



UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

# Implémentation d'un service web de réduction de données de la mission XMM

Rapport de stage ST42 - A2022

**VIALA Alexandre**

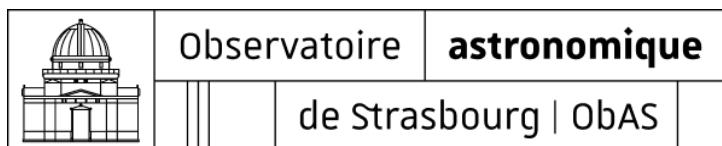
Département Informatique

## Observatoire Astronomique de Strasbourg

11 rue de L'Université  
67 000 Strasbourg  
[astro.unistra.fr](http://astro.unistra.fr)

Tuteur en entreprise  
**MICHEL Laurent**

Suiveur UTBM  
**GAUD Nicolas**





# Remerciements

Tout d'abord je souhaiterais remercier mon tuteur de stage, M. Laurent MICHEL, ingénieur de recherche au sein de l'équipe GALHECOS de l'observatoire astronomique de Strasbourg. Il m'a aidé tout au long de mon stage en me présentant en détail mes missions et en m'aidant toujours à améliorer les solutions que je concevais. Je le remercie également pour avoir relu mon rapport et ma soutenance de stage en me donnant de nombreux conseils pour les concevoir et les améliorer.

Je tiens également à remercier les astrophysiciens de l'équipe SSC-XMM, MM. Georgios VASILOPOULOS, Pierre MAGGI et Christian MOTCH qui m'ont aidé à concevoir des logiciels correspondant à leurs besoins. Je les remercie également pour m'avoir expliqué à plusieurs reprises des notions d'astronomie des rayons X qui m'ont permis d'élargir ma culture générale tout en comprenant mieux l'intérêt de ma mission.

Je tiens également à remercier M. André SCHAAFF, ingénieur de recherche au CDS, pour avoir organisé à plusieurs reprises des rencontres avec les personnels de l'observatoire astronomique. Cela m'a permis d'en découvrir davantage sur les corps de métiers présents dans une telle infrastructure.

Je souhaiterais remercier l'Observatoire Astronomique de Strasbourg et son Directeur M. Pierre-Alain DUC pour m'avoir permis de réaliser ce stage au sein de leurs locaux.

Je remercie également Mme. Mireille JACQUOT, responsable du service des stages en informatique à l'UTBM, pour avoir suivi le processus de recherche de stage et M. Nicolas GAUD, mon enseignant-suiveur, pour avoir veillé au bon déroulement de mon stage.

Enfin j'exprime ma gratitude envers toutes les personnes qui m'ont aidé durant ce stage. Merci notamment aux ingénieurs de recherche du CDS et de l'équipe GALHECOS qui m'ont régulièrement donné leurs conseils avisés pour résoudre certains de mes problèmes et pour améliorer les logiciels que j'ai conçus.

## I.Table des matières

Remerciements.....	1
I. Introduction .....	4
II. Présentation de l'Entreprise .....	4
1. Historique et évolution .....	4
2. Activités.....	5
3. Équipes au sein de l'Observatoire .....	6
III. Présentation du Sujet de Stage .....	7
4. L'astronomie X .....	7
a. Le but de l'astronomie X .....	7
b. La récupération des photons.....	8
c. Le traitement des données X (du photon au spectre).....	9
d. La mission XMM-Newton .....	9
Les objectifs de ma mission.....	10
5. Outils et protocoles.....	11
a. Outils génériques .....	11
b. High Energy Science Software (HEASoft).....	13
c. Science Analysis Software (SAS) .....	14
d. Multi Order Coverage Map (MOC) .....	14
IV. Organisation du travail.....	15
XCatDB .....	15
Architecture & fonctionnalités.....	16
AliX .....	17
Méthodologie.....	18
Planning du stage .....	18
Co-encadrement scientifique.....	19
Organisation au quotidien .....	19
Communication avec l'équipe .....	20
Réunions hebdomadaires .....	20
“peer programming” pour concevoir les scripts .....	20
Utilisation de gitlab pour les issues .....	21
V. Tâches réalisées.....	22
Indexation des observations .....	22
a. Préambule .....	22
b. Les MOCSet .....	23

c. Analyse .....	24
d. Conception .....	26
e. Implémentation .....	28
Adaptation de l'outil de tracé de région d'AliX .....	30
a. Préambule .....	30
b. Analyse .....	30
c. Conception .....	33
d. Implémentation .....	37
Le Service AMORA .....	38
a. Préambule .....	38
b. Analyse .....	39
c. Conception .....	41
d. Implémentation .....	46
VI. Conclusion .....	48
1. Travail accompli .....	48
2. Prospectives.....	48
VII. Bibliographie .....	50
Documents consultés .....	50
Illustrations utilisées .....	51

# I. Introduction

Dans le cadre de ma deuxième année en école d'ingénieur à l'Université de Technologie de Belfort-Montbéliard en spécialité Informatique, j'ai eu l'occasion d'effectuer mon stage de ST40 à l'Observatoire Astronomique de Strasbourg. Le stage de ST40 est un stage long d'une durée de 24 semaines en entreprise en tant qu'assistant ingénieur. J'ai pris mes fonctions le 6 septembre 2022 jusqu'au 10 février 2023.

