Rapport de TER

Encadrants :
Paola SALLE
Sandra BRINGAY

Etudiants :
Charles-Edouard COSTE
Karima ZAYRIT
Wei XING
Xin HU

Version 1.0, *26/04/2008*

Table des matières

Remerciements

L'équipe tient à remercier comme il se doit ses encadrantes :

- Sandra BRINGAY, maître de conférence à l'Université Montpellier III
- et Paola SALLE doctorante dans l'équipe TATOO

pour leur soutien et leur disponibilité ainsi que pour leurs précieux conseils (et leur patience).

Introduction

La recherche biomédicale s'appuie en grande partie sur l'extraction de connaissances. A partir de données récupérées de manière expérimentale, il est nécessaire de pouvoir interpréter les résultats.

L'équipe TATOO basée au LIRMM est spécialisée dans ce domaine. Une étape du processus d'extraction de connaissances est la fouille de données. Or, certains problèmes peuvent apparaître lors ce traitement :

- Les données proviennent parfois de différentes sources et peuvent donc être peu ou pas cohérentes.
- Les données peuvent être très volumineuses en terme de quantité.
- Les données peuvent aussi s'avérer incomplètes de temps à autres.

Ces facteurs peuvent rendre difficile l'utilisation des algorithmes classiques de fouille. C'est pourquoi l'équipe TATOO souhaitait intégrer les données de différentes sources spécialisées dans le domaine biomédical.

Les récentes avancées en biologie moléculaire et en génomique ainsi que les progrès technologiques au niveau de l'équipement ont permis un flux très important des connaissances dans ce domaine.

En conséquence, les bases de données biomédicales ont connu une croissance exponentielle ces dernières années.

Ces banques de données sont devenues des éléments centraux de la recherche dans ce domaine et il est donc crucial pour les biologistes de disposer d'outils informatiques adéquats leur permettant d'analyser ces données.

Voilà donc l'objectif du projet KEFT (Knowledge Extractor For TATOO) que notre groupe a initié dans le cadre de notre master informatique « professionnel et recherche unifié » à l'université Montpellier 2, à l'occasion d'un travail d'étude et de recherche de 3 mois :

Fournir aux experts un logiciel capable d'interroger différentes bases et de récupérer les données sous une forme exploitable.

A. Matériels et méthodes

Cette partie du rapport présente l'essentiel de notre démarche. Nous y aborderons un peu plus en détail les spécifications du logiciel, et les choix techniques et stratégiques que nous avons effectués afin de répondre aux besoins de la maîtrise d'ouvrage.

A.1. Objectifs

1. Besoins

Comme vu précédemment, le but du projet est d'intégrer différentes données biomédicales. Mais pas n'importe lesquelles et pas n'importe comment !

Selon le cahier des charges fournis par la maîtrise d'ouvrage, le produit devra être capable de récolter des informations concernant des gènes, des maladies, et des pathways. Il devra aussi être capable d'établir les relations entre ces entités.

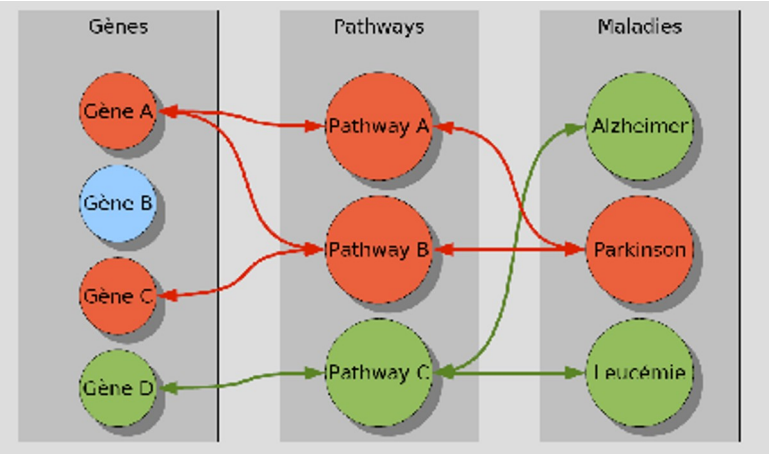


illustration 1 : Relations gènes/pathway/maladie

Gènes

Un gène est un "morceau" d'ADN contenu dans le noyau de nos cellules, et qui porte le plan de fabrication d'une protéine. Les gènes sont porteurs des informations relatives aux caractéristiques d'un individu (couleur des yeux par exemple).

L'homme possède environ 30 000 gènes, ce qui ne représente que 5% de tout son ADN. Certaines espèces animales et végétales ont plus de gènes que l'homme.

Pathways

Le métabolisme est un ensemble de transformation, de processus biologiques, qui se manifestent dans tous les tissus de l'organisme vivant. Le métabolisme est constitué d'un réseau de réactions biochimiques. Lorsqu'un ensemble déterminé de réaction de ce réseau se produit à partir d'un élément, appelé précurseur, jusqu'à un ou des produits finaux, cet ensemble de réactions est appelé voie métabolique. Chacune de ces réactions est catalysée par une enzyme, (protéine très spécialisée qui intervient dans la cinétique chimique de la rection c a d la vitesse à laquelle elle se produit.

