variable selon les cas, ne sont pas nécessairement tous produits à l'occasion d'une modélisation. Les plus utiles pour la maîtrise d'ouvrage sont les diagrammes d'activités, de cas d'utilisation, de classes, d'objets, de séquence et d'états-transitions. Les diagrammes de composants, de déploiement et de communication sont surtout utiles pour la maîtrise d'œuvre à qui ils permettent de formaliser les contraintes de la réalisation et la solution technique.

GLOSSAIRE

ADN (Acide Désoxyribonucléique) : L'ADN est le support de l'hérédité ou de l'information génétique, car il constitue le génome des êtres vivants et se transmet en totalité ou en partie lors des processus de reproduction. L'ADN détermine la synthèse des protéines. La molécule d'ADN est formée de répétitions de nucléosides constitués de quatre bases différentes (adénine, guanine, thymine, cytosine) qui se présentent en simple brin ou en double brin complémentaires et antiparallèles.

Ad Hoc : Réseau dans lequel l'infrastructure n'est composée que des stations elles-mêmes. Ces dernières acceptent de jouer le rôle de routeur pour permettre le passage de l'information d'un terminal vers un autre sans que ces terminaux soient reliés directement.

API : Application Programming Interface (Interface de programmation publique). Bibliothèque de fonctions destinées à être utilisées par les programmeurs dans leurs applications. Ces fonctions facilitent l'écriture des programmes en fournissant des procédures pour gérer des éléments particuliers : affichage, connexion une base de données, pilotage de périphériques...

Blast : Basic Local Alignment Search Tool est un algorithme de comparaison de séquences. En fonction du site sur lequel l'utilisateur pose sa requête, l'algorithme utilisé pourra s'avérer différent, donc fournir des résultats divergents.

Chaînes de Markov : Une chaîne de Markov est un processus aléatoire portant sur un nombre fini d'états, avec des probabilités de transition sans mémoire.

Chromosome : Ensemble d'éléments d'information liés entre eux dans une même molécule d'ADN. (en biologie cellulaire) : le chromosome est une structure cytologique résultant d'une hypercondensation de la chromatine, permettant la répartition du matériel génétique entre les cellules filles lors de la mitose ou de la méiose. Chromosome vient de "chromos", couleur : allusion à leur capacité de fixer les colorants. Les chromosomes ne sont visibles, en général, que durant la division cellulaire.

Clause de Horn : c'est une règle sous forme d'une implication d'un ensemble de formules atomiques booléennes vers une conclusion (une seule formule atomique). Avec la condition que toutes les variables apparaissant en conclusion des règles apparaissent au moins une fois en prémisses des règles.

Data Mining : ensemble des technologies avancées susceptibles d'analyser l'information d'un entrepôt de données pour en tirer des tendances, pour segmenter l'information ou pour trouver des corrélations dans les données.

Datalog : est un langage de requête et de règles pour les bases de données déductives. Il correspond à un sous ensemble de Prolog. Ses origines remontent aux débuts de la programmation logique.

DTD : Une DTD, acronyme anglais signifiant Document Type Définition, se traduisant par Définition de Type de Document, est un document permettant de décrire un modèle de document SGML ou XML. Une DTD indique les noms des éléments pouvant apparaître et leur contenu constitué par leurs sous-éléments et leurs attributs.

Espèce : Ensemble d'individus ayant des caractéristiques génétiques semblables. Chez les organismes à reproduction sexuée, les individus sont interféconds : le produit de leur croisement est fertile. Chez les procaryotes, l'unité repose sur les similitudes du génome et du phénotype.

Eucaryote : Organisme vivant dont les cellules possèdent un noyau au sein duquel est isolé le génome nucléaire.

FASTA : Un outil d'alignement de séquences ADN ou protéiques proposé par David J. Lipman et William R. Pearson en 1985 dans l'article "Rapid and sensitive protein similarity searches". Le programme original "FASTP" était destiné à la recherche de similarités entre protéines.

Génome : Ensemble du matériel génétique (patrimoine héréditaire) d'un individu ou d'une espèce. Il est constitué de molécules d'acides nucléiques (ADN ou ARN). Les gènes, c'est-à-dire les parties d'ADN porteuses d'une information génétique, ne constituent qu'une partie du génome.

Graphe Conceptuel : c'est un formalisme de représentation de connaissances et de raisonnements. Ce formalisme a été introduit par John F. Sowa (en) en 1984. Depuis cette date, ce formalisme a été développé suivant trois directions principales: interface graphique de la logique du premier ordre, système diagrammatique pour la logique du premier ordre, formalisme de représentation de connaissances et de raisonnement basé sur les graphes.

Homonymie : Caractère d'un mot identique à un autre par la prononciation, mais de sens différent. Des mots homonymes peuvent de plus être homographes : ils se prononcent de la même façon, ont des sens différents, mais s'écrivent de manière identique.

HTML (HyperText Markup Language) **:** Langage de description de pages Web. Un standard initié par le W3C et compatible tous systèmes.

Interopérabilité : c'est le fait que plusieurs systèmes, qu'ils soient identiques ou radicalement différents, puissent communiquer sans ambiguïté et opérer ensemble.

Isoforme : Les isoformes d'une protéine sont les différentes formes qu'elle prend lorsqu'elle est issue de gènes différents, ou du même gène par épissage alternatif.

Locus : Emplacement d'un gène sur un chromosome.

Orthologues : désigne des gènes d'espèces différentes dont les séquences sont homologues et dérivent d'un même gène ancestral puis ont divergés à la suite d'un évènement de spéciation. Peuvent ou non avoir la même fonction.

Perl : un langage optimisé pour extraire des informations de fichiers texte et imprimer des rapports basés sur ces informations. C'est aussi un bon langage pour de nombreuses tâches d'administration système. Il est écrit dans le but d'être pratique (simple à utiliser, efficace, complet) plutôt que beau (petit, élégant, minimaliste). Perl combine les meilleures fonctionnalités de C, sed, awk et sh, de manière telle que les personnes familières de ces langages ne devraient avoir aucune difficulté avec celui-ci.