Lors de ce stage, j'ai été amené à épauler le travail de mon maître de stage, M. Laurent MICHEL, en ajoutant des fonctionnalités sur un projet existant : la XCatDB. Il s'agit d'un système de base de données créé par M. MICHEL, dont le but est de donner un accès complet aux données du télescope à rayons X, XMM-Newton.

Durant ces quelques mois à l'observatoire, j'ai été chargé d'implémenter une extension de la XCatDB permettant d'appliquer des traitements scientifiques sur des régions du ciel tracées à l'écran.

Pour présenter mon stage, nous procéderons en plusieurs étapes. Nous commencerons par présenter l'Observatoire Astronomique de Strasbourg, ses équipes et les gens qui y travaillent. Nous présenterons ensuite le sujet de mon stage et tous les prérequis nécessaires à la compréhension de ce rapport. Nous détaillerons les démarches suivies afin de réaliser mes tâches et nous évoquerons les outils et protocoles auxquels j'ai eu recours durant cette période. Nous présenterons ensuite les tâches que j'ai été amené à réaliser et nous conclurons en prenant le soin de parler de l'état de mon travail à l'achèvement de ces 6 mois.

# II. Présentation de l'Entreprise

## 1. Historique et évolution

L'observatoire astronomique de Strasbourg est un observatoire inauguré en 1881. A cette époque l'Alsace est gouvernée par la Prusse puis l'empire Allemand (l'occupation a lieu de 1870 à 1918 et la Prusse devient l'empire allemand en 1871). Strasbourg, quant à elle, est la capitale du Reichsland d'Alsace-Lorraine.

Pendant cette période, d'autres bâtiments historiques ont été construits comme le palais universitaire, la gare centrale ou encore la place de la République et ses bâtiments. Tous ces édifices sont venus étendre la ville de Strasbourg d'alors en formant un nouveau quartier : la Neustadt. L'observatoire de Strasbourg a été conçu par l'architecte Hermann Eggert en collaboration avec l'astronome August Winnecke afin de construire un bâtiment faisant rayonner l'architecture et la science allemande. La construction de l'édifice se fera de 1876 à 1880. A son inauguration, le bâtiment sera composé de trois bâtiments : Une Grande-Coupole, dotée d'une ouverture pour utiliser un télescope, un bâtiment en longueur dans le sens de l'axe nord-sud disposant de deux coupoles (le bâtiment est) et d'un bâtiment disposant de bureaux et



Figure 1 : Tour de l'hôpital civil

d'habitats (le bâtiment sud). Ces trois bâtiments sont reliés par un couloir en Y. Aujourd'hui une verrière a été ajoutée au bâtiment de la coupole afin d'étendre l'amphithéâtre de ce bâtiment.

Cet observatoire n'est pas vraiment l'unique à avoir été construit à Strasbourg. On retrouve l'édification d'un premier observatoire en 1673 sur l'une des tours d'enceinte de la ville à l'entrée de l'hôpital. Aujourd'hui on peut encore observer cette tour bien qu'elle ne soit plus vraiment à l'extérieur de la ville. Elle se trouve à côté de l'entrée du nouvel hôpital civil de Strasbourg. Un deuxième observatoire a par ailleurs été construit sur le toit de l'actuelle académie de Strasbourg en 1828. Elle abritait alors la faculté de Sciences et de Médecine.

Au sein de la grande coupole, on trouve une lunette de 48.7 cm d'ouverture et 7 m de focale. Cette lunette, appelée « Grand Réfracteur » a été installée en 1877, elle était alors la plus grande lunette d'Europe. Il est possible d'ouvrir la grande coupole à l'aide d'un rail pour dégager l'horizon pour une plus petite lunette. La coupole peut, en temps normal, être visitée par le grand public mais depuis le deuxième semestre 2022, la coupole est en travaux pour rénovation. Les coupoles du bâtiment Est sont également équipées de télescope : deux lunettes de 36 cm équipées d'une caméra CCD et d'un spectrographe et une lunette de 21 cm.

L'observatoire a souvent changé de nationalité au cours de son histoire à cause des multiples changements de nationalité de la région Alsace. Pendant l'occupation allemande de la Seconde Guerre Mondiale, un directeur allemand fut mis au poste de l'observatoire tandis que le directeur français, André Danjon, continuait de diriger ses équipes à Clermont-Ferrand, lieu où l'université de Strasbourg s'était réfugiée.

En 1981, un planétarium a été installé dans une partie inutilisée des locaux du bâtiment Est de l'observatoire.

## 2. Activités

L'Observatoire astronomique de Strasbourg est un Observatoire des Sciences de l'Univers (OSU). En cette qualité, il doit remplir différentes missions. Il doit en premier lieu permettre d'observer le ciel. Bien que des lunettes soient présentes dans les locaux de l'Observatoire, elles ne permettent pas d'effectuer des recherches. Toutefois, l'observatoire astronomique de Strasbourg est particulièrement impliqué dans plusieurs services nationaux d'observation (SNO) de l'Institut National des Sciences de l'Univers (INSU). L'observatoire est tout particulièrement impliqué dans trois de ces services : Le Centre de Données astronomiques de Strasbourg, le Survey Science Centre du Satellite XMM-Newton et le segment sol de la caméra MXT du satellite SVOM.