Voici la représentation graphique d'un pathway :

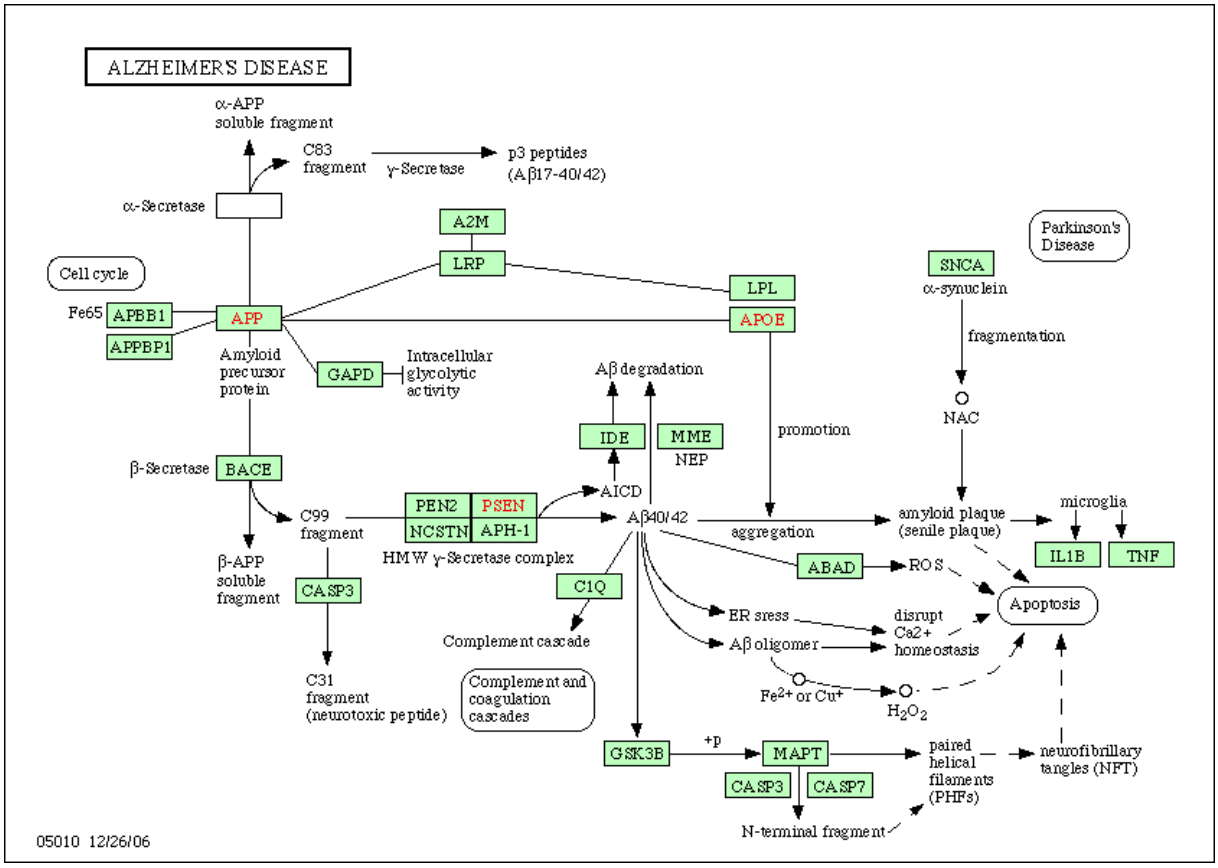


illustration 2 : Alzheimer's disease, source : http://www.genome.jp/kegg/

2. Spécifications

Une fois la capture des besoins réalisée nous avons pu établir les spécifications de notre futur logiciel.

Par rapport aux besoins exprimés, il a été défini que notre produit devrait pouvoir interroger différentes bases de connaissances. L'engouement pour les services web nous a poussé à choisir ce moyen d'interrogation dans la mesure du possible.

Une grande partie du travail aura d'ailleurs été la recherche de services disponibles via le protocole SOAP.

Ces nouveaux protocoles ont l'avantage d'être basés sur des requêtes http standard. Cela signifie que peu importe où l'on soit, si on peut accéder à des sites internet, alors on peu faire communiquer un logiciel avec un service web.

Ce qui n'est pas le cas lorsque l'on interroge une base de données directement car si un pare-feu est en place, il se peut qu'il empêche ce type de communication.

A.2. Services web

Un aspect important des nouvelles technologies du web est l'apparition des services web. Ces derniers sont en quelque sorte des sites web pour programmes. Les données sont directement envoyées d'un site web à un autre ou à un logiciel tournant sur votre ordinateur.

Cependant, il arrive souvent qu'un service web ait son équivalent en html (format standard des pages web) pour qu'un humain puisse aussi s'en servir.

Pour utiliser un service web, il faut connaître son adresse URL exactement comme pour un site web. Par contre il nous faut aussi la liste des fonctions que l'on peut appeler. Un service de gestion de photos pourrait proposer par exemple :

- addPhoto
- getPhoto
- deletePhoto

Pour ne pas avoir à vérifier constamment que les fonctions soient toujours disponibles et pour ne pas avoir à mettre à jour l'URL du service s'il change souvent, certains services web proposent un fichier WSDL qui contient la description complète du service. Une application pourra donc vérifier si elle est toujours en mesure d'interroger le service.

Voici quelques ressources biomédicales connues, disponibles en ligne et interrogeables par des services web :

1. KEGG

De son vrai nom "Kyoto Encyclopedia of Genes and Genomes", KEGG[1] est un projet japonais de grande envergure dont l'objectif ambitieux est d'intégrer toute la connaissance mondiale en génétique.

KEGG est composée d'une vingtaine de bases de données mais celles qui nous intéressent sont les suivantes :

- KEGG PATHWAY
- KEGG DISEASES
- KEGG GENES

Ce service web nous permettra de connaître les relations entre chaque entité et utilise le protocole SOAP.

Côté technique, KEGG fournit une API sous forme de classes JAVA ou d'un fichier WSDL. Les premières sont directement utilisables dans un programme écrit en JAVA alors que le deuxième contient les spécifications du service qui permet de programmer ses propres classes dans le langage que l'on souhaite.

1 - http://www.genome.jp/kegg

L'inconvénient de KEGG est que son API n'est pas très normalisée. On pourra par exemple obtenir les noms des pathways si on lui demande la liste de ces derniers, par contre on sera dans l'incapacité totale d'obtenir le nom d'un pathway à l'aide de son identifiant. Les données ne sont pas toujours atomiques. On se retrouvera souvent donc à devoir segmenter des chaînes de caractères pour avoir l'information qui nous intéresse ou à combiner deux résultats de fonctions. Ce qui rend donc notre application difficile à maintenir car si la mise en forme de KEGG change, alors il faut changer une partie de notre programme. Une raison supplémentaire donc de bien définir notre architecture.

