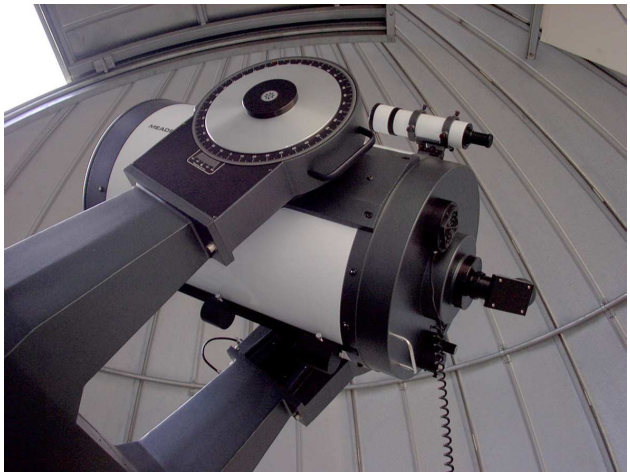
CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

GRENOBLE 1
UNIVERSITE
JOSEPH FOURIER
SCIENCES.TECHNOLOGIE.SANTÉ

LAOG
Laboratoire d'AstrOphysique de Grenoble

Conception et réalisation d'un logiciel de recherche de calibrateurs pour des observations interférométriques



Sylvain CETRE
Stage MAI3
Maître de stage : Gérard ZINS

"Celui qui s'oriente sur l'étoile ne se retourne pas."
Léonard de Vinci

Remerciements

Je tiens à remercier en premier mon maître de stage, Gérard Zins, pour m'avoir proposé ce stage. Sa gentillesse et sa disponibilité ont été d'un grand secours tout au long du travail effectué.

Je souhaite remercier aussi Gilles Duvert, le Directeur Technique du JMMC pour son aide sur certains sujets où sa compétence a toujours été un secours utile.

Merci à Alain Chelli, le Directeur du JMMC de m'avoir accueilli dans le laboratoire.

Je remercie tout particulièrement Laurence Gluck, Sylvain Lafrasse et Guillaume Mella, les membres du JMMC sur Grenoble qui n'ont jamais hésité à répondre à mes questions, et cela même quand leur travail ne leur en laissait peut-être pas le temps.

Je remercie aussi Xavier Delfosse pour son écoute et ses conseils sur les points difficiles de la partie scientifique du projet. Je remercie également Daniel Bonneau et Jean-Michel Clausse pour leur travail sur Search Calibrators.

Je remercie bien évidemment Thierry Stein, dont le stage s'est déroulé en même temps que le mien, et qui par sa bonne humeur a participé aussi au bon déroulement du stage.

Table des matières

Table des figures

1 Introduction

Ce document présente le stage de troisième année d'Institut Universitaire Professionnalisé en Génie Mathématique et Informatique spécialité Mathématiques Appliquées et Industrielles. Ce stage s'est déroulé du 5 Avril au 31 Août 2004 au Laboratoire d'Astrophysique de l'Observatoire de Grenoble. Le titre du stage est "Conception et réalisation d'un logiciel de recherche de calibrateurs pour des observations interférométriques".

Le document présente dans un premier temps le contexte dans lequel s'est déroulé le stage, notamment par la présentation du laboratoire et du "Jean Marie Mariotti Center". Il explique de manière plus précise le stage et les notions fondamentales nécessaires à la compréhension des termes employés. Ce rapport insiste aussi sur le travail déjà effectué par les membres du laboratoire, et présente l'environnement de développement propre au centre Mariotti.

Une deuxième partie présente la partie technique du stage, avec une explication du programme réalisé ainsi que les décisions prises pour arriver à des résultats corrects. Ce stage faisant appel à la notion d'interrogation de centres de données distants ainsi qu'à une notion d'analyse d'informations, il est présenté dans ce document des exemples d'informations et de résultats obtenus. Enfin, une version du code développé pendant ce stage est inséré en annexe.

2 Présentation

2.1 Le LAOG

Le Laboratoire d'Astrophysique de l'Observatoire de Grenoble est une UMR[1] de l'UJF[2] et du Centre National de la Recherche Scientifique. Créé en 1979, le laboratoire participe activement à l'enseignement et à la formation scientifique des étudiants de Grenoble.

FIG. 1 – Observatoire de Grenoble

Le personnel du LAOG est composé de 30 chercheurs et enseignants-chercheurs, de 20 ingénieurs, techniciens, administratifs, ainsi que d'une vingtaine de visiteurs, post-docs, stagiaires et thésitifs.

Le laboratoire travaille sur plusieurs disciplines en astronomie et physique des particules : les activités d'observation, de modélisation, théoriques et

[1]Unité Mixte de Recherche
[2]Université Joseph Fourier

en instrumentation. Les thèmes de recherche sont divers : il y est étudié les étoiles jeunes, les étoiles de très faible masse et les naines brunes, par exemple. Le LAOG s'est aussi concentré sur les observations à haute résolution angulaire comme l'optique adaptative, l'interférométrie, l'optique intégrée, etc... Le laboratoire est aussi présent au niveau international. Les chercheurs du LAOG partent régulièrement observer sur les plus grands télescopes mondiaux. Le laboratoire collabore avec de nombreux observatoires français et étrangers. Il est aussi en relation avec le monde industriel notamment pour la construction de pièces pour les instruments d'observation. Ces informations sont disponibles sur le site internet du laboratoire dont l'adresse est donnée dans la suite de la présentation.

Le LAOG est situé sur le domaine universitaire de Grenoble. Son adresse géographique est la suivante :

Laboratoire d'Astrophysique
Observatoire de Grenoble
414, Rue de la Piscine
Domaine Universitaire
Saint-Martin d'Hères

Le laboratoire peut être contacté à l'adresse postale :

Laboratoire d'Astrophysique
Observatoire de Grenoble
BP 53
F-38041 GRENOBLE Cédex 9
(France)

Téléphone :

04 76 51 48 77 ou par numéros directs
+33 4 76 51 48 77 (de l'étranger)

Fax :

04 76 44 88 21 (de France) +33 4 76 44 88 21 (de l'étranger)

De plus amples renseignements sont disponibles à l'adresse suivante :

http ://www-laog.obs.ujf-grenoble.fr/

2.2 Le JMMC

2.2.1 Ses missions

Le JMMC[3] a pour mission de structurer et d'encadrer le développement de logiciels pour préparer les observations interférométriques et exploiter les données qui en résultent. La devise du JMMC est :

"We Interfere Constructively"

Il est nécessaire en interférométrie de développer de nouveaux logiciels afin de :

1. préparer les observations.
2. manipuler les données.
3. interpréter les résultats en termes de modèles simples.
4. interpréter les résultats en termes de reconstruction d'images.

Le but du JMMC est donc de concevoir de nouveaux types d'instruments où il est pris en compte lors de la conception, les logiciels qui utiliseront ces instruments. Le centre garde également un axe recherche en aidant notamment à définir des modèles géométriques pour interpréter les résultats expérimentaux, ou en développant des algorithmes de reconstruction d'images.

Le JMMC fournit les logiciels, mais assure aussi la gestion documentaire des travaux effectués dans le cadre des recherches interférométriques du laboratoire. Il contribue aussi au stockage du grand nombre de données présent en astronomie. Enfin, le centre assure un rôle de formation en organisant des conférences scientifiques pour les étudiants et les investisseurs.

[3]Jean Marie Mariotti Center

Le JMMC est composé de deux pôles scientifiques qui fournissent des services à la communauté scientifique :

- le LAOG
- l'OCA[4].

et de dix laboratoires ayant une grande expertise en interférométrie :

1. CRAL (Observatoire de Lyon)
2. OPM (Observatoire de Meudon)
3. IRCOM (Université de Limoges)
4. LISE (Observatoire de Haute Provence)
5. LAOG (Observatoire de Grenoble)
6. OBX (Observatoire de Bordeaux)
7. OCA (Observatoire de la Côte d'Azur)
8. OMP (Observatoire Midi-Pyrenées)
9. ONERA (Paris)
10. UNSA (Université de Nice Sophia-Antipolis)

2.2.2 Le centre de coordination

Le centre de coordination s'occupe de gérer tous les projets du JMMC. Il est composé d'un directeur technique, d'un chef de projet et de plusieurs développeurs. Le stage présenté dans ce rapport s'est déroulé dans le cadre d'un des groupes d'études, recherche et développement que gère le centre de coordination (Catalogue de Calibrateurs). La figure suivante présente l'organigramme du JMMC

[4]Observatoire de la Côte d'Azur

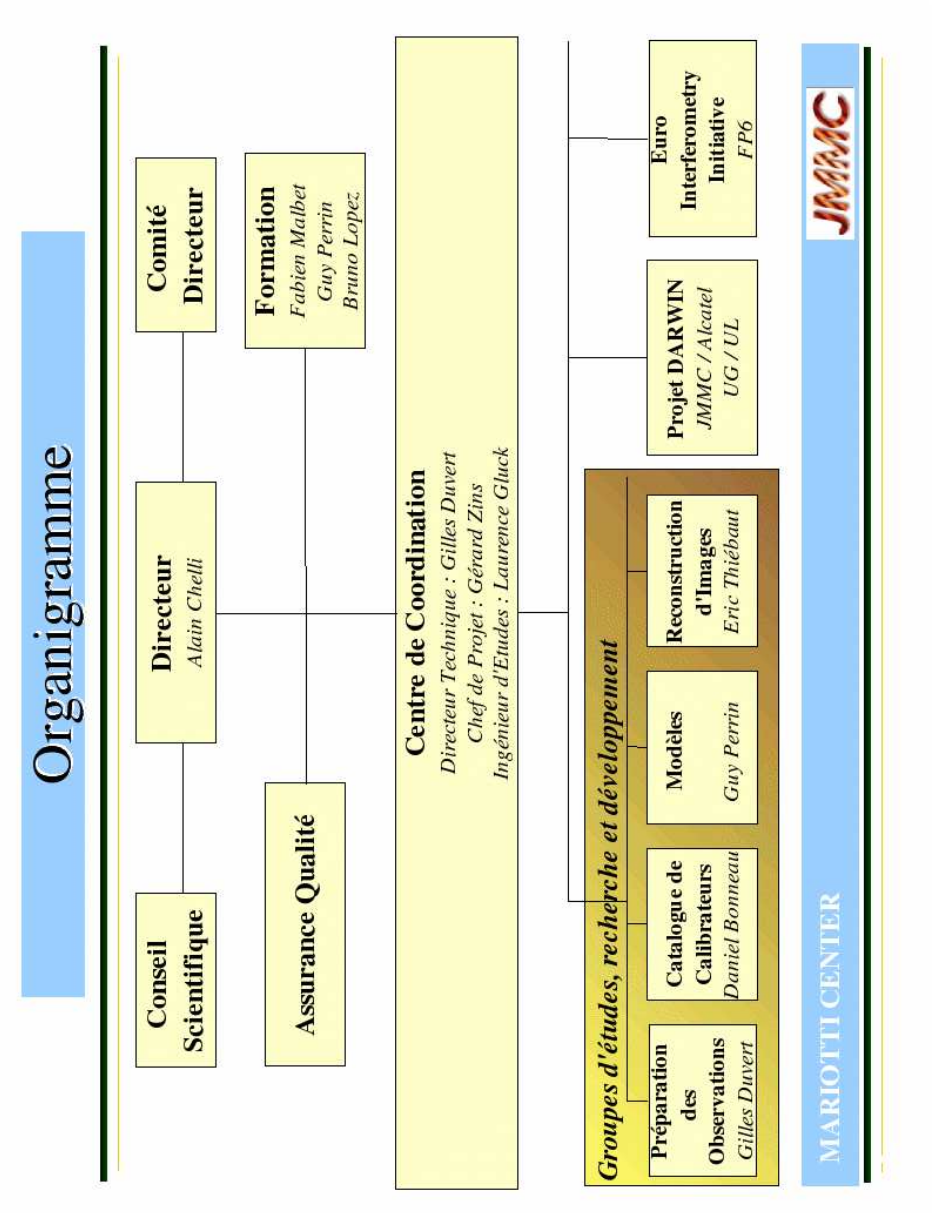


FIG. 2 – Organigramme du JMMC

2.3 Le sujet du stage

Le but du stage est de concevoir et de réaliser un logiciel "Orienté Objet" de recherche de calibrateurs pour des observations interférométriques. Ce logiciel sera utilisé par les astronomes pour la calibration des appareils de mesures lors des observations. Le principe est d'interroger des bases de données astronomiques dans l'optique de trouver, grâce à certaines contraintes des calibrateurs potentiels. Ce logiciel est une partie d'un grand logiciel d'astronomie disponible sur le web : ASPRO[5].

2.3.1 Les observations interférométriques

L'interférométrie est une technique d'observation permettant aux astronomes d'atteindre une résolution angulaire au delà de celle accessible avec des télescopes monolithiques (de l'ordre de la milliseconde d'arc dans le visible) et donc de discerner des détails très fins sur les objets observés. L'interférométrie est basée sur la nature ondulatoire de la lumière et la possibilité de faire interférer les ondes lumineuses. Le principe est de séparer la lumière émise par une source en deux systèmes d'ondes qui après avoir parcourues différents chemins sont recombinées. Cela revient à observer en stéréo les étoiles, donnant donc une information plus précise. L'un des plus célèbres interféromètre est le VLTI[6] situé à Paranal au Chili.

