Table des matières

Remerciements

Je remercie, pour le bon accueil, toutes les personnes que j'ai pu côtoyer durant toute la période du stage à l'Observatoire de Strasbourg. Principalement mon tuteur André Schaaff, pour tous ses bons conseils.

Introduction

Ce rapport concerne mon stage de troisième année d'IUP Génie Mathématique et Informatique à Strasbourg. Celui-ci s'étendait sur la période du 1er avril au 31 août 2005. Il a été accompli à l'Observatoire Astronomique de Strasbourg et plus particulièrement au Centre de Données Astronomiques de Strasbourg (CDS).

Ce rapport est construit de la manière suivante : je débuterai par une présentation du centre de recherche, puis je présenterai le sujet du projet et enfin je décrirai plus précisément la solution développée.

1. Présentation du laboratoire

1.1. Structure juridique

L'Observatoire de Strasbourg est une Unité de Formation et de Recherche (UFR) de l'Université Louis Pasteur. Il est également une Unité Mixte de Recherche du CNRS et de l'Université Louis Pasteur(UMR 7550). *[1]*

1.2. Présentation générale

L'Observatoire est situé sur le campus de l'Esplanade ; ses bâtiments font partie du campus historique de l'université de Strasbourg.

Enseignements dispensés :

– DEA « Analyse et Traitement des Données sur les Milieux Astronomiques »

– DEUG Sciences et autres DEUG (enseignements d'ouverture).

– Licence de Géosciences.

– Maîtrise de Géosciences, Maîtrise de Physique, Maîtrise de Sciences Naturelles.

– Préparation au CAPES et à l'Agrégation.

– DESS Applications des Technologies Spatiales.

– Diffusion de la Culture

La partie publique de l'Observatoire, le Planétarium, est ouvert pour la vulgarisation de l'astronomie.

L'Observatoire se compose de quatre équipes de recherche :

1.2.1. Hautes énergies

L'équipe Astrophysique des Hautes énergies *[2]* a pour thème l'étude des astres et sites de l'univers émetteurs de photons de haute énergie. Cette thématique générale recouvre des aspects variés, comme l'étude des astres compacts en fin d'évolution, la physique de leur activité, les phénomènes de haute énergie intéressant les étoiles jeunes ou le soleil, ou l'étude de ces phénomènes à l'échelle galactique. Ces recherches se sont largement appuyées sur les données acquises par le satellite ROSAT et s'appuieront dans l'avenir sur celles des satellites X de nouvelle génération, tout spécialement XMM.

1.2.2. Etoiles et systèmes stellaires

Les recherches menées par l'équipe « Etoiles et systèmes stellaires » *[3]* recouvrent un domaine étendu, incluant les étoiles, les milieux interstellaires, la Galaxie et les galaxies proches. De l'étoile, objet individuel, l'intérêt s'est porté aux groupes d'étoiles, témoins de l'évolution stellaire mais aussi traceurs des grandes structures de la Voie Lactée.

1.2.3. Galaxies

Les activités de l'équipe *[4]* sont centrées sur les problèmes de la structure du Groupe Local, de ses populations stellaires et sur la dynamique gravitationnelle. De plus, l'équipe possède un savoir-faire sur les outils statistiques d'analyse de données et sur les méthodes inverses non paramétriques. Un des objectifs consiste à combiner les informations d'évolution des populations stellaires et celles de dynamique afin de reconstituer les événements déterminants liés aux processus de formation et d'évolution galactique.

1.2.4. Le centre de données (CDS)

L'activité de recherche du CDS *[5]* s'est concentrée sur l'étude de la dynamique galactique et des populations d'étoiles binaires, sur une participation importante à la mission HIPPARCOS de l'Agence Spatiale Européenne, ainsi que sur le développement de méthodologies nouvelles applicables à l'analyse et au traitement de données astronomiques.

1.3. Organisation du personnel

Le CDS est un laboratoire de l'Institut National des Sciences de l'Univers (INSU), rattaché au CNRS. L'Observatoire de Strasbourg est un institut de l'Université Louis Pasteur. Le personnel du CDS comprend 10 chercheurs, 6 ingénieurs, 3 techniciens, et plusieurs collaborateurs à contrat temporaire et invités.

1.4. Centre de données astronomique de Strasbourg (CDS)

J'ai effectué mon stage au sein du Centre de Données astronomiques de Strasbourg (CDS) qui est un centre de données dédié `a la collection et `a la distribution dans le monde entier de données astronomiques.

Le CDS héberge la base de données *Simbad*, la base de référence mondiale pour l'identification d'objets astronomiques. Le but du CDS est de :

– rassembler toutes les informations utiles, concernant les objets astronomiques, disponibles

sous forme informatisée : données d'observations produites par les observatoires

du monde entier, au sol ou dans l'espace ;

– mettre en valeur ces données par des évaluations et des comparaisons critiques ;

– distribuer les résultats dans la communauté astronomique ;

– conduire des recherches utilisant ces données.

Le CDS joue, ou a joué, un rôle dans d'importantes missions astronomiques spatiales : contribuant aux catalogues d'étoiles guides, aidant `a identifier les sources observées ou organisant l'accès aux archives, etc. Le CDS contribue au XMM Survey Science Center, sous la responsabilité de l'équipe « Hautes-Energies » de l'Observatoire de Strasbourg.

Le CDS a signé des accords d'échanges internationaux avec les organismes suivants :

– NASA,

– National Astronomical Observatory (Tokyo, Japon),

– l'Académie des Sciences de Russie,

– le réseau PPARC Starlink au Royaume-Uni,

– l'Observatoire de Beijing (Chine),

– l'Université de Porto Allegre au Brésil,

– l'Université de La Plata en Argentine,

– InterUniversity Center for Astronomy and Astrophysics (Inde).

Le CDS est membre de la Fédération des Services d'Analyse de Données Astrophysiques et Géophysiques.

Le CDS coopère aussi avec l'Agence spatiale Européenne (transfert au CDS du service de catalogues du projet ESIS : le projet *VizieR*), et avec la NASA : en particulier le CDS abrite une copie miroir du Système de Données Astrophysiques (ADS) et ADS abrite une copie miroir de *Simbad*. Le CDS contribue aussi au projet NASA AstroBrowse. Le CDS abrite aussi les copies miroirs Européennes des journaux de l'American Astronomical Society (AAS).