KEGG est donc une ressource indispensable à notre projet mais l'utilisation de ce seul service ne permettrait pas de répondre complètement aux besoins. En effet KEGG, comme la plupart des bases de connaissance a adopté sa propre nomenclature et il arrive donc que l'utilisateur ne sache pas réellement comment s'appelle ce qu'il cherche. Pour cela, il nous faut compléter les informations recueillies.

2. Gene ontology

Gene Ontology[2] (GO) est considéré par beaucoup comme une espèce de dictionnaire dans lequel on peut retrouver une grande quantité de termes biomédicaux.

Nous avons vu que KEGG a adopté une nomenclature bien spécifique. GO, quant à lui tente de lister un maximum de termes afin de connaître ce à quoi ils se réfèrent.

C'est un bon complément à KEGG dans le sens il permet par rapport à un terme donné, de savoir à quoi il se réfère.

le site de Gene Ontology ne présente pas de moyen d'accéder aux données par les services web. Cependant après de longues recherches, nous avons trouvé le site WABI[3] qui fournit un accès par le protocole SOAP aux ressources de GO.

2 - http ://www.geneontology.org

3 - http://xml.nig.ac.jp/index.html

3. MESH

MeSH est l'acronyme de "Medical Subject Headings". Le MeSH, produit par la NLM (U.S. National Library of Medicine), est une liste normalisée de termes utilisée pour l'analyse documentaire dans le domaine biomédical. Le vocabulaire MeSH est un outil d'indexation d'articles pour l'Index Medicus® et MEDLINE et de catalogage pour les ouvrages et les documents audiovisuels. Le vocabulaire contrôlé de MeSH est un trait distinctif de MEDLINE. L'indexation de la documentation biomédicale, est effectuée de manière homogène et cohérente. Les descripteurs MeSH sont organisés de façon hiérarchique dans le MeSH Tree Structures (descripteurs en structures arborescentes) mis à jour chaque année.

Malheureusement, faute de temps nous n'avons pas été en mesure d'intégrer ce service dans notre application.

A.3. Méthodologie

Tout projet se doit de démarrer par le choix de la méthode de travail qui conditionnera toute la démarche.

1. Cycle de vie

Un projet peut suivre différents types de cycles de vie. Le plus connu est sans doute le cycle en "V" cependant nous avons opté pour une autre solution.

Il suffit de chercher un peu sur internet pour se rendre compte que chacun a un peu sa propre conception de chaque type de cycle de vie. Cependant quelques livres récents comme UML pour les décideurs *(doc. 2)* considère que le cycle en "V" impose un *"effet tunnel"*. En effet, d'après l'auteur, ce type de cycle est supposé ne faire intervenir la maîtrise d'ouvrage qu'au début et à la fin du projet. Ce qui laisse le loisir à la maîtrise d'œuvre de produire un logiciel à l'opposée des attentes du client.

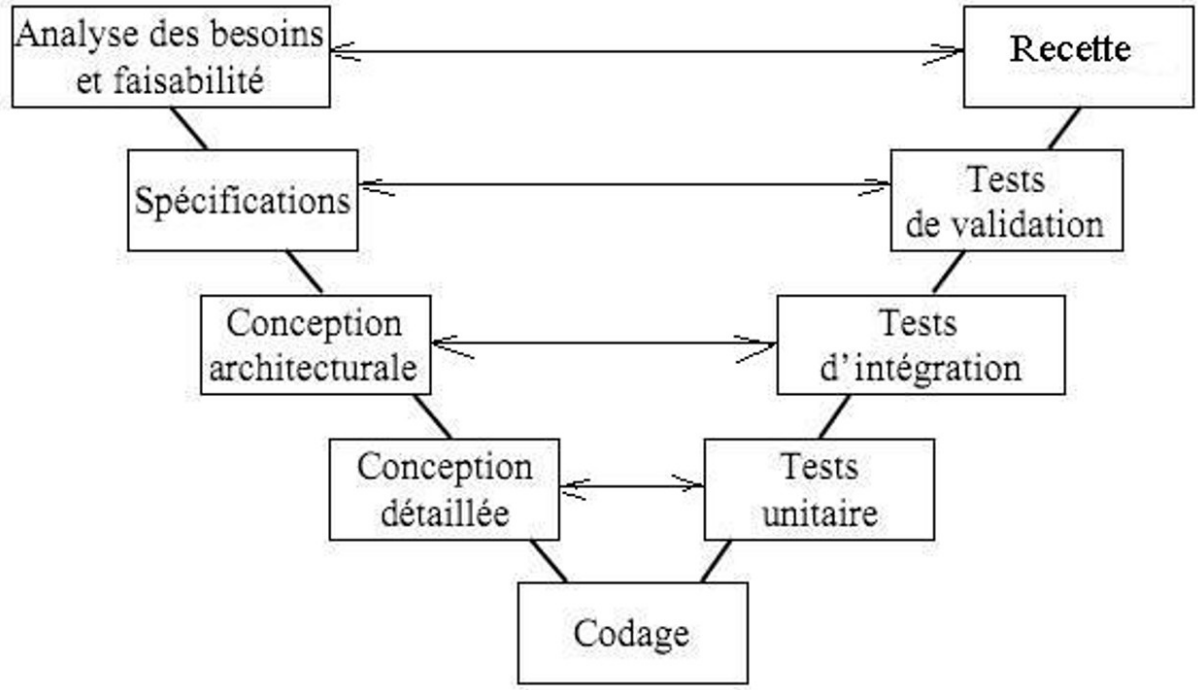


illustration 3 : Le cycle de vie en "V", source : http://wikipedia.org

La littérature à ce sujet suggère donc de n'utiliser ce type de méthode que dans les cas où il est considéré que de toute manière la maîtrise d'ouvrage n'interviendra qu'au niveau des spécifications et au niveau des recettes. Elle suggère aussi, dans la mesure où il peut y avoir une communication régulière entre les deux parties, d'utiliser les méthodes dites "agiles".