L'observatoire astronomique doit ensuite remplir des fonctions d'enseignement en association avec l'Université de Strasbourg. L'observatoire est associé dans ce cadre à l'Ecole Doctorale de Physique et de Chimie Physique pour la formation de doctorants.



Figure 2: Bâtiment Est et Grande Coupole de l'Observatoire

L'observatoire présente une importante activité de recherche et de services grâce aux nombreux documentalistes, chercheurs et ingénieurs qui y travaillent. Les recherches sont axés autour de trois grands axes scientifiques :

- Astrophysique des hautes énergies : physique des astres compacts en fin d'évolution, accrétion, éjection et phénomènes magnétohydrodynamiques.
- Galaxies : populations stellaires, propriétés chimiques et dynamiques de la Galaxie et des galaxies proches, milieu intergalactique, grandes structures, et dynamique gravitationnelle.
- Méthodes de gestion de l'information et exploitation scientifique des grands relevés, en relation avec le centre de données astronomiques de Strasbourg.

Les équipes de recherche du CDS, entretiennent notamment leurs bases de données (VizieR et Simbad) en répertoriant de nombreuses publications scientifiques chaque semaine.

Pour accompagner cette activité de recherche, les équipes de l'observatoire astronomique organisent de nombreuses actions afin de sensibiliser le grand public. Lors de cet automne 2022, une éclipse partielle du soleil, particulièrement visible à Strasbourg pouvait être observée le 25 octobre. A cette occasion, l'observatoire astronomique, en collaboration avec le planétarium de Strasbourg, a organisé différentes activités ouvertes au public avec la mise à disposition d'outils permettant de regarder l'éclipse sans séquelle.

Des séminaires sont également organisés tous les vendredis dans l'amphithéâtre de la grande coupole et présentés, en règle générale, par des membres de la communauté astrophysique. L'intervenant présente alors les résultats de ses recherches avant d'échanger avec l'audience pendant une séance de questions-réponses.

### 3. Équipes au sein de l'Observatoire

L'Observatoire astronomique de Strasbourg présente de nombreuses activités. Toutefois, les effectifs de l'Observatoire, environ 80 permanents et 20 temporaires, sont répartis en deux équipes :

- Le Centre de Données de Strasbourg (CDS) chargé de développer des services pour la communauté scientifique internationale, qu'elle soit amateur ou professionnelle. Le CDS est notamment chargé de l'entretien de Simbad, VizieR et Aladin. Ces trois logiciels sont interconnectés entre eux et permettent de parcourir des catalogues d'astres célestes de diverses manières.
- L'équipe "Galaxies, High Energy, Cosmology, Compact Objects & Stars" (GALHECOS) chargée de l'étude des galaxies par l'étude des astres célestes les composant. Dans cette équipe, le groupe Hautes Énergies s'intéresse tout particulièrement aux sources émettant des rayons X, aux objets compacts (étoiles à neutron, naines blanches, etc.) et aux noyaux actifs de galaxie (émission dans les ondes radios).

Au sein de l'équipe GALHECOS, on retrouve les deux SNO (Services Nationaux d'Observation) mentionnés plus haut, le SSC XMM-Newton et SVOM :

- L'équipe du Survey Science Centre (SSC) du satellite XMM-Newton (SSC-XMM) a pour mission de développer et de maintenir en opération un noeud du segment sol de la mission qui corrèle les données du satellite avec les données du CDS. Il a en outre la charge d'une interface donnant accès aux données de ma mission, il s'agit de la XCatDB sur laquelle j'ai travaillé durant mon stage.

- L'équipe SVOM travaille en collaboration avec la Chine sur le segment sol de la mission SVOM. Cette mission a pour objectif d'observer les explosions d'étoiles les plus lointaines, ce que l'on appelle des sursauts gamma. A cet effet, le satellite d'observation est équipé d'une caméra à rayons X, le MXT, d'une caméra infrarouge, et de caméras à rayons gamma. Une batterie d'instruments au sol complète la mission. Les compétences acquises à l'Observatoire dans le cadre de XMM ont été réinvesties pour assurer le développement du traitement sol des données du MXT. Cette mission sera lancée en 2024.