1.5. Les Services du CDS

1.5.1. Simbad

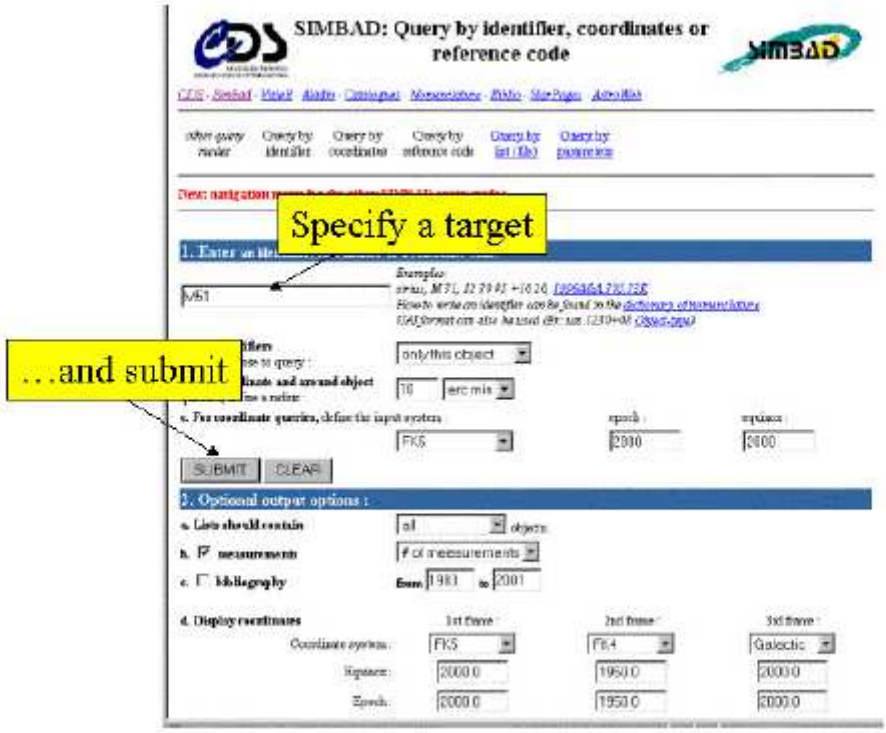


Figure. 1.1 - Page Web permettant d'effectuer une requête sur Simbad

Simbad est une base de données de référence pour les identifications et la bibliographie d'objets astronomiques. *[6] Simbad* contient plus 7,5 millions d'identificateurs pour plus de 2,8 millions d'objets différents. Pour chaque objet figurent dans la base quelques mesures (position, magnitude dans différents domaines de longueurs d'ondes), ainsi que les références bibliographiques où l'objet est cité (plus de 110 000 articles sont concernés). L'utilisateur peut choisir le format du fichier o`u seront entreposés les résultats de la requête(***Fig. 1.1***). En effet, *Simbad* peut générer des fichiers HTML, XML ou XLS (fichiers Excel).

Cet ensemble de données résulte d'un long travail d'identification croisée entre de nombreux catalogues, listes d'objets et articles de journaux, entrepris au début des années 1980, et constamment développé et mis `a jour depuis.

1.5.2. VizieR

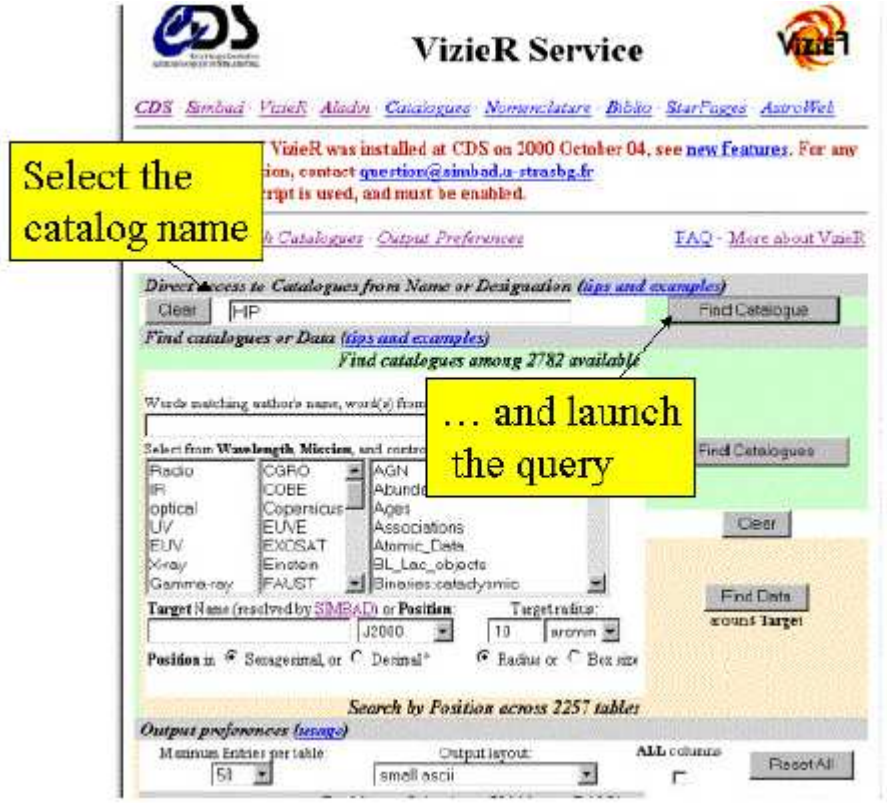


Figure. 1.2 – Page Web permettant d'effectuer une requête sur VizieR

VizieR est une base de données rassemblant plusieurs milliers de catalogues astronomiques sous un format homogène. *[7]* Une description standardisée du contenu des catalogues permet leur inclusion dans un système de gestion de base de données (SGBD) relationnel. Un ensemble de liens, entre les tables de *VizieR*, et avec des services externes (bibliographiques, archives externes, serveurs d'images), permettent de naviguer entre les données des catalogues et d'autres données associées (***Fig. 1.2***). Il faut noter que les très grands catalogues (plus de 107 enregistrements) ne peuvent pas être gérés par un SGBD relationnel pour des raisons de performances. Des outils spécifiques doivent être utilisés.

1.5.3. Aladin

Figure. 1.3 – Exemple d'utilisation d'Aladin

Aladin est un atlas interactif du ciel permettant d'accéder simultanément à des images numérisées du ciel, ainsi qu''`a des catalogues et bases de données astronomiques. *[8]* Cet outil permet de superposer, sur des images du ciel optique, les objets présents dans *Simbad*, des sources de catalogues contenus dans *VizieR*, mais aussi d'autres données, locales ou situées sur des serveurs distants (archives, HST, . . .).

2. Analyse de la situation actuelle

2.1. Origine de la proposition

Il y a quelques mois, au CDS (Centre de données astronomiques de Strasbourg), une personne a été engagée afin de permettre la manipulation de traitement d'image. Ces traitements ou tâches sont sur un serveur distant. Après analyse de la situation, la personne a décidé de faire un accès par CGI car c'est une manière simple d'accéder à un traitement. Au départ, il s'agissait d'exécuter sur le serveur une des tâches disponibles et de récupérer le résultat de cette requête. Puis, puisque cela était possible, l'enchaînement de plusieurs tâches consécutives pouvait être réalisé. Cependant, pour ceci, il fallait une application qui se charge de faire toutes les requêtes et de récupérer le résultat final afin de le remonter auprès du demandeur. Après mainte réunions, le groupe qui s'occupait de ce projet, prit l'initiative d'émettre un nouveau sujet de stage afin de s'occuper de la partie *Workflow* ou enchaînement de tâches.

2.2. Objectif et champs d'application

L'objectif de se stage consiste à permettre la création d'un *Workflow*, ou graphe représentant l'enchaînement d'un certain nombre de tâches consécutives. N'importe quelle personne doit pouvoir en concevoir. Les traitements sont divers et variés, mais sont principalement des calculs sur les images. Grâce aux différentes interfaces (web ou « *standalone* »), le CDS propose de récupérer des images astronomiques (étoiles, constellations, etc..). Les traitements sur ces images peuvent donc être très utiles pour les chercheurs.