Le but de ces méthodes est de s'assurer tout au long du processus de développement que le résultat final correspondra bien aux attentes du client et non pas aux spécifications qui ont la fâcheuses tendance d'être incomplètes (voir "La théorie de la réationnalité limitée" et "les théories contractuelles des organisations").

Pour ce faire, la plupart des méthodes agiles sont fondées sur un cycle de vie itératif. A chaque fin de phase, le produit doit être opérationnel et à chacune d'elles doit correspondre l'ajout de nouvelles fonctionnalités. Cela nécessite généralement un plus grand travail de programmation car il faut souvent faire du refactoring, cependant on est certain de fournir autre chose qu'une analyse ou un logiciel à moitié fini.

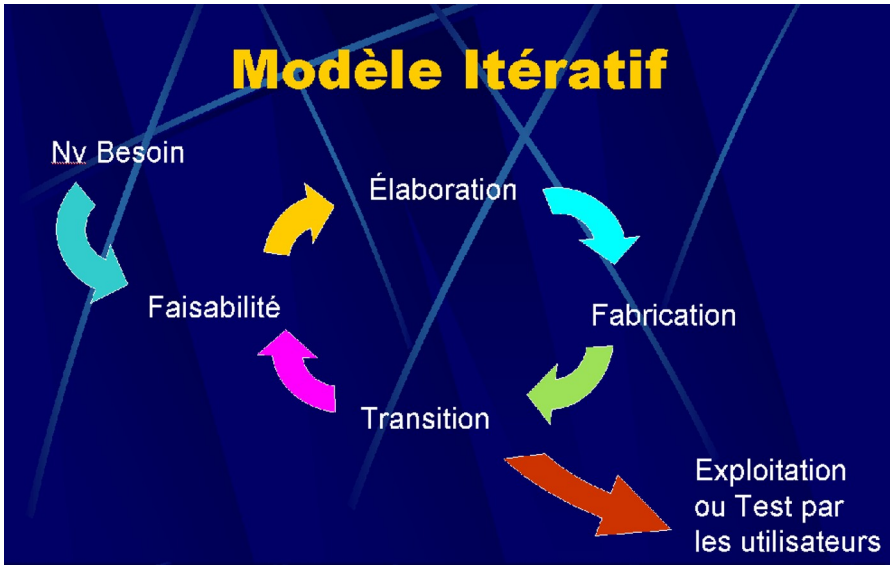


illustration 4 : Cycle de vie itératif, source : http://wikipedia.org

2. Méthode de développement

Le cycle de vie défini, il faut définir une méthodologie de travail pour organiser ce dernier au sein de l'équipe. La plus connue est sans nul doute le pilotage par les cas d'utilisation.

Il existe un nombre conséquent de méthodes de développement. Nous avons vu que nous nous orientons vers des méthodes "agiles". Nous avons identifié deux méthodes connues : l'Extrem Programming (XP) et le Rational Unified Process (RUP).

Nous n'entrerons pas dans le détail de ces méthodes qui mériteraient un rapport à elles seules. Cependant la grande différence entre ces deux méthodes est le niveau de formalisme nécessaire ou possible.

En effet la méthode XP (dont nous avons eu un aperçu dans UML pour les décideurs *(doc. 2)* ainsi que Swing *(doc. 1)* dans la collection des "Cahiers du programmeur) n'est pas très formaliste. Elle n'oblige pas à produire de document d'analyse et se base beaucoup sur la "virtuosité" des développeurs. Comme nous voulions une méthode permettant de structurer notre travail, nous avons préférer opter pour la méthode RUP.

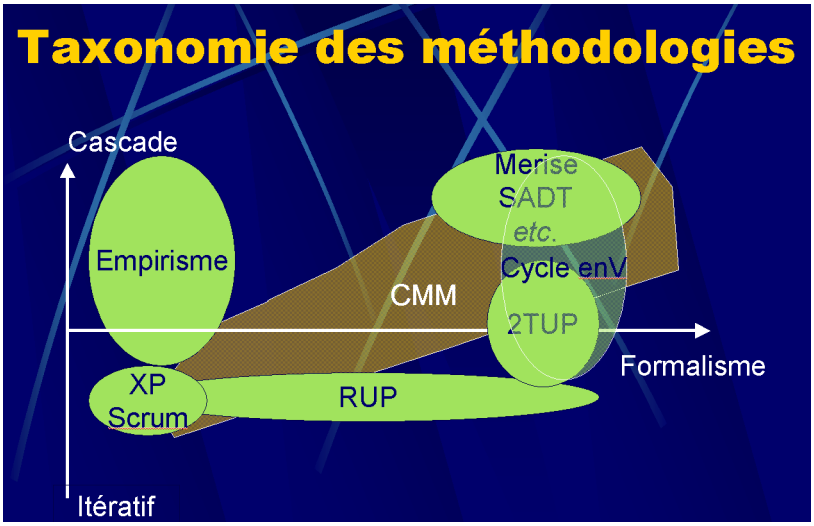


illustration 5 : Taxonomie des méthodologies, source : http://wikipedia.org

Dans le modèle RUP, les différentes tâches "Analyse", "Modélisation","Développement", "Test" sont considérées comme des disciplines s'exécutant simultanément au cours du projet mais pas avec la même intensité. Ainsi, par exemple, nous considérons que la modélisation intervient tout au long du projet mais représente une charge de travail plus importante au début du projet.

