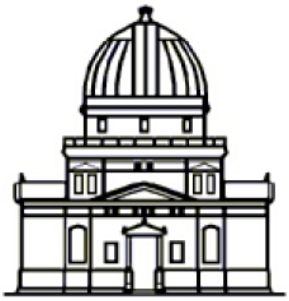
IUT Nancy Charlemagne
Université de Lorraine
2 ter Boulevard Charlemagne
54052 Nancy Cedex
Dépt. Informatique

Chargement d'une table et interfaçage

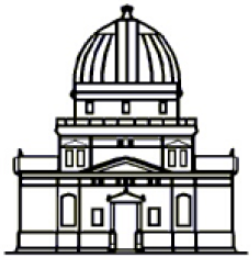


Observatoire astronomique
de Strasbourg

Mallory MARCOT
DUT Informatique AS

IUT Nancy Charlemagne
Université de Lorraine
2 ter Boulevard Charlemagne
54052 Nancy Cedex
Dépt. Informatique

Chargement d'une table et interfaçage



Observatoire astronomique
de Strasbourg

11, Rue de l'Université
67000 Strasbourg
03 68 85 24 10

Stagiaire :

Mallory Marcot
DUT Informatique AS

Tuteur en entreprise :

François Bonnarel
Ingénieur de recherche

Parrain de stage :

Bernard Mangeol

Remerciements

Je tiens à remercier :

- François Bonnarel, mon tuteur en entreprise, pour sa patience, ses explications et son implication dans le bon déroulement du stage.
- Françoise Genova, la directrice du CDS pour l'intérêt porté à mon stage, et son approbation concernant mon intégration en CDD.
- Hervé Wozniak, le directeur de l'observatoire, pour m'avoir accueillit dans son établissement.
- Laurent Michel pour ses explications sur le logiciel Saada et sur les bonnes habitudes de développement.
- Pierre Fernique pour m'avoir expliqué le fonctionnement des plug-in autour du logiciel Aladin.
- Mireille Louys pour la relecture du rapport.
- L'équipe pédagogique de l'IUT Charlemagne, et plus particulièrement Bernard Mangeol pour avoir suivit mon stage.

Table des matières

Introduction

Travail demandé

Le service de catalogues astronomiques VizieR contient aux cotés des tables et de leurs descriptions de nombreuses données additionnelles (spectres, courbes de lumière, etc...) dont certaines sont des images. 142 catalogues, contant les listes des sources observées sur une portion de ciel sont ainsi associés à des images de façon très diverse. Nous souhaitons donner un accès standardisé à ces images.

La première partie du stage consistera à structurer les descriptions de ces images autour d'un modèle de données standardisé (ObsCore) defini par l'IVOA pour pouvoir faire des requêtes à la base des images via des paramètres tels que la position, la longueur d'onde, l'origine ou le nom de l'objet du ciel associé à cette image. Cela nous permettra de créer des conditions de navigation depuis les pages de VizieR vers ces images mais aussi les images découvertes indépendamment vers VizieR.

La seconde partie du stage portera sur le développement d'un plug-in en Java, qui viendra se greffer sur le logiciel Aladin (un atlas interactif du ciel). Il devra permettre à l'utilisateur de naviguer interactivement dans les descriptions d'images préalablement structurées.

Annonce du plan

Dans un premier temps, nous verrons les solutions apportées concernant la structuration de données hétérogènes (fichiers *.fits) autour d'un modèle de données standardisé (ObsCore Data Model) et les difficultés que nous avons rencontrées.

Dans une seconde partie, nous verrons plus en détail le développement d'un interface en Java qui viendra se greffer sur le logiciel Aladin.

Présentation de l'entreprise

Présentation générale

L'observatoire astronomique de Strasbourg est situé sur le campus de l'Université de Strasbourg. Il a été fondé en 1881. C'est un établissement qui effectue de la recherche et de l'enseignement dans le domaine de l'astronomie. Il inclut le Centre de Données astronomiques de Strasbourg (CDS) qui s'occupe de collecter et de diffuser des données issues de l'observation ou de traitements (données fondamentales, mesures, catalogues, images et spectres) servant à l'identification et à la caractérisation d'objets astronomiques ou permettant des études scientifiques ultérieures.

La **recherche** s'effectue dans les domaines :

- de l'astrophysique des hautes énergie : étude des objets tels que les trous noirs, naines blanches.. qui produisent une énergie considérable.
- des galaxies : étude des galaxies proches et de la notre
- des méthodes de gestion et d'exploitation de l'information (en relation avec le CDS)

L'observatoire est aussi impliqué dans l'**enseignement**. En effet, il a pour mission d'assurer la formations des étudiants et la diffusion des connaissances. De plus, il délivre la spécialité astrophysique de la mention physique du master.

fig. 1 : Lunette d'observation de l'observatoire de Strasbourg

Présentation du CDS

Le Centre de Données astronomiques de Strasbourg est dédié à la collecte et la diffusion au niveau international de données astronomiques.

Ses missions sont de :

- collecter des informations sur des objets astronomiques de manière informatisé,
- évaluer et comparer les données ainsi collectées,
- diffuser les résultats à la communauté internationale,
- utiliser ces données pour la recherche.

Contexte logiciel

Le CDS héberge la base de données SIMBAD qui permet aux astronomes de facilement connaître les propriétés de base (coordonnées, magnitudes ..) de chacun des objets recensés dans un catalogue astronomique. Il est possible d'exécuter des requêtes sur SIMBAD directement depuis le site internet.

VizieR est un service qui permet de visualiser des astres classés sous forme de catalogues. Un catalogue peut correspondre à une mission spatiale particulière, à une région du ciel, ou encore à une étude scientifique. Ils sont saisis informatiquement par les documentalistes du CDS. Il existe près de 12 500 catalogues à l'heure actuelle dans VizieR. C'est un service qui connaît un fort succès avec près de 400 000 requêtes par jour en moyenne.

Aladin est un logiciel permettant d'afficher des images du ciel référencées en position et de superposer des catalogues de sources détectées dans ces champs, des objets astronomiques étudiés dans les publications scientifiques ou d'autres images obtenues par différents instruments. Ses fonctionnalités de navigation sont pensées comme un Google Map du ciel, pour faire une analogie avec les outils web courants. Il est développé en Java à l'observatoire de Strasbourg pour l'ensemble de la communauté.

C’est un projet de grande envergure qui a été créé en 1999 et qui ne cesse d’évoluer depuis. Il est développé par Pierre Fernique, Thomas Boch, Anaïs Oberto et François Bonnarel. Aladin est disponible sous forme d’application autonome, d’applet ou encore de canvas HTML5 (permet de dessiner à la volé via des scripts).

Saada est un logiciel développé en Java par Laurent Michel à l’observatoire de Strasbourg. Il permet de générer une base de données à partir d’un ensemble de fichiers FITS[1].

Un fichier au format FITS peut être une image un spectre un cube ou hypercube. Il est composé :

- d’un en-tête qui contient un ensemble de paires “mots-clés / valeur” permettant d’apporter des renseignements sur le contenu du fichier,
- d’une matrice de valeurs qui est interprétée grâce au header.

Une même base de données Saada peut être divisée en plusieurs collections. Typiquement, cela peut servir pour regrouper des entrés par catalogues. Saada recherche des mots-clés dans l’en-tête des fichiers FITS et crée la base en conséquence.

Une fois la base de données créée, le logiciel est capable de l’exposer via une interface web, et surtout il met à notre disposition un service TAP[2]. Il s’agit d’un protocole qui définit des normes permettant d'accéder à des tables de données. Cela peut concerner des catalogues astronomiques ou bien une

[1] Flexible Image Transport System, format de fichier pouvant contenir des spectres, images, cubes ...
[2] Table Access Protocol

base de données plus générale. L'interrogation d'un service de ce type passe par le protocole HTTP[3] et se fait en ADQL, un langage de reqête spécialisé pour l'astronomie et basé sur SQL.