2.3. Intérêts et avantages

Seul un groupe de personne pouvait utiliser ses traitements. La manière de faire n'était pas triviale car les tâches étaient disponibles uniquement sous la forme de fonctions *MatLab*. Tout le monde ne connaissant pas ce langage, ces fonctions n'étaient pas accessibles à l'ensemble des chercheurs. Si l'on développe un produit simple et utilisable de n'importe où, il est certain qu'un plus grand nombre de personnes pourront utiliser ces traitements.

2.4. Conséquences prévisibles

Si le produit est suffisamment générique, d'autres possédant également des traitements, pourront développer à leur tour à l'aide de ce produit, leur propre logiciel de traitements de tâches ou *Workflow*.

2.5. Etude et critique de l'existant

Il existe de nombreuse applications permettant de schématiser un *Workflow*. La plupart utilise un langage XML pour décrire celui-ci. Ils permettent de faire énormément de choses, mais ils sont aussi souvent extrêmement spécifiques. Etant donné que notre cas est bien précis, il a été nécessaire d'élaborer un produit adapté à notre environnement mais pouvant néanmoins l'être à d'autres.

Lorsque nous avons été sûr qu'il fallait concevoir un produit personnel, je me suis orienté vers les librairies graphiques permettant de construire des graphes ou des *Workflow*. Pour ce dernier, il n'existe pas de librairie, mais uniquement des applications prêtent à l'emplois. Par contre, il y a de nombreux outils pour fabriquer son propre logiciel de construction de graphe. Il y en a un très puissant, et totalement personnalisable : *Tom Sawyer*. Cependant celui-ci est payant. Grappa est un ensemble de packages Java qui nous donne la possibilité de créer notre propre application de graphe. Il utilise le format dot[12] pour la sauvegarde. Il est utilisé par le logiciel *GraphViz* qui permet la visualisation de graphe. Cette librairie était l'une de celle que j'ai eu à tester afin de déterminer laquelle nous prendrions. Ensuite, il y a *JGraph*[13], qui a attiré tout de suite mon attention car le site officiel avait l'air d'être mise à jour régulièrement. Des logiciels de démonstration sont directement téléchargeables sur leurs pages Internet. Ils donnent un bon aperçu des capacités de la librairie. Je l'ai repéré alors que je cherchais des logiciels de *Workflow*. En effet, la librairie est utilisée par une application de *Workflow* (JAWE[9]) utilisant le langage de *Workflow* XPDL[10]. De plus, il existe un tutorial, un forum actif, et une API complète sur leur site. Ceci n'est pas négligeable lorsqu'on doit développer avec une librairie. J'ai dû tester ou explorer d'autres logiciels comme *Triana*, *Taverna*, AVS pour ne citer qu'eux, mais aucun m'a donné autant de satisfaction que *JGraph*. C'est donc ce dernier, que j'ai choisi pour développer mon produit.

3. Le Workflow

Un des mes premiers travail fût de rechercher des informations concernant les *Workflow*. Pendant un mois, j'ai recueilli une foule de document s'occupant de les décrire, de proposer des logiciels pour en concevoir ou en exécuter, et d'analyser les différents éléments les constituant.

3.1. Définition

Un *Workflow* est un flux d'informations au sein d'une organisation, par exemple en transmettant automatiquement des documents entre des personnes.

On appelle « *WorkFlow* » (traduisez littéralement "flux de travail") la modélisation et la gestion informatique de l'ensemble des tâches à accomplir et des différents acteurs impliqués dans la réalisation d'un processus métier (aussi appelé processus opérationnel). Le terme de *Workflow* pourrait donc être traduit en français par Gestion électronique des processus métier. De façon plus pratique le *WorkFlow* décrit le circuit de validation, les tâches à accomplir entre les différents acteurs d'un processus, les délais, les modes de validation et fournit à chacun des acteurs les informations nécessaires pour la réalisation de sa tâche. Pour un processus de publication en ligne par exemple, il s'agit de la modélisation des tâches de l'ensemble de la chaîne éditoriale.

Il permet généralement un suivi et identifie les acteurs en précisant leur rôle et la manière de le remplir au mieux.

Un *Workflow* peut être très complexe. Il organise l'enchaînement de tâches, de conditions de contrôle (XOR, AND, OR …), et met en jeux différentes entités. Vous devez décrire votre *Workflow* à l'aide d'un logiciel de *Workflow* afin qu'il puisse être compris et exécuté. Un langage de description de processus est un langage spécifique qui vous aide à concevoir votre *Workflow* avec des mots compréhensibles par vous et par un logiciel. Depuis que le XML est aussi populaire, de nombreux langages de description de processus, comme XPDL [10], BPML ou encore WS-BPEL [11], ont vu le jour.

3.2. Les types de Workflow

Dans les applications de *WorkFlow*, on distingue classiquement quatre catégories :

- le ***Workflow* de production**, qui correspond à la gestion des processus de base de l'entreprise. Les procédures supportent peu de changements dans le temps, et les transactions sont répétitives. On peut y trouver par exemple la production de contrats d'assurance, la gestion de litiges, la gestion de réclamations clients, etc.
- le ***Workflow* administratif**, qui correspond à tout ce qui est routage de formulaires, basé en général sur une infrastructure de messagerie.
- le ***Workflow* ad-hoc** pour la gestion des procédures non déterminées, ou mouvantes.
- le ***Workflow* coopératif**, gérant des procédures évoluant assez fréquemment, et liées à un groupe de travail restreint dans l'entreprise.

3.3. La nature des tâches

On trouve essentiellement trois types de tâches dans un système de *WorkFlow*. Certains systèmes n'en supporteront qu'un, d'autres en supporteront d'emblée deux ou trois. Les types de tâches sont les suivantes :

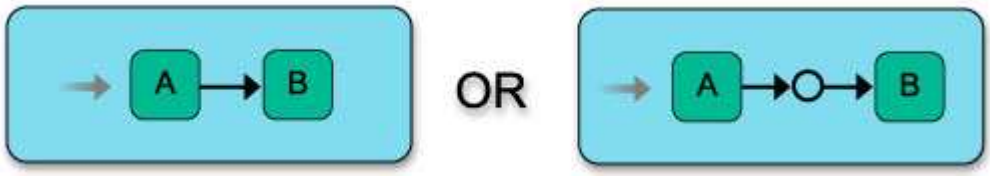
- les tâches qui sont en fait des formulaires de données, généralement définis à partir du produit de *Workflow* lui-même, à compléter au fur et à mesure de l'avancement de la procédure. Ce sont les tâches que l'on trouve dans les ***Workflow* administratifs**.
- les tâches qui sont des services du système d'informations, tels que la saisie de transactions gros systèmes, ou l'appel à un exécutable spécifique, etc. Le ***Workflow* de production** est entièrement basé sur ce genre de tâches, puisque son travail consiste à coordonner l'ensemble des actions possibles au sein du système d'information.
- les tâches qui correspondent à un routage de fichiers bureautiques. On retrouve ces tâches essentiellement dans les ***Workflow* administratif** ou dans les ***Workflow* ad-hoc**, comme peuvent le proposer les différents traitements de texte du marché en se servant des messageries comme infrastructure.

3.4. Les types de contrôles

La notion de contrôles est très important dans le *Workflow*. Il faut permettre la manipulation du flux de données qui transite à travers les différentes tâches du *Workflow*. Pour ceci, il existe des Patterns rangés dans plusieurs catégories.