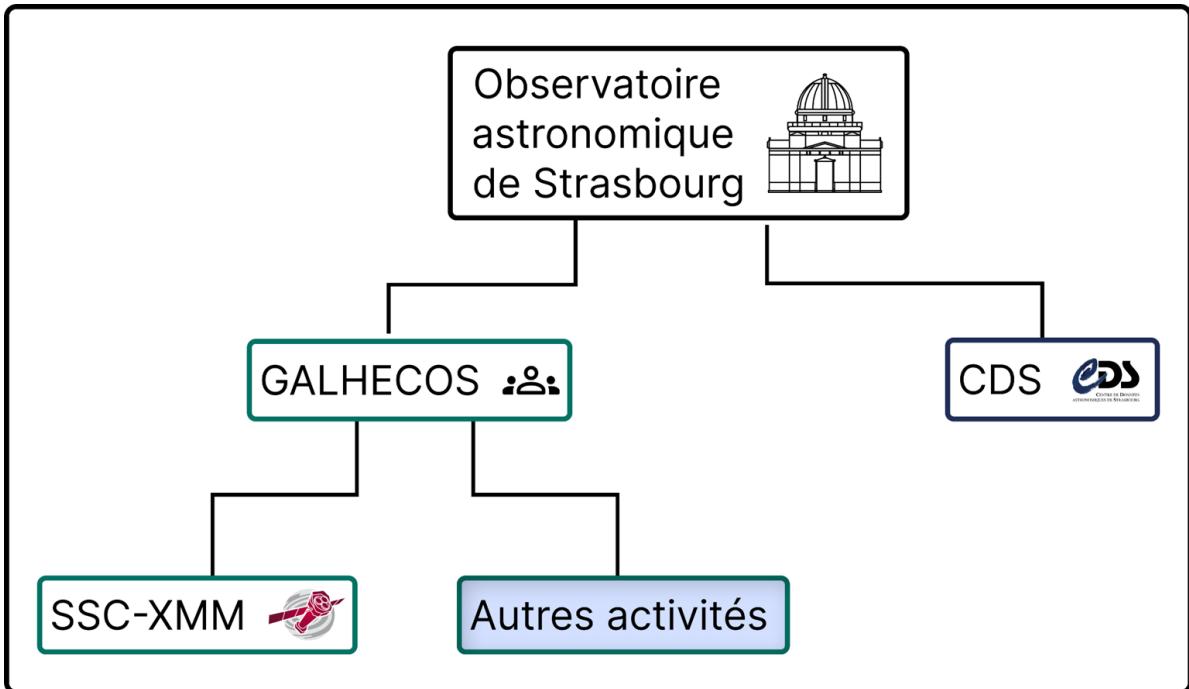


Figure 3: Organigramme de l'Observatoire astronomique de Strasbourg

### III. Présentation du Sujet de Stage

#### 4. L'astronomie X

Au cours de mon stage à l'Observatoire Astronomique de Strasbourg, j'ai dû intégrer le XSS-XMM-Newton. Dans ce contexte j'ai dû me familiariser avec les bases de l'astronomie X.

##### a. Le but de l'astronomie X

L'astronomie X est l'étude des objets célestes et des phénomènes qui émettent des rayons X. Ces rayons X sont une forme de radiations électromagnétiques avec une longueur d'onde comprise entre **0.01** et **10 nanomètres** (nm). Cette zone du spectre des ondes est comprise entre les régions des rayons ultraviolets et la région des rayons gamma.

L'astronomie X se porte sur l'observation d'une large variété d'objets célestes et de phénomènes comme les **trous noirs**, les **étoiles à neutrons**, les **restes de supernova** et les **noyaux actifs de Galaxie**. Les rayons X sont également émis par les **gaz à haute température** dans les amas de galaxies et par certaines étoiles, en particulier, les étoiles de type solaire. Les rayons servent enfin à étudier les propriétés du milieu intergalactique.

Les sources de rayons X sont essentiellement observées à l'aide de télescopes spatiaux étant donné que l'atmosphère terrestre absorbe lesdits rayons. Il n'existe aucun observatoire terrestre pouvant observer les rayons X. Toutefois, certains rayons gamma de haute énergie peuvent créer des particules dans la haute atmosphère. Ces dernières peuvent être alors observées depuis le sol depuis observatoires bien spécifiques : les observatoires de Cerenkov atmosphériques.