Exemple d'une requête à un service TAP via HTTP POST :

HTTP POST http://example.com/tap/sync
REQUEST=doQuery
LANG=ADQL
QUERY=SELECT * FROM magnitudes as m where m.r>=10 and m.r<=16

Présentation de l'IVOA

L'IVOA (International Virtual Observatory Alliance) a été créé en 2002 avec pour but de standardiser les échanges, les outils d'interrogation, les systèmes d'extraction de l'information, de manière à globaliser les données et plus généralement l'information en astronomie, au niveau international. C'est dans ce but qu'est créé le modèle de donnée ObsCore qui définit les règles à respectée pour structurer les descriptions d'un ensemble de données hétérogènes (images, cubes, spectres…)

L'IVOA est également à l'origine du langage ADQL (Astronomical Data Query Langage) qui est basé sur du SQL, mais qui apporte quelques fonctionnalités supplémentaires.

[3] Hypertext Transfer Protocol

Le service informatique à l'observatoire

Le service informatique à l'observatoire est composé d'une dizaine de personnes reparties dans différents bâtiments et bureaux. Il dispose aussi de quelques baies de serveurs permettant d'héberger ses principaux services. Des copies miroir sont déployées dans plusieurs autres pays : USA, Inde, Japon, Russie, Espagne...etc.

Ma mission

Ma mission consiste à offrir un accès direct et simplifié aux images liées aux sources des catalogues VizieR. L'objectif est d'avoir à la fin du stage une interface exploitable pour accéder à ces images. Cela correspond à une réelle demande formulée par le conseil du CDS. Le conseil scientifique du CDS, nommé par l'INSU (Institut national des sciences de l'univers du CNRS) est composé de scientifiques français et étrangers extérieurs à l'observatoire. Il formule des avis et des recommandations sur les activités du centre.

Travail réalisé

Chargement de la table ObsCore

Introduction

La première partie de mon stage consiste à charger une table dans une base de donnée PostgreSQL. Cette table doit être conforme à un modèle de donnée reconnu dans le monde de l'astronomie : ObsCore Data Model.

Pour ce faire, je dispose du logiciel pgAdmin III qui permet d'interfacer la base PostgreSQL et d'exécuter des requêtes dans un environnement dédié. J'ai ajouté à Eclipse le plug-in PyDev pour me permettre de développer des scripts Python dans un environnement complet, pratique et familier.

Description des attributs du modèle obscore

nom de la colonne	type	unité	description
dataproduct_type	VARCHAR		le type de la donnée (image, cube…)
calib_level	INTEGER		niveau de calibration (0, 1, 2, 3)
obs_collection	VARCHAR		collection à laquelle la donnée appartient
obs_id	VARCHAR		identifiant unique de l'observation
obs_publisher_did	VARCHAR		l'identifiant coté éditeur de la collection
access_url	GLOB		url permettant de télécharger la donnée

access_format	VARCHAR		le format de la donnée
access_estsize	INTEGER	kb	la taille en kilobytes
target_name	VARCHAR		le nom de la cible de l'observation
s_ra	DOUBLE	degrés	l'ascension droite par rapport au centre
s_dec	DOUBLE	degrés	la déclinaison par rapport au centre
s_fov	DOUBLE	degrés	la taille de la région couverte par la donnée
s_region	REGION	degrés	le contour de la région couverte
s_resolution	DOUBLE	arcsec	résolution spatiale
t_min	DOUBLE	date	date départ d'observation en MJD
t_max	DOUBLE	date	date fin d'observation en MJD
t_exptime	DOUBLE	s	le temps total d'exposition de l'observation
t_resolution	DOUBLE	s	résolution temporelle
em_min	DOUBLE	m	la longueur d'onde minimal de l'observation
em_max	DOUBLE	m	la longueur d'onde maximale de l'observation
em_res_power	DOUBLE		résolution spectrale
o_ucd	VARCHAR		tag sémantique identifiant la quantité observée
pol_states	VARCHAR		liste des états de polarisation
facility_name	VARCHAR		le nom du télescope
instrument_name	VARCHAR		le nom de l'instrument de mesure (camera..)

Téléchargement des fichiers FITS

Avant de me plonger dans le gros du travail, il faut que je télécharge un ensemble de fichiers FITS (image ou cube). Pour cela, on me donne une liste des catalogues VizieR.

Je récupère les fichiers distants grâce à la commande :

wget -r -nd [url du catalogue par FTP]

wget -r -nd ftp://cdarc.u-strasbg.fr/pub/cats/V/127A/fits/*.fits

L'option *-r* permet de télécharger un répertoire, tandis que l'option *-nd* (no directory) demande à *wget* de ne pas recréer toute l'arborescence présente sur le serveur.

Je fais cela pour une centaine de catalogues, en vérifiant à chaque fois :

- l'intégrité d'un fichier récupéré par catalogue,
- s' il possède bien les mots-clefs WCS[4] pour le calcul des coordonnées astronomiques.

A la fin de cette opération, nous obtenons un total de 61 catalogues intéressants avec environ 10 000 images ou cubes FITS attachés.

[4] World Cordinate System

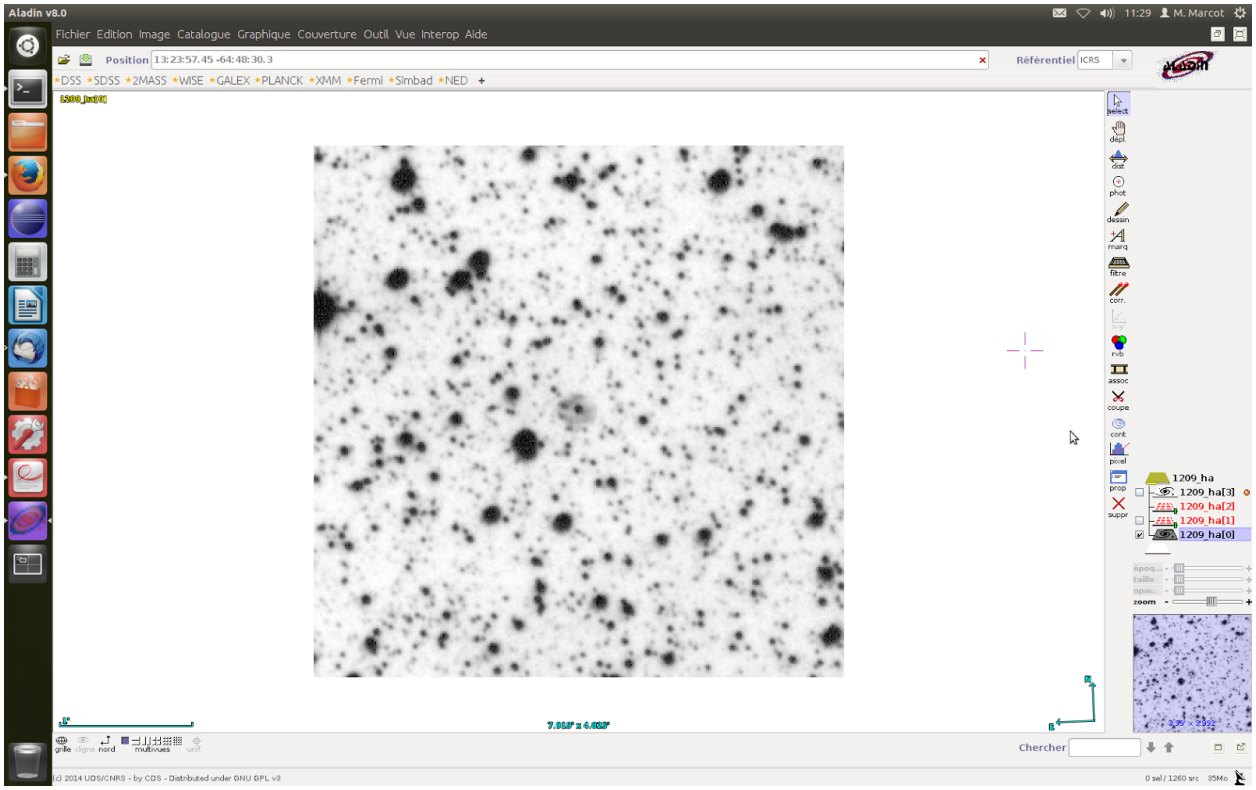


fig. 2 : un fichier FITS ouvert avec Aladin (capture d'écran)

Utilisation de Saada

D'abord, il faut que je me familiarise avec Saada. Je me met donc à la lecture de la documentation de ce logiciel. Je reçois aussi des explications et des démonstrations du développeur principal de Saada.