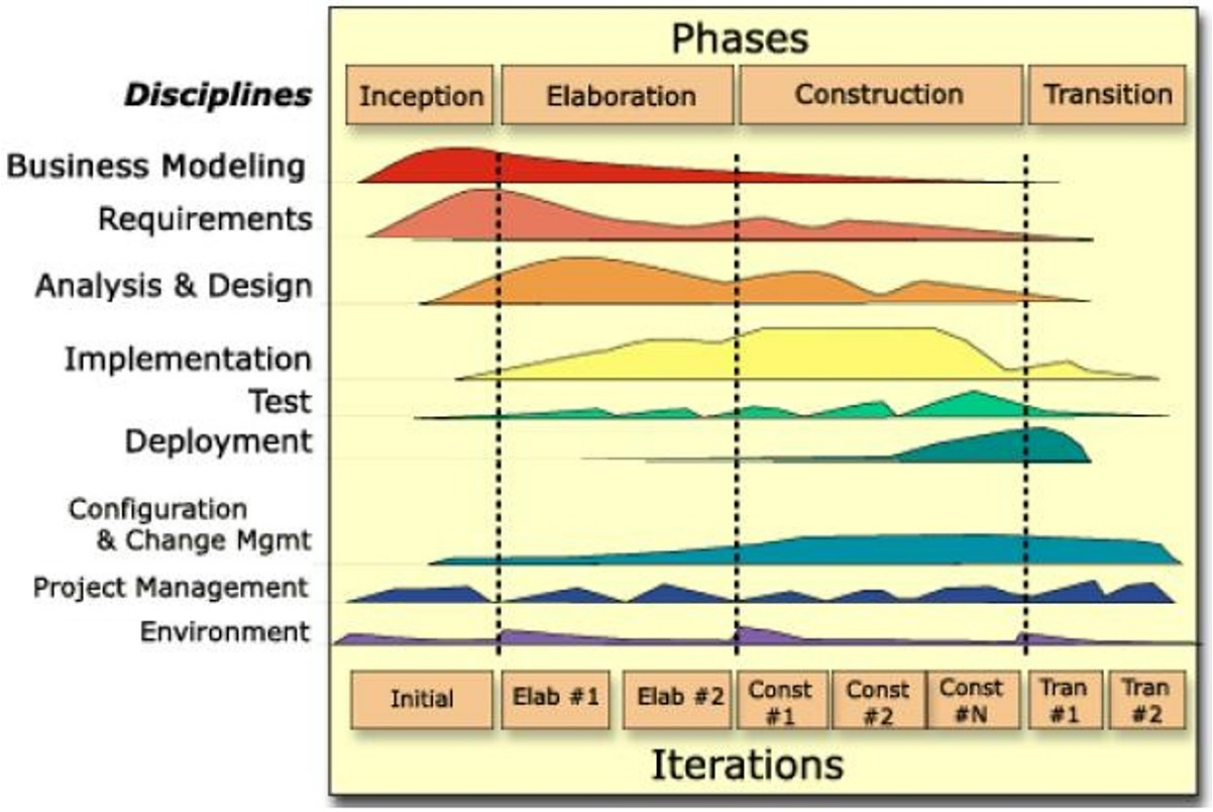


illustration 6 : Exemple de découpage en disciplines, source : http://dn.codegear.com

Cette vision est plus représentative de la réalité et convient parfaitement au contexte car il semblait évident étant donné la complexité du sujet que nous aurions des corrections à apporter à la modélisation au fur et à mesure que nos connaissances s'améliorent, tout au long du projet.

C'est pourquoi nous avons choisi ce modèle de méthode.

A.4. Outils logiciels

Notre projet reposant en grande partie sur la communication, il nous fallait des outils permettant d'améliorer celle-ci.

Google Code Project Hosting est un service gratuit d'hébergement de projets Open Source sur la plateforme Google Code. Il a permis de gérer les différentes versions du logiciel grâce au système Subversion (SVN). Afin de faciliter la mise en commun du travail de chacun et de bénéficier du suivi de version sur chaque fichier, nous avons décidé d'utiliser un serveur SVN. Google code a semblé être le plus adapté à nos besoins, c'est pourquoi dorénavant, chaque membre de l'équipe mettra son travail en ligne sur http://keft.googlecode.com qui correspond à notre site projet. La communication entre maîtrise d'oeuvre et maîtrise d'ouvrage se fait, quant à elle, via le groupe google « ProjetTer » créé par Sandra Bringay dans cette optique.

1. Serveur de fichier

Lorsque l'on souhaite travailler en groupe sur du code il devient important de gérer les versions des fichiers. C'est le rôle d'un serveur de fichier.

Différents sites web proposent gratuitement d'héberger des projets informatiques. Les plus en vogue sont SourceForge.net[4] et GoogleCode.com[5]. Le second propose moins de fonctionnalités mais se révèle plus simple à utiliser. Pour ne pas se perdre dans une jungle de fonctions dont nous n'avons pas besoin, nous avons préféré utiliser GoogleCode.

4 - http://sourceforge.net

5 - http://googlecode.com

Evidemment, si le service est gratuit il y a une contrepartie : nous devions attribuer une licence libre à notre projet. Nous nous sommes entendus et cela n'a pas semblé poser de problème d'autant plus que notre intérêt est qu'un maximum de personnes puisse utiliser le fruit de notre travail.

Pour travailler avec ce serveur, il nous a alors suffit d'installer un plug-in Subversion pour Eclipse dont nous allons parler ci-après.

2. Plate-forme de développement

Il y a généralement trois niveau en matière d'outils de développement. L'éditeur de texte simple (exemple : bloc note de windows), les IDE qui sont en quelque sorte des éditeurs de texte évolués et les plate-formes de développement qui sont des systèmes aux fonctionnalités multiples.