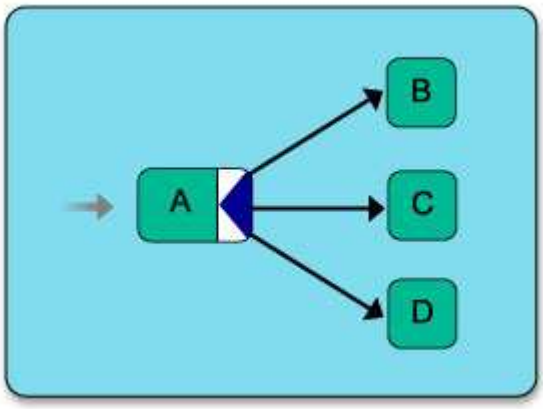
Les plus élémentaires Patterns sont ceux-ci :

1. *Séquence*



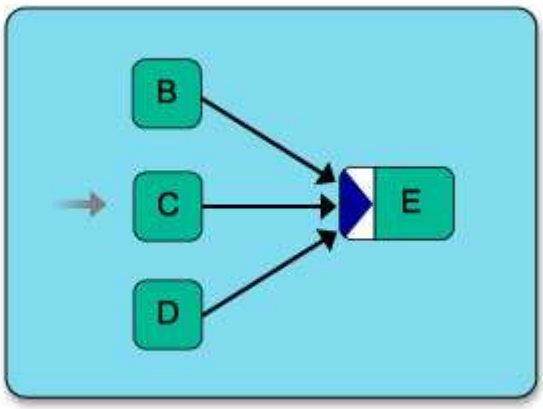
Après l'exécution de la tâche A, la B peut être lancée.

2. *Parallèle*



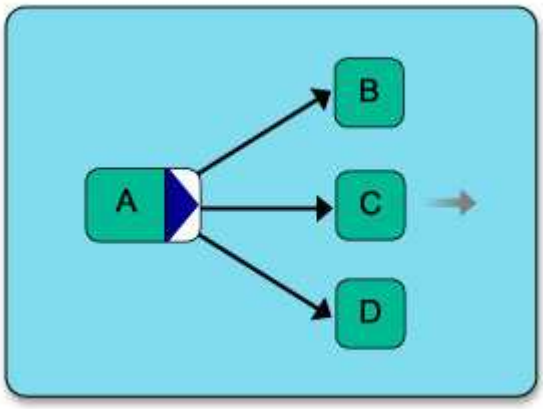
Lorsque la tâche A est terminée, B, C, et D s'exécutent en parallèle.

3. *Synchronisation*



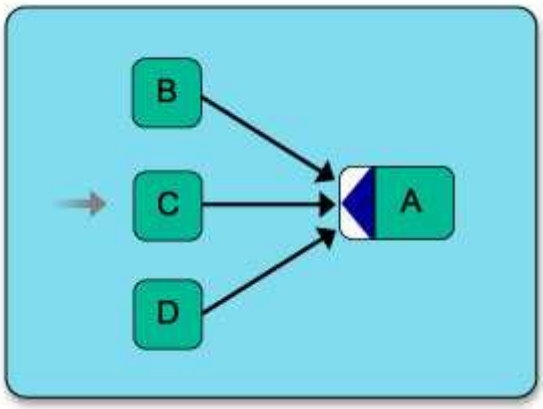
Suite à l'exécution des tâches B, C, et D, E peut commencer.

4. *Choix Exclusif*



Un choix est fait pour savoir quelle tâche entre B, C, et D, sera exécutée après que A est terminée.

5. *Fusion*



La tâche A peut commencer dès que l'une des taches B, C, ou D est terminée.

4. Le Projet

Le projet comporte deux parties. L'une est une application cliente qui peut être installée sur n'importe quelle machine, l'autre est un serveur qui interagit avec le programme client.

4.1. La mission

Le client doit être en mesure de concevoir un *Dataflow*, c'est à dire un enchaînement précis de calculs prédéfinis, et s'échangeant parfois des données. Il fournit des boîtes et des flèches qui lui permettront de constituer une suite de traitements. Une boîte peut être différentes opérations comme un calcul, une condition logique (OR, AND, XOR) ou la représentation d'un paramètre d'entrée de fonction. Les flèches quant à elles, illustreront la jonction entre les boites et correspondront à un échange de données. Une fois le graphe ou *Workflow* constitué, le programme client aura la charge d'effectuer la conversion de ce dernier en un fichier diffusable sur le réseau et compréhensible par la partie serveur notamment. Le client recevra le ou les résultats finaux du *Dataflow* ainsi qu'éventuellement les résultats intermédiaires. Un dialogue bilatéral est donc nécessaire entre les entités clientes et serveurs. On doit également admettre l'exécution d'une seule tache et éventuellement d'un sous-ensemble du *Workflow* total.

Le serveur sera capable de recueillir un fichier venant du réseau. Il interprétera celui-ci afin de pouvoir le traiter. L'exécution des tâches ou calculs ne se fera pas par le serveur mais par une autre entité. Il sert donc d'intermédiaire permettant de gérer de manière avancée l'agencement des différents traitements. Il permet également d'éviter que l'utilisateur effectue les calculs un par un et donne la possibilité d'une plus grande complexité dans l'enchaînement des tâches.

4.2. Les Résultats

Nous avons finalement opté pour une librairie qui serait utilisable par quiconque voulant manipuler les *Workflow*. Une application servant à appeler des opérations distantes a été conçue par une personne du CDS. Etant donné qu'on ne voyait pas l'utilité de faire deux programmes séparés, l'une appelant une tâches distantes et l'autre effectuant une suite d'appel (*Workflow*), une seule et même application sera distribuée. Ma mission consista alors à

développer une librairie afin qu'elle puisse servir à l'implémentation de cette application. J'ai donc dû collaborer avec la personne du CDS. Je travaille actuellement afin de lui procurer les outils dont il a besoin. Nous entretenons des réunions régulièrement pour nous tenir au courant des avancées.