Je commence par créer des collections, une collection correspondant à un catalogue VizieR d'objets astronomiques. Il va donc falloir que j'en crée 61. Conscient de la fastidiosité de ce travail avec l'interface graphique, nous décidons d'utiliser la commande *ant* couplé à un fichier descriptif XML.
Exemple de commande :

ant -Dcollection_name *collec123* -Ddir_name *directory123*

La commande ant remplace les variables collection_name et dir_name dans le fichier build.xml et ainsi exécute les commandes Saada (création et chargement) pour chacune des collections. Je crée alors un script Python qui va exécuter à la chaîne ces 61 commandes.

Le seconde partie du travail avec Saada consiste à analyser la sortie, c'est à dire la base de données produite. Elle comporte plus de 500 tables. Deux tables Saada correspondent à 1 catalogue VIzieR car le logiciel distingue collection et classe. Le but est donc de peupler la table ObsCore à l'aide de ces données. Logiquement ce serait au logiciel Saada de réaliser cette opération, mais celui-ci étant encore en cours de développement, et la fonctionnalité n'étant pas encore implémentée. Je me vois effectuer un travail de défrichage en cherchant un ensemble de règles afin de transposer correctement ces données dans la table ObsCore. Ces règles seront bien entendu reprises pour le développement de cette fonctionnalité dans Saada.

Je vais détailler l'opération pour quelques colonnes ObsCore qui me semblent intéressantes et représentatives.

La colonne dataproduct_type

Ce champ ObsCore qui contient le type de la donnée peut prendre 2 valeurs différentes dans notre cas : image ou cube. Un cube peut être représenté mentalement par une succession d'images, où chaque image serait graduée sur un troisième axe mesurant la longueur d'onde par exemple. Ainsi, il est possible dans un seul fichier de connaître, par exemple, l'intensité lumineuse à un endroit précis du ciel découpé par tranches de longueur d'onde.

Pour distinguer un cube d'une image, il suffit de compter le nombre d'axes à l'aide d'une requête SQL :

```
SELECT name_class
FROM saada_metaclass_image
WHERE name_origin LIKE 'NAXIS_'
```

GROUP BY name_class
HAVING count(*) > 2;

Cette requête nous retourne la liste de toutes les classes qui ont plus de 2 axes, c'est à dire la liste de tous les cubes ou hypercubes.

Il suffit ensuite d'effectuer un parcours de tous les enregistrements dans la table obscore, et d'écrire dans le champ dataproduct_type 'cube' si l'image appartient à la classe présente dans la liste, ou 'image' sinon.

La colonne t_min

La colonne t_min qui correspond à la date de début d'observation en MJD (Modified Julian Day), il s'agit d'un décompte de jours depuis le 17 novembre 1858 utilisé principalement dans le monde de l'astronomie.

Ce champ s'est montré particulièrement difficile à traiter dans la mesure où les mots-clés dans les header FITS ne sont pas standardisés. Il faut donc :

- chercher dans la base Saada un ensemble de mots-clés possible, (_date, _jd, _mdj et _date_obs)
- parser pour chaque résultat le contenu de la colonne (le format de date peut être différent à chaque fois),
- convertir la sortie du parseur en MJD,
- insérer le résultat dans notre table ObsCore.

Les dates dans la base Saada sont stockées sous forme de chaîne de caractère et non pas au format DATE. Pour effectuer la conversion d'un masque standard de type MM-JJ-YYYY à un objet python de type date facilement manipulable, j'utilise un package python 'dateutil' qui contient notamment un

parser permettant de convertir une chaîne de caractère avec un masque couramment utilisé en un objet python de type DATE. Seulement, le parser ne sait pas faire la différence entre un masque JJ-MM-YY et MM-JJ-YY lorsque les jours sont inférieurs à 13. Cependant, il est possible de lui passer en paramètre un booléen permettant de privilégier une situation ou une autre. Pour résoudre ce problème, j'ai dû regrouper les chaînes de caractères par catalogue. Si une date d'un catalogue donné contient pour ses 2 premiers chiffres un nombre supérieur à 12, alors toutes les dates de ce catalogue correspondent au masque JJ-MM-YYYY, sinon on considère que le masque à privilégier est MM-JJ-YYYY.

Les colonnes em_min et em_max

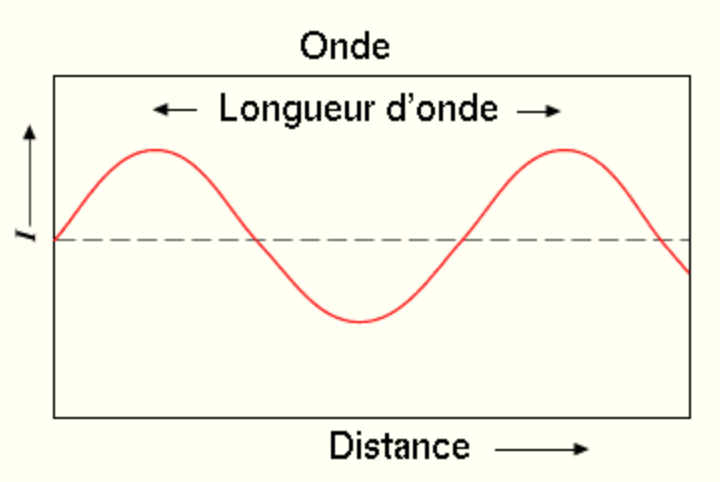


fig. 3 : représentation d'une longueur d'onde : en abscisse la distance, et en ordonnée une quantité variable (magnitude, amplitude, pression…)

Une observation est faite sur un intervalle de longueur d'onde : em_min correspond à la borne minimale de cet intervalle, et em_max à sa borne maximale. On mesure donc un flux entre 2 valeurs de longueur d'onde.

Pour rappel, une onde est une oscillation d'une source qui se propage dans un milieu environnant la source. La période est la durée d'une oscillation. La longueur d'onde est la distance parcourue par

l'onde pendant une période et séparant deux pic. La fréquence correspond au nombre de périodes par seconde. Un rayonnement se déplaçant toujours à la vitesse de la lumière, on peut donc dire que la fréquence est inversement proportionnelle à la longueur d'onde.

Longueur d'onde = vitesse de la lumière / fréquence

ou

fréquence = vitesse de la lumière / longueur d'onde

La base de données associée au logiciel Saada contient un ensemble de données permettant de calculer la fréquence :

- naxis : le nombre de pixels sur l'axe
- cdelt : taille du pixel en unité physique (selon unité de l'axe par ex . arcsec)
- crpix : position de référence de l'image ou du cube en pixels
- crval : position de référence en coordonnées physiques
- ctype : le type de la donnée. Ex: FREQ pour fréquence

```
em_min = crval3 + (1-crpix3) * cdelt3
em_max = crval3 + (naxis3-crpix3) * cdelt3

# conversion frequence => longueur d'onde :
if em_min != 0 :
    em_min = c/em_min
if em_max != 0 :
    em_max = c/em_max
```

fig. 4 : Calcul de la fréquence et conversion en longueur d'onde (capture d'écran).

J'ai présenté ici un des cas les plus simples, car suivant la valeur du ctype on peut se trouver confronté à des calculs plus compliqués.