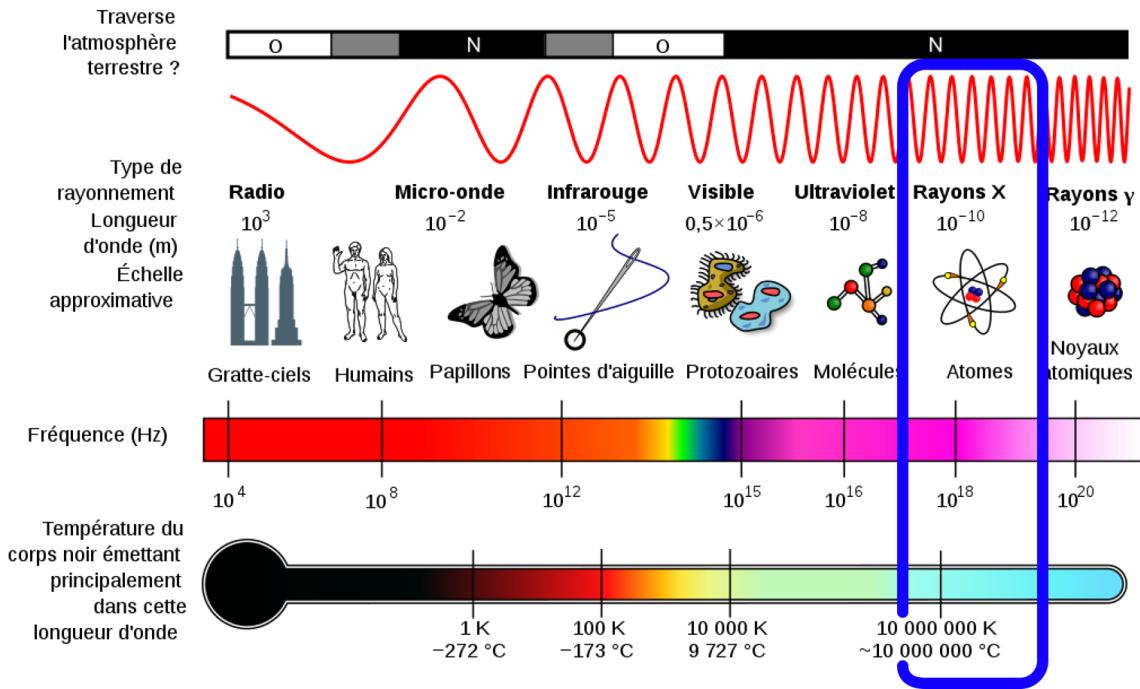


Figure 4: Spectre des ondes

## b. La récupération des photons

Grâce à l'astronomie X, nous sommes donc capables d'observer un grand nombre d'objets célestes montrant des conditions de température, de densité, ou de gravité extrême. Du fait de la complexité de l'interaction des photons X avec les pixels des CDDs de la caméra et en tenant compte du faible flux de photons, la création d'images est plutôt réalisée au sol à partir des listes des évènements détectés par la caméra (les event-lists). Un événement contient l'identification du pixel touché (2D), l'amplitude du signal électrique qui est relié à l'énergie du photon X, ainsi que la date précise. Quelle que soit la manière de générer une image, il est d'abord nécessaire de **focaliser** ces rayons. Pour cela, on utilise un télescope à forme concave pour que chaque point du capteur reçoive des photons provenant d'un seul point du ciel. Ainsi, on arrive à identifier d'où vient le photon qui a été récupéré sur le capteur.

La focalisation dans le domaine des rayons X n'est pas simple à mettre en œuvre. Contrairement à la lumière visible, les rayons X ne peuvent pas être réfléchis par des miroirs du fait de leur longueur d'onde subatomique. La déviation de rayons X peut toutefois se faire sous incidence rasante. Après une double réflexion la focalisation peut se faire. Cette configuration, dite de télescope de Wolter, est utilisée par les télescopes de XMM-Newton (Figures 3 & 4), qui comporte en plus des miroirs concentriques emboîtés afin d'augmenter la surface réflectrice totale.

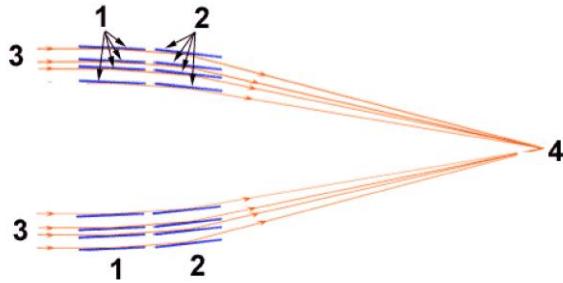


Figure 5: Miroir de Wolter

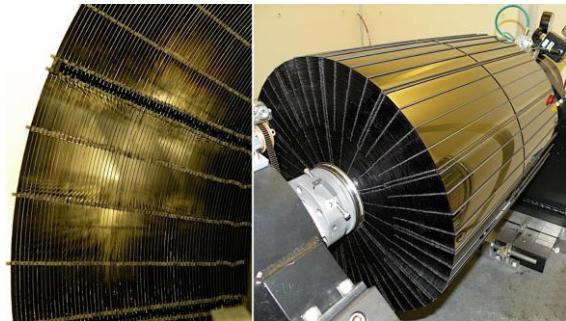


Figure 6: Miroirs de Newton-XMM

### c. Le traitement des données X (du photon au spectre)

Une fois que les photons sont entrés en contact avec un ou des capteurs CCD, il faut réussir à les caractériser. Plusieurs obstacles se présentent alors :

- Un photon peut entrer en contact avec plusieurs pixels voisins. Il faut pouvoir identifier le nombre de photons détectés.
- un photon arrachera plus ou moins de charge électrique au capteur en fonction de son énergie. Il faut pouvoir identifier la quantité de charge afin de mesurer l'énergie du photon incident.
- Le problème de "pile-up", plusieurs photons peuvent arriver sur un capteur pendant la même trame caméra (en général de 73 ms et 2.6 s pour les détecteurs à bord de XMM-Newton), ou sur deux pixels contigus. Il faut pouvoir corriger cet effet.

Compte tenu de ces contraintes, il est nécessaire de réaliser des études statistiques sur les signaux qui ont été récupérés sur les CCD à un instant t.

### d. La mission XMM-Newton

XMM-Newton a été déployé par l'ESA le 10 décembre 1999 depuis la station Kourou en Guyane Française. Le satellite a été propulsé en orbite à l'aide d'une fusée Ariane 5 et est toujours en activité aujourd'hui. Afin de focaliser les rayons X, le satellite est doté de trois télescopes identiques, chacun comportant 58 cylindres en nickel recouvert d'une fine couche d'or augmentant la réflectivité, déviant les rayons X sur les trois différentes caméras associées à chaque télescope. Ces caméras sont séparées en 2 types : les caméras EMOS (EMOS-1 et EMOS-2) et la caméra EPN. Grâce à ces outils, il est possible d'observer la position d'objets célestes mais également la luminosité, la variabilité et les caractéristiques spectrales de ces sources.