Pseudomycélium : ensemble de cellules allongées, sans cloisons les séparant

Puce à CGH : La technique d'hybridation génomique comparative (CGH), permet de caractériser les gains et pertes de segments chromosomiques qui ont lieu dans les cellules cancéreuses. Le principe d'une puce à CGH est, comme la puce à ADN, fondé sur l'hybridation. Dans une puce à CGH, on dépose sur une matrice une représentation complète d'un génome sain, chaque spot contenant un BAC marqué par un fluorochrome rouge. On hybride alors la puce avec un ADN tumoral, marqué par un fluorochrome vert. Si dans la tumeur un segment chromosomique était sur-représenté, il y aura un excès d'ADN vert correspondant à ce segment, et après hybridation du mélange de sondes, le segment chromosomique correspondant sera plus vert que rouge. De manière symétrique, si un segment chromosomique était perdu dans la tumeur, le segment correspondant du chromosome normal sera plus rouge que vert. Cette technique permet ainsi de caractériser, avec une résolution d'environ 10-20 mégabases, l'ensemble des gains et pertes présents dans une tumeur donnée, et où pourraient se trouver localisés respectivement des oncogènes et des suppresseurs de tumeurs.

Sémantique : La sémantique est, dans les sciences du langage, opposée à la syntaxe. La syntaxe concerne les règles formelles, alors que la sémantique concerne la signification. Dans le domaine informatique, le but du "Semantic Web" est de permettre aux machines d'échanger des informations en utilisant le sens des mots comme dans les langages naturels. Cet objectif ambitieux nécessite un travail important sur les langages, la structure des systèmes, et les ontologies.

Service Web : Technologie permettant à des applications de dialoguer à distance via Internet indépendamment des plates-formes et des langages sur lesquelles elles reposent.

SGBD : Un SGBD est une collection de logiciels permettant de créer, de gérer et d'interroger efficacement une base de données indépendamment du domaine d'application.

Synonymie : Relation sémantique entre des mots ou des expressions dont les sens sont identiques ou très proches. Le synonyme d'un mot appartient nécessairement à la même catégorie grammaticale que celui-ci. La synonymie absolue, qui fait que deux mots sont interchangeables dans tous les contextes, est très rare. Dans la majorité des cas, la

synonymie est relative ou partielle et les deux mots ne sont interchangeables que dans certains contextes.

Taxonomie : Science des lois de la classification des formes vivantes. Elle inclut la reconnaissance, l'identification des formes vivantes et leur rangement dans une classification.

Terminologie : La terminologie est l'ensemble des termes, rigoureusement définis, qui sont spécifiques d'une science, d'une technique, d'un domaine particulier de l'activité humaine. C'est une discipline qui a pour objet l'étude théorique des dénominations des objets ou des concepts utilisés par tel ou tel domaine du savoir, le fonctionnement dans la langue des unités terminologiques, ainsi que les problèmes de traduction, de classement et de documentation qui se posent à leur sujet

Transcriptome : L'ensemble des ARN messagers (molécules servant de matrice pour la synthèse des protéines) issu de l'expression d'une partie du génome d'un tissu cellulaire ou d'un type de cellule. La caractérisation et la quantification du transcriptome, dans un tissu donné et dans des conditions données, permet d'identifier les gènes actifs, de déterminer les mécanismes de régulation d'expression des gènes et de définir les réseaux d'expression des gènes.

URI (de l'anglais Uniform Resource Identifier) : une courte chaîne de caractères identifiant une ressource sur un réseau (par exemple une ressource Web) physique ou abstraite, et dont la syntaxe respecte une norme d'Internet mise en place pour le World Wide Web.

URL (Uniform Resource Locator) : un URL qui, outre le fait qu'il identifie une ressource sur un réseau, fournit les moyens d'agir sur une ressource ou d'obtenir une représentation de la ressource en décrivant son mode d'accès primaire ou "emplacement" réseau.

URN (Uniform Resource Name) : un URI qui identifie une ressource par son nom dans un espace de noms. Un URN peut être employé pour parler d'une ressource sans que cela préjuge de son emplacement ou de la manière de la référencer.

Web sémantique : N'est pas un Web distinct mais bien un prolongement du Web que l'on connaît et dans lequel on attribue à l'information une signification clairement définie, ce qui permet aux ordinateurs et aux humains de travailler en plus étroite collaboration.

Web : Système basé sur des liens hypertextes, permettant l'accès aux ressources du réseau Internet.

XML (eXtensible Markup Language) : Standard du W3C qui permet de décrire les données et de les structurer de telle sorte qu'elles puissent être échangées entre un large nombre d'applications en différents environnements hardware et software.

Xquery (XML Query) : Langage de requête permettant d'accéder à chacun des éléments d'information d'un document XML, d'en sélectionner des listes et de les manipuler. XQuery est un sur-ensemble de XPath.

BIBLIOGRAPHIE

BioMART Project.

Abdulrehman, D., et al. (2011) YEASTRACT: providing a programmatic access to curated transcriptional regulatory associations in Saccharomyces cerevisiae through a web services interface, Nucleic Acids Research, 39, D136-D140.

Aberer, K. (1995) The Use of Object-Oriented Datamodels for Biomolecular Databases In Proceedings of the 2nd International Workshop on Object-Oriented Computing in the Natural Sciences (OOCNS'95). Grenoble, France, pp. 21-24

Abiteboul, S. and Duschka, O.M. (1998) Complexity of answering queries using materialized views. Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. ACM, Seattle, Washington, United States, pp. 254-263.

Achard, F., Vaysseix, G. and Barillot, E. (2001) XML, bioinformatics and data integration, Bioinformatics, 17, 115-125.

Adam., E. (2000) Modèle d'organisation multi-agent pour l'aide au travail coopératif dans les processus d'entreprise : application aux systèmes administratifs complexes. Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis.

Apweiler, R., et al. (2004) UniProt: the Universal Protein knowledgebase, Nucleic Acids Research, 32, D115-D119.

Arena, A. (2012) Le web sémantique : un projet pour amener le web à son plein potentiel.