Développement d'un interface en Java

Introduction

La seconde partie de mon stage consiste en le développement d'un plug-in nommé ObsTAPQuery qui viendra se greffer sur le logiciel Aladin. Il devra permettre à l'utilisateur d'interroger la table ObsCore à l'aide d'un interface graphique simple et intuitif. Le résultat de cette interrogation apparaîtra dans Aladin où l'utilisateur pourra alors interagir avec sa sélection.

Analyse des besoins

La première étape dans ce travail consiste à analyser les besoins. Qu'est ce que le CDS et plus globalement la communauté astronomique souhaite réaliser à travers ce plug-in ? Autrement dit, que doit-il être capable de faire ?

Les fonctionnalités suivantes sont attendues :

- Sélectionner ou non certains champs ObsCore
- Ajouter et supprimer des contraintes sur ces champs
- Générer et afficher la requête ADQL correspondante
- Pouvoir modifier directement la requête ainsi générée
- Se connecter à un service TAP
- Exécuter la requête dans Aladin

L'objectif est d'arriver à avoir un plug-in qui exécute correctement les fonctions de bases et qui soit suffisamment bien structuré pour pouvoir lui ajouter facilement des fonctionnalités supplémentaires.

Réflexion sur l’interface

La seconde étape revient à se poser la question suivante :
Comment proposer une vue à l’utilisateur qui permet d’imbriquer intuitivement les fonctionnalités et les besoins attendus ?

L’enjeu est d’avoir une vue simple et compartimentée qui permet à l’utilisateur (novice ou avancé) de s’y retrouver facilement sans avoir besoin de lire la documentation au préalable. Car rares sont ceux prêts à lire des dizaines de pages avant d’utiliser un logiciel. L’aspect intuitif est donc particulièrement important.

Pour cela nous nous sommes mis d’accord sur une interface qui comporte trois parties distinctes :

- Une pour le choix des colonnes obscore à sélectionner
- Une pour le choix des contraintes
- Une dernière présentant la requête correspondante avec la possibilité de la modifier avant de l'exécuter.

fig. 5 : Schéma après réflexion sur l'interface

Environnement de développement

La troisième étape consiste à se créer un environnement de travail efficace en terme de productivité et de fonctionnalités.

Pour ce faire je décide d'utiliser le logiciel Eclipse, que je commence à bien connaître. J'ai la chance de pouvoir travailler sur Linux Ubuntu, un système d'exploitation qui m'est familier et que je maîtrise convenablement.

Pour parfaire cet environnement de travail, je décide d'utiliser le gestionnaire de version Git. Cela me permet de :

- mieux gérer l'évolution du contenu de l'arborescence du projet
- tester de nouvelles fonctionnalités facilement (système de branches)
- pouvoir tout aussi facilement revenir en arrière en cas d'erreur.

De plus, couplé à la forge GitHub, je peux :

- diffuser le code de façon simple
- travailler sur différents postes sans avoir la contrainte du support amovible
- imaginer du travail collaboratif.

Le développement de la version de base

Nous allons d'abord nous affairer à produire une version basique mais fonctionnelle d'ObsTAPQuery (prototype). Puis, nous verrons les améliorations apportées dans un second temps.

S'agissant d'un projet conséquent, je décide d'implémenter le pattern MVC[5] afin d'imposer une structure claire.

Les classes sont donc divisées en 3 parties :

- Le Modèle contenant l'ensemble des classes métiers, mais aussi des classes techniques (pour la connexion à la base de donnée par exemple)
- La Vue contenant les composants Swing[6], et tout ce qui est en lien direct avec l'interface homme machine.
- Les Contrôleurs qui permettent de gérer les évènements utilisateurs.

Chacune des ces parties correspond à un package, créé encore une fois dans un souci structurel.

[5] Modèle Vue Contrôleur
[6] Il s'agit d'une bibliothèque graphique pour Java

La vue

Je décide de commencer le développement d'ObsTAPQuery par la vue. Étant donné que le principal objectif du plug-in est de proposer une interface, le gros du travail se situe ici.

La difficulté consiste à imbriquer les blocs entre eux et à les positionner correctement. Pour cela j'ai procédé par tâtonnement, en coloriant chaque panneau d'une couleur différente, et en essayant de les positionner correctement. C'est une tâche fastidieuse car chaque gestionnaire de position a ses subtilités.

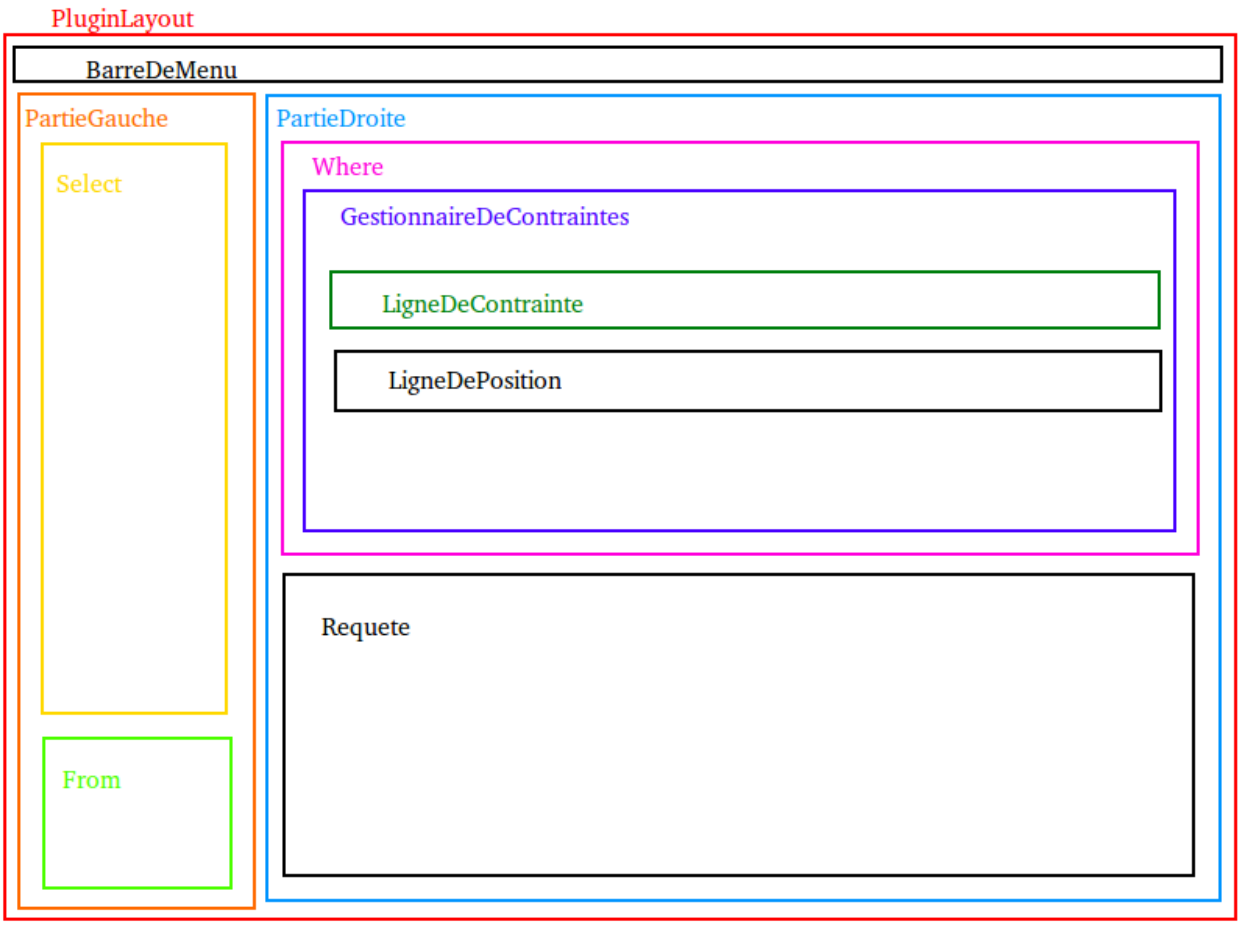


fig. 6 : encapsulation des JPanels composant la vue

Le modèle

Une fois la vue terminée, je m'attaque au modèle qui lui s'avère relativement simple. En effet, il se résume au chargement et à la gestion des différents champs ObsCore.