La plate-forme de développement couramment utilisée pour JAVA est Eclipse. C'est un environnement complet disposant de nombreux outils pour faciliter le développement, ainsi que les tests et le déploiement. En utilisant cet environnement muni d'un plug-in pour Subversion, il nous a été possible de travailler à plusieurs sur le code même lorsque nous n'étions pas ensemble.

3. Wiki

L'innovation de ces dernières années en matière de knowledge management reste le wiki. Ce système qui permet la mise en commun des connaissances par mise à jour successive et qui a montré son efficacité.

Etant donné la méthode de développement choisie, la rédaction du rapport pouvait se faire en parallèle des autres tâches. Afin de mettre en commun nos recherches et de conserver régulièrement une trace de ce que nous faisions, le wiki fournit par GoogleCode en même temps que le serveur de fichier s'avérait être l'outil idéal. Il nous suffirait alors pour la rédaction du rapport, de récupérer tout ce qui aura été écrit.

4. UML

Nous avons vu que la communication prenait une part importante dans notre projet. Notre équipe présentant plusieurs nationalités les erreurs de d'interprétation risquaient d'être fréquentes.

Comme son nom l'indique UML (Unified Modeling Language) et un langage. Un langage connu de tous les membres du groupe. De ce fait son utilisation a permis de mieux clarifier les choses et dissiper les malentendus.

Cependant, les outils libres d'utilisation ont un niveau de qualité encore limité dans ce domaine et nous n'avons pas pu utiliser ce langage autant que nous l'aurions souhaité.

B. Résultats

B.1. Interrogation

Le minimum syndicale pour notre application est bien évidemment qu'il soit capable d'interroger les bases de données qui nous intéressent. Nous allons voir que c'est bien le cas.

L'application se connecte parfaitement bien aux services web KEGG et GO. Les classes que nous avons développé grâce à la documentation de KEGG[6] et l'interface vers GO fournie par le site WABI[7] permettent l'interrogation de ces services.

B.2. Ergonomie

Le logiciel est relativement facile d'utilisation.

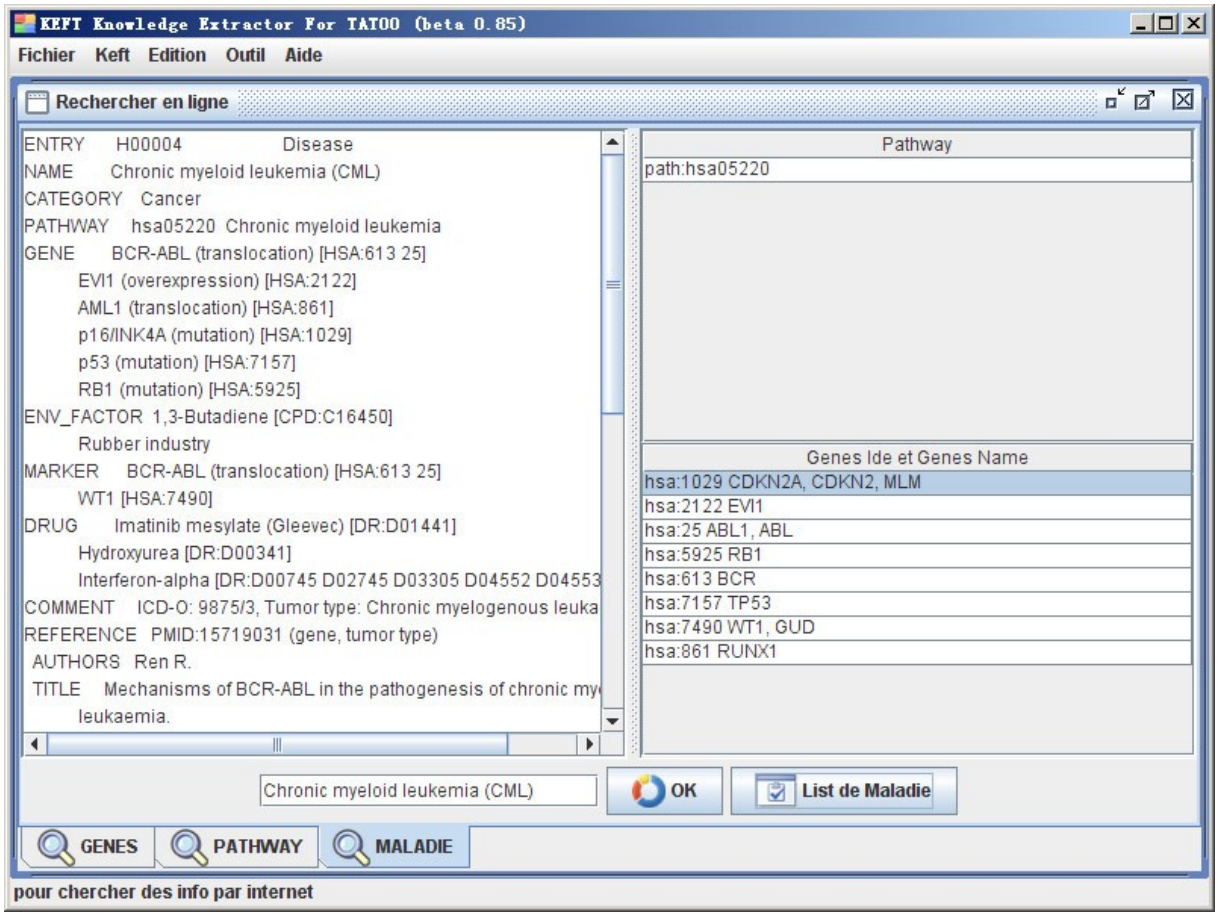
Nous avons essayé de limiter le nombre d'actions nécessaires pour un scénario. Le résultat est qu'un utilisateur pourra passer d'une recherche à une autre uniquement en deux cliques : "sélection d'un résultat" puis "action à effectuer".

Nous vous invitons à lire le manuel d'utilisation pour plus d'informations.