FIG. 3 – Vue du VLT au Chili

[5]Astronimical Software to PrepaRe Observations
[6]Very Large Telescope Interferometer

2.3.2 Les étoiles de calibrations

Afin de régler les appareils de mesures, il est nécessaire de calibrer ceux-ci à l'aide d'étoiles de calibration ou calibrateurs. Ces étoiles, dont on suppose connaître les principales caractérisques (magnitude, position exacte dans le ciel, etc...), vont permettre d'interpréter les données résultant de l'observation de l'objet de science. Le sujet du stage est de rechercher en particulier les objets dit "brillants", c'est à dire lumineux. Il existe certains calibrateurs "faible", donc peu lumineux qui seront intégrés par la suite dans le logiciel.

calibrateur = *Objet proche (coordonnées et propriétés) de l'objet de science qui va servir de référence lors de l'analyse et l'interprétation des données d'observations.*

2.3.3 Les observatoires virtuels et le CDS

Le projet d'observatoire virtuel est une initiative de la communauté astronomique internationale en Europe, Amérique du Sud et du Nord, Asie et Australie sous l'égide de l'IVOA[7].

Il a pour but d'offrir un accès électronique aux bases de données astronomiques de différents centres de stockage (comme le centre de Strasbourg par exemple) et des différents observatoires à travers le monde. Le deuxième objectif est d'offrir une analyse technique des données qui fournira des normes communes et des outils d'analyse novateurs.

Pour cela, le très grand nombre de données existantes doivent être archivées et être accessibles de manière systématique et uniforme afin de pouvoir profiter au maximum des nouveaux potentiels d'observation en astronomie.

Le CDS est un ensemble de serveurs situés à Strasbourg, qui gèrent d'importantes bases de données astronomiques. Il est composé de plusieurs catalogues appartenant à plusieurs bases : Simbad, Vizier, Aladin.

La quantité d'informations présentes est assez impressionante : la base de données Simbad contient :

1. 3 313 810 étoiles

[7]International Virtual Observatory Allinace

2. 5620 catalogues
3. 8 688 406 identifieurs
4. 148 187 références bibliographique
5. 4 524 779 citations sur les étoiles dans les publications

Les catalogues du CDS sont accessibles par internet à l'URL :

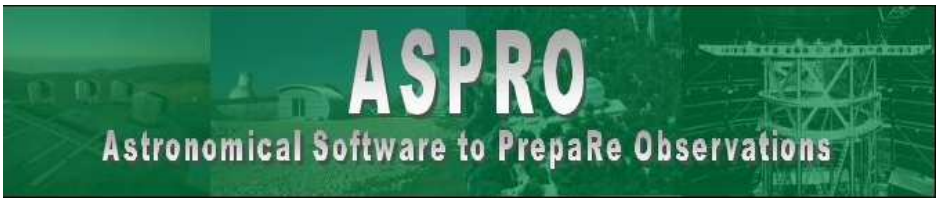
http ://cdsweb.u-strasbg.fr/

Toute la difficulté d'utilisation de ces catalogues réside dans le tri de l'information. Il est nécessaire de connaître la structure des requêtes à envoyer à Strasbourg, ainsi que la structure de sortie des données. Pour cette dernière, la norme est actuellement le XML. L'interrogation peut être visualisée sur un navigateur internet. Un exemple de requête est présenté dans ce rapport ainsi que des exemples de sorties XML.

2.3.4 Le logiciel ASPRO

L' "Astronomical Software to PrepaRe Observations" est un logiciel qui, comme son nom l'indique, permet de préparer des observations interférométrique (Duvert et al., 2002). Il est accessible par le web à l'adresse suivante :

http ://mariotti.ujf-grenoble.fr/ aspro/



Ce logiciel se présente sous la forme d'une application JAVA. Les menus déroulants permettent de choisir quel programme lancer. Dans l'exemple ci-dessous, nous lançons Search Calibrator :

FIG. 4 – Interface du logiciel ASPRO

2.3.5 Historique du logiciel Search Calibrators

Lors des observations interférométriques, il est nécessaire de passer par une phase de calibration afin d'obtenir les visibilités réelles pour que ces dernières puissent être interprétées comme de réels paramètres en astrophysique. La sélection de calibrateurs appropriés est cruciale pour obtenir une grande précision avec des instruments interférométriques comme le VLTI par exemple. C'est dans cet optique qu'il a été développé un logiciel pour créer un catalogue d'étoile évolutif donnant toutes les informations nécessaires pour effectuer un choix de calibrateurs qui respectent les besoins des astronomes et des instruments d'observations.

Une liste de calibrateurs potentiels devait être récupérée depuis un ensemble de catalogues disponibles au CDS. Les requêtes a effectuer au CDS étaient et sont toujours basées sur certains critères comme la distance angulaire ou la magnitude autour de la cible.

Afin de pouvoir utiliser les calibrateurs dans les programmes numériques, ceux-ci doivent avoir des propriétés proches des cibles que l'on souhaite obtenir : dans un premier temps, les calibrateurs doivent être dans la zone du ciel où l'on souhaite observer et doivent avoir une magnitude apparente proche pour pouvoir être observés avec un même instrument avec la même configuration optique. Dans un deuxième temps, le type spectral (ou couleur) doit être similaire dans le cas d'observations interférométriques dans de larges bandes pour limiter l'effet chromatique.

Le logiciel doit à partir de la magnitude et de la couleur de l'objet observé, et des contraintes sur la position et la magnitudes des calibrateurs :

1. Récupérer les liste des étoiles proches de l'objet de science en interrogeant le CDS.
2. Compléter les propriétés de ces étoiles à partir de formules analytiques.
3. Sélectionner les calibrateurs potentiels.
4. Présenter la liste à l'opérateur.

Il a été décidé que le logiciel Search Calibrators serait une partie du logiciel ASPRO. L'interface de Search Calibrators est, dans la continuité de ASPRO, une application JAVA qui est appelé depuis les menus déroulants de ce dernier :

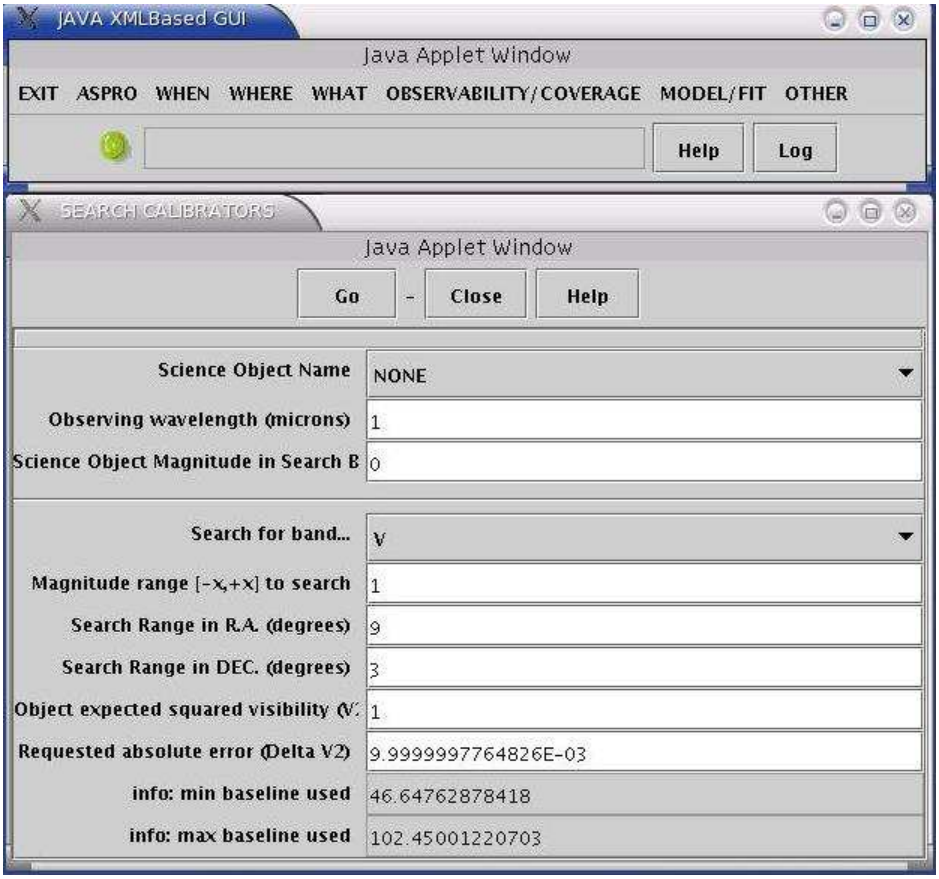


FIG. 5 – Interface du logiciel Search Calibrators

Les résultats sont affichés dans la fenêtre suivante :

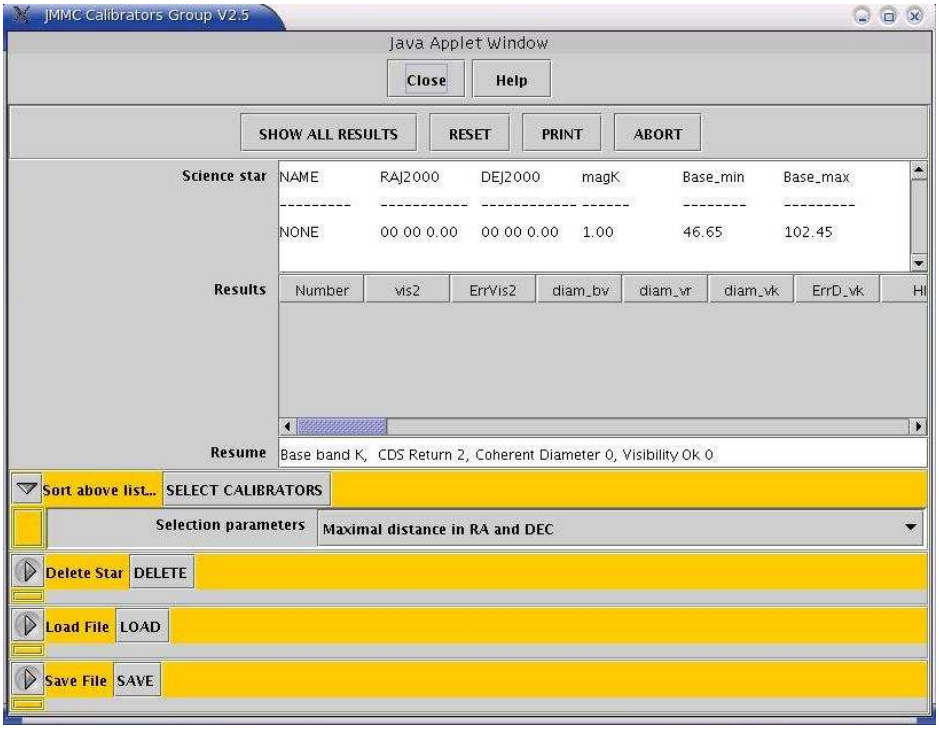


FIG. 6 – Fenêtre de résultats du logiciel Search Calibrators

Le logiciel a pour objectif de pouvoir obtenir des calibrateurs, et ce quelle que soit la magnitude de l'objet de science. En effet, certaines étoiles dont la magnitude est élevée, appelées objets faibles (faible luminosité), peuvent poser des problèmes car les informations les caractérisant peuvent être inexistantes ou très peu nombreuses. Une première version de Search Calibrators a été développée en C. Cette version ne gère que les objets brillants (lumineux). La partie recherche d'objets faibles est actuellement en cours d'étude dans les laboratoires de Grenoble et de Nice (OCA).

Afin de pouvoir implémenter dans Search Calibrators la partie objet faible, et de permettre une maintenance plus aisée du logiciel, il a été décidé de reprendre complètement Search Calibrators en programmation orientée objet en C++.

3 Réalisation

3.1 Objectif de la réalisation

L'objectif du stage est de réaliser la conception orientée objet du logiciel Search Calibrators. Ceci implique de repartir depuis le début. Le nouveau Search Calibrators doit s'appuyer sur l'ancien, notamment pour l'aspect scientifique et protocolaire (comment récupérer l'information, quels scénarios utiliser, ...). Par contre, la conception doit être redéfinie et le code doit se baser sur l'environnement de développement mis en place par l'équipe du JMMC : il doit respecter les règles de programmation, utiliser des librairies spécifiques pour la gestion d'erreur, le déboguage, etc... L'environnement de développement sera présenté par la suite.

Le travail à réaliser porte sur la partie objet brillant, comme le Search Calibrators actuel. Le logiciel doit être structuré de telle sorte que tout le travail effectué par les astronomes sur la partie objet faible puisse être intégrée facilement dans le nouveau Search Calibrators.