Je décide de créer 3 classes :

- UnChampObscore.java : modélise un unique champs ObsCore avec ses attributs (nom, type de donnée, unité…)
- ColonnesObscore.java : qui charge depuis le service TAP l'ensemble des noms de colonnes ObsCore et crée ainsi une liste de type *UnChampsObscore*
- GenerateurRequete.java : cette classe s'occupe de générer la requête ADQL qui correspond à l'ensemble des champs sélectionnés par l'utilisateur, en prenant en compte les éventuelles contraintes.

Ces trois classes permettent d'avoir une version basique mais fonctionnelle de notre plug-in, avec la possibilité de l'améliorer facilement par la suite.

Les contrôleurs

Pour relier la vue au modèle, il faut réaliser des auditeurs qui vont attendre un évènement utilisateur (typiquement un clic sur un bouton). Pour cela, je crée une classe auditrice par évènement que je souhaite capturer, c'est un choix qui me semble essentiel pour des projets de cet ampleur. En effet, l'utilisation de classe anonyme viendrai compliquer le code et rendre sa lecture plus difficile. Le fait d'éclater les différentes tâches en classes va permettre de faire évoluer le code plus facilement.

Le principal défi ici est la manipulation des références pour arriver à appeler des méthodes dans des blocs parents. Pour résoudre ce problème je récupère le bouton à l'origine de l'évènement, puis j'appelle la méthode *getParent()* jusqu'à tomber sur le bloc qui m'intéresse. Cela est relativement

pratique car je n'ai pas besoin de passer de référence aux auditeurs, mais en contre partie cela impose une certaine rigidité au code.

Une fonctionnalité intéressante

La fonctionnalité qui mérite d'être expliquée concerne la façon dont notre plug-in fait pour interagir avec Aladin. Pour cela, notre classe principale hérite d'une classe nommée AladinPlugin. Cette classe dispose d'un attribut qui est une référence à l'instance d'Aladin en train de se dérouler. Ainsi, il devient possible en manipulant cette référence de dialoguer entre les deux applications. En particulier via la méthode *execCommande(String str)* qui permet d'exécuter une commande script dans Aladin. Elle se présente de la forme suivante :
get_File("*hostname:port/DBname/tap/sync?**REQUEST**=doQuery&**LANG**=ADQL&**QUERY**=S ELECT%20*%20FROM%20ivoa.ObsCore*")
La requête ADQL est encodée pour pouvoir passer dans un URL.

Quelques figures (copie d'écran) pour mieux comprendre

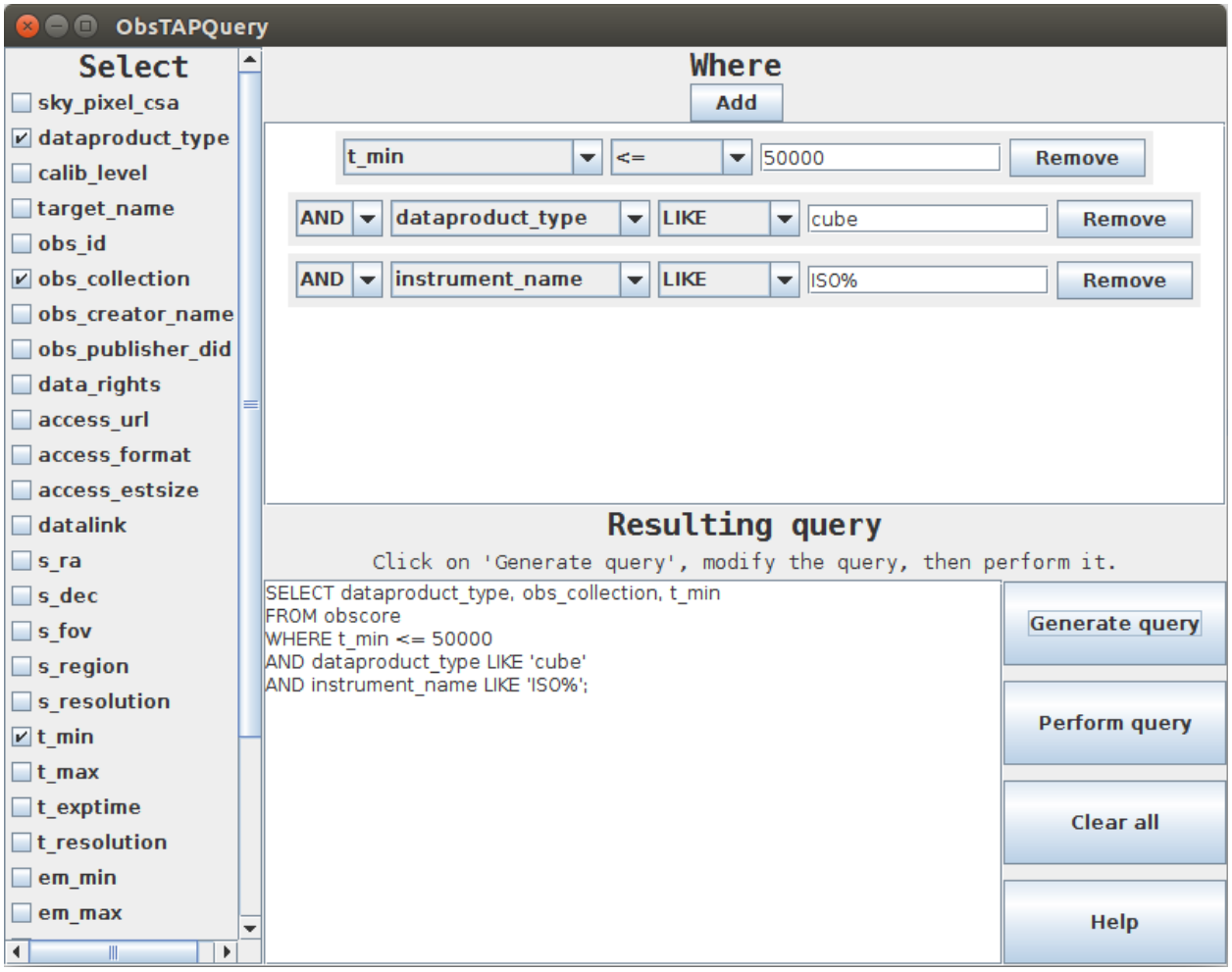


fig. 7 : Version basique mais fonctionnelle d'ObsTAPQuery (capture d'écran)

Et lorsque l'on clique sur le JButton "Perform query" la requête est envoyée à Aladin qui va s'occuper d'afficher le résultat.