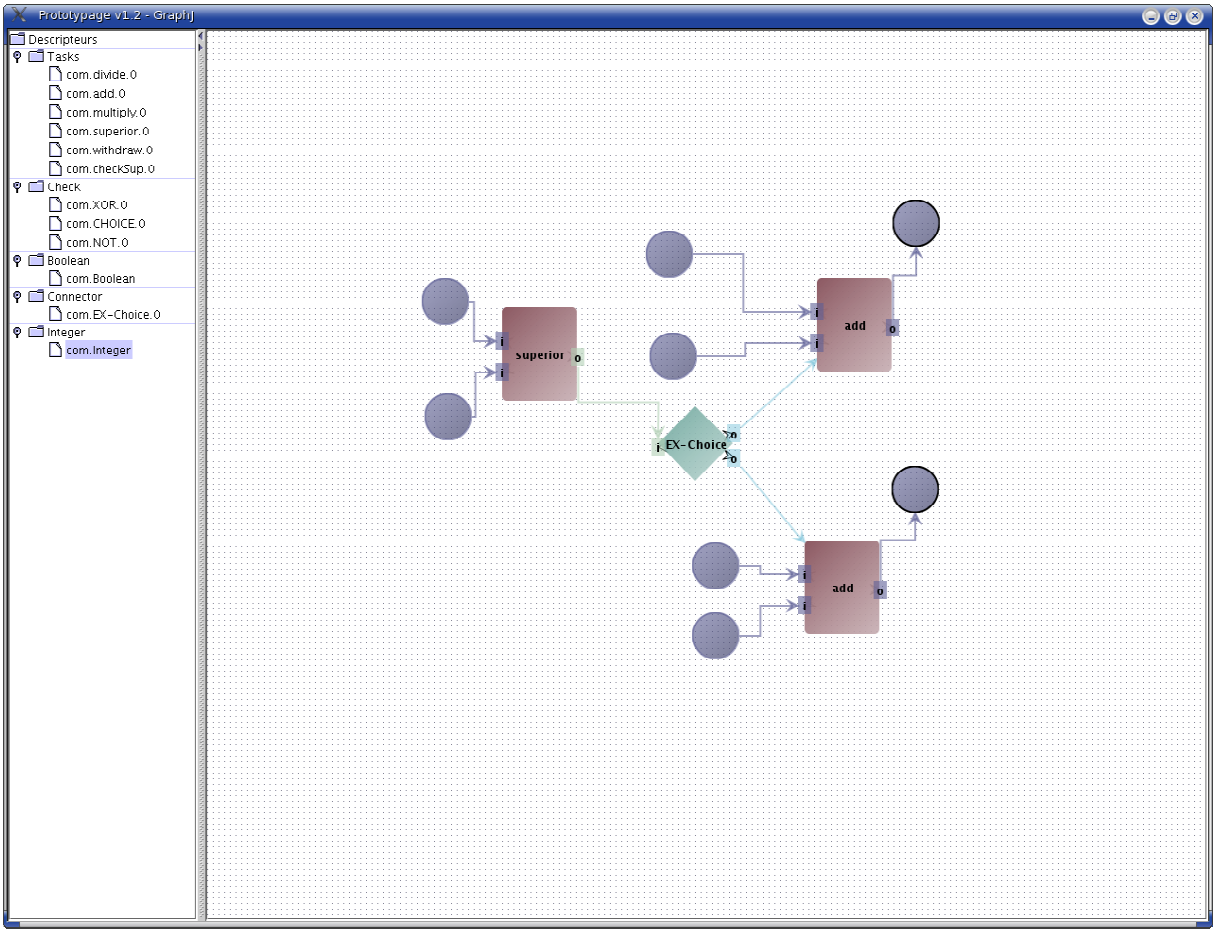


Figure 2.1 - La librairie dispose d'un exemple d'implémentation

Une page de présentation de la librairie est mise en place dans une section du site du CDS. Quelques pages m'ont été demandées, pour permettre de télécharger la librairie, avoir des informations sur la façon de l'utiliser, et divers renseignements que l'on peut trouver sur n'importe quel site. Elles ont disponible en français mais aussi en anglais.

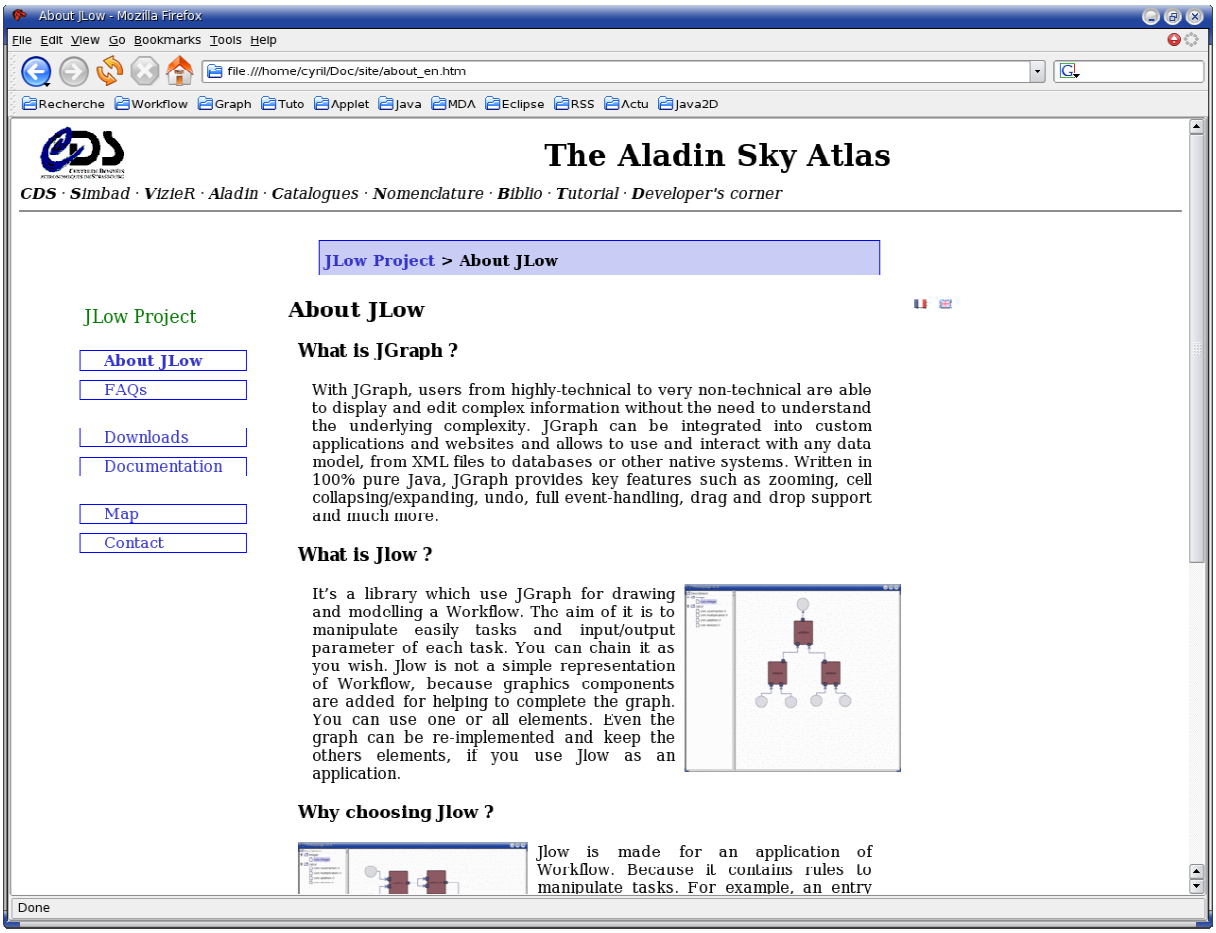


Figure 2.2 - La présentation de la librairie est disponible sur le site

Plusieurs versions en téléchargement sont disponibles. La version light contient uniquement la librairie, c'est à dire un fichier jar comportant les classes nécessaires à l'implémentation d'un programme voulant manipuler des chaînes de *Workflow*. Une autre version comporte en plus, des exemples d'implémentation. Une sorte d'application de démonstration qui se contente de faire des calculs simples. Il faut savoir que la librairie pour concevoir le serveur est disponible dans un jar séparé. Il y a aussi une version, sous la forme d'une application java web start. Cela permet à n'importe qui sur le net, de tester un programme utilisant la librairie. Evidemment, c'est une implémentation simple et légère. Le programme ne fait pas d'échange avec un serveur mais effectue le calcul directement. Tout se passe au niveau du client. Cela permet de donner une idée des capacités de la librairie.