Une première étape du travail consiste donc à étudier le logiciel Search Calibrators existant. Cette étude est nécessaire pour repérer toutes les actions que celui-ci effectue. Cette étape est importante car elle permet de mettre en avant les raisons de la nécessité de créer un nouveau Search Calibrators.

La deuxième partie du travail consiste à définir la structure du nouveau logiciel. Il faut définir les objets à utiliser, donc les classes qui composent le nouveau soft. Cette étape est très importante car les contraintes pour le développement sont multiples : premièrement, Search Calibrators ne sera jamais un produit "fini", cela signifie qu'il est amené à évoluer en fonction des changements de formats des centres de données ou des recherches scientifiques par exemple. Il faut aussi prendre en compte que le logiciel s'inscrit dans le cadre d'un développement JMMC, c'est à dire qu'il doit utiliser l'environnement de développement propre à ce centre. Une troisième partie sera dédié à une série de tests permettant de valider le travail effectué.

3.2 L'environnement de développement

La présentation de l'environnement de développement est importante car les outils mis à la disposition du développeur détermine en partie l'allure du code qui est fourni. Le système d'exploitation sur lequel a été développé le logiciel est une distribution linux : Mandrake 9.2.

Le logiciel est stocké sur le serveur Mariotti du LAOG. Mariotti est disponible à l'adresse **http ://mariotti.ujf-grenoble.fr**.

L'environnement de développement s'appelle MCS[8]. Cet environnement est constitué de différentes bibliothèques ainsi que de certains outils comme par exemple des utilitaires de génération de code suivant les règles de programmation du JMMC, des utilitaires de remplacement de nom, d'extension, etc... L'environnement de développement est actuellement en cours de réalisation. Le logiciel Search Calibrators se sert de l'environnement mais profite aussi à ce dernier pour valider et perfectionner le travail effectué par les développeurs.

Actuellement, il est possible d'utiliser une bibliothèque de gestion d'erreurs dans les programmes. Une bibliothèque "log" servant au suivi des traces dans les programmes est aussi disponible. Il est important de préciser qu'une librairie de gestion de "buffer" à taille dynamique a été développé car comme les précédentes, elle est utilisée dans Search Calibrators. En résumé, l'environnement se compose pour l'instant de :

err : bibliothèque de gestion d'erreur

log : bibliothèque de suivi de trace

misc : bibliothèque de dynamique buffer

ctoo : outils de génération, modification, déplacement de code ou de fichiers

bcc : bibliothèque de classes C++ de base

[8]Mariotti Common Software

Les règles de programmation sont basées sur les règles mises en place pour le projet VLT. Ces règles définissent de manière précise la structure des dossiers des logiciels. Il y est aussi expliqué par exemple la manière de déclarer des variables, du nombre de colonnes que chaque ligne du code ne doit pas dépasser ou encore des règles à suivre pour nommer les classes...

Tous les développements du JMMC sont organisés en modules. Un module peut-être une librairie, un logiciel, ou un outils de développement. Chaque module doit être organisé de manière standard, c'est à dire respecter l'architecture logiciel fixée par le JMMC.

La documentation du code source suit les normes de Doxygen. Cet utilitaire "open source" permet à l'aide de balises situées à des endroits spécifiques dans le code de créer une documentation html ou latex. Il est présenté en Annexe la documentation générée pour le logiciel Search Calibrators.

3.3 Les modules de Search Calibrators

La structure du logiciel Search Calibrators a été un point très important à définir car ce logiciel doit pouvoir répondre à certaines contraintes. La première des contraintes est de créer un logiciel maintenable facilement et surtout évolutif. En effet, le domaine de la recherche auquel il est destiné peut le contraindre à évoluer : par exemple si les astronomes veulent récupérer des informations supplémentaires depuis le CDS, il est nécessaire que la modification dans le logiciel soit minime et intuitive. Bien sûr, la structure doit permettre aux développeurs de se replonger dans le logiciel aisément.
L'autre contrainte est de créer un logiciel qui soit transportable sur des serveurs distants. En effet, dans le cadre des observatoires virtuels, le fait de mettre une partie du logiciel à distance, comme à Strasbourg par exemple permet de faire ressortir l'idée même d'observatoire virtuel. La structure du logiciel doit donc être assez claire, afin que le passage d'objets puisse se faire sans contraintes pour le reste du programme.

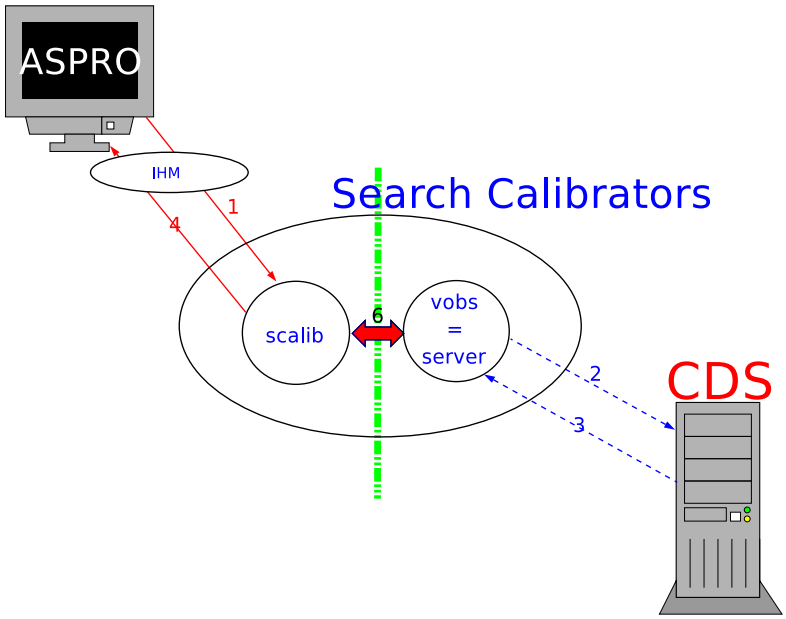


FIG. 7 – Organisation en module de Search Calibrators

Le programme est actuellement divisé en deux modules :

scalib : Ce module offre des méthodes d'interface avec ASPRO (1)(4). Il passe des objets requête à **vobs** (6).

vobs : Ce module est dédié à l'interrogation (2) du CDS et à la récupération des données (3). Il offre donc les méthodes d'interface avec les serveurs du CDS. Il passe les objets résultats, à savoir les listes d'étoiles de calibration à **scalib**.

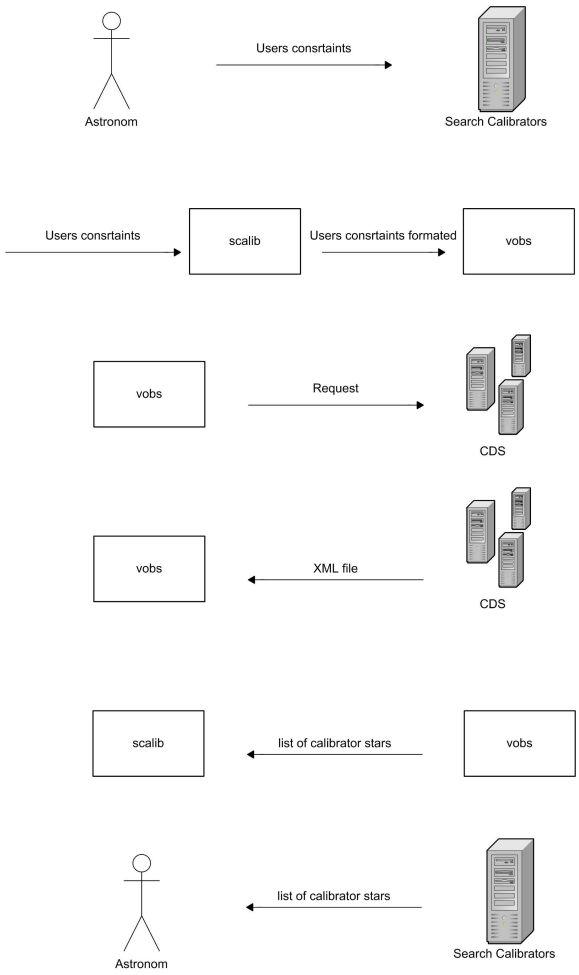


FIG. 8 – Etapes de Search Calibrators

3.4 scalib

Le module **scalib** de Search Calibrators est en cours de développement. Son rôle est d'assurer la partie calcul du logiciel, c'est-à-dire qu'il doit contenir des méthodes permettant de compléter les informations sur les étoiles. En effet, les données provenant du CDS sont incomplètes. Par exemple, certaines magnitudes dans différentes bandes sont manquantes. Les chercheurs du JMMC ont donc entrepris des études sur le développement d'algorithmes afin de les compléter. **scalib** va utiliser les objets définis dans **vobs**. Il offrira une interface à ce dernier et au logiciel ASPRO grâce à des méthodes spécifiques. Le développement de ce module nécessite la finalisation de **vobs**.

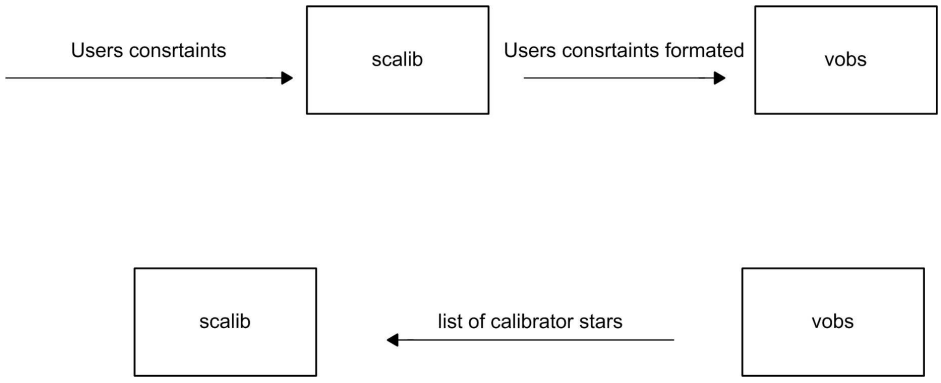


Fig. 9 – Principe de fonctionnement du module scalib

3.5 vobs

Le rôle de vobs est le suivant : après avoir récupéré les contraintes de l'utilisateur, il interroge les catalogues en créant des requêtes, analyse les données, les trie et retourne une liste de calibrateurs potentiels à scalib.

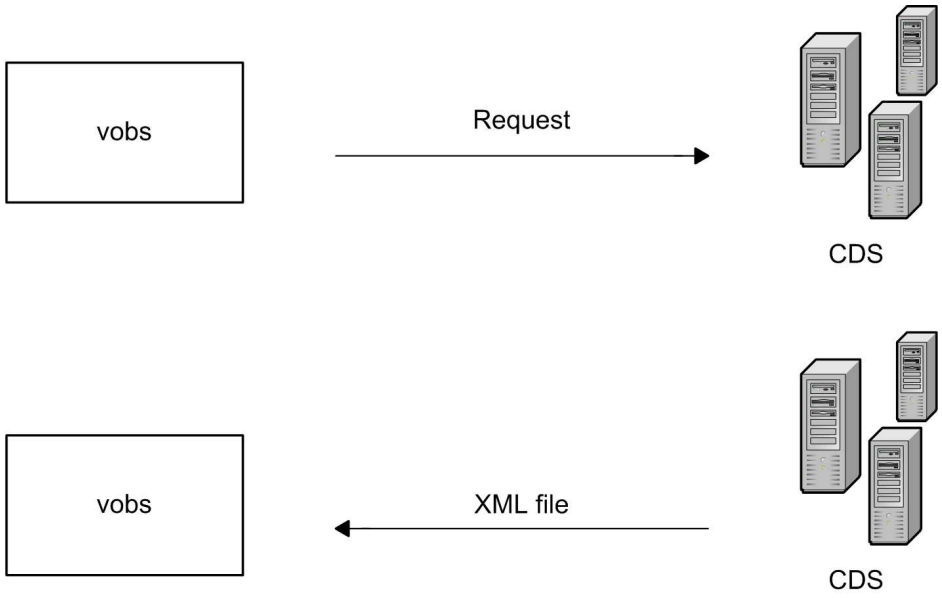


FIG. 10 – Principe de fonctionnement du module vobs

3.5.1 Le diagramme de classes

Il ressort de cette première approche de vobs les objets de base du programme. Il est mis en évidence l'objet requête. Ici, le mot requête doit être vu comme un ensemble de contraintes dictées par les nécessités de l'astronome. L'objet de calibration a été défini, ainsi que l'objet liste d'objet de calibration. L'objet catalogue est aussi présent. La classe catalogue est dérivée afin de créer un objet par catalogue. La recherche s'effectuant sur différents catalogues, il est nécessaire de créer des classes différentes à cause de la spécificité de chacun.