Le satellite transmet des données brutes qui sont traitées au sol. C'est l'objectif du Survey Science Center (SSC). Le SSC est un consortium de 8 instituts européens nommé par l'ESA. Ces instituts sont situés en Allemagne, France, Royaume-Uni et Espagne. La recherche est dirigée par l'Institut de Recherche en Astrophysique et Planétologie de Toulouse. Cet institut a pour rôle de coordonner les actions des autres instituts afin de fixer les objectifs de recherches. Ces objectifs peuvent être de 4 natures différentes :

- Fournir des logiciels pour traiter et analyser les données du XMM-Newton
- Traiter l'ensemble des données
- Construire des catalogues de sources X détectés par l'observatoire
- Analyser scientifiquement les sources pour essayer de comprendre leur nature (étoile à neutron, étoile, supernova, etc.)

Aujourd’hui, grâce aux efforts conjoints des différents instituts du SSC-XMM, les logiciels permettant de traiter et d’analyser les données sont nombreux. On trouve aussi bien des logiciels comme la XCatDB qui permettent d’accéder aux données facilement que des logiciels qui permettent de générer des spectres d’une zone du ciel (cf. § II.3.g). C’est pourquoi les efforts du consortium sont d’abord centrés sur la compilation d’un catalogue à intervalles de temps réguliers (le dernier en date est le catalogue<sup>1</sup> 4XMM dr12 sorti en Juillet 2022). Les efforts de recherche portent à la fois sur l’étude de sources particulières observées par XMM que sur l’étude statistique de population d’étoiles référencées par le catalogue du SSC.

## Les objectifs de ma mission

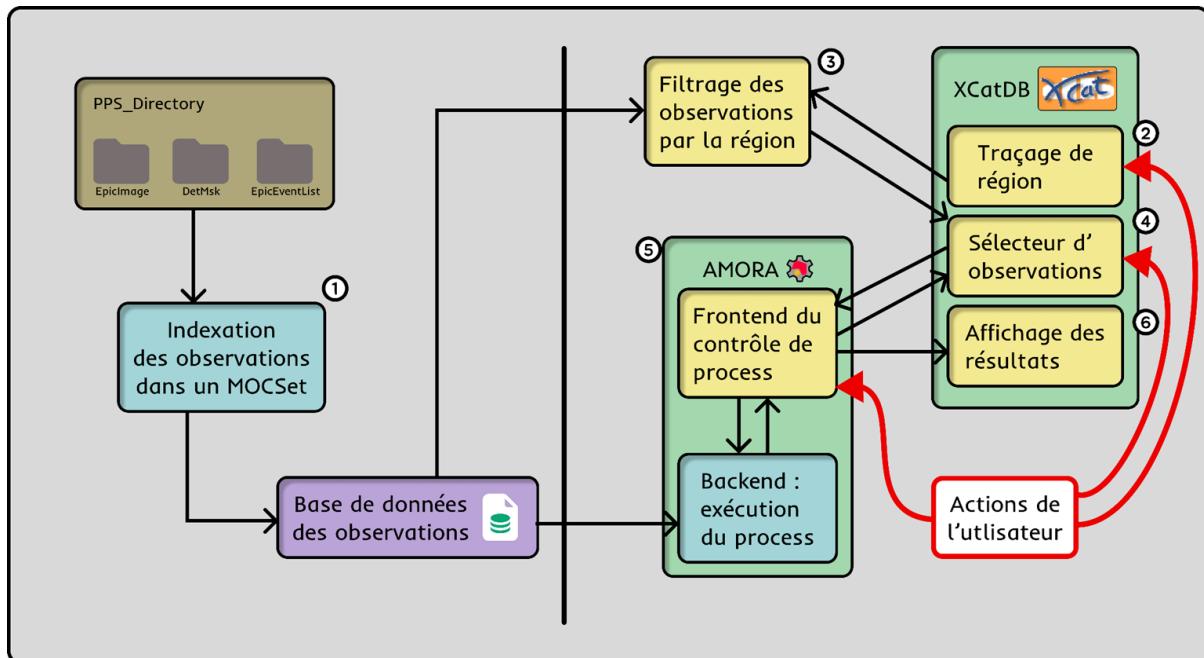


Figure 7: Structure finale du projet

Au cours de ses années d’activité, le satellite Newton-XMM a récolté un grand nombre de données. Toutes ces données sont traitées par les scientifiques et elles sont ensuite disponibles par l’intermédiaire du logiciel XCatDB. On peut ainsi visualiser les différentes sources de rayons X, les classer et obtenir des produits scientifiques. Néanmoins, il peut arriver, qu’un utilisateur souhaite explorer une région du ciel qui ne contient pas forcément de source détectée par le traitement automatique du SSC.