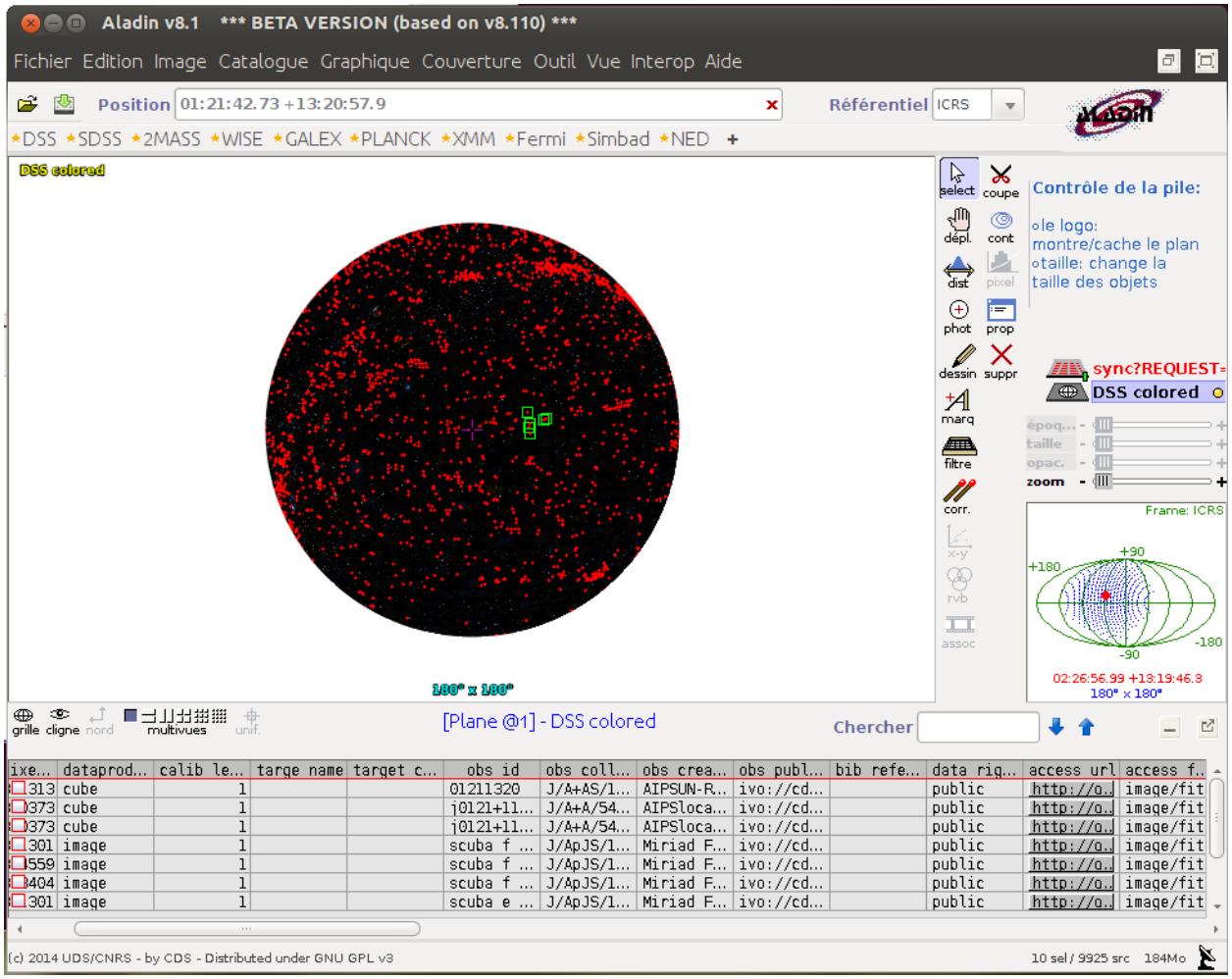


fig. 8 : Résultat de la requête dans Aladin (capture d'écran)

Chaque point rouge correspond à une entrée de la table ObsCore, c'est à dire un fichier FITS. Ce fichier peut être affiché directement dans Aladin en cliquant sur le bouton présent dans le champs access_url.

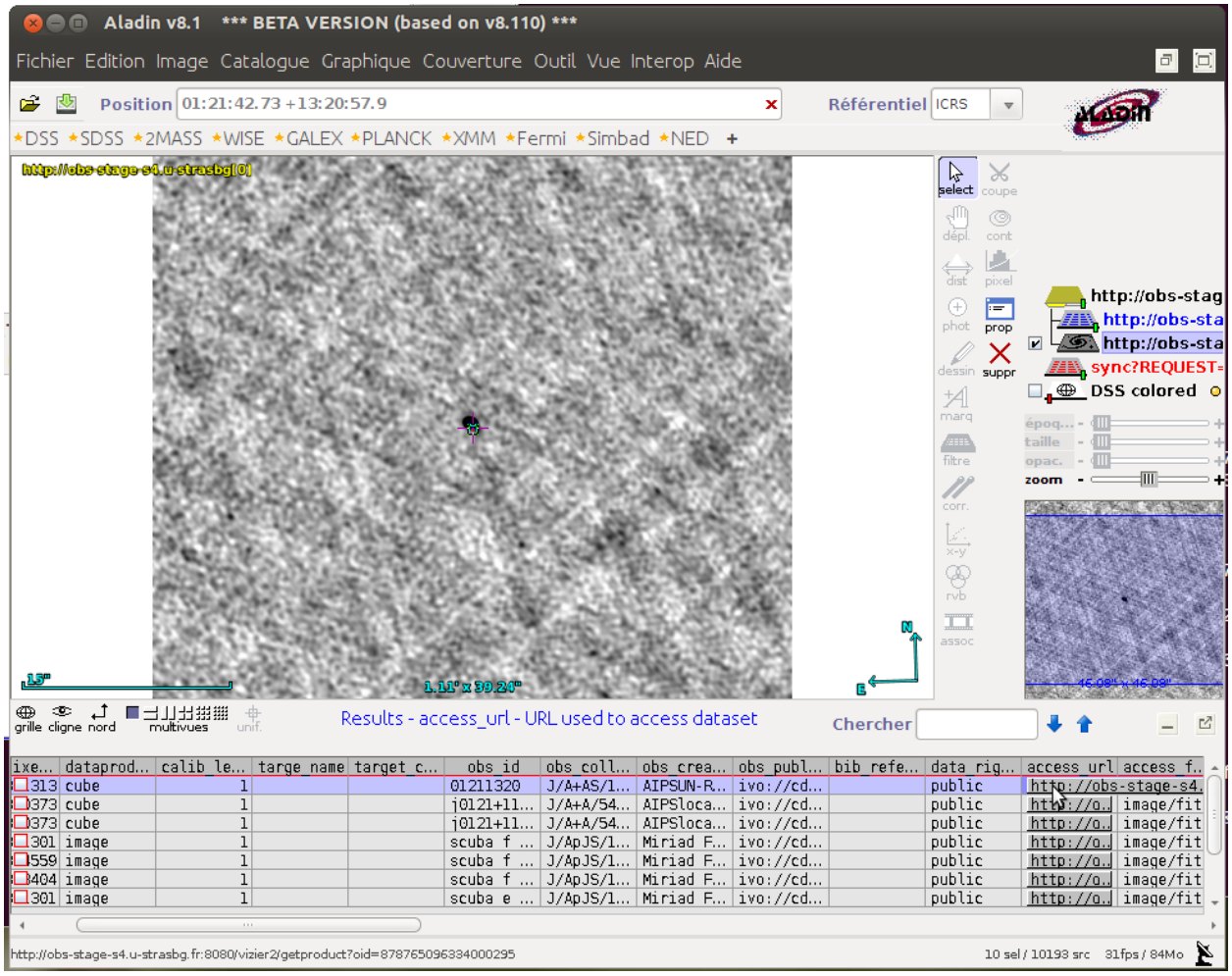


fig. 9 : Résultat lorsque l'on clique sur le lien présent dans access_url (capture d'écran)

Tout cela m'a permis de livrer un premier prototype fonctionnel du plug-in mais le but n'est pas de s'arrêter là. En effet, il faut améliorer l'application pour proposer une version plus complète et intuitive.

Les améliorations du plug-in

La récupération des colonnes depuis le service TAP distant

Cela représente une avancée majeure pour le plug-in, car une table ObsCore peut contenir en plus des colonnes requises par le standard ObsTAP, des colonnes additionnelles laissées au choix de l'utilisateur. De plus nous verrons ci-après que nous pouvons récupérer des tables non contraintes par le standard ObsCore. Il est donc important que notre application s'adapte et les prennent en compte pour la génération de la requête ADQL. Cela est entièrement flexible et permet même d'interroger une table générale avec des colonnes quelconques. La seule restriction à cela est que la base de donnée respecte le protocole TAP qui impose la présence d'un TAP schéma décrivant les différentes tables et leur contenu.