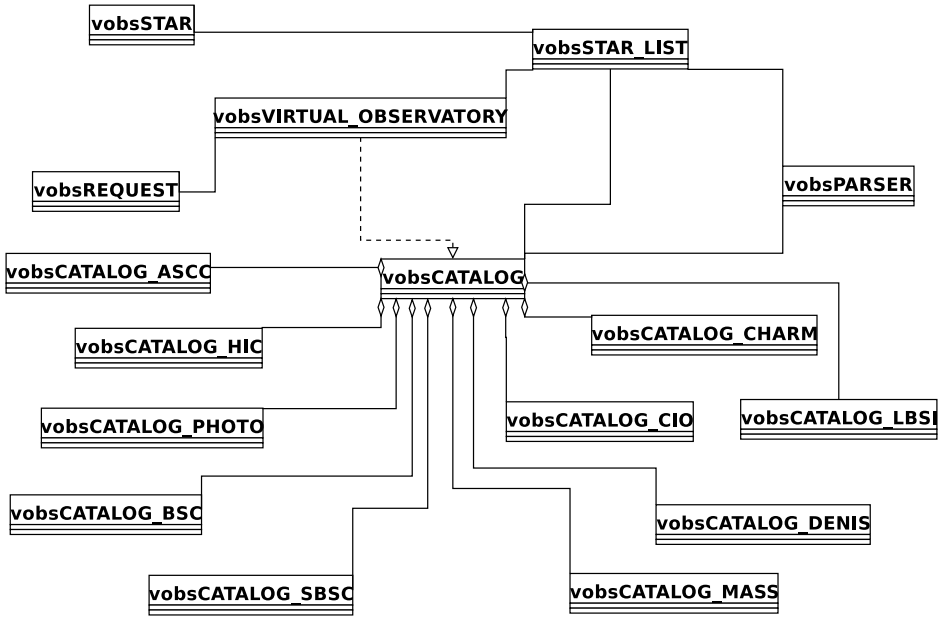


FIG. 11 – Diagramme de classes de vobs

La liste de toutes les classes du programme est la suivante :

1. `vobsVIRTUAL_OBSERVATORY`
2. `vobsREQUEST`
3. `vobsSTAR`
4. `vobsSTAR_LIST`
5. `vobsPARSER`
6. `vobsCATALOG`
7. `vobsCATALOG_ASCC : :vobsCATALOG`
8. `vobsCATALOG_BSC : :vobsCATALOG`
9. `vobsCATALOG_CHARM : :vobsCATALOG`
10. `vobsCATALOG_CIO : :vobsCATALOG`
11. `vobsCATALOG_DENIS : :vobsCATALOG`
12. `vobsCATALOG_HIC : :vobsCATALOG`
13. `vobsCATALOG_LBSI : :vobsCATALOG`

14. `vobsCATALOG_MASS : :vobsCATALOG`
15. `vobsCATALOG_PHOTO : :vobsCATALOG`
16. `vobsCATALOG_SBSC : :vobsCATALOG`

Les particularités de chaque classe seront expliquées plus en détails dans la suite du rapport.

Le déroulement du programme est représenté par le diagramme suivant :

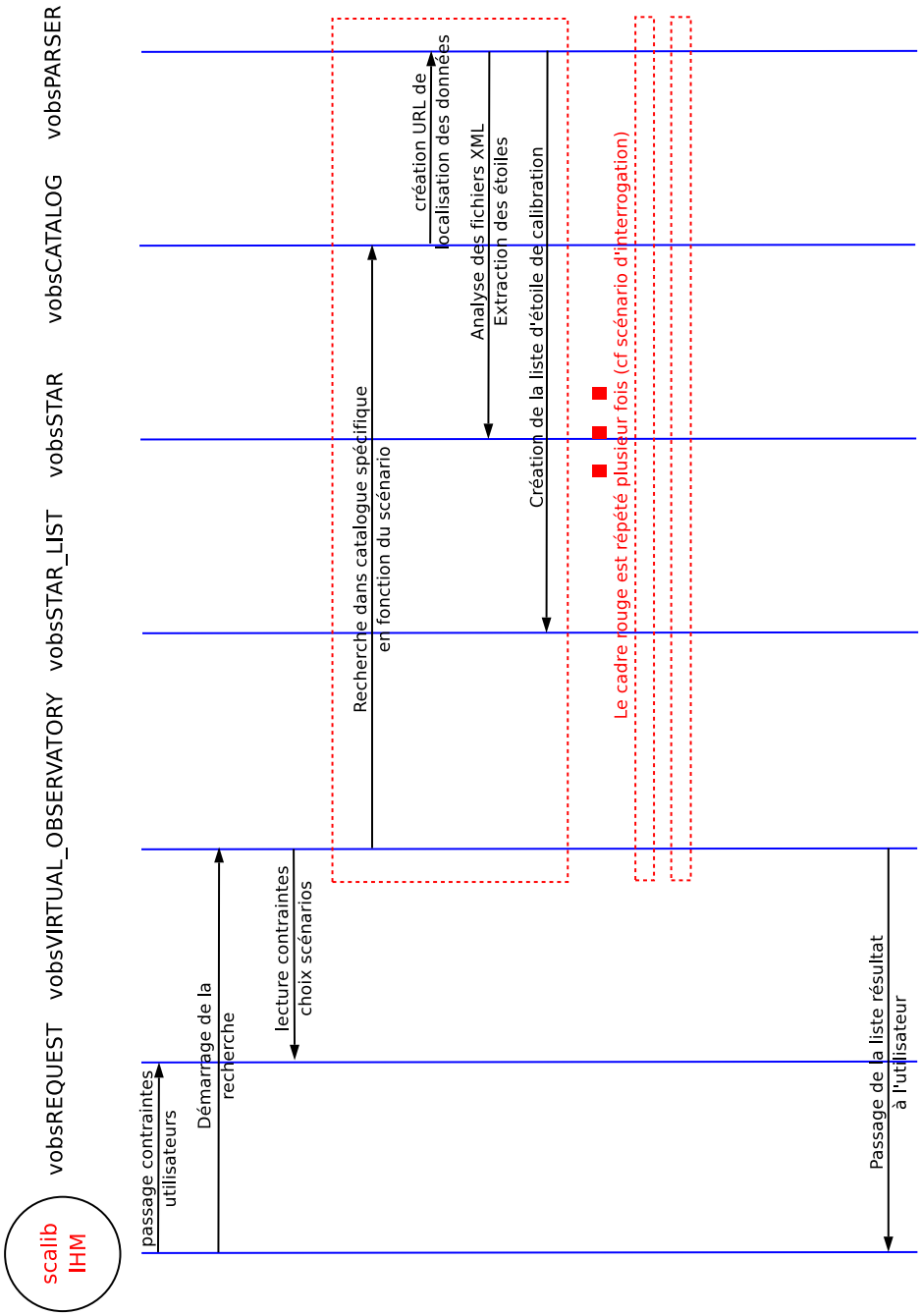


FIG. 12 – Diagramme de séquence de vobs

3.5.2 les contraintes utilisateurs

La recherche de données est réalisée en fonctions des contraintes utilisateurs. Les principales contraintes sont les suivantes :

1. Le nom de l'objet
2. Les coordonnées de la boîte d'observation
3. La longueur d'onde
4. La magnitude
5. La fourchette de magnitude
6. La bande dans laquelle on observe
7. La visibilité
8. L'erreur de visibilité

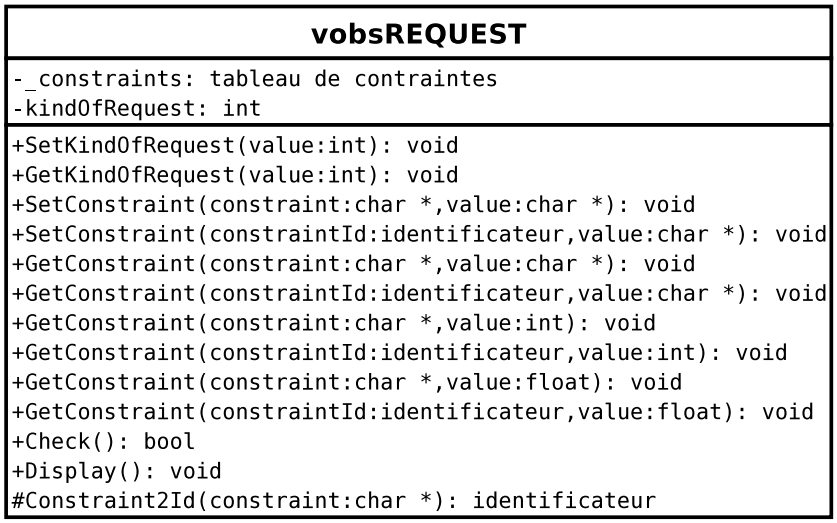


FIG. 13 – class vobsREQUEST

A l'aide de ces différents paramètres, l'objet `vobsREQUEST` est créé. Ce dernier est composé de plusieurs méthodes, essentiellement des méthodes d'affectation et de récupération de paramètres :

`SetConstraint(...), GetConstraint(...).`

Cette classe utilise une objet `vobsCONSTRAINT_ID` qui affecte un identificateur à chaque contraintes. Les contraintes sont "rangées" dans un tableau de contraintes, `_constraints[ ]`.

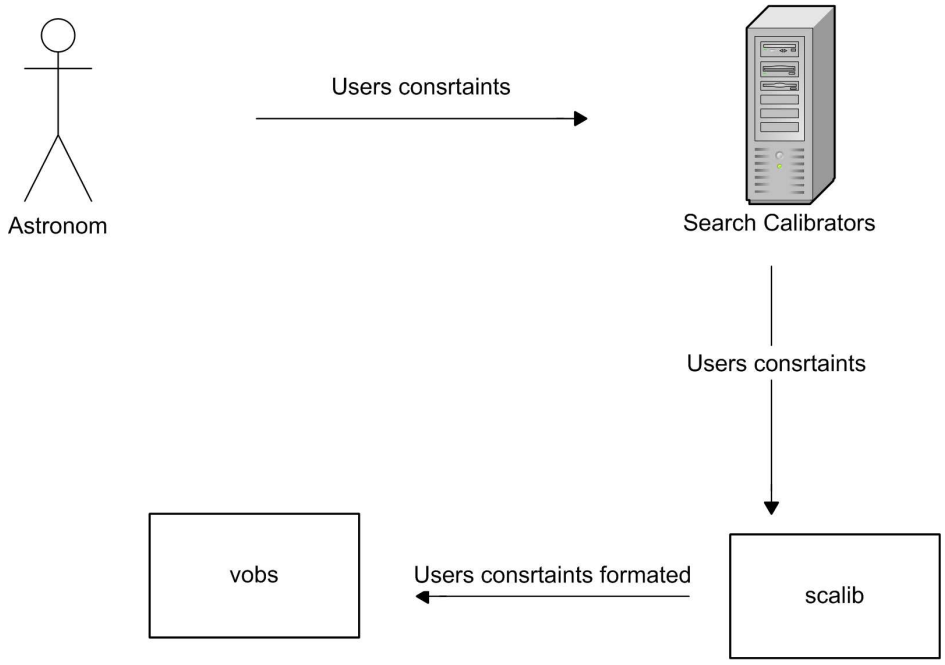


FIG. 14 – Arrivée d'une contrainte

Une méthode de vérification de construction, `Check()`, a été développée afin d'éviter au programme de démarrer une recherche si les objets ne sont pas construits correctement. Le nombre de contraintes étant fixe, la gestion des contraintes se fait de manière statique. Une fois cet objet créé il est plus aisé de "passer" d'une classe à l'autre ces contraintes.

FIG. 15 – Etapes pour la construction de l'objet requête

3.5.3 Les objets étoile de calibration et liste d'étoiles de calibration

L'étoile de calibration

Le but du logiciel étant de donner à l'astronome une ou des étoiles de calibration, il est apparu nécessaire de créer `vobsSTAR`, l'objet étoile. Une étoile est caractérisée par ses propriétés.

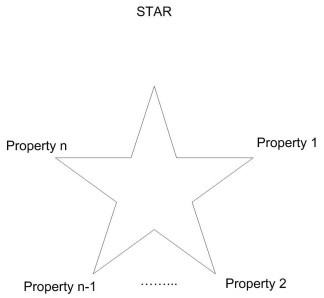


FIG. 16 – Objet étoile

Ces dernières sont nombreuses, les principales sont, pour une étoile de calibration, la magnitude dans les différentes bandes (U, B, V, R, I, J, H, K, L, M, N), la position exacte dans le ciel, le type spectral, la méthode d'observation, etc...

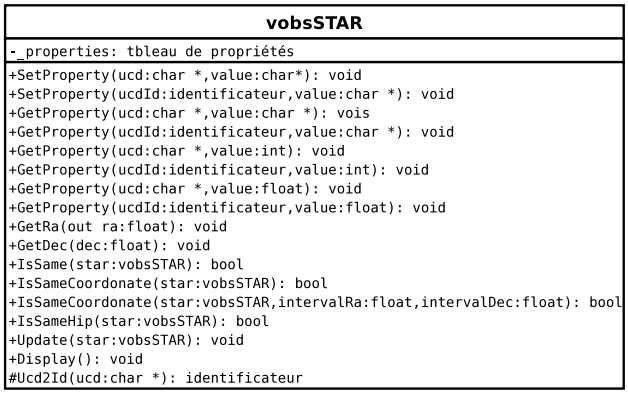


FIG. 17 – class vobsSTAR

Les propriétés correspondant à une étoile sont rangées dans un tableau de propriétés `_properties[ ]`. La classe `vobsSTAR` utilise un énumérateur `vobsUCD_ID` qui permet d'affecter un identificateur à chaque propriété pour pouvoir avoir un accès simple dans le tableau.