L’objectif de mon stage fut de concevoir une nouvelle fonctionnalité, intégrée dans la XCatDB, permettant de réaliser des traitements sur des zones du ciel préalablement tracées par l’utilisateur. Pour réaliser cette tâche le problème a dû être séparé en plusieurs sous-fonctions :

- (1) L’Indexation spatiale. Il était nécessaire d’indexer les nombreuses observations du Newton-XMM dans une structure de donnée afin de récupérer certaines observations

<sup>1</sup> Un catalogue astronomique est une liste de différents objets astronomiques observés d’une manière similaire. Ces catalogues sont entièrement consultables sur des logiciels du CDS comme Simbad ou VizieR. On peut également attacher d’autres produits scientifiques (images spectres...) aux données des catalogues.

par requête quasi instantanément. Nous avons utilisé pour cela les MOCSet<sup>2</sup>. (Voir section [Indexation des observations](#)).

- **(2)** L'amélioration de l'outil de tracé de région. Il était nécessaire d'ajouter une fonctionnalité de traçage de cercle ainsi qu'une fonctionnalité de trace de "bruits de fonds" de l'observation. (Voir section [Adaptation de l'outil de tracé de région d'AliX](#)).
- **(3)** Le Serveur MOCSet. Il s'agit d'un endpoint de la REST API qui permet de filtrer les observations à l'aide de la région sélectionnée. Cette région va être utilisée pour réaliser une requête sur le MOCSet précédemment créée. (Voir sections [Indexation des observations](#) et [Le Service AMORA](#)).
- **(4)** Le Sélecteur d'observations. L'utilisateur va parfois vouloir sélectionner des observations particulières parmi celles déjà filtrées. Ce sélecteur va permettre de sélectionner manuellement des observations. Il allie interface graphique et endpoint de la REST API pour fonctionner. (Voir section [Le Service AMORA](#)).
- **(5) / (6)** Le service AMORA. Pour pouvoir effectuer des traitements sur les données sélectionnées, le service AMORA (Asynchronous Multi Observation Region-based Analysis) a dû être créé. Il permet de contrôler un processus (script bash ou python) à distance. (Voir section [Le Service AMORA](#)).

## 5. Outils et protocoles

Au cours de mon stage j'ai été amené à utiliser différents outils et protocoles.

Certains sont spécifiques à l'astronomie X tandis que d'autres ont un usage plus générique. Les outils plus génériques seront présentés succinctement dans le tableau ci-dessous. Veuillez consulter les annexes pour plus d'informations sur ces logiciels. Les logiciels plus spécifiques seront présentés après les outils génériques.

### a. Outils génériques

Outil	Type	Description
Git	Gestionnaire de sources	Un outil en lignes de commande permettant de gérer des versions de fichiers.
Gitlab	Logiciel de forge en ligne	Logiciel permettant d'héberger des versions de fichiers en ligne permettant ainsi la collaboration entre différents utilisateurs. Ce logiciel supporte le gestionnaire de version git.
Eclipse	Environnement de Développement Intégré	Un environnement de développement utilisé pour divers langages de

---

<sup>2</sup> Fichier binaire permettant de stocker des observations dans une table. Il est alors possible de requêter ce fichier en indiquant une position du ciel et d'obtenir toutes les observations qui incluent cette position.

		programmation dont Javascript et Java.
Visual Studio Code	Éditeur de code	Visual Studio Code est un éditeur de code disposant de nombreuses extensions permettant de l'utiliser dans de nombreux langages de programmation dont Python et Javascript.
Python	Langage de programmation	Langage de programmation multiparadigmes doté d'une grande quantité de bibliothèques.
Flask	Framework Python	Framework Python permettant d'héberger un site web.
Jinja	Moteur de template	Moteur de templates HTML utilisé par différents framework Python. Il est en l'occurrence utilisé avec Flask.
HTML	Langage de balisage	Langage de balisage permettant de concevoir l'architecture visuelle d'une page internet.
CSS	Langage de feuille de style	Langage permettant de décrire par des propriétés, l'apparence de balises HTML.
Javascript	Langage de programmation	Langage de programmation multiparadigmes utilisé notamment pour rendre les pages HTML dynamiques.
JQuery	Bibliothèque Javascript	JQuery est une bibliothèque grandement utilisée. Elle ajoute, par exemple, des fonctions permettant de raccourcir la syntaxe Javascript classique.
Bootstrap	Bibliothèque Javascript, HTML et CSS	Bootstrap fournit un ensemble cohérent de composants classiques de page internet. On retrouve notamment des accordéons, des boutons, etc.

DataTable	Bibliothèque Javascript	Bibliothèque permettant d'ajouter des fonctionnalités sur un tableau HTML comme un tri par colonne, une barre de recherche, une pagination améliorée, etc.
SQLite	Moteur de base de données	Moteur permettant de créer des bases de données relationnelles sous la forme d'un fichier binaire local.

## b. High Energy Science Software (HEASoft)

L'astronomie X utilise une variété d'outils afin d'observer les phénomènes dans les rayons X. Ces outils sont pour la plupart des logiciels informatiques en ligne de commande et sont disponibles dans la suite de logiciels HEASoft. Cette suite de logiciels a été créée par la NASA dans les années 80, et plus spécifiquement par le Centre de Recherches des Archives Scientifiques sur les Hautes Énergies (HEASARC).