Artimo, P., et al. (2012) ExPASy: SIB bioinformatics resource portal, Nucleic Acids Research, 40, W597-W603.

Ashburner, M., et al. (2000) Gene ontology: tool for the unification of biology. The Gene Ontology Consortium, Nat Genet, 25, 25-29.

Ault, M., et al. (2003) Oracle Database 10g New Features : Oracle10g Reference for Advanced Tuning and Administration. Rampant TechPress.

Baader, F., et al. (2003) the Description Logic Handbook Theory, Implementation and Applications. Cambridge University Press.

Bairoch, A. (2000) The ENZYME database in 2000, Nucleic Acids Research, 28, 304-305.

Baker, P.G., et al. (1999) An ontology for bioinformatics applications, Bioinformatics, 15, 510-520.

Balakrishnan, R., et al. (2012) YeastMine—an integrated data warehouse for Saccharomyces cerevisiae data as a multipurpose tool-kit, Database, 2012.

Barrett, T., et al. (2011) NCBI GEO: archive for functional genomics data sets—10 years on, Nucleic Acids Research, 39, D1005-D1010.

Bateman, A., et al. (2000) The Pfam Protein Families Database, Nucleic Acids Research, 28, 263-266.

Baumgartner, W.A., et al. (2007) Manual curation is not sufficient for annotation of genomic databases, Bioinformatics, 23, i41-i48.

Bechhofer, S., et al. (2001) OilEd: A Reason-able Ontology Editor for the Semantic Web. Proceedings of the Joint German/Austrian Conference on AI: Advances in Artificial Intelligence. Springer-Verlag, pp. 396-408.

Bellwood, T., et al. (2005) UDDI Version 3.0.2. OASIS Standard.

Ben Miled, Z., et al. (2004) On the Integration of a Large Number of Life Science Web Databases. In Rahm, E. (ed), Data Integration in the Life Sciences. Springer Berlin Heidelberg, pp. 172-186.

Benedek, J. and Miner, T. (2002) Measuring desirability: New methods for evaluating desirability in a usability lab setting. Proceedings of the Usability Professionals Association (UPA 2002). Orlando, FL.

Benson, D.A., et al. (2012) GenBank, Nucleic Acids Research, 40, D48-D53.

Berman, H., et al. (2007) The worldwide Protein Data Bank (wwPDB): ensuring a single, uniform archive of PDB data, Nucleic Acids Research, 35, D301-D303.

Berners-Lee, T. (2004) W3c semantic web layer cake.

Berners-Lee, T. and Cailliau, R. (1990) WorldWideWeb: Proposal for a HyperText Project.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001) The Semantic Web, Scientific American.

Bernstein, P.A., et al. (2002) Data Management for Peer-to-Peer Computing : A Vision. Workshop on the Web and Databases, WebDB. pp. 89-94.

Bertails, A., Herman, I. and Hawke, S. (2010) Le Web sémantique, Réalités Industrielles, 84 - 89.

Birrell, A.D. and Nelson, B.J. (1984) Implementing remote procedure calls, ACM Trans. Comput. Syst., 2, 39-59.

Bishr, Y. (1998) Overcoming the semantic and other barriers to GIS interoperability, International Journal of Geographical Information Science, 12, 299-314.

Blomberg, A. (2002) Use of two-dimensional gels in yeast proteomics. In Christine, G. and Gerald, R.F. (eds), Methods in Enzymology. Academic Press, pp. 559-584.

Bodenreider, O. (2004) The Unified Medical Language System (UMLS): integrating biomedical terminology, Nucleic Acids Research, 32, D267-D270.

Boeckmann, B., et al. (2003) The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003, Nucleic Acids Research, 31, 365-370.

Bon, E., et al. (2003) Molecular evolution of eukaryotic genomes: hemiascomycetous yeast spliceosomal introns, Nucleic Acids Res, 31, 1121-1135.

Bonnefoy, N. and Fox, T.D. (2002) Genetic transformation of Saccharomyces cerevisiae mitochondria. In Christine, G. and Gerald, R.F. (eds), Methods in Enzymology. Academic Press, pp. 97-111.

Booch, G., Rumbaugh, J. and Jacobson, I. (2005) Unified Modeling Language User Guide. Addison-Wesley Professional, pp. 496.

Boyd, M., McBrien, P. and Tong, N. (2002) The AutoMed Schema Integration Repository. Proceedings of the 19th British National Conference on Databases: Advances in Databases. Springer-Verlag, pp. 42-45.

Brazma, A., et al. (2003) ArrayExpress—a public repository for microarray gene expression data at the EBI, Nucleic Acids Research, 31, 68-71.

Brickley, D. and Guha, R.V. (2004) RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation.

Brickley, D. and Guha, R.V. (2004) RDF Vocabulary Description Language 1.0: RDF Schema. In McBride, B. (ed). W3C Recommendation

Brohee, S., et al. (2010) YTPdb: a wiki database of yeast membrane transporters, Biochim Biophys Acta, 1798, 1908-1912.

Brooke, J. (1996) SUS: A Quick and Dirty Usability Scale. In P. W. Jordan, B.T., B. A. Weerdmeester, & A. L. McClelland. (ed), Usability Evaluation in Industry. Taylor & Francis., London.

Buneman, P., et al. (1995) A Data Transformation System for Biological Data Sources. In Dayal, U., Gray, P.M.D. and Nishio, S. (eds), Proceedings of the 21th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., Zurich, Switzerland, pp. 158-169.

Burger, E., Link, J. and Ritter., O. (1997) A Multi-Agent Architecture for the Integration of Genomic Information. Proceedings of the 1st International Workshop on Intelligent Information Integration (III'97). Fribourg, Allemagne.

Buttler, D., et al. (2002) Querying multiple bioinformatics information sources: can semantic web research help?, SIGMOD Rec., 31, 59-64.

Cadag, E., et al. (2007) Biomediator data integration and inference for functional annotation of anonymous sequences, Pac Symp Biocomput, 343-354.

Calí, A., et al. (2002) On the Expressive Power of Data Integration Systems. Proceedings of the 21st International Conference on Conceptual Modeling. Springer-Verlag, pp. 338-350.