L'objet vobsSTAR contient plusieurs méthodes d'affectation qui permettent d'affecter une propriété à l'aide du nom de la propriété (UCD[9]) ou de son identificateur dont nous avons parlé avant : ces méthodes sont nommées `SetProperty(...)`. Il sera expliqué dans la partie CDS, ce à quoi correspond un UCD. La classe `vobsSTAR` contient six méthodes pour récupérer une propriété : `GetProperty(...)`. Il est possible ainsi d'obtenir celle-ci sous la forme d'une chaîne de caractère, d'un réel, si l'on souhaite effectuer des calculs, et cela à partir d'un UCD ou de l'identificateur.

Les coordonnées d'une étoile étant données en heures, minutes, secondes (*hms*) et en degrés, minutes, secondes (*dms*), il a été développé deux méthodes qui permettent d'obtenir les coordonnées en arcsecondes, `GetRa()` et `GetDec()`
Pour la conversion *hms* en *arcsec*, la position *hms* est de la forme :

$$ra(hms) = hh : mm : ss$$

La formule est la suivante :

$$ra(arcsec) = (hh + \frac{mm}{60} + \frac{ss}{3600}) \times 15$$

Pour la conversion *dms* en *arcsec*, la position *dms* est de la forme :

$$dec(dms) = dd : mm : ss$$

La formule est la suivante :

$$dec(dms) = dd + \frac{mm + \frac{ss}{60}}{60}$$

[9]Unified Content Description

Des méthodes de la classe `vobsSTAR` permettent de comparer l'objet avec un autre : `IsSame()`, `IsSameCoordonate()`. Il est possible de comparer de manière précise toutes les propriétés de l'étoile de calibration, ou de ne comparer que les coordonnées, ou de ne comparer les coordonnées que dans une fourchette à indiquer (Cette méthode est nécessaire car certaines étoiles ont pu être observées par deux astronomes différents, avec des positions sensiblement différentes). Enfin, il est possible de comparer les étoiles à l'aide de leur identifiant ou numéro HIP[10] grâce à la méthode `IsSameHip (vobsSTAR &star)`. Ce numéro est l'identifiant des étoiles dans un catalogue nommé Hipparcos qui sert de référence lors de recherches.

L'objet étoile de calibration est aussi capable de se "mettre à jour" à l'aide d'une méthode `Update (vobsSTAR &star)` qui, à partir d'une étoile de calibration acceptée comme identique, va remplir les informations manquantes.

La liste d'étoiles de calibration

Le résultat que le logiciel Search Calibrators doit renvoyer à l'utilisateur est une liste de calibrateurs potentiels.

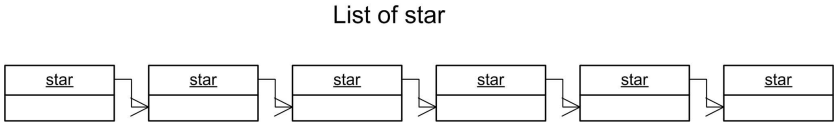


FIG. 18 – Liste d'étoiles

Il a été créé un objet `vobsSTAR_LIST` : cet objet permet de gérer des listes d'étoiles de calibration.

[10]Hipparcos number

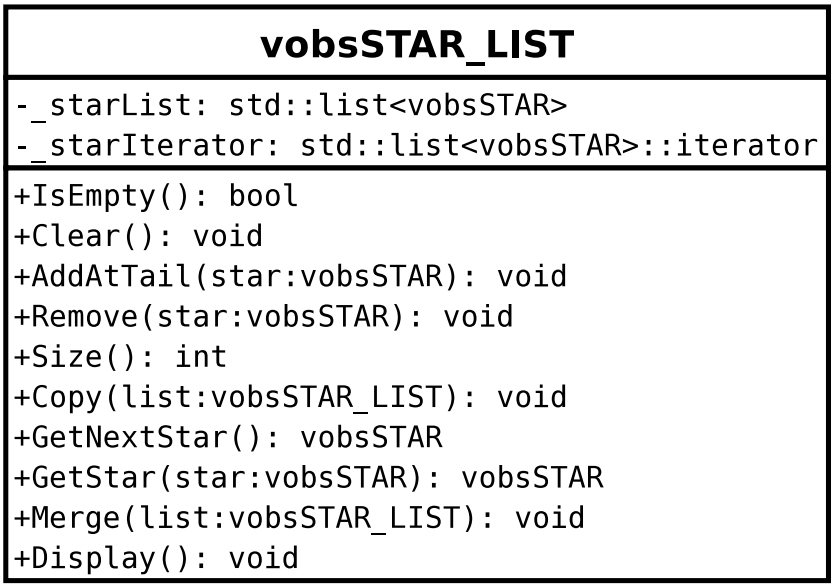


FIG. 19 – class vobsSTAR_LIST

Basée sur la STL[11], il a été redeveloppé des méthodes de déplacement, de suppression d'éléments, etc... Cette encapsulation de la liste sans utiliser les méthodes déjà existantes de la STL a été nécessaire.
En effet, `std : :list<vobsSTAR> _starList` ,qui contient concrètement les étoiles de calibrations, a été passée comme un membre privé de la classe de même que son itérateur : `iterator _starIterator`. Les raisons de ce passage en membre privé, est qu'il est souhaité que les opérations sur cette liste soient limitées pour que chaque objet qui utilise l'objet liste d'étoiles de calibration ne puisse effectuer des traitements inappropriés.

Une méthode `IsEmpty()` de la classe permet de savoir si la liste est vide. Il existe aussi une méthode `Clear()` permettant de supprimer tous les éléments de la liste.

Il a été créée une méthode d'ajout `AddAtTail(vobsSTAR &star)` et une méthode de suppression `Remove(vobsSTAR &star)`. Il est possible d'obtenir la taille de la liste à l'aide de la méthode `Size()`.

[11]Standard Template Library

Lorsqu'il est nécessaire de manipuler plusieurs listes en effectuant des copies, la méthode `Copy(vobsSTAR_LIST& list)` permet de copier des listes par valeurs. Les méthodes `GetNextStar()` et `GetStar()` permettent d'obtenir les étoiles de calibrations de la liste, tout en plaçant l'itérateur privé sur un emplacement spécifique. Sur la même idée que celle de "mise à jour" de l'étoile de calibration, il est possible d'utiliser une méthode `Merge(vobsSTAR_LIST &list)` qui va récupérer l'information manquante dans une autre liste d'étoiles de calibration.

3.5.4 l'interrogation du CDS, l'utilisation des catalogues

Comme il a été précisé précédemment, les données que l'on souhaite récupérer sont pour le moment situées au CDS. Lors du déroulement, nous allons interroger dix catalogues différents. Il est donc nécessaire de définir l'objet catalogue. La première des propriétés d'un catalogue est son nom. Voici la liste des noms des dix catalogues a utiliser :

1. *ASCC* : All Sky Compiled Catalog
2. *HIC* : Hipparcos Input Catalog, version 2
3. *PHOTO* : Photoelectric catalog
4. *MASS* : The 2MASS All Sky Catalog of Point Source
5. *CIO* : Catalog of Infrared Observations
6. *CHARM* : Catalog of High Angular Resolution Measurements
7. *LBSI* : Catalog of Calibrators Stars for LBSI
8. *BSC* : Bright Star Catalog
9. *SBSC* : Supplement to the BSC
10. *DENIS* : the DENIS database

Chaque catalogue possède un autre élément privé, la requête. Les données sont situées à des URL précises, il est donc important de pouvoir recréer ceux-ci pour chaque catalogue. Prenons un exemple simple : si l'on souhaite voir le contenu du catalogue Hipparcos, sans spécifier de contraintes particulières, il faut écrire une requête du type :

http ://vizier.u-strasbg.fr/viz-bin/asu-xml ?-source=I/280

pour voir les données à l'aide d'un navigateur.
Pour obtenir les informations directement depuis la console :

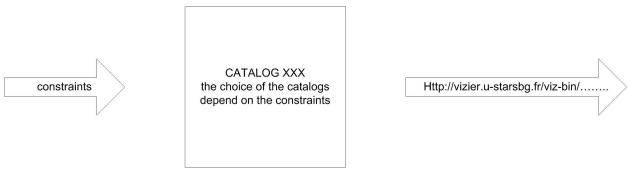


FIG. 20 – Création de la requête depuis les contraintes

GET http ://vizier.u-strasbg.fr/viz-bin/asu-xml ?-source=I/280

Il existe dans cette classe `vobsCATALOG` une méthode

```
Search(vobsREQUEST request, vobsSTAR_LIST &list)
```

qui va préparer la requête, récupérer l'information, l'analyser, et la renvoyer sous la forme d'une liste d'étoiles de calibration. Cette méthode publique va appeler des méthodes privées propres pour la plupart à chaque catalogue. C'est pourquoi il a été créé plusieurs objets `vobsCATALOG_XXXX` ou XXXX correspondant au nom du catalogue. Toutes ces classes dérivent de la même classe `vobsCATALOG`. Elles ne contiennent pas de méthodes particulières, mais elles surchargent quelques méthodes de `vobsCATALOG`.

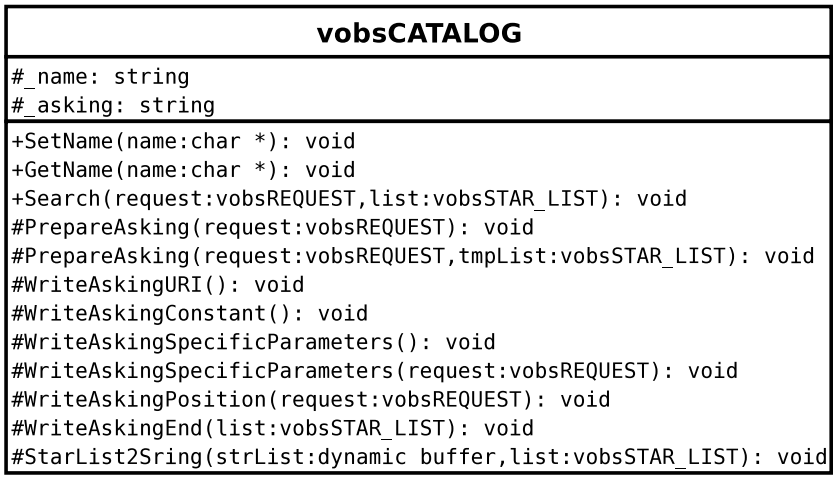


FIG. 21 – class vobsCATALOG

Il existe deux types de requêtes dans le logiciel Search Calibrators : un premier type va concerner l'interrogation brute, c'est-à-dire en spécifiant les contraintes que l'utilisateur a rentré. D'une manière générale, on peut schématiser cela par la phrase : Au début on a rien, on espère avoir une liste à la fin.

Le deuxième type de requête qu'il est possible de rencontrer est la requête avec une liste d'étoiles comme paramètre. On considère dans ce cas que l'on a déjà obtenu une liste d'étoiles, et que l'on souhaite écrire dans la requête *http ://...+<liste_etoile>*. La recherche va dans ce cas compléter à l'aide des données spécifiques du catalogue interrogé, la liste d'étoiles passée en paramètre. La structure d'une requête de premier type est la suivante :

URI + POSITION + PARAMETRES SPECIFIQUES POUR TYPE1

La structure d'une requête de deuxième type est la suivante :

URI + CONSTANTE + PARAMETRES SPECIFIQUES POUR TYPE2 + LISTE ETOILE

La méthode `Search()` va donc utiliser une des méthodes `PrepareAsking()` en fonction des informations présentes dans l'objet `vobsREQUEST`. `PrepareAsking()` fait ensuite appel à plusieurs méthodes :

1. `WriteAskingURI()`
2. `WriteAskingConstant()`
3. `WriteAskingSpecificParameters()`
4. `WriteAskingPosition()`
5. `WriteAskingEnd()`

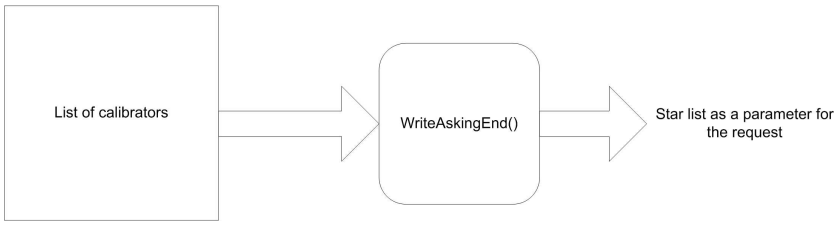


FIG. 22 – Passage de la liste détoiles comme paramètre

La requête à passer dans le navigateur est donc écrite à ce niveau. Un point important qui a été traité de manière transparente lors du traitement, est le passage d'une liste d'étoiles de calibration à une suite de caractères

représentant le paramètre de la requête.
Ce traitement est effectué dans la méthode `WriteAskingEnd()` par l'appel d'une méthode `StarList2String()`.