Pour réaliser cela, il faut interroger le service TAP distant à l'aide d'une requête ADQL qui va sélectionner les colonnes qui nous intéressent dans le TAP schéma. J'utilise la méthode GET du protocole HTTP :
hostname:port/DBname/tap//sync?REQUEST=doQuery&LANG=ADQL&QUERY=SELECT+*+FROM+TAP_SCHEMA.columns
Cette requête nous retourne un résultat sous forme de VOTable, c'est un standard IVOA basé sur XML qui permet de représenter des données sous forme de table. La VOTable doit ensuite être parsée grâce à Savot[7] pour pouvoir extraire le résultat.

Le procédé consiste à :

- vérifier la présence d'un TAP schéma,
- formuler la requête ADQL
- extraire le nom des colonnes depuis la VOTable résultante,
- afficher les noms de colonnes et permettre à l'utilisateur de travailler avec.

[7] Projet également maintenu par le CDS

La réalisation d'un from dynamique

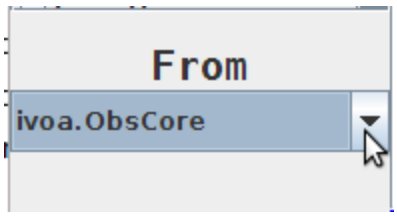


fig. 10 : From dynamique (capture d'écran)

Cette amélioration vient compléter la précédente dans la mesure où il sera maintenant possible de choisir la table qu'on souhaite interroger parmi celles qui sont exposées grâce au service TAP. La principale difficulté consiste à mettre à jour dynamiquement les colonnes disponibles (à la sélection et pour les contraintes) en fonction de la table choisie. De plus, il faut revoir l'interface graphique afin d'implémenter cette fonctionnalité. Il me semble judicieux de placer la combo box de sélection des tables dans l'alignement du *select*, c'est à dire en bas à gauche afin d'optimiser la place et de garder une bonne visibilité.

Les étapes sont les suivantes :

- récupération du nom de toutes les tables présentes grâce au TAP schéma,
- récupération du nom des colonnes qui correspondent à la table sélectionnée,
- affichages dynamique de ces colonnes dans la partie select

Pouvoir changer de service TAP

Il est essentiel que notre plug-in ne se limite pas à un seul service TAP, mais que l'utilisateur puisse choisir le service qu'il souhaite interroger. Pour réaliser cela il faut que notre plug-in et le service TAP interrogé respectent tous les 2 le protocole TAP à la lettre afin de permettre l'interopérabilité.

Il me semble judicieux d'intégrer au programme une barre de menu afin de pouvoir rajouter des fonctionnalités sans bouleverser l'interface graphique.

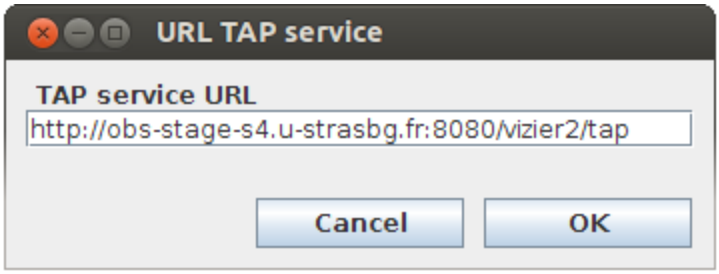


fig. 11 : fenêtre de changement de service TAP (capture d'écran)

La recherche par position

La recherche par position dans le ciel est aussi un des points central de notre application. En effet, une des fonctionnalité attendue est de pouvoir faire facilement des recherches dans le ciel autour d'un point du ciel connu grâce à ses coordonnées astronomiques.
Pour cela l'utilisateur est invité à entrer :

- l'ascension droite du centre de la zone de recherche, elle est l'équivalent sur la sphère céleste de la longitude terrestre, RA pour Right Ascension
- la déclinaison du centre de la zone de recherche, elle est l'équivalent de la latitude terrestre projetée sur la sphère céleste, DEC pour declination
- le rayon de la zone de recherche souhaité en degré, minute ou seconde, Radius

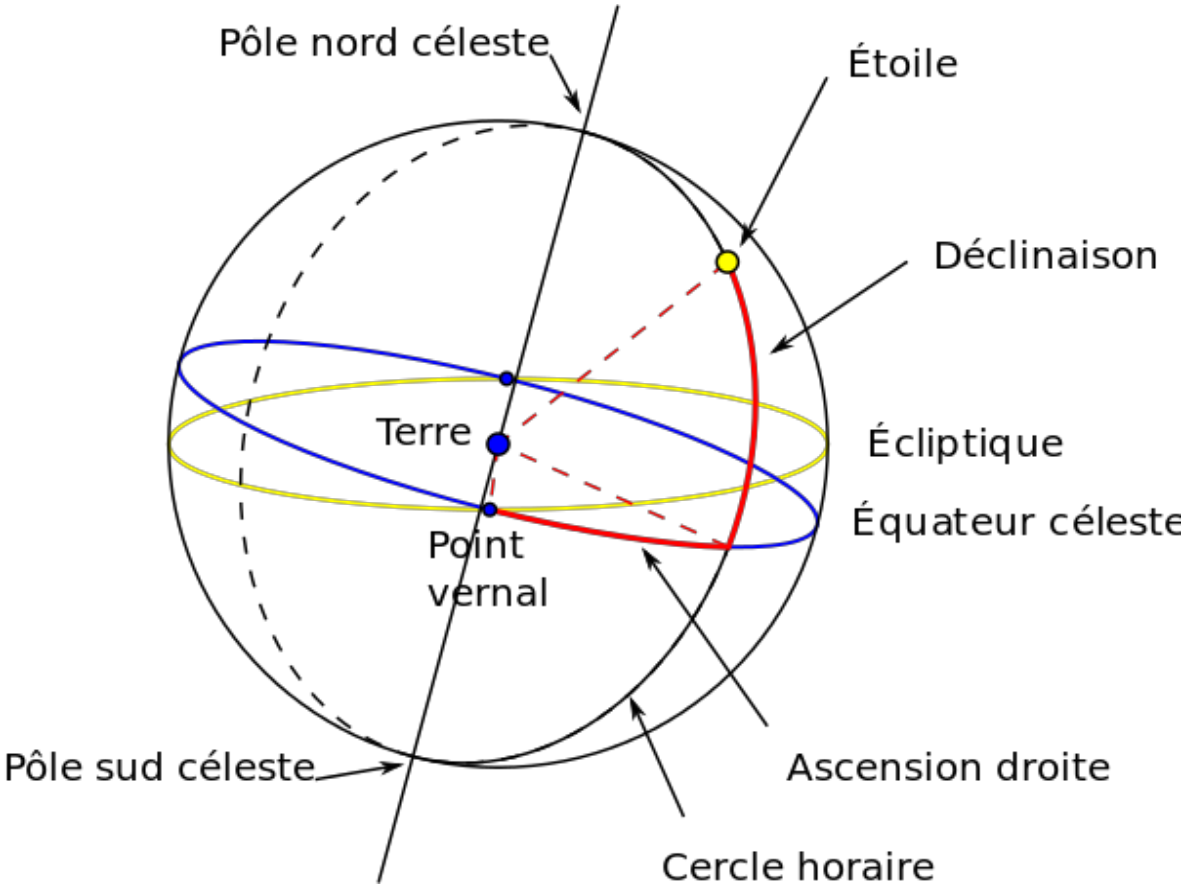


fig. 12 : Coordonnées équatoriales (source Wikipédia)

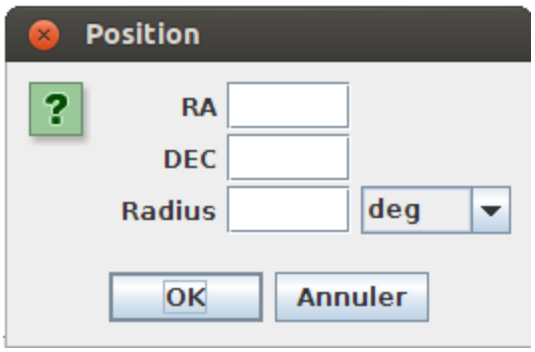


fig. 13 : Fenêtre de saisie des coordonnées (capture d"écran)

Une fois la saisie effectuée, une nouvelle ligne est créée dans le gestionnaire de contraintes résumant la position. Il est possible de rentrer plusieurs positions et de les combiner (opérateur OU), ou de ne chercher qu'à leur croisement (opérateur ET).