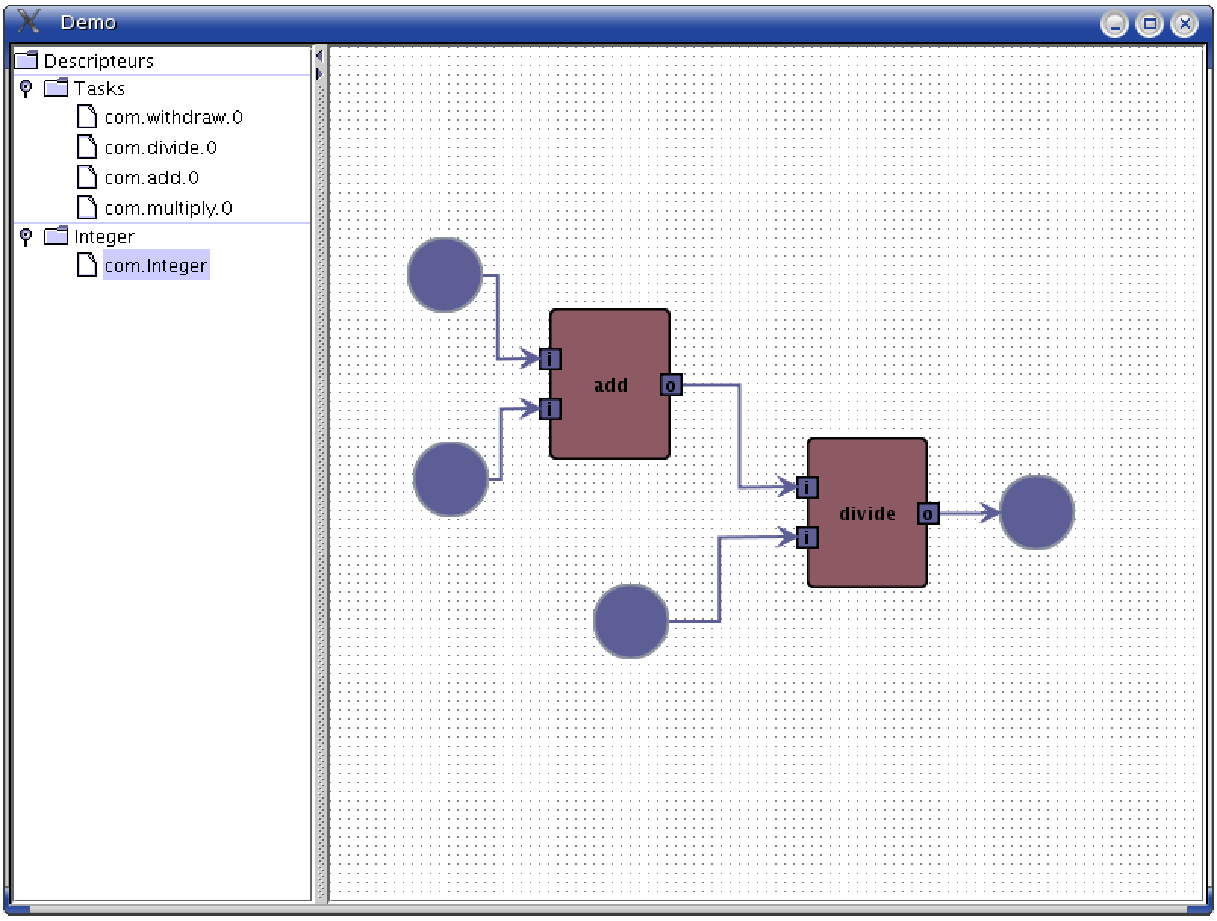


Figure 2.4 - Le programme de démonstration disponible sur Internet

Cette même démonstration est disponible dans la version *Applet Java*. Maintenant, tous les navigateurs téléchargeables sur le net savent faire fonctionner les *Applets*. Nous espérons ainsi donner la possibilité de tester la démonstration à un maximum de personne.

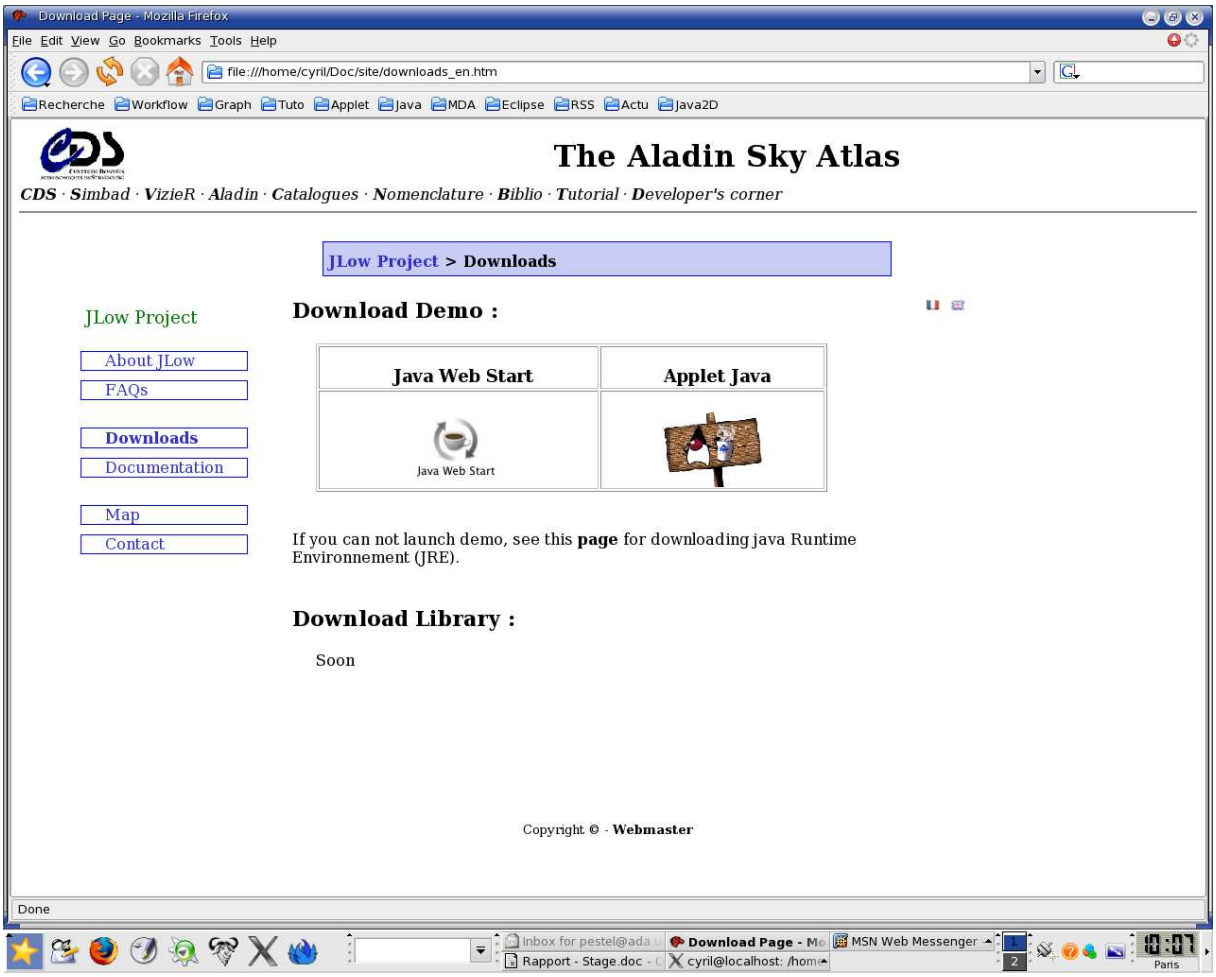


Figure 2.3 - Page de téléchargement des démonstrations

Afin de permettre la prise en main aisée pour un développeur, un tutorial a été rédigé en français et bientôt en anglais. Le produit y est présenté de manière simple tout d'abord. Puis, pour ceux désirant pouvoir manipuler de façon plus précise et plus poussée, les dernières parties sont consacrées à l'explication de code, provenant du produit lui même. Ce tutorial est fait tout aussi bien pour une personne cherchant des informations sur le produit que pour ceux qui veulent apprendre à se servir de la librairie.