6 - http://www.genome.jp/kegg/soap/doc/keggapi_manual.html

7 - http://xml.nig.ac.jp/index.html

Résultats



L'utilisation d'un système de "panier" permet de mettre en mémoire les résultats trouvés ainsi que de les visualiser.

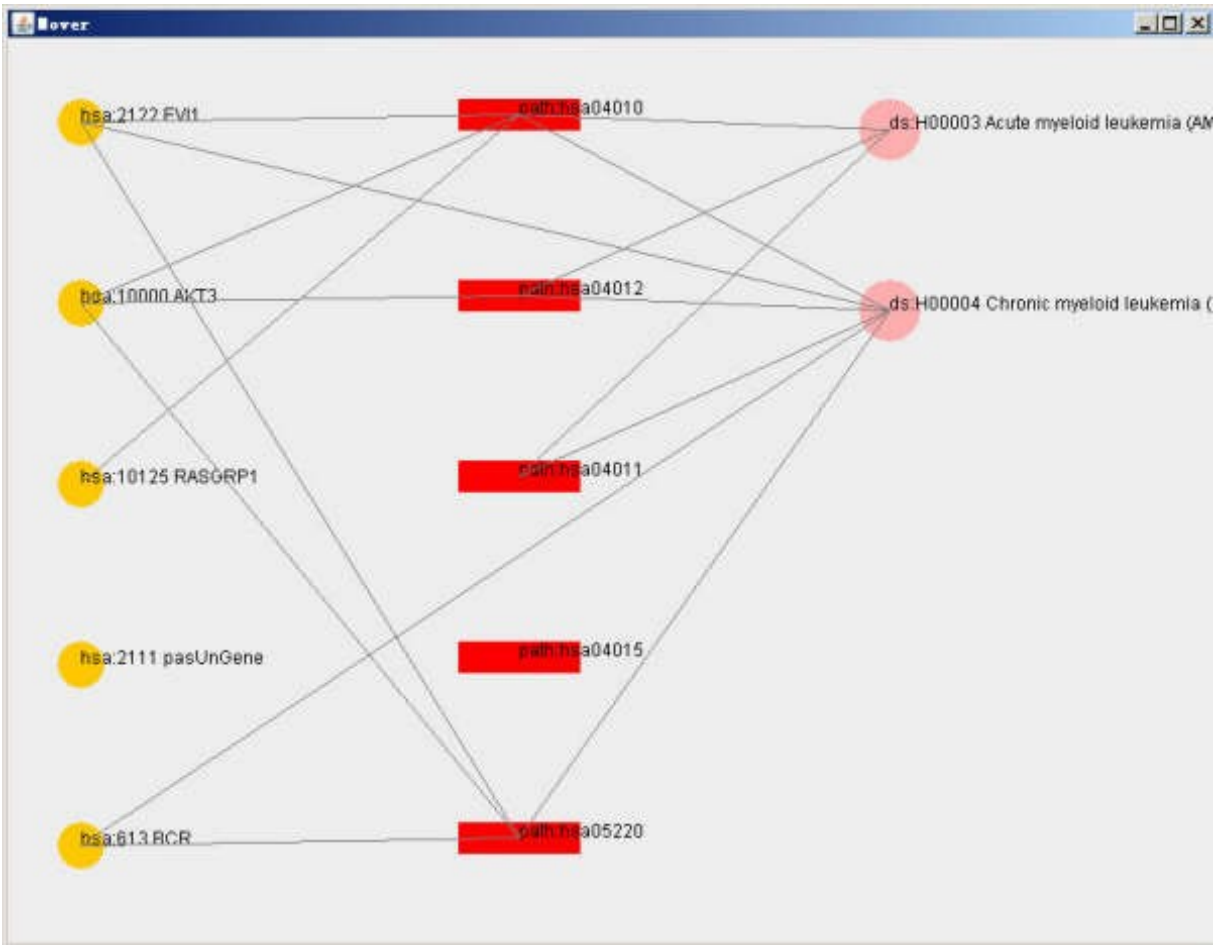


illustration 7 : Visualisation du résultat

B.3. Performances

Lors des premiers tests nous avons pu nous rendre compte que les temps d'interrogation des services web avaient tendance à être très longs (environ une demi-seconde par requête), de ce fait il a fallut trouver une solution.

Grâce à notre méthodologie de développement, nous nous sommes rendu compte assez tôt, lors de la première itération, qu'il fallait trouver un moyen d'accélérer le traitement des données. Pour cela, nous avons opté pour un système de cache qui puisse conserver les résultats en locale, pendant une certaine durée.

Pour cela nous avons utilisé le design pattern "Façade" qui est le plus approprié. En y ajoutant une classe d'abstraction pour une base de données, nous avons pu obtenir des temps de traitement quasi-instantanés pour autant que la recherche ait été faite antérieurement.

B.4. Réutilisabilité

Une architecture qui permet de réutiliser les classes principales.

L'architecture de Keft s'articule autour de la classe du même nom. Cette classe a été conçue pour pouvoir être réutilisable par d'autres développeurs. Ainsi, n'importe qui pourrait récupérer le package principal de notre application et développer sa propre interface utilisateur tout en conservant les mêmes fonctionnalités.

C. Discussion

Cette partie présente le bilan du projet. Nous allons voir les points positifs et les points négatifs.

C.1. Réussite ou échec ?

La première question à se poser concernant le résultat de notre projet est : "Avons-nous rempli les objectifs fixés?"

Dans la mesure où l'objectif de notre travail était l'interrogation de sources de données biomédicales et leur intégration sous une forme exploitable pour de le fouille de données, nous pouvons dire que l'objectif est atteint.

En effet, notre application est capable de récupérer des données provenant de KEGG et de GO et de les intégrer dans une base de données SQL, qui permet de récupérer facilement les données.