Calí, A., Giacomo, G.D. and Lenzerini, M. (2001) Models for Information Integration: Turning Local-as-View into Global-as-View. In Proceedings of the International Workshop on Foundations of Models for Information Integration (FMII 2001).

Calí, A., et al. (2003) Source integration for data warehousing. In, Multidimensional databases. IGI Publishing, pp. 361-392.

Cantor, C. (1990) Orchestrating the Human Genome Project, Science, 248, 49-51.

Casati, F. and Shan, M.-C. (2001) Models and languages for describing and discovering E-services, SIGMOD Rec., 30, 626.

Chandra, A.K. and Merlin, P.M. (1977) Optimal implementation of conjunctive queries in relational data bases. Proceedings of the ninth annual ACM symposium on Theory of computing. ACM, Boulder, Colorado, United States, pp. 77-90.

Chaudhri, V., et al. (1997) The Generic Frame Protocol 2.0.

Cherry, J.M., et al. (1998) SGD: Saccharomyces Genome Database, Nucleic Acids Research, 26, 73-79.

Cherry, J.M., et al. (2012) Saccharomyces Genome Database: the genomics resource of budding yeast, Nucleic Acids Research, 40, D700-D705.

Chin, J.P., Diehl, V.A. and Norman, K.L. (1988) Development of an instrument measuring user satisfaction of the human-computer interface. In Soloway, E., Frye, D. and Sheppard, S.B. (eds), Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, Washington, D.C., United States, pp. 213-218.

Chinnici, R., et al. (2007) Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation.

Christensen, E., et al. (2001) Web Services Description Language (WSDL) 1.1. W3C Note.

Claypool, K.T. and Madria, S. (2006) A P2P Integration Architecture for Protein Resources. Proceedings of the 17th International Conference on Database and Expert Systems Applications. IEEE Computer Society, pp. 717-721.

Cliften, P., et al. (2003) Finding Functional Features in Saccharomyces Genomes by Phylogenetic Footprinting, Science, 301, 71-76.

Cohen-Boulakia, S. (2005) Intégration de données biologiques : Sélection de sources centrée sur l'utilisateur. Faculté des Sciences d'Orsay. Thèse de doctorat. Université Paris-Sud XI.

Collas, P. (2010) The Current State of Chromatin Immunoprecipitation, Molecular Biotechnology, 45, 87-100.

Connolly, D., et al. (2001) DAML+OIL (March 2001) Reference Description. W3C.

Consortium, T.F. (2002) The FlyBase database of the Drosophila genome projects and community literature, Nucleic Acids Research, 30, 106-108.

Consortium, T.U. (2012) Reorganizing the protein space at the Universal Protein Resource (UniProt), Nucleic Acids Research, 40, D71-D75.

Corcho, Ó. and Gómez-Pérez, A. (2000) Evaluating knowledge representation and reasoning capabilities of ontology specification languages. Proceedings of the ECAI'00 Workshop on Applications of Ontologies and Problem Solving Methods. Berlin, Alemania.

Corpet, F., Gouzy, J. and Kahn, D. (1998) The ProDom database of protein domain families, Nucleic Acids Research, 26, 323-326.

Curwen, V., et al. (2004) The Ensembl Automatic Gene Annotation System, Genome Research, 14, 942-950.

Danflous, D. (2002) Déploiment National des Systèmes d'Information Multimodale : DELFI, l'exemple Allemand. Rapport technique. CERTU (Centre d'études sur les réseaux, les transports, l'urbanisme et les constructions publiques).

Daniel Gietz, R. and Woods, R.A. (2002) Transformation of yeast by lithium acetate/single-stranded carrier DNA/polyethylene glycol method. In Christine, G. and Gerald, R.F. (eds), Methods in Enzymology. Academic Press, pp. 87-96.

Davidson, S.B., et al. (2001) K2/Kleisli and GUS: Experiments in Integrated Access to Genomic Data Sources, IBM System Journal, 40, 512 - 531.

Davidson, S.B., Overton, C. and Buneman, P. (1995) Challenges in integrating biological data sources, J Comput Biol, 2, 557-572.

Davidson, S.B., et al. (1997) BioKleisli: A Digital Library for Biomedical Researchers., Int. J. on Digital Libraries, 1, 36-53.

Davis, A.P., et al. (2011) The curation paradigm and application tool used for manual curation of the scientific literature at the Comparative Toxicogenomics Database, Database, 2011.

Davis, R., Shrobe, H. and Szolovits, P. (1993) What is a knowledge representation? , AI Magazine. Spring, pp. 17 – 33.

De Hertogh, B., et al. (2002) Phylogenetic classification of transporters and other membrane proteins from Saccharomyces cerevisiae, Funct Integr Genomics, 2, 154-170.

Degtyarenko, K., et al. (2008) ChEBI: a database and ontology for chemical entities of biological interest, Nucleic Acids Res, 36, D344-350.

Dhombres, F., et al. (2010) Choix méthodologiques pour la construction d'une ontologie de domaine en médecine prénatale. 21èmes Journées Francophones d'Ingénierie des Connaissances. Nîmes, France

Duschka, O.M. and Genesereth, M.R. (1997) Query planning in infomaster. Proceedings of the 1997 ACM symposium on Applied computing. ACM, San Jose, California, United States, pp. 109-111.

Eckman, B.A., Kosky, A.S. and Jr., L.A.L. (2001) Extending traditional query-based integration approaches for functional characterization of post-genomic data, Bioinformatics, 17, 587-601.

Eppig, J.T., et al. (2012) The Mouse Genome Database (MGD): comprehensive resource for genetics and genomics of the laboratory mouse, Nucleic Acids Research, 40, D881-D886.

Etzold, T., Ulyanov, A. and Argos., P. (1996) SRS : Information Retrievat System for Molecular Biology Databanks, Methods in Enzymology, 266, 114-128.

Fasman, K. (1994) Restructuring the Genome Data Base : A model for a federation of biological databases., Journal of Computational Biology, 1, 165-171.