5. Le client

5.1. Personnalisable

Chaque partie du client est bien séparée. Il y a tout d'abord les exemples d'implémentations qui ont été placés dans des packages Java, séparés de la partie principale. Puis ce qui constitue le noyau de la librairie, est divisé à son tour en plusieurs packages. Les classes sont pour la plupart dissociable des autres. Les interfaces Java ont été placées stratégiquement afin de permettre la génération de nouvelle classe sans avoir à modifier les autres qui vont l'utiliser. Pour faire simple, si une classe utilise une autre qui implémente une interface, il suffit de décrire chacune des méthodes de l'interface pour constituer la nouvelle classe désirée. Aucune modification n'est alors à faire du coté de la classe qui utilise cette nouvelle classe. Des implémentations pour chacune de ces interfaces sont disponibles. Ainsi, on ne change que ce que l'on veut, sans ce soucier du reste.

5.2. Evolutif

Dans le but de rendre la librairie la plus modulable et la plus évolutive possible, les données sont séparées de la vue (interface graphique). C'est à l'aide d'une classe qu'il est possible de contrôler la vue des objets manipulés. Ces derniers sont stockés séparément. C'est à la charge d'une autre entité d'aller puiser les informations dont elle a besoin, dans la classe où ces objets sont enregistrés. Puis il construit la vue de l'objet en fonction des informations recueillies précédemment. De cette manière, un programmeur peut éventuellement soit changer la manière dont sont stockées les informations, soit modifier l'apparence des objets à l'écran ou bien encore faire de façons différentes la jonction entre la vue et les données, et tout cela sans impacter sur l'une des autres parties.

5.3. Les fonctionnalités

Afin de montrer l'étendues des possibilités de la librairie, j'ai développé des outils. Certains sont liés au *Workflow*, comme les patterns élémentaires (voir Les types de contrôles), d'autres sont souvent utilisés dans les applications, par exemple la sauvegarde et l'ouverture

de fichiers, enfin il y a les indispensables si l'on désire vérifier que le serveur fonctionne correctement, bien entendu c'est l'exécution à distance du *Workflow* créé.

De manière à simplifier la vue du *Workflow*, j'ai implémenté l'exportation et l'importation d'un *Workflow*. Ainsi, on peut constituer son graphe, puis l'exporter dans un fichier sur le disque. Enfin, importer ce même fichier quand on le souhaite. Cette importation permet de constituer un nouvel objet contenant certaines informations sur l'ensemble des tâches et données du *Workflow* importé. Cette objet sera représenté comme une seule tâche alors qu'il en contient plusieurs.

5.4. Evolution possible

Avec le temps qu'il me reste, avant la fin de stage, je pense qu'il serait intéressant de permettre de tester la librairie à l'aide d'une application « *standalone* ». Le programme serait alors installé sur la machine d'un utilisateur. Il se présenterait avec les éléments habituels : barre de menu, barre d'état, etc. Il doit permettre le plus de fonctionnalités que peut offrir la librairie. On pourrait également permettre la création de plusieurs graphes, dans des onglets différents. Cela donnerait, à quelqu'un qui n'aurait pas le temps de développer sa propre application, une moyen rapide et sans difficulté d'utiliser la librairie.

La gestion des erreurs est disponible, mais on aimerait bien avoir une gestion plus sophistiquée, avec par exemple des commentaires pour chaque erreur, comme avec les exceptions Java.

La possibilité de vérifier le *Workflow* avant l'envoi au serveur, permettrait moins de trafic sur le réseau. La gestion des erreurs pourrait donc être réalisée également au niveau du programme client.

6. Le Serveur

6.1. Le moteur de Workflow

Le moteur permet l'enchaînement, de manière logique, des tâches. Il faut alors contrôler les dépendances temporelles et logiques. En effet, une tâche doit parfois attendre une autre quand cela est demandé. L'utilisateur peut indiquer explicitement qu'il souhaite qu'une tâche soit traitée avant une autre. Ceci constitue une dépendance temporelle. Il arrive aussi qu'un des traitements à besoin d'une valeur qu'un autre va lui donner plus tard. Il doit alors attendre que toutes ses valeurs en entrées soient disponibles.

Le moteur doit permettre également d'indiquer au demandeur, les erreurs qui peuvent empêcher le bon déroulement du *Workflow*. On ne contrôle pas l'affichage de cette erreur, car cela doit être à la charge de la partie client. Celui-ci reçoit un message du serveur, pour lui indiquer l'erreur. Après, c'est au client de l'afficher ou non.

6.2. La manipulation du Moteur

Comme nous l'avons vu précédemment, c'est un fichier XML qui est le plus souvent utilisé pour la description du *Workflow*. C'est pourquoi nous avons choisi de prendre le langage GXL. Il permet de décrire des graphes extrêmement complexes. Dans notre cas, il suffit amplement, car contrairement aux autres langages, il est souple et léger. Les langages de *Workflow* sont largement plus lourd. Cependant, on peut à tout moment en utiliser un autre. Etant donné que ce sont deux classes Java qui permettent la manipulation des fichiers contenant le *Workflow*, il suffit de les changer par d'autres, spécifiques à un langage différent.

C'est l'interprétation de ce fichier XML qui permet de constituer les éléments du moteur. Une fois le moteur prêt, il se charge lui même du déroulement des opérations et des différents échanges nécessaire entre les entités.

6.3. L'utilisation de thread

Afin de permettre l'exécution de tâches parallèles, nous avons eu recourt aux threads. Celles-ci sont assez simple à mettre en place en Java. Cela permet de ne pas être bloqué sur une tâche qui mettrait du temps à s'exécuter. En effet, le serveur ne peut pas se permettre d'être inactif.

6.4. Les différentes interfaces d'accès

- **socket** : Pour effectuer une requête sur le serveur, en passant par le réseau, il nous faut dérouler une structure le permettant. Le plus simple en Java, est l'utilisation des *sockets*. Pour le moment, ce sont des liens non sécurisés qui sont utilisés. Cependant ceci ne constitue pas une solution diffusable à n'importe qui sur le réseau. C'est pourquoi, nous avons décidé de déployer un autre moyen d'accès : les *servlets*.