C.2. Perspectives

Il est évidemment impossible de faire un projet parfait : question de temps et de moyens. Voici quelques éléments que nous n'avons pas pu traiter et qui pourrait faire l'objet de mises-à-jour.

Faute de temps nous ne sommes pas allés jusqu'à exporter nos résultats sous le formatOWL . Cela semble être la fonctionnalité la plus importante à mettre en œuvre dans de futures versions.

Nous regrettons aussi de ne pas avoir pu intégrer les données de MESH. Cela aurait permis d'obtenir plus de données sur les maladies.

Nous aurions aussi aimé générer une javadoc complète afin de bien documenter les classes de notre application. Cette partie devrait cependant être faite assez facilement.

C.3. Les point forts du projet

Malgré quelques regrets quant au résultat, nous avons quand même eu la satisfaction d'aller au delà de la simple réponse aux besoins.

1. L'interface

A la base, l'interface n'a pas vraiment fait l'objet d'exigences fonctionnelles fortes car étant secondaire par rapport au traitement. Cependant nous avons réussi à obtenir un résultat intéressant pour l'utilisateur.

Rappelons que l'interface que nous avons élaborée permettra à l'utilisateur d'enchaîner différentes recherches en simplement deux cliques de souris.

Nous nous félicitons aussi de la visualisation des résultats qui n'était pas nécessaire, mais que nous avons tout de même inclue et donne un petit plus à notre application.

2. Les performances

Aucune spécification particulière ne faisait état des performances de l'application. C'est pourtant un facteur que nous avons pris en considération et le système de cache permettra l'économie de plusieurs heures de traitements.

3. La maintenance

le problème d'un grand nombre de projets d'étudiant réside dans le fait que la dernière étape qui est la "maintenance" est généralement réduite au stricte minimum du fait qu'elle ne rapporte pas de points. Statistiquement, il apparait donc que la durée de vie d'un projet de ce type est de l'ordre de 2 ans. Après quoi, le logiciel devient obsolète et/ou impossible à mettre à jour. De ce fait, il est généralement remplacé par un nouveau projet.

Nous avons essayé d'être le plus professionnel possible en prenant en compte cette étape du développement.

Ainsi, les utilisateurs pourront envoyer leurs rapports de bug ou leurs suggestions à support@keft.fr, ou ajouter un ticket sur keft.googlecode.com. N'importe qui pourra aussi contribuer au projet grâce à la licence GPL3[8] qui autorise l'utilisation, la distribution et la modification de l'application.

De notre côté, il nous sera toujours possible, même au bout de plusieurs années, de mettre à jour le code source grâce à googlecode.

De plus l'architecture du logiciel a été pensée de manière à pouvoir être réutilisable par n'importe quel développeur.

8 - http://www.gnu.org/licenses/gpl-3.0.html

Conclusion

Ce projet a été l'occasion de mieux cerner les enjeux d'une bonne communication au sein d'une équipe de développement. De plus, chacun de nous aura acquis certaines connaissances qui pourront s'avérer utiles dans une future vie professionnelle.

Nous sommes d'autant plus satisfait que le résultat répond plutôt bien aux besoins du cahier des charges. Et même si certaines améliorations sont encore à apporter, notre stratégie de développement et les choix techniques que nous avons fait favorisent le support à long terme de ce logiciel.

Glossaire

eXtrem Programming

Méthode itérative et incrémentale de développement logiciel, focalisée sur l'aspect programmation.

Voir p. 18

Integrated Developpement Environment

Logiciels d'édition de code source.

Voir p. 21

Refactoring

La refactorisation (anglicisme venant de refactoring) est une opération de maintenance du code informatique. Elle consiste à retravailler le code source non pas pour ajouter une fonctionnalité supplémentaire au logiciel mais pour améliorer sa lisibilité, simplifier sa maintenance, ou changer sa généricité (on parle aussi de remaniement). Une traduction plus appropriée serait réusinage. C'est donc une technique qui s'approche de l'optimisation du code, même si les objectifs sont radicalement différents.

source : wikipedia.org

Voir p. 17

Simple Object Access Protocol

Protocole de commande à distance basé sur XML

Voir p. 13 - 14 - 15

SQL

Langage de programmation utilisé pour effectuer des requêtes sur des bases de données de type dit "relationnel"

Voir p. 30

TATOO

Extraction de connaissances dans les bases de données : motifs séquentiels et ontologies

Voir p. 5 - 7

Unified Process

Méthode générique, itérative et incrémentale, de développement de logiciels orientés objet.

Voir p. 18 - 19

Uniform Resource Locator

Synonyme de "adresse web"

Voir p. 13

Web Ontology Language

Langage de description de données basée sur XML et permettant de représenter des ontologies.

Voir p. 30

Web Services Description Language

C'est un un format XML qui indique comment utiliser le service web qui lui est associé.

Voir p. 14

Références bibliographiques

1 Les cahiers du programmeur : Swing

Titre	Les cahiers du programmeur : Swing
Auteurs	Emmanuel Puybaret

Voir p. 18

2 UML pour les décideurs

Titre	UML pour les décideurs
Auteurs	Franck Vallée

Voir p. 16 - 18

Références webographiques

1 Gene Ontology

Adresse du site web :

http ://www.geneontology.org

Voir p. 15

2 GNU general public licence

Adresse du site web :

http://www.gnu.org/licenses/gpl-3.0.html

Voir p. 32

3 GoogleCode

Adresse du site web : http://googlecode.com

Voir p. 20

4 KEGG API reference

Adresse du site web :

http://www.genome.jp/kegg/soap/doc/keggapi_manual.html

Voir p. 24

5 Kyoto Encyclopedia of Genes and Genomes

Adresse du site web :

http://www.genome.jp/kegg

Voir p. 14

6 SourceForge

Adresse du site web : http://sourceforge.net

Voir p. 20

7 Web API for BioInformatics

Adresse du site web :

http://xml.nig.ac.jp/index.html

Voir p. 15 - 24

Illustrations