Dans la requête ADQL le résultat est généré sous la forme suivante :
WHERE CONTAINS(POINT('ICRS', ivoa.ObsCore.s_ra, ivoa.ObsCore.s_dec),
CIRCLE('ICRS',1.0, 2.0, 3.0)) = 1;

La fonction POINT définit un point grâce à ses coordonnées (issues du centre de l'image ou du cube).
La fonction CIRCLE définit un cercle sur la sphère céleste par les coordonnées de son centre et son rayon.
La méthode CONTAINS cherche si le point de l'enregistrement est contenu dans le cercle d'ascension droite 1.0, de déclinaison 2.0 et de rayon 3.0 degrés. Elle renvoie 1 si c'est vrai, 0 sinon.

Sauvegarder et charger une requête

Il me semble important que le plug-in soit en mesure de sauvegarder puis de charger une requête. Cela dans un souci d'ergonomie pour que l'utilisateur n'ait pas à faire la séquence : copier, ouvrir un éditeur de texte, ouvrir un fichier, coller, sauvegarder le fichier. Mais simplement cliquer sur File > Save. Et de pouvoir la charger plus tard tout aussi facilement.

La mise à disposition d'exemples

Notre plug-in étant amené à être utilisé par une communauté variée d'astronomes, d'informaticiens ou de curieux, il est judicieux de proposer des exemples de requêtes ADQL permettant aux non initiés de faire leurs premiers pas. Pour ne pas décourager l'utilisateur il faut garder une bonne visibilité sur ce

bouton et donc ne pas le cacher dans la barre de menu. Ainsi il faut quelque peu bouleverser l'interface et créer un panneau de boutons latéral dans la partie where.

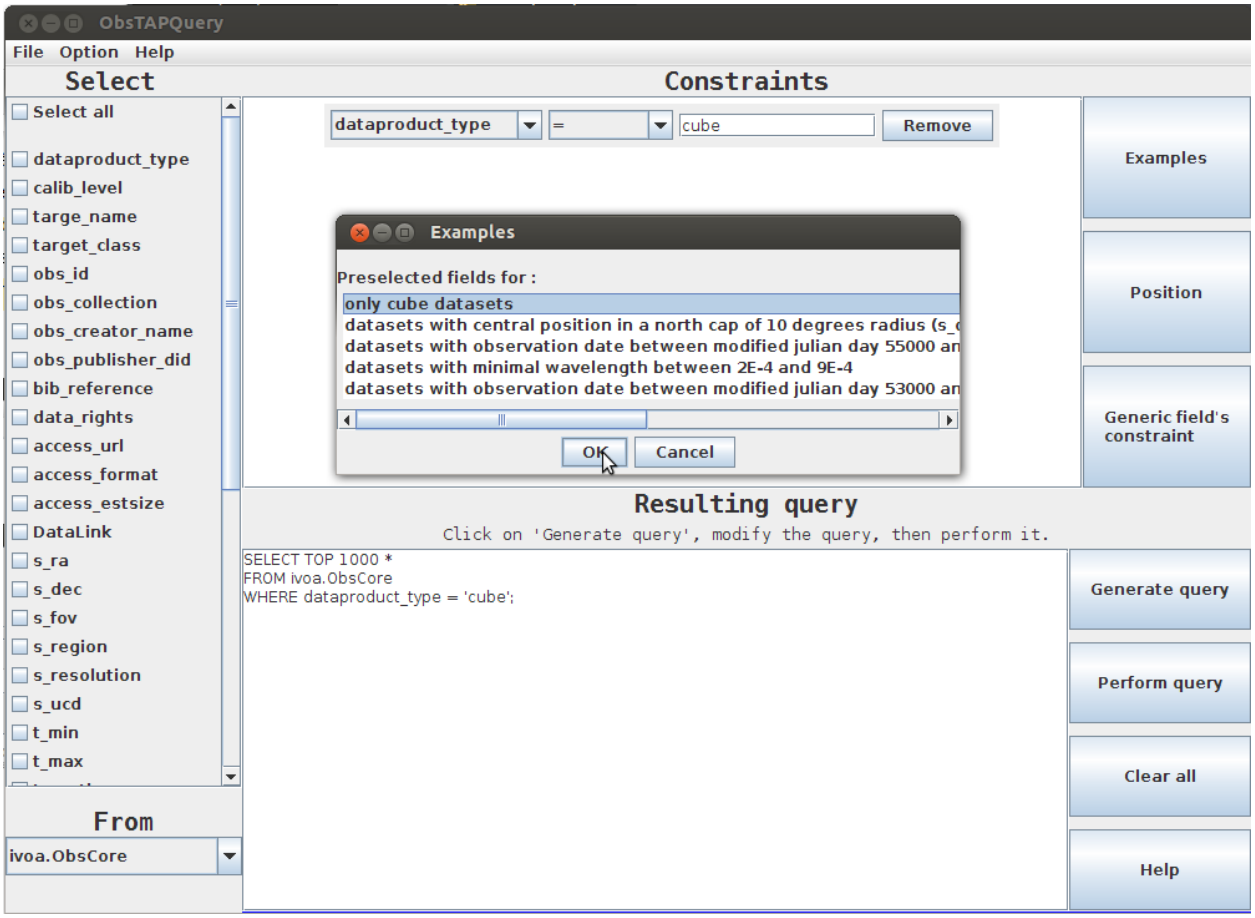


fig. 14 : Nouvelle interface, intégration des exemples de requêtes (capture d'écran)

Conclusion

Sur le plan professionnel, les apports de ce stages ont été évidents, et m'ont permis de vérifier que c'est bien par l'expérience que l'on apprend et assimile le mieux des connaissances. Depuis la fin du stage je me sens plus à l'aise en programmation, j'ai rencontré des problèmes et des difficultés qui m'ont fait apprendre de nouvelles connaissances informatiques, des nouvelles techniques et façon de faire. J'ai aussi appris à m'adapter à un environnement métier riche avec ses contraintes et ses bonnes pratiques. Je me suis rendu compte de l'importance de la structuration pour les gros projets : un code sans ligne directrice et qui part dans tous les sens est voué à l'échec. Tout doit être compartimenté, flexible et réutilisable. Je pense que la vision langage objet s'apprend beaucoup par l'expérience et que le DUT ne fourni qu'une base permettant d'évoluer par la suite.

Sur le plan personnel, l'intégration à l'observatoire s'est très bien passée, l'établissement étant habitué à recevoir et former des stagiaires cela n'a posé aucun problème au contraire. J'ai apprécié le fait que mon tuteur en entreprise soit vraiment impliqué dans le bon déroulement du stage et qu'en conséquence les tâches que j'ai réalisées étaient toutes intéressantes et cohérentes dans le contexte d'un projet scientifique réel.

Bibliographie

Sites web

Obs. astro, 18/06/14, http://astro.unistra.fr/

CDS, 19/06/14, http://cdsweb.u-strasbg.fr/

IVOA, 19/06/14, http://www.ivoa.net/index.html

Wikipédia, 18/06/14, http://fr.wikipedia.org/wiki/Observatoire_astronomique_de_Strasbourg

Saada, 23/06/14, http://amwdb.u-strasbg.fr/saada/

Wikipédia, 21/07/14, http://fr.wikipedia.org/wiki/D%C3%A9clinaison_%28astronomie%29