Fayech, B. (2003) Régulation des réseaux de transport multimodal : Systèmes multi-agents et algorithmes évolutionnistes. Thèse de doctorat. Ecole Centrale de Lille/ Université des Sciences et Technologies de Lille, Lille.

Fensel, D. (2002) Semantic Web Enabled Web Services. Proceedings of the 25th Annual German Conference on AI: Advances in Artificial Intelligence. Springer-Verlag, pp. 319-322.

Fensel, D., Bussler, C. and Maedche, A. (2002) Semantic Web Enabled Web Services. International Semantic Web Conference. Sardinia, Italy, pp. 1–2.

Fensel, D., et al. (2000) OIL in a Nutshell. Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management. Springer-Verlag, pp. 1-16.

Ferber, J. (1995) Les systèmes multi-agents. Vers une intelligence collective. InterEditions, Paris.

Finin, T., et al. (1994) KQML as an agent communication language. Proceedings of the third international conference on Information and knowledge management. ACM, Gaithersburg, Maryland, United States, pp. 456-463.

Flicek, P., et al. (2012) Ensembl 2012, Nucleic Acids Research, 40, D84-D90.

French, S.L., et al. (2011) Distinguishing the roles of Topoisomerases I and II in relief of transcription-induced torsional stress in yeast rRNA genes, Mol Cell Biol, 31, 482-494.

Friedman, M., Levy, A. and Millstein, T. (1999) Navigational plans for data integration. Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence. American Association for Artificial Intelligence, Orlando, Florida, United States, pp. 67-73.

Froidevaux, C. and Cohen-Boulakia, S. (2002) Integration de Sources de Donnees Genomiques du Web. Journees scientifiques du Web Semantique (actes electroniques).

Galas, D.J. and Schmitz, A. (1978) DNAase footprinting a simple method for the detection of protein-DNA binding specificity, Nucleic Acids Research, 5, 3157-3170.

Galperin, M.Y. (2007) The Molecular Biology Database Collection: 2007 update, Nucleic Acids Research, 35, D3-D4.

Galperin, M.Y. and Cochrane, G.R. (2011) The 2011 Nucleic Acids Research Database Issue and the online Molecular Biology Database Collection, Nucleic Acids Research, 39, D1-D6.

Galperin, M.Y. and Fernández-Suárez, X.M. (2012) The 2012 Nucleic Acids Research Database Issue and the online Molecular Biology Database Collection, Nucleic Acids Research, 40, D1-D8.

Gambin, A., Lasota, S. and Rutkowski, M. (2006) Analyzing stationary states of gene regulatory network using petri nets, In silico biology, 6, 93-109.

Garrels, J.I., et al. (1997) Proteome studies of saccharomyces cerevisiae: Identification and characterization of abundant proteins, Electrophoresis, 18, 1347-1360.

Gasch, A.P. (2002) Yeast genomic expression studies using DNA microarrays. In Christine, G. and Gerald, R.F. (eds), Methods in Enzymology. Academic Press, pp. 393-414.

Geen, S., et al. (1997) Software Agents : A review. Rapport technique. Trinity College Dublin.

Geib, J.-M., Gransart, C. and Merle., P. (1999) Corba, des concepts `a la pratique. InterEditions,.

Goasdoué, F. and Rousset, M.-C. (2004) Answering queries using views: A KRDB perspective for the semantic Web, ACM Trans. Internet Technol., 4, 255-288.

Goble, C.A. (2002) Position Statement: Musings on Provenance, Workflow workflow and (semantic web) annotations for bioinformatics. Workshop on Data Derivation and Provenance.

Goffeau, A., et al. (1996) Life with 6000 genes, Science, 274, 546, 563-547.

Griffth, A. (2005) Java, XML, and the JAXP. Wiley.

Gruber, T.R. (1991) The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases. Principles of Knowledge Representation and Reasoning. Proceedings of the Second International Conference. Morgan Kaufmann, Cambridge, MA, pp. 601-602.

Gruber, T.R. (1992) Ontolingua : A mechanism to support portable ontologies. Rapport technique KSL 91-66. Stanford University, Knowledge Systems Laboratory.

Gruber, T.R. (1993) A translation approach to portable ontology specifications, Knowl. Acquis., 5, 199-220.

Gruber, T.R. (1995) Toward principles for the design of ontologies used for knowledge sharing?, International Journal of Human-Computer Studies, 43, 907-928.

Guarino, N. (1999) Formal Ontology and Information Systems. Proceedings of FOIS'98. IOS Press, Trento, Italy, , pp. 3-15.

Gudgin, M., et al. (2007) SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation.

Guérin, E., et al. (2005) Integrating and warehousing liver gene expression data and related biomedical resources in GEDAW. international conference on Data Integration in the Life Sciences. Springer-Verlag, San Diego, CA, pp. 158-174.

Guldener, U., et al. (2005) CYGD: the Comprehensive Yeast Genome Database, Nucleic Acids Res, 33, D364-368.

Guldener, U., et al. (2006) MPact: the MIPS protein interaction resource on yeast, Nucleic Acids Res, 34, D436-441.

Haas, H. and Brown, A. (2004) Web Services Glossary. W3C Working Group Note.

Haas, L.M., et al. (1999) Transforming Heterogeneous Data with Database Middleware: Beyond Integration, IEEE Data Eng. Bull., 22, 31-36.

Haas, L.M., et al. (2001) DiscoveryLink: a system for integrated access to life sciences data sources, IBM Syst. J., 40, 489-511.

Haas, L.M., et al. (2001) DiscoveryLink: A system for integrated access to life sciences data sources, IBM Systems Journal, 40, 489-511.

Halevy, A., Rajaraman, A. and Ordille, J. (2006) Data integration: the teenage years. In Dayal, U., et al. (eds), Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment, Seoul, Korea, pp. 9-16.

Halevy, A.Y. (2001) Answering queries using views: A survey, The VLDB Journal, 10, 270-294.

Hamosh, A., et al. (2002) Online Mendelian Inheritance in Man (OMIM), a knowledgebase of human genes and genetic disorders, Nucleic Acids Research, 30, 52-55.

Harger, C., et al. (2000) The Genome Sequence DataBase, Nucleic Acids Research, 28, 31-32.