Les méthodes de préparation de la requête étant terminées, il peut être considéré que l'on sait aller chercher l'information et que l'on arrive à localiser le lieu de stockage des données. Il faut ensuite analyser ces données, qui sortent dans un format défini : le XML[12].

[12]eXtensible Markup Language

3.5.5 Le parser de fichiers XML

Les informations récupérées au CDS sont fournies sous forme de fichiers XML. Les informations sont dispatchées dans l'arbre. Il est nécessaire de définir un objet `vobsPARSER` qui va analyser les fichiers XML, et en extraire les données. Afin de se faire une idée des fichiers que renvoie le CDS, il est possible de visualiser l'arbre XML à l'adresse suivante : (cf annexe A)

http ://vizier.u-strasbg.fr/viz-bin/asu-xml ?=I/280

Schématiquement, la structure des fichiers est la suivante :

```
<ASTRO>
        <DEFINITION>
        </DEFINITION>
        <RESOURCE>
                <NAME>
                </NAME>
                <TITLE>
                </TITLE>
                <TABLE>
                        <FIELD>
                        </FIELD>
                        ...
                        <DATA>
                        </DATA>
                </TABLE>
        </RESOURCE>
</ASTRO>
```

Le fichier XML caractérise une table de données présents sous forme de CDATA dans la partie <DATA>...</DATA> de l'arbre. Ces CDATA sont en fait un tableau de n colonnes (le nombre de colonnes varie d'un catalogue à l'autre) et m lignes. Ce qui signifie que l'on a n informations pour m étoiles. L'organisation de l'information dans les CDATA du fichier est donné par les balises <FIELD> présentes au dessus. Chaque balise donne le nom d'une colonne, sa taille et le type de valeurs qui s'y trouvent. Afin d'obtenir

l'information sur les m étoiles, il faut dans un premier temps repérer les balises <FIELD>, lire l'information qu'elles contiennent, repérer la balise <DATA> et la lire.

En résumé, afin de parser le fichier XML, il faut lancer une méthode récursive qui va partir de la racine de l'arbre, parcourir tout l'arbre, en récupérant au fur et à mesure les informations ainsi que les CDATA présentes. Une autre méthode de l'objet parser servira à analyser les CDATA.

Un fichier XML peut être assimilé à un arbre. Il a donc les caractéristiques d'un arbre : il possède une racine, et différents noeuds permettant d'accéder aux différentes branches de l'arbre. Il existe deux manières d'utiliser un fichier XML : soit le traitement se fait "noeud par noeud" (SAX), en récupérant l'information à la volée, soit il s'effectue en stockant l'arbre en mémoire (DOM). SAX a l'avantage de ne pas avoir à stocker l'arbre en mémoire lors de l'utilisation de gros fichier, mais nécessite la connaissance complète de la structure utilisée : en effet, pour pouvoir se promener dans l'arbre, si celui-ci n'est pas stocké en mémoire, atteindre un enfant spécifique d'un noeud nécessite de connaître exactement l'existence de celui-ci, c'est-à-dire son numéro s'il existe plusieurs enfants, etc... Lors de l'utilisation de fichier pas trop volumineux, comme ceux fournis par le CDS, il est intéressant d'utiliser DOM car, une fois le fichier stocké en mémoire, il suffit de se "promener" dans l'arbre à l'aide des fonctions existantes dans les librairies : il n'y a pas de risque de se perdre ou de tomber sur un enfant qui n'existe pas car les méthodes fournies permettent de dire si ces opérations sont possibles. Les fichiers XML respectent les normes fixées par le W3C[13].

Il est présenté dans les pages suivantes la partie analyse de fichiers XML de manière technique. Le travail effectué sur les librairies d'analyse a été conséquent. Search Calibrators utilise la librairie **libgdome**. Après discussion, **libxml2** n'a pas été retenue, mais le temps passé à l'étudier a été

[13]World Wide Web Consortium

suffisament important pour qu'une présentation de cette librairie figure dans ce rapport.

libxml2

Un premier travail a été effectué afin de tester la librairie **libxml2**. Le site internet qui diffuse librement cette librairie est disponible à l'adresse :

http ://www.xmlsoft.org/

La complexité de l'utilisation de libxml2 consiste à comprendre la structure employée pour utiliser l'arbre XML. L'élément de base de la structure est `xmlDoc`. Pour utiliser celui-ci, il faut définir un pointeur sur cet élément : `xmlDocPtr`. Cette grosse structure contient tous les éléments de l'arbre XML. Ces derniers, sont des `xmlNode` ou "noeuds". De manière assez succinte, nous pouvons représenter la structure comme cela :

struct `xmlDoc` {
-xmlElementType : le type du document
(fichier XML par exemple)
-xmlNode fils : une référence vers un noeud "fils"
-xmlNode dernier : le dernier fils
-xmlNode parent : le(s) parent(s) des fils
-xmlNode suivant
-xmlNode précédent
-... (plusieurs élément faisant référence à
diverses informations : compression,
URL de provenance ...)
}

Il est donc nécessaire de déclarer l'arbre de la manière suivante :

```
xmlDocPtr doc ;
```

Il faut maintenant créer le noeud `root_element` qui donne une référence sur la racine de l'arbre :

```
xmlNode *root_element=NULL ;
```

La création d'une fonction `extractFile` de type `xmlDocPtr` est aussi nécessaire :

```
static xmlDocPtr extractFile(const char *filename, const
                    xmlCahr * pattern){}
```

où `const char *filename` est l'adresse où est situé le fichier XML à extraire et `xmlChar *pattern` est un renseignement qui permet de préciser si le fichier XML peut être modifié dans l'optique (qui n'est pas la notre), par exemple, de rajouter ou de détruire de l'information. Dans la situation de parser le fichier XML sans le modifier, le pattern sera "preserved".
La fonction `extractFile()` a les variables suivantes :

`xmlDocPtr doc ;` *//un pointeur sur le xmlDoc que l'on construit*
`xmlTextReaderPtr reader ;` *//un pointeur sur un xmlTextReader qui lit le fichier*

La méthode `xmlReaderForFile(filename, NULL, 0)` renvoie un xmlTextReader à partir du fichier XML. Le fichier est considéré comme "ouvert" et peut être dorénavant lu.

Afin de ne rien oublier dans l'arbre, c'est-à-dire de bien parcourir toutes les branches et ce jusqu'à leur extrémité, la librairie **libxml2** fournit une méthode `xmlTextReaderRead(reader)` qui renvoie un entier indiquant s'il reste des étages à parcourir, si une erreur est survenue ou si le parcours est fini. S'il reste des noeuds non explorés dans l'arbre, cette méthode fait pointer le `reader` sur les parties non encore explorées. Une fois l'arbre parcouru, `reader` pointe sur l'arbre entier. Il est nécessaire d'utiliser la méthode `xmlTextReaderCurrentDoc(reader)` afin de convertir en `xmlDoc` toute l'information sur laquelle pointe `reader` :

```
doc = xmlTextReaderCurrentDoc(reader) ;
```

La libération de l'espace mémoire sur lequel pointe `reader` s'effectue à l'aide de la méthode :

```
xmlFreeTextReader(reader) ;
```

Il ne reste plus qu'a `extractFile()` de renvoyer `doc` :

```
return(doc) ;
```

L'arbre est donc construit par la méthode `extractFile()` :

```
doc=extractFile(filename, (const xmlChar *) pattern) ;
```

L'arbre étant construit, il faut faire pointer `root_element` sur la racine de l'arbre ou `doc` : ceci est réalisé à l'aide d'une autre méthode de la librairie **libxml2**, `xmlDocGetRootElement(xmlDoc doc)` :

```
root_element=xmlDocGetRootElement(doc) ;
```

La difficulté réside maintenant dans le déplacement à l'intérieur de l'arbre. Le déplacement de noeuds en noeuds se fait assez simplement : par exemple, pour atteindre un fils de `root_element`, il faut définir un nouveau xmlNode :

```
xmlNode -root_element_child=root_element->children ;
```

L'opération de déplacement en elle-même n'est pas compliquée, mais pour pouvoir l'appliquer, il faut connaître, par exemple, le numéro du fils si l'élément a plusieurs enfants, ou savoir si l'enfant existe vraiment. C'est sur ce point que réside la plus grosse difficulté : il faut connaître précisemment la structure employée par **libxml2** pour pouvoir créer des méthodes donnant le nombre d'enfants d'un noeud, ou de l'existence de ceux-ci.

Il ressort que la librairie **libxml2** offre des possibilités intéressante pour parser les fichiers XML. Une étude plus approfondi de la structure d'un `xmlDoc` doit permettre de réaliser un parsing complet et efficace des fichiers. Cette librairie est donc complète, puissante, mais complexe. C'est pourquoi l'étude de la librairie **libgdome** qui offre une "interface" avec **libxml2** est présentée dans la suite du document.

libgdome

Le site internet relatif à cette librairie se situe à l'adresse :

http ://gdome2.cs.unibo.it/

Comme indiqué précédemment, libgdome offre une interface de **libxml2**, c'est à dire qu'elle est basée sur **libxml2** : elle en reprend la structure de données générale, mais elle offre de nombreuses méthodes qui permettent une utilisation plus simple des outils de parsing.

Afin d'utiliser cette librairie, il est nécessaire de définir un `GdomeDOMImplementation` qui fournit un certain nombre de méthodes pour effectuer des opérations sur le modèle de document. Elle se définit de la manière suivante :

```
GdomeDOMImplementation *domimpl ;
```

De la même manière qu'avec libxml2, l'arbre correspondant au fichier XML est stocké dans de type `GdomeDocument` :

```
GdomeDocument *doc ;
```

struct `GdomeDocument` {
fonctions permettant d'accéder à un
noeud de l'arbre, de connaître son type,
de créer un noeud et/ou de le supprimer
...
}

Ce `GdomeDocument`, qui contient l'arbre, est constitué de noeuds ou éléments : par exemple, pour définir l'élément racine qui pointe comme son nom l'indique sur la racine de l'arbre, il faut déclarer `root` de la manière suivante :

```
GdomeElement *root ;
```

La première chose à réaliser est de récupérer la référence de l'implémentation :

```
domimpl=gdome_di_mkref() ;
```

Il faut ensuite charger `doc` depuis le fichier XML :

```
doc=gdome_di_createDocFromURI(domimpl, filename,
          GDOME_LOAD_PARSING, &exc) ;
```

où `exc` est une `GdomeException` qui permet de gérer les exceptions. Ceci est le parfait exemple de l'avantage de libgdome : toute l'extraction, qui nécessite la création d'une fonction `extractFile` lors de l'utilisation de **libxml2**, est effectuée à l'aide d'une seule méthode déjà existante.
La librairie **libgdome** permet à l'aide de la méthode `gdome_doc_documentElement(doc, &exc)` de faire pointer l'élément `root` sur la racine :

```
root=gdome_doc_documentElement(doc, &exc) ;
```

Pour parcourir l'arbre entièrement, et de manière automatique, l'appel à une méthode récursive est nécessaire. Afin de lancer la récursivité, il faut avoir l'élément `root` et l'ensemble de ses enfants. Une méthode de la librairie permet d'avoir une référence sur la liste de tous les enfants d'un élément. Il faut donc définir une liste de noeuds, `child` de type `GdomeNodeList` :

```
GdomeNodeList *childs ;
```

En appliquant la méthode `gdome_el_childNodes` à l'élément `root`, il est créé une liste de ses enfants :

```
childs=gdome_el_childNodes(root,&exc) ;
```

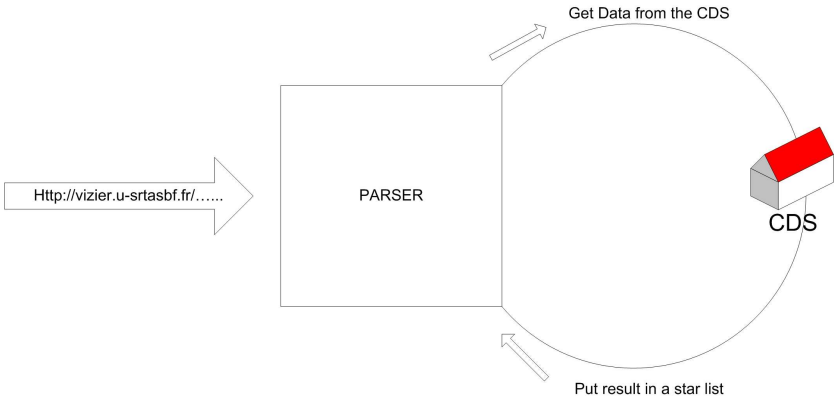
La création d'une méthode récursive `Parcours(GdomeNodeList *childs, GdomeDocument *doc)` permet de parser le fichier XML.