- **servlet** : Pour le moment, ceci n'est pas développé, mais je pense que d'ici la fin du stage, j'aurai le temps de le faire. C'est assez important, car cela permettrait de diffuser notre service de *Workflow* à travers un serveur HTTP.

6.5. Evolution possible

Si j'ai le temps j'aimerais essayer une autre interfaces d'accès : le *WebService*. Celle-ci est déjà utilisée au CDS, c'est pourquoi je souhaiterais profiter de cette opportunité.

Conclusion

Durant le premier mois de ce stage, j'ai du m'informer sur ce qu'était un *Workflow*, les différents produits en relation avec le sujet, et d'autres renseignements que l'on peut trouver pour analyser la mission qui m'a été asservie. Mon premier réflexe fut de chercher tout ce que je pouvais trouver sur le net, avec les mots-clefs tels que *Workflow*, *Dataflow*. Très rapidement, j'ai eu beaucoup de sites traitant le sujet du *Workflow*. Certains étaient des articles, d'autres des sites entièrement consacrés à ce sujet. En parcourant les différentes pages Internet, j'ai découvert qu'il y avait de nombreuses applications destinées à créer un *Workflow* complet. Il y a également une importante quantité de langage (Xml pour la plupart) ayant pour but de décrire le *Workflow*. C'est pourquoi, je me suis mis à penser que mon stage allait terminer rapidement, puisqu'il suffisait de reprendre un logiciel qui utilise un certain langage. J'aurais eu juste à comparer ces différents programmes et langages pour voir lequel était le mieux où lequel donnait le plus de possibilités. C'est en discutant avec mon tuteur qu'il m'a expliqué qu'il ne souhaitait pas avoir une application qui peut tout faire, un programme qui utilise un langage en particulier mais ne peut en utiliser d'autre. Or c'est ce qui existait sur le net. Le but du stage était donc de faire un produit capable de créer un *Workflow,* et qui serait suffisamment générique pour permettre l'évolutivité. Mon tuteur voulait également que cela serve pour l'Observatoire mais pour éventuellement d'autres personnes. Toute la difficulté finalement a été de rendre le produit suffisamment malléable pour être adapté dans n'importe quelle situation, et n'importe quel environnement.

J'ai eu l'opportunité de présenter la librairie à une personne ayant travaillé sur le projet *AstroGrid* (projet anglais), de passage à l'observatoire de Strasbourg. Mon tuteur m'avez notifié au début du stage, que le produit que j'allais développer pourrait intéresser d'autres personnes comme ceux sur le projet *AstroGrid*.

Les Apports

L'Observatoire de Strasbourg est un centre dans lequel de nombreux chercheurs et chercheuses peuvent travailler. L'environnement y est convivial, il y a assez d'espace pour que tout le monde s'y sent à l'aise. C'est, selon moi, le cadre idéal pour travailler tout en prenant plaisir à accomplir sa tâche. J'ai vraiment apprécié la tranquillité, ce qui m'a permis de me concentrer sur mon travail. Lorsque j'avais des questions, je trouvais toujours

quelqu'un pour y répondre. J'ai eu même des informations ne concernant pas mon travail mais plutôt l'astronomie. J'ai pu participer à un Workshop [Annexe 1] organisé par mon tuteur. Celui-ci était sur les *Grid*. Ce fut une journée très enrichissante, pour moi qui n'avais pas encore assisté à autant de conférences (en anglais) dans une même journée. C'est une expérience que j'ai prise avec plaisir et que je n'oublierai jamais. Tous ses événements et toutes les personnes que j'ai rencontré pendant ce stage, m'ont permit de me donner une petite idée de ce qu'est la vie d'un chercheur. Cela m'attire de plus en plus. Je suis content d'avoir pu trouver un stage dans un tel endroit plutôt que dans une entreprise. Mais je me rends compte que vu mon manque d'expérience, je ne peux avoir un avis définitif. Tous les centres de recherche ne se ressemblent pas et certaines entreprises doivent permettrent un environnement aussi bénéfique que celui dans lequel j'ai baigné pendant toute la durée de mon stage.

Du point de vue professionnel, j'ai gagné en expérience. J'ai respecté un planning afin de terminer dans les temps. Un planning comportant des phases de récupération d'informations, de testes, et de développement. J'ai pris énormément plaisir à développer une librairie. C'est différent d'une application classique. Je n'ai rien appris de nouveau concernant les réseaux, mais j'espère le faire dans les prochains jours, en mettant en place un *WebService*. J'ai pris du plaisir tout le long du stage, et je pense que c'est l'essentiel.

Les Difficultés

La plus grande difficulté que j'ai eu à surmonter est le fait qu'une librairie telle que la mienne doit être conçue de manière à être entièrement malléable. On doit permettre la personnalisation de toutes les classes (au sens Java). Il existe de nombreuses librairies en Java, desquelles j'ai pu m'inspirer. Le fait que mon produit soit utilisé dans un programme (développé par une personne en CDD à l'Observatoire), prouve qu'il fonctionne correctement. C'est donc une grande satisfaction pour moi.

Webographie

Les sites proposant les différents Patterns :

- http://www.yawl.fit.qut.edu.au/
- http://www.workflowpatterns.com/

Les sites décrivant les Workflow :

- http://www.mayetic.fr/Home.nsf/Pages/ARCHIVEWorkflow
- http://en.wikipedia.org/wiki/Workflow

Annexe 1 : Grid Workshop

Annexe 2 : Références

[1] Site Internet de l'Observatoire astronomique de Strasbourg : http://astro.u-strasbg.fr/Obs.html

[2] Site Internet de l'équipe Hautes Energies : http://astro.u-strasbg.fr/Obs/ENERGIE/ENERGIE.html

[3] Site Internet de l'équipe Etoiles et système stellaire : http://astro.ustrasbg.fr/Obs/PHYSTEL/PHYSTEL.html

[4] Site Internet de l'équipe Galaxies : http://astro.ustrasbg.fr/Obs/GALAXIES/

[5] Site Internet du **CDS** : http://astro.u-strasbg.fr/Obs/CDS/CDS.html

[6] Site Internet du service **Simbad** : http://simbad.u-strasbg.fr/Simbad

[7] Site Internet du service **VizieR** : http://vizier.u-strasbg.fr/

[8] Site Internet du service **Aladin** : http://aladin.u-strasbg.fr/aladin.gml

[9] Site internet de **JaWE** : http://jawe.objectweb.org

JaWE is a tool for Process Definition modelling. The final output of this process modelling is a **XPDL** output file, which can be interpreted at runtime by the workflow engine(s). JaWE accomplished three main goals:

- Graphical representation of process definition (Jgraph)
- Export of process definitions to **XPDL**
- Import of any valid **XPDL** and its graphical representation

[10] **XPDL** (XML Process Definition Language ou langage (dérivé du) XML de définition de processus).

C'est un standard de la Workflow Management Coalition (WfMC) qui permet de définir un processus métier à l'aide du langage XML. Processus métier qui sera ensuite utilisé par un moteur de workflow.

[11] **WS-BPEL** :
(http://www.smartcomps.org/confluence/pages/viewpage.action?pageId=182)

[12] **DOT** : C'est un langage permettant de décrire de manière simple un graphe. (Texte simple)

[13] **JGraph** : Petit mais totalement fonctionnel, JGraph est un composant Java qui facilite le développement de représentations graphiques de réseau grâce aux principes de la théorie des graphes.