Hartmann, J., et al. OMV - Ontology Metadata Vocabulary In Welty, C. (ed), Proceeding of the ISWC 2005 - In Ontology Patterns for the Semantic Web.

Heijst, G.v., Schreiber, A.T. and Wielinga, B.J. (1997) Using explicit ontologies in KBS development, Int. J. Hum.-Comput. Stud., 46, 183-292.

Hermjakob, H., et al. (2004) The HUPO PSI's molecular interaction format--a community standard for the representation of protein interaction data, Nat Biotechnol, 22, 177-183.

Hernandez, T. and Kambhampati, S. (2004) Integration of biological sources: current systems and challenges ahead, SIGMOD Rec., 33, 51-60.

Hitzler, P., et al. (2009) OWL 2 Web Ontology Language Primer. W3C Recommendation.

Hizicky, E.M., et al. (2002) Biochemical genomics approach to map activities to genes activities to gene. In Christine, G. and Gerald, R.F. (eds), Methods in Enzymology. Academic Press, pp. 546-559.

Holm, C. (1982) Clonal lethality caused by the yeast plasmid 2μ DNA, Cell, 29, 585-594.

Hong, E.L., et al. (2008) Gene Ontology annotations at SGD: new data sources and annotation methods, Nucleic Acids Research, 36, D577-D581.

Ito, T., et al. (2000) Toward a protein-protein interaction map of the budding yeast: A comprehensive system to examine two-hybrid interactions in all possible combinations between the yeast proteins, Proceedings of the National Academy of Sciences of the United States of America, 97, 1143-1147.

Ives, Z.G., et al. (2011) Piazza: Mediation and Integration Infrastructure for Semantic Web Data. 2011. Elsevier.

Jamison, D.C., Mills, B. and Schatz., B. (1996) An extensible network query unification system for biological databases. , Computer Applications In the Biosciences, 12, 145-150.

Jarras, I. and Chaib-draa, B. (2002) Aperçu sur les systèmes multiagents. In CIRANO (ed).

Karolchik, D., et al. (2003) The UCSC Genome Browser Database, Nucleic Acids Research, 31, 51-54.

Karp, P.D., et al. (2010) Pathway Tools version 13.0: integrated software for pathway/genome informatics and systems biology, Brief Bioinform, 11, 40-79.

Kataoka, T. and Satou, K. (1999) A Full-Text Search System Covering the

Whole GenomeNet. In Proceedings of the 10th Workshop on Genome Informatics (GIW '99). Universal Academy Press., Tokyo, Japon, pp. 308-309.

Kellis, M., et al. (2003) Sequencing and comparison of yeast species to identify genes and regulatory elements, Nature, 423, 241-254.

Kifer, M., Lausen, G. and Wu, J. (1995) Logical foundations of object-oriented and frame-based languages, J. ACM, 42, 741-843.

Klug, A. (1988) On conjunctive queries containing inequalities, J. ACM, 35, 146-160.

Kolaitis, P.G., Martin, D.L. and Thakur, M.N. (1998) On the complexity of the containment problem for conjunctive queries with built-in predicates. Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. ACM, Seattle, Washington, United States, pp. 197-204.

Kumar, A. and Snyder, M. (2001) Emerging technologies in yeast genomics, Nat Rev Genet, 2, 302-312.

Lacroix, Z., et al. (2004) Links and Paths through Life Sciences Data Sources. In Rahm, E. (ed), Data Integration in the Life Sciences. Springer Berlin Heidelberg, pp. 203-211.

Lapp, H. (2006) BioSQL.

Lassila, O. and Swick, R.R. (1999) Resource description framework (rdf) model and syntax specification. w3c recommendation.

Lee, T.J., et al. (2006) BioWarehouse: a bioinformatics database warehouse toolkit, BMC Bioinformatics, 7, 170.

Lemer, C., et al. (2004) The aMAZE LightBench: a web interface to a relational database of cellular processes, Nucleic Acids Res, 32, D443-448.

Lenzerini, M. (2002) Data integration: a theoretical perspective. Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, Madison, Wisconsin, pp. 233-246.

Letovsky, S.I., et al. (1998) GDB: The Human Genome Database, Nucleic Acids Research, 26, 94-99.

Levy, A. (1999) Combining Artificial Intelligence and Databases for Data Integration. In Wooldridge, M. and Veloso, M. (eds), Artificial Intelligence Today. Springer Berlin / Heidelberg, pp. 249-268.

Levy, A.Y. (2000) Logic-based techniques in data integration. In, Logic-based artificial intelligence. Kluwer Academic Publishers, pp. 575-595.

Levy, A.Y., et al. (1995) Answering Queries Using Views.

Levy, A.Y., Rajaraman, A. and Ordille, J.J. (1996) Querying Heterogeneous Information Sources Using Source Descriptions. Proceedings of the 22th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., pp. 251-262.

Levy, A.Y. and Sagiv, Y. (1993) Queries Independent of Updates. Proceedings of the 19th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., pp. 171-181.

Lewis, J.R. (1995) IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use, Int. J. Hum.-Comput. Interact., 7, 57-78.

MacGregor, R.M. (1991) Inside the LOOM description classifier, SIGART Bull., 2, 88-92.

Marc, P., Devaux, F. and Jacq, C. (2001) yMGV: a database for visualization and data mining of published genome-wide yeast expression data, Nucleic Acids Research, 29, e63.

Martzen, M.R., et al. (1999) A Biochemical Genomics Approach for Identifying Genes by the Activity of Their Products, Science, 286, 1153-1155.

Matys, V., et al. (2003) TRANSFAC: transcriptional regulation, from patterns to profiles, Nucleic Acids Res, 31, 374-378.

McBride, B. (2004) The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS. In Staab, S. and Studer, R. (eds), Handbook on Ontologies. Springer, pp. 51-66.

McBrien, P. and et Poulovassilis, A. (2003) Data Integration by Bi-Directional Schema Transformation Rules. In Proceedings. 19th International Conference on Data Engineering. pp. 227-238.