De manière synthétique, le parsing des fichiers XML s'effectue de manière récursive. L'information concernant les CDATA est récupérée et le bloc CDATA est stocké. Il ne reste alors qu'à effectuer un parsing de ce bloc à l'aide d'un buffer. La classe `vobsPARSER` contient une méthode qui peut être caractérisée comme "générale" , `Parse()`, qui servira d'interface avec les autres classes. Cette méthode prendra en entrée un fichier XML et en ressortira une liste d'étoile de calibration. La classe utilise une structure `vobsCDATA` qui représente le bloc CDATA. Cette structure, une fois initialisée, doit permettre un parsing des CDATA. La méthode `Parse()` va créer la structure XML en utilisant la librairie `libgdome`. Elle appelle ensuite une méthode privée `ParseXmlSubTree()`. Cette méthode récursive va extraire toute l'information nécessaire du fichier XML. Enfin, la méthode `ParseCData()` va, comme son nom l'indique, parser les CDATA.

vobsPARSER
+Parse(uri:char *,starList:vobsSTAR_LIST): void -ParseXmlSubTree(node:GdomeNode,listCDATA:std::vector<vobsCDATA *>,cData:vobsCDATA): voic -ParseCData(cData:vobsCDATA,starList:vobsSTAR_LIST): void

FIG. 23 – class vobsPARSER

Les actions que réalise **vobsPARSER** sont résumées dans le schéma suivant :



3.5.6 Les scénarios d'interrogation

Les travaux menés par les chercheurs des laboratoires de Grenoble et de Nice ont permis, à l'aide de tests et d'études des résultats des fichiers du CDS, de définir des scénarios d'interrogation des différents catalogues. Pour les objets brillants, il a été défini deux types de scénarios en fonction de la magnitude cherchée.

Le scénario bande K

Pour la recherche en bande K, les catalogues à interroger doivent respecter le scénario suivant :
soit la liste d'étoile L, qui sera considérée comme la liste résultat de la recherche. Au début, L est vide.

1. requête primaire sur CIO → obtention d'une liste d'étoile L1.
2. requête primaire PHOTO → obtention d'une liste d'étoile L2.
3. requête secondaire sur ASCC avec comme paramètre la liste L1 mergée avec L2 → obtention d'une liste d'étoile L3.
4. requête secondaire sur CIO avec comme paramètre la liste L3 → la liste obtenue est mergée dans L.
5. requête secondaire sur HIC avec comme paramètre la liste L3 → la liste obteneu est mergée dans L.
6. requête secondaire sur MASS avec comme paramètre la liste L3 → la liste obtenue est mergée dans L.
7. requête secondaire sur LBSI avec comme paramètre la liste L3 → la liste obtenue est mergée dans L.
8. requête secondaire sur CHARM avec comme paramètre la liste L3 → la liste obtenue est mergée dans L.
9. requête secondaire sur PHOTO avec comme paramètre la liste L3 → la liste obtenue est mergée dans L.
10. requête secondaire sur BSC avec comme paramètre la liste L3 → la liste obtenue est mergée dans L.
11. requête secondaire sur SBSC avec comme paramètre la liste L3 → la liste obtenue est mergée dans L.
12. requête secondaire sur DENIS avec comme paramètre la liste L3 → la liste obtenue est mergée dans L.

Le scénario bande V

Pour la recherche dans la bande V, le scénario défini est :
En considérant toujours que la liste L est vide.

1. requête primaire sur ASCC → obtention d'une liste d'étoile L1.
2. requête secondaire sur HIC avec comme paramètre la liste L1 → la liste obtenue est mergée dans L.
3. requête secondaire sur MASS avec comme paramètre la liste L1 → la liste obtenue est mergée dans L.
4. requête secondaire sur CIO avec comme paramètre la liste L1 → la liste obtenue est mergée dans L.
5. requête secondaire sur LBSI avec comme paramètre la liste L1 → la liste obtenue est mergée dans L.
6. requête secondaire sur CHARM avec comme paramètre la liste L1 → la liste obtenue est mergée dans L.
7. requête secondaire sur PHOTO avec comme paramètre la liste L1 → la liste obtenue est mergée dans L.
8. requête secondaire sur BSC avec comme paramètre la liste L1 → la liste obtenue est mergée dans L.
9. requête secondaire sur SBSC avec comme paramètre la liste L1 → la liste obtenue est mergée dans L.
10. requête secondaire sur DENIS avec comme paramètre la liste L1 → la liste obtenue est mergée dans L.

Les scénarios d'interrogation sont gérés par une classe `vobsVIRTUAL_OBSERVATORY`. Cette classe est composée de deux méthodes : une première, publique, s'intitulant `Research()`, va appeler une autre nommée `LoadScenario()`. Cette dernière est une méthode protégée.

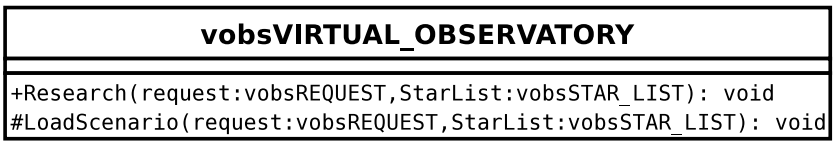


FIG. 24 – class vobsVIRTUAL_OBSERVATORY

3.6 Manuel utilisateur

L'utilisation de **vobs** est présentée sous la forme d'un exemple. Considérons que l'astronome souhaite observer l'étoile ETA TAU. Il cherche des calibrateurs dans la zone du ciel définie par les coordonnées :

- 03 : 47 : 29.08 (hms) right ascension
- +24 : 06 : 18.5 (dms) declinaison

Il souhaite que les calibrateurs aient une magnitude proche de 2.96. Pour être large il fixe une fourchette de magnitude entre -1 et 5. Il désire une longueur d'onde de 0.65 et une visibilité de 0.922. Enfin, il souhaite observer dans la bande K. Pour simplifier, ne tenons pas compte des unités des contraintes.

Afin de lancer le traitement, il doit commencer par construire un objet `vobsREQUEST`

```
vobsREQUEST request ;
```

et affecter à chaque contrainte la valeur qui lui est propre.

```
request.SetConstraint(STAR_NAME_ID,"ETA TAU");
request.SetConstraint(STAR_WLEN_ID,"0.65");
request.SetConstraint(STAR_MAGNITUDE_ID,"2.96");
request.SetConstraint(MAGNITUDE_RANGE_ID,"-1..5");
request.SetConstraint(SEARCH_BOX_RA_ID,"03+47+29.08");
request.SetConstraint(SEARCH_BOX_DEC_ID,"+24+06+18.5");
request.SetConstraint(STAR_EXPECTED_VIS_ID,"0.922");
request.SetConstraint(STAR_MAX_ERR_VIS_ID,"0.09");
request.SetConstraint(OBSERVED_BAND_ID,"K");
```

Bien sûr, il faudra automatiser ceci en passant juste une ligne de commande. Il est conseillé de vérifier que la requête a bien été construite.

```
request.Check() ;
```

On construit alors l'objet observatoire virtuel. C'est donc lui qui va gérer l'interrogation du CDS et la récupération des données.

```
vobsVIRTUAL_OBSERVATORY vobs ;
```

Les résultats seront stockés dans un objet liste d'étoiles de calibration.

```
vobsSTAR_LIST starList ;
```

Il suffit à l'utilisateur de **vobs** de lancer la recherche

```
vobs.Research(request, starList) ;
```

pour obtenir dans `starList` tous les calibrateurs potentiels. Tous les traitements présentés seront gérés bientôt par le module **scalib** qui ajoutera certaines propriétés à la liste d'étoiles grâce à certains calculs. L'atronome n'aura plus qu'à utiliser la souris en lançant Search Calibrators depuis ASPRO.

Voici un exemple de code complet, utilisant l'environnement de développement, qui effectue le traitement présenté.

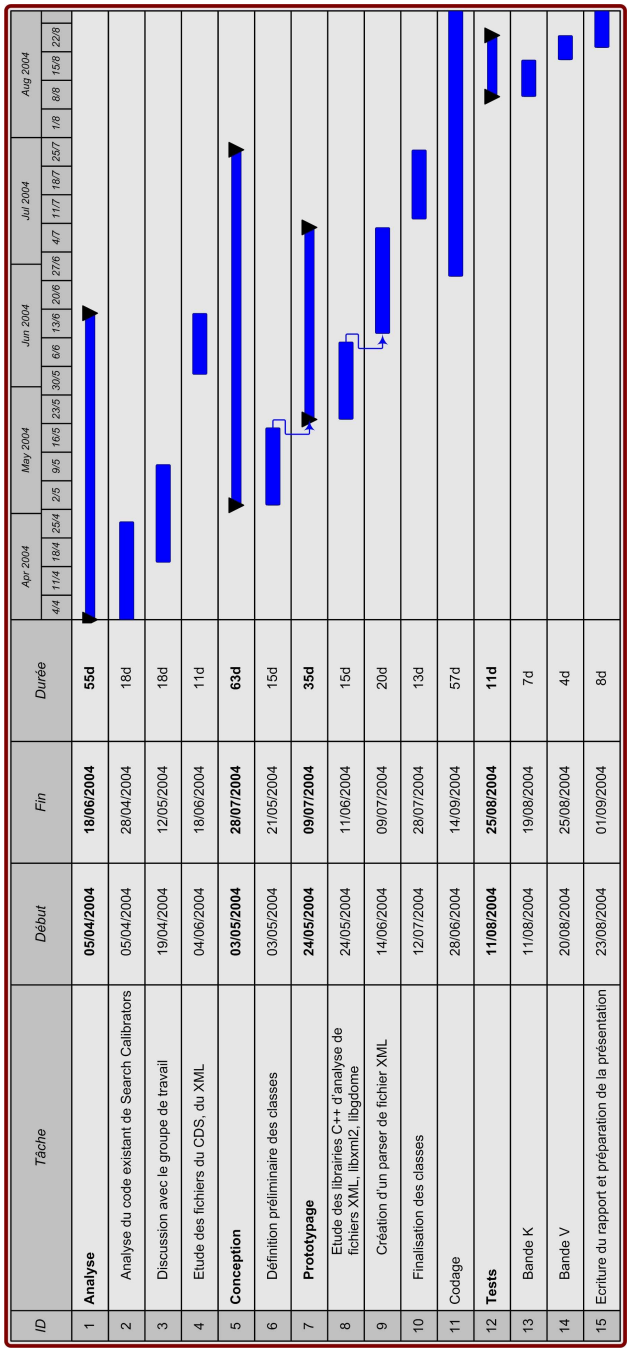
```
/*
 * System Headers
 */
#include <iostream>
using namespace std;


/*
 * MCS Headers
 */
#include "mcs.h"
#include "log.h"
#include "err.h"


/*
 * Local Headers
 */
#include "vobsSTAR.h"
#include "vobsSTAR_LIST.h"
#include "vobsREQUEST.h"
#include "vobsVIRTUAL_OBSERVATORY.h"
#include "vobsPrivate.h"


int main(int argc, char *argv[])
{
    // Initialize MCS services
    if (mcsInit(argv[0]) == FAILURE)
```

```
{
    // Error handling if necessary

    // Exit from the application with FAILURE
    exit (EXIT_FAILURE);
}

logSetStdoutLogLevel(logEXTDBG);

logInfo("Starting ...");
vobsREQUEST request;

// Affect the kind of request
if (request.SetKindOfRequest(1)==FAILURE)
{
    errDisplayStack();
    errCloseStack();
    exit(EXIT_FAILURE);
}

if ((request.SetConstraint(STAR_NAME_ID,"ETA TAU") == SUCCESS) &&
    (request.SetConstraint(STAR_WLEN_ID,"0.65") == SUCCESS) &&
    (request.SetConstraint(STAR_MAGNITUDE_ID,"2.96") == SUCCESS) &&
    (request.SetConstraint(MAGNITUDE_RANGE_ID,"-1..5") == SUCCESS) &&
    (request.SetConstraint(SEARCH_BOX_RA_ID,"03+47+29.08") == SUCCESS) &&
    (request.SetConstraint(SEARCH_BOX_DEC_ID,"+24+06+18.5") == SUCCESS) &&
    (request.SetConstraint(STAR_EXPECTED_VIS_ID,"0.922") == SUCCESS) &&
    (request.SetConstraint(STAR_MAX_ERR_VIS_ID,"0.09") == SUCCESS) &&
    (request.SetConstraint(OBSERVED_BAND_ID,"K") == SUCCESS ))
{
    // Check if the request is correctly build
    if (request.Check()==mcsTRUE)
    {
        // Display of the request correctly build
        request.Display();
    }
    else
    {
        errDisplayStack();
        errCloseStack();
```

```
            exit(EXIT_FAILURE);
        }
    }
    else
    {
        errDisplayStack();
        errCloseStack();
        exit(EXIT_FAILURE);
    }
    errDisplayStack();
    // Declaration of the list of star
    vobsSTAR_LIST starList;

    // declaration of the virtual observatory
    vobsVIRTUAL_OBSERVATORY vobs;

    // start the research in the virtual obs built
    if (vobs.Research(request, starList)==FAILURE)
    {
        errDisplayStack();
        errCloseStack();
        exit(EXIT_FAILURE);
    }

    errDisplayStack();

    logInfo("Exiting ...");
    exit(EXIT_SUCCESS);

}
```

3.7 Bilan

3.7.1 Diagramme de Gantt



ID	Tâche	Début	Fin	Durée
1	**Analyse**	**05/04/2004**	**18/06/2004**	**55d**
2	Analyse du code existant de Search Calibrators	05/04/2004	28/04/2004	18d
3	Discussion avec le groupe de travail	19/04/2004	12/05/2004	18d
4	Etude des fichiers du CDS, du XML	04/06/2004	18/06/2004	11d
5	**Conception**	**03/05/2004**	**28/07/2004**	**63d**
6	Définition préliminaire des classes	03/05/2004	21/05/2004	15d
7	**Prototypage**	**24/05/2004**	**09/07/2004**	**35d**
8	Etude des librairies C++ d'analyse de fichiers XML, libxml2, libgdome	24/05/2004	11/06/2004	15d
9	Création d'un parser de fichier XML	14/06/2004	09/07/2004	20d
10	Finalisation des classes	12/07/2004	28/07/2004	13d
11	Codage	28/06/2004	14/09/2004	57d
12	**Tests**	**11/08/2004**	**25/08/2004**	**11d**
13	Bande K	11/08/2004	19/08/2004	7d
14	Bande V	20/08/2004	25/08/2004	4d
15	Ecriture du rapport et préparation de la présentation	23/08/2004	01/09/2004	8d

FIG. 25 – Diagramme de Gantt

Le travail s'est organié autour de cinq tâches principales :

Analyse : analyse du code existant, discussion avec le groupe de travail et étude des fichiers du CDS et du XML. Cette partie a commencé dès le début du stage et s'est déroulé sur deux mois environ.