McGuiness, D.L., et al. (2000) An Environment for Merging and Testing Large Ontologies. In A. G. Cohn, F.G.a.B.S. (ed), Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'2000). Morgan Kaufmann, San Francisco, CA, pp. 483-493.

McGuinness, D.L., et al. (2002) DAML+OIL: an ontology language for the Semantic Web, Intelligent Systems, IEEE, 17, 72-80.

McGuinness, D.L. and Harmelen, F.v. (2004) OWL Web Ontology Language Overview. W3C Recommendation.

McGuinness, D.L. and Wright, J.R. (1998) Conceptual modelling for configuration: A description logic-based approach, Artif. Intell. Eng. Des. Anal. Manuf., 12, 333-344.

McIlraith, S.A., Son, T.C. and Zeng, H. (2001) Semantic Web Services, IEEE Intelligent Systems, 16, 46-53.

McLaughlin, B. (2002) Java & XML Data Binding. O'Reilly & Associates, Inc.

Médigue, C., et al. (1999) Imagene: an integrated computer environment for sequence annotation and analysis, Bioinformatics, 15, 2-15.

Mewes, H.W., et al. (1997) Overview of the yeast genome, Nature, 387, 7-65.

Mewes, H.W., et al. (2011) MIPS: curated databases and comprehensive secondary data resources in 2010, Nucleic Acids Research, 39, D220-D224.

Michael Rebhan, V.C.-C., Jaime Prilusky, et Doron Lancet (1998) GeneCards : a novel functional genomics compendium with automated data mining and query reformulation support, Bioinformatics, 14, 656-664.

Milo, T., et al. (2005) Exchanging intensional XML data, ACM Trans. Database Syst., 30, 1-40.

Mork, P., Halevy, A. and Tarczy-Hornoch, P. (2001) A model for data integration systems of biomedical data applied to online genetic databases. Proceedings of the Symposium of the American Medical Informatics Association. pp. 473-477.

Motta, E. (1999) Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving. IOS Press.

Muller, H.M., Kenny, E.E. and Sternberg, P.W. (2004) Textpresso: an ontology-based information retrieval and extraction system for biological literature, PLoS Biol, 2, e309.

Mungall, C.J. and Emmert, D.B. (2007) A Chado case study: an ontology-based modular schema for representing genome-associated biological information, Bioinformatics, 23, i337-346.

Musen, M.A. (1992) Dimensions of knowledge sharing and reuse, Comput. Biomed. Res., 25, 435-467.

Nasmyth, K. and Dirick, L. (1991) The role of SWI4 and SWI6 in the activity of G1 cyclins in yeast, Cell, 66, 995-1013.

Navas-Delgado, I. and Aldana-Montes, J.F. (2009) Extending SD-Core for Ontology-based Data Integration, Journal of Universal Computer Science, 15, 3201-3230.

Nejdl, W., et al. (2002) EDUTELLA: a P2P networking infrastructure based on RDF. Proceedings of the 11th international conference on World Wide Web. ACM, Honolulu, Hawaii, USA, pp. 604-615.

Noy, N. and Deborah (2001) Ontology Development 101: A Guide to Creating Your First Ontology.

O'Donovan, C., et al. (2002) High-quality protein knowledge resource : SWISS-PROT and TrEMBL, Briefings in Bioinformatics, 3, 275–284.

Okuda, S., et al. (2008) KEGG Atlas mapping for global analysis of metabolic pathways, Nucleic Acids Research, 36, W423-W426.

Oliver, S.G. (1996) From DNA sequence to biological function, Nature, 379, 597-600.

Olken, F. and Jagadish., H.V. (2003) Data Management for Integrative Biology., OMICS, 7, 1-2.

Overbeek, R., Disz, T. and Stevens, R. (2004) The SEED: a peer-to-peer environment for genome annotation, Commun. ACM, 47, 46-51.

Ploger, R., et al. (2000) XREFdb: Cross-referencing the genetics and genes of mammals and model organisms, Nucleic Acids Research, 28, 120-122.

Pontius, J., Wagner, L. and Schuler, G. (2003) UniGene: a unified view of the transcriptome. In, The NCBI Handbook. Bethesda (MD): National Center for Biotechnology Information.

Pottinger, R. and Halevy, A. (2001) MiniCon: A scalable algorithm for answering queries using views, The VLDB Journal, 10, 182-198.

Pruitt, K.D., et al. (2009) The consensus coding sequence (CCDS) project: Identifying a common protein-coding gene set for the human and mouse genomes, Genome Research, 19, 1316-1323.

Pruitt, K.D., et al. (2012) NCBI Reference Sequences (RefSeq): current status, new features and genome annotation policy, Nucleic Acids Research, 40, D130-D135.

Pu, S., et al. (2009) Up-to-date catalogues of yeast protein complexes, Nucleic Acids Res, 37, 825-831.

Rebhan, M., et al. (1997) GeneCards: integrating information about genes, proteins and diseases, Trends in Genetics, 13, 163.

Rechenmann, F. (2002) GenoStar: A Bioinformatics Platform for Exploratory Genomics. ERCIM News.

Rector, A.L., et al. (1997) The GRAIL concept modelling language for medical terminology, Artif Intell Med, 9, 139-171.

Rector, A.L., et al. (1993) A framework for modelling the electronic medical record, Methods of Information in Medicine 32, 109-119.

Rifaieh, R., et al. (2007) SWAMI: integrating biological databases and analysis tools within user friendly environment. In Cohen-Boulakia, S. and Tannen, V. (eds), Proceedings of the 4th international conference on Data integration in the life sciences. Springer-Verlag, Philadelphia, PA, USA, pp. 48-58.

Ross-Macdonald, P., et al. (1999) Transposon mutagenesis for the analysis of protein production, function, and localization. pp. 512.

Rosse, C., et al. (1995) Enhancements of Anatomical Information in UMLS Knowledge Sources. Proceedings of19th Annual Symposium on Computer Applications in Medical Care. pp. 873-877.

Rousset, M.-C., et al. (2006) SomeWhere in the semantic web. Proceedings of the 32nd conference on Current Trends in Theory and Practice of Computer Science. Springer-Verlag, Měřín, Czech Republic, pp. 84-99.