Conception : définition des classes, prototypage, et finalisation des classes. Ce travail s'est déroulée sur une période de trois mois.

Codage : durée d'environ deux mois.

Tests : Des tests ont d'abord été effectués sur la recherche en bande K, puis en bande V.

Rapport et Soutenance : durant les deux dernières semaines du stage.

Ce diagramme de Gantt a été mis à jour tout au long du stage.

Bien que déjà existant, le logiciel Search Calibrators n'a pas été facile à mettre en oeuvre car le passage en Orienté Objet a nécessité une remise en question de tout ce qui était déjà codé. Il fallait cependant que les résultats soit comparables à l'ancien. L'environnement de développement, qui semblait une contrainte au début du stage, notamment sur tout ce qui concerne les déclarations, ou l'utilisation de librairie de bases, s'est avéré être une aide grâce à la rigueur que son utilisation impose. En effet, la documentation du code source, par exemple, est facilité car on connaît précisement le format à utiliser grâce à Doxygen. La librairie de log permet un débogage plus aisé du programme grâce à la trace qu'elle permet d'obtenir. Le logiciel a permis aussi d'aider au développement de ces librairies car chacune a été testée sur le module **vobs**. C'est la raison pour laquelle le développement de Search Calibrators a été retardé. En effet, il était prévu que le logiciel devait pouvoir fonctionner durant le mois de Septembre. La partie interrogation du CDS marche, elle est utilisable en ligne de commande et elle donne les mêmes résultats que l'ancien Search Calibrators. Le travail restant sera finalisé dans les mois à venir avec comme grands axes à suivre :

- Effectuer un ensemble de tests poussés afin de valider officiellement la partie interrogation.
- Implémenter la partie calcul.
- Implémenter la partie "objets faibles" dans le nouveau Search Calibrators.

3.7.2 Problèmes rencontrés

La première difficulté a été de définir précisement la structure du programme. Les premières versions du diagramme de classes étaient assez éloignées du diagramme actuel. Certaines classes, par exemple, ont complètement disparu. L'autre grosse difficulté a été d'utiliser l'environnement de développement du JMMC. La nécessité de celui-ci ne saute pas aux yeux lorsque l'on a déjà travaillé sur des petits projets. Ces outils mis à la disposition du développeur ne semblent pas vitaux pour l'avancement du projet. Mais au bout de quelque semaines d'utilisation, on mesure toute l'importance de cet encadrement lorsque les premiers "bugs" sérieux apparaissent.

4 Conclusion

Ce stage effectué dans le cadre de ma troisième année de formation à l'IUP MAI a été personnellement une réussite et ce pour plusieurs raisons : Premièrement, d'un point de vue technique, le travail réalisé m'a permis d'étudier la programmation "Orienté Objet" de manière approfondie dans la suite logique des enseignements suivis pendant la formation. La lecture d'ouvrage sur l'UML pour pouvoir définir la sructure des classes du logiciel ainsi que les discussions avec les développeurs m'ont permis de découvrir la gestion de projet et comment coder de manière structurée et rigoureuse. J'ai également découvert la programmation dans un environnement de développement professionnel. La connaissance de nouveau language comme le XML sera à, n'en pas douter, un avantage. Enfin, le stage qui s'est déroulé dans des conditions excellentes m'a permis de comprendre et d'appliquer certaines règles vitales que tous développeurs doit suivre. Voici personnellement les règles qui régissent une bonne programmation :

1. Ne pas coder tout de suite tu devras.
2. Analyser le sujet tu t'imposeras.
3. Analyser encore tu devras.
4. Partager tes analyses avec des personnes compétentes tu devras.
5. Ecouter leurs suggestions surtout tu devras.
6. Coder enfin tu pourras. (Enfin !)
7. Documenter ton code, tu devras. (et t'y prendre dès le début afin de ne pas passer des jours à tout reprendre...)

Le travail réalisé sera poursuivi puisqu'un contrat m'a été proposé. Je devrai développer dans un premier temps la partie calcul et la partie objet faible du logiciel Search Calibrators. Il sera aussi nécessaire de travailler sur l'aspect graphique de l'application.

A Exemple de fichier XML du CDS

```
<?xml version = "1.0"?>
<!DOCTYPE ASTRO SYSTEM "http://vizier.u-strasbg.fr/vizier/lib/astrores.dtd">


<!-- Parameters used for the Query:
 -source=I/280
 -out.max=10
 -->

<ASTRO ID="v2.94" xmlns:ASTRO="http://vizier.u-strasbg.fr/doc/astrores.htx">
 <DESCRIPTION about="http://vizier.u-strasbg.fr/">
   VizieR Astronomical Server: http://vizier.u-strasbg.fr 2004-08-20T07:57:00
   Explanations and Statistics of UCDs: See LINK below
   In case of problem, please report to: question@simbad.u-strasbg.fr
 </DESCRIPTION>
 <LINK content-role="hints" href="http://vizier.u-strasbg.fr/dm-bin/ucd_show?${UCD}"/>
<!-- AstroRes Description at  http://vizier.u-strasbg.fr/doc/astrores.htx -->

<DEFINITIONS>
  <COOSYS ID="J2000" system="eq_FK5" equinox="2000"/>
</DEFINITIONS>

<INFO ID="Ref" name="-ref" value="Vxml4940" />

<RESOURCE ID="1280">
  <NAME>I/280A</NAME>
  <TITLE>All-sky Compiled Catalogue of 2.5 million stars (Kharchenko 2001)</TITLE>
  <TABLE ID="I_280A_ascc01">
    <NAME>I/280A/ascc01</NAME>
    <TITLE>The all-sky catalogue of 2.5million stars</TITLE>
    <COOSYS ID="J2000_1991.250" system="eq_FK5" equinox="2000_1991.250"/>

    <!-- Now comes the definition of each field -->
    <FIELD name="recno" ucd="RECORD" datatype="I" width="8" type="hidden">
      <DESCRIPTION>Record number within the original table (starting from 1)</DESCRIPTION>
      <LINK href="http://vizier.u-strasbg.fr/viz-bin/VizieR-5?-info=XML&amp;-out.add=.&amp;-source=I/280A/ascc01&amp;recno=${recno}"/>
    </FIELD>
    <FIELD name="RAJ2000" ucd="POS_EQ_RA_MAIN" ref="J2000_1991.250" datatype="F" width="12" precision="8" unit="deg">
      <DESCRIPTION>Right Ascension J2000.0, epoch 1991.25</DESCRIPTION>
      <LINK href="http://vizier.u-strasbg.fr/viz-bin/VizieR-5?-info=XML&amp;-out.add=.&amp;-source=I/280A/ascc01&amp;recno=${recno}"/>
    </FIELD>
    <FIELD name="DEJ2000" ucd="POS_EQ_DEC_MAIN" ref="J2000_1991.250" datatype="F" width="12" precision="8" unit="deg">
      <DESCRIPTION>Declination J2000.0, epoch 1991.25</DESCRIPTION>
    </FIELD>
    <FIELD name="Plx" ucd="POS_PARLX_TRIG" datatype="F" width="7" precision="2" unit="mas">
      <DESCRIPTION>? Trigonometric parallax</DESCRIPTION>
    </FIELD>
    <FIELD name="e_Plx" ucd="ERROR" datatype="F" width="6" precision="2" unit="mas">
      <DESCRIPTION>? Standard error in Plx</DESCRIPTION>
    </FIELD>
    <FIELD name="pmRA" ucd="POS_EQ_PMRA" ref="J2000_1991.250" datatype="F" width="8" precision="2" unit="mas/yr">
      <DESCRIPTION>Proper Motion in RA*cos(DE)</DESCRIPTION>
    </FIELD>
    <FIELD name="pmDE" ucd="POS_EQ_PMDEC" ref="J2000_1991.250" datatype="F" width="8" precision="2" unit="mas/yr">
      <DESCRIPTION>Proper Motion in Declination</DESCRIPTION>
    </FIELD>
    <FIELD name="Bmag" ucd="PHOT_JHN_B" datatype="F" width="6" precision="3" unit="mag">
      <DESCRIPTION>? B magnitude in Johnson system</DESCRIPTION>
    </FIELD>
    <FIELD name="Vmag" ucd="PHOT_JHN_V" datatype="F" width="6" precision="3" unit="mag">
      <DESCRIPTION>? V magnitude in Johnson system</DESCRIPTION>
    </FIELD>
    <FIELD name="SpType" ucd="SPECT_TYPE_MK" datatype="A" width="12">
      <DESCRIPTION>Spectral type in MK or HD system</DESCRIPTION>
    </FIELD>
    <FIELD name="TYC1" ucd="ID_ALTERNATIVE" datatype="I" width="4">
      <DESCRIPTION>? TYC1 number from Tycho-2</DESCRIPTION>
      <LINK href="http://vizier.u-strasbg.fr/viz-bin/VizieR-5?-info=XML&amp;-out.add=.&amp;-source=Tycho-2&amp;TYC1=${TYC1}&amp;TYC2=${TYC2}&amp;TYC3=${T
    </FIELD>
    <FIELD name="TYC2" ucd="ID_ALTERNATIVE" datatype="I" width="5">
      <DESCRIPTION>? TYC2 number from Tycho-2</DESCRIPTION>
      <LINK href="http://vizier.u-strasbg.fr/viz-bin/VizieR-5?-info=XML&amp;-out.add=.&amp;-source=Tycho-2&amp;TYC1=${TYC1}&amp;TYC2=${TYC2}&amp;TYC3=${T
    </FIELD>
    <FIELD name="TYC3" ucd="ID_ALTERNATIVE" datatype="I" width="1">
```

```
      <DESCRIPTION>? TYC3 number from Tycho-2</DESCRIPTION>
    </FIELD>

<DATA><CSV headlines="3"><![CDATA[
recno RAJ2000 DEJ2000 Plx e_Plx pmRA pmDE Bmag Vmag SpType TYC1 TYC2 TYC3
deg deg mas mas mas/yr mas/yr mag mag
--------------------------------------------------------------------------------------------
       1   0.00338579 +72.16412166                  -5.79    12.96 13.255 10.773             4302  1966 1
       2   0.00439113 +76.40701777                   4.19     1.07 13.059 11.455             4492  2249 1
       3   0.00679762 +70.88723796  182.10  47.40   -53.92    17.65 11.844 10.937             4298  1232 1
       4   0.00855139 +73.66289182                  -1.05    -2.98 12.994 11.923             4306  1190 1
       5   0.01247266 +75.27709117                   2.33    -2.68 13.300 12.721             4492    18 1
       6   0.01908149 +83.10556149   -1.50  17.10     6.11    -6.94 11.031  9.319 K5          4615   214 1
       7   0.02051416 +77.79328652                  -4.36    -4.98 13.005 12.868             4496  1722 1
       8   0.02731868 +75.48327160   -1.39  19.50    35.63     6.75 10.595 10.200             4492   308 1
       9   0.05026688 +77.75313183                  12.23     2.91 12.878 11.680             4496  2025 1
      10   0.05526252 +73.08768283                  15.04     3.91 12.437 11.557             4302   921 1
]]>
</CSV></DATA>
</TABLE>

</RESOURCE>
</ASTRO>
```

B Poster JMMC Evolutive Search Calibrator Tool

C La documentation Doxygen de Search Calibrators

D Le code source de Search Calibrators objets brillants