Rousset, M.-C. and Reynaud, C. (2004) Knowledge representation for information integration, Information Systems, 29, 3-22.

Roze, C.P. (2003) Organisation multi-agents au service de la personnalisation de l'information. Thèse de doctorat. Université de valenciennes et du Hainaut-Cambrésis.

Ruepp, A., et al. (2004) The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes, Nucleic Acids Res, 32, 5539-5545.

Sagiv, Y. and Yannakakis, M. (1980) Equivalences Among Relational Expressions with the Union and Difference Operators, J. ACM, 27, 633-655.

Sayers, E.W., et al. (2011) Database resources of the National Center for Biotechnology Information, Nucleic Acids Research, 39, D38-D51.

Schulze-kremer, S. (1998) Ontologies For Molecular Biology. In Proceedings of the Third Pacific Symposium on Biocomputing. AAAI Press, pp. 693-704.

Shah, S.P., et al. (2005) Atlas - a data warehouse for integrative bioinformatics, BMC Bioinformatics, 6, 34.

Shakshuki, E., Ghenniwa, H. and Kamel, M. (2003) An architecture for cooperative information systems, Knowledge-Based Systems, 16, 17-27.

Sherlock, G., et al. (2001) The Stanford Microarray Database, Nucleic Acids Research, 29, 152-155.

Sherman, F. (2002) Getting started with yeast. In Christine, G. and Gerald, R.F. (eds), Methods in Enzymology. Academic Press, pp. 3-41.

Sheth, A.P. and Kashyap, V. (1992) So far (schematically) yet so near (semantically). In DS-5. pp. 283-312.

Sheth, A.P. and Larson, J.A. (1990) Federated database systems for managing distributed, heterogeneous, and autonomous databases, ACM Comput. Surv., 22, 183-236.

Shi, G. (2002) Data Integration using Agent based Mediator-Wrapper Architecture, tutorial report for Agent Based Software Engeneering. Rapport technique. University of calgary.

Siepel, A.C., et al. (2001) An integration platform for heterogeneous bioinformatics software components, IBM Syst. J., 40, 570-591.

Stal, M. (2002) Web services: beyond component-based computing, Commun. ACM, 45, 71-76.

Stark, C., et al. (2005) BioGRID: a general repository for interaction datasets, Nucleic Acids Research, 34, D535-D539.

Stark, C., et al. (2011) The BioGRID Interaction Database: 2011 update, Nucleic Acids Res, 39, D698-704.

Stark, C., et al. (2010) PhosphoGRID: a database of experimentally verified in vivo protein phosphorylation sites from the budding yeast Saccharomyces cerevisiae, Database (Oxford), 2010, bap026.

Stevens, R., et al. (2000) TAMBIS: transparent access to multiple bioinformatics information sources, Bioinformatics, 16, 184-185.

Su, X. and Ilebrekke, L. (2002) A Comparative Study of Ontology Languages and Tools. Proceedings of the 14th International Conference on Advanced Information Systems Engineering. Springer-Verlag, pp. 761-765.

Sweeney, R. and Zakian, V.A. (1989) Extrachromosomal elements cause a reduced division potential in nib 1 strains of Saccharomyces cerevisiae, Genetics, 122, 749-757.

Sycara, K. and Zeng, D. (1996) Multi-Agent Integration of Information Gathering and Decision Support. Proceedings of the 12th European Conference on Artificial Intelligence. John Wiley & Sons, Ltd.

Teixeira, M.C., et al. The YEASTRACT database: a tool for the analysis of transcription regulatory associations in Saccharomyces cerevisiae, Nucleic Acids Research, 34, D446-D451.

Tong, A.H.Y., et al. (2001) Systematic Genetic Analysis with Ordered Arrays of Yeast Deletion Mutants, Science, 294, 2364-2368.

Tullis, T. and Stetson, J.N. (2004) A comparison of questionnaires for assessing website usability. Proceedings of the Usability Professionals Association (UPA). Minneapolis, Minnesota

Uetz, P., et al. (2000) A comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae, Nature, 403, 623-627.

Ullman, J.D. (2000) Information integration using logical views, Theor. Comput. Sci., 239, 189-210.

Ullman., J.D. (1989) Principles of Database and Knowledge-base Systems. Computer Science Press, Rockville MD,.

Uschold, M. and Grüninger, M. (1996) Ontologies: principles, methods, and applications, Knowledge Engineering Review, 11, 93-155.

Velculescu, V.E., et al. (1997) Characterization of the Yeast Transcriptome, Cell, 88, 243-251.

Wang, H., Li, J. and He, Z. (2003) An Effective Wrapper Architecture to Heterogeneous Data Source. Proceedings of the 17th International Conference on Advanced Information Networking and Applications. IEEE Computer Society, pp. 565.

Welty, C. and Ide, N. (1999) Using the Right Tools: Enhancing Retrieval from Marked-up Documents, Computers and the Humanities, 33, 59-84.

Wiederhold, G. (1998) Mediators in the architecture of future information systems. In Michael, N.H. and Munindar, P.S. (eds), Readings in agents. Morgan Kaufmann Publishers Inc., pp. 185-196.

Wilde, E. and Lowe, D. (2002) XPath, XLink, XPointer, and XML : A Practical Guide to Web Hyperlinking and Transclusion. Pearson Education.

Yang, B. and Garcia-Molina, H. (2002) Improving Search in Peer-to-Peer Networks. Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02). IEEE Computer Society, pp. 5.

Zamboulis, L., Martin, N. and Poulovassilis, A. (2007) Bioinformatics service reconciliation by heterogeneous schema transformation. In Cohen-Boulakia, S. and Tannen, V. (eds), Proceedings of the 4th international conference on Data integration in the life sciences. Springer-Verlag, Philadelphia, PA, USA, pp. 89-104.

Zhang, X. and Ozsoyoglu, Z. (1993) On efficient reasoning with implication constraints

Deductive and Object-Oriented Databases. In Ceri, S., Tanaka, K. and Tsur, S. (eds). Springer Berlin / Heidelberg, pp. 236-252.