Mon projet étant centré sur le traitement de données X, j'ai été amené à utiliser partiellement ces logiciels. Mon utilisation de ce logiciel est toutefois restée très sommaire. Pour les utiliser correctement il est nécessaire d'avoir de grandes connaissances en astronomie X, ce qui n'était pas vraiment mon cas. J'ai surtout utilisé l'un de ces logiciels : fv.

fv est un logiciel, assez ancien mais toujours utilisé, doté d'une interface graphique qui permet de visualiser les données d'un produit scientifique<sup>3</sup> au format FITS. Il permet notamment de visualiser les métadonnées présentes dans le "header" et dans les extensions du produit.

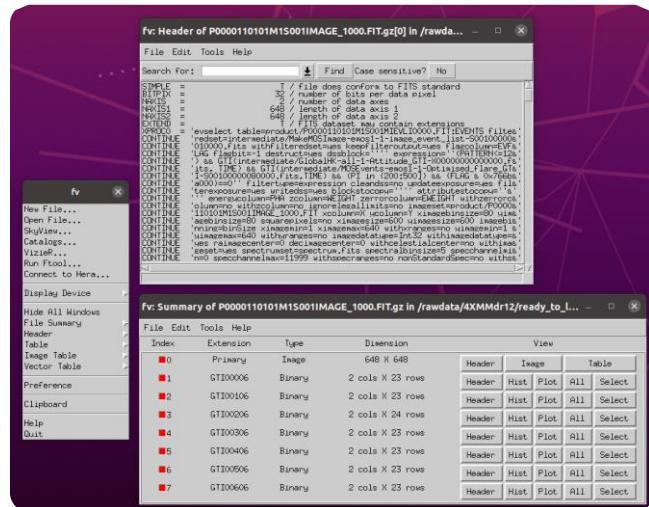


Figure 8: Interface de l'outil fv par HEASoft

<sup>3</sup> Un produit scientifique est un fichier de type "Flexible Image Transport System". Il s'agit d'un standard en astronomie permettant de stocker différents types de données astronomiques. Ces fichiers comportent notamment un "header" comportant des métadonnées communes et des extensions contenant des jeux de données plus spécifiques. L'extension de ces fichiers est .fits, .fit ou .fts. Plus d'informations sont disponibles en ligne à l'adresse [https://fr.wikipedia.org/wiki/Flexible\\_Image\\_Transport\\_System](https://fr.wikipedia.org/wiki/Flexible_Image_Transport_System)

### c. Science Analysis Software (SAS)

Pour compléter HEASoft, la mission Newton-XMM dispose d'outils supplémentaires spécifiquement conçus pour traiter les données de ce satellite. Ces outils sont rassemblés dans une suite de logiciels appelée SAS (Science Analysis Software). Elle a été créée par l'Agence Spatiale Européenne (ESA) et dispose de solutions complémentaires à celles de HEASoft.

Dans le cadre de mon stage, je n'ai quasiment pas eu recours directement à la SAS. J'ai toutefois dû mettre en œuvre des scripts shell<sup>4</sup> utilisant les principales fonctionnalités de la SAS.

### d. Multi Order Coverage Map (MOC)

Finalement, parmi les outils que j'ai utilisés, le Multi Order Coverage Map, est celui le plus spécifique au domaine de l'astronomie. Les MOC, sont des structures de données définies par le VO<sup>5</sup>. Pour comprendre ce que sont ces MOC, il est nécessaire de bien comprendre comment sont référencés les observations du ciel.

Les observations astronomiques se font en regardant le ciel. Le référentiel de ces observations est en général la Terre. On représente les observations comme faisant partie d'une sphère entourant la Terre. Sur cette sphère, on détermine les coordonnées des points du ciel à l'aide de l'ascension droite (right ascension ou ra) et de la déclinaison (declination ou dec) qui correspondent à la latitude et la longitude respectivement. Ra et dec sont toujours exprimés en degrés et dans ses sous unités (la minute d'arc, la seconde d'arc, les millisecondes d'arc, etc.).

Le ciel est donc un repère non-euclidien que l'on projette en général sur un plan afin de pouvoir être affiché sur des illustrations ou un écran d'ordinateur. Afin de simplifier les repères dans l'espace, le VO a fait le choix d'effectuer une tessellation<sup>6</sup> du ciel en cellules HEALPix<sup>7</sup>.

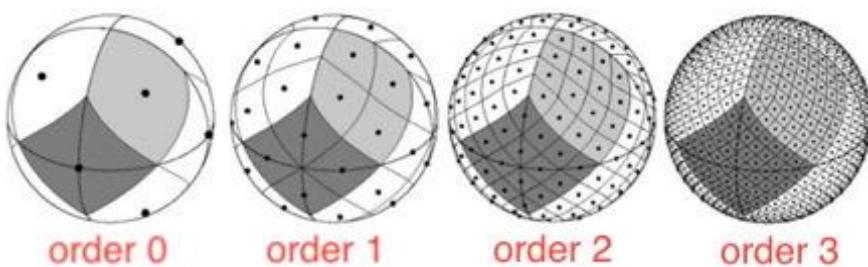


Figure 9 : Tessellation HEALPix d'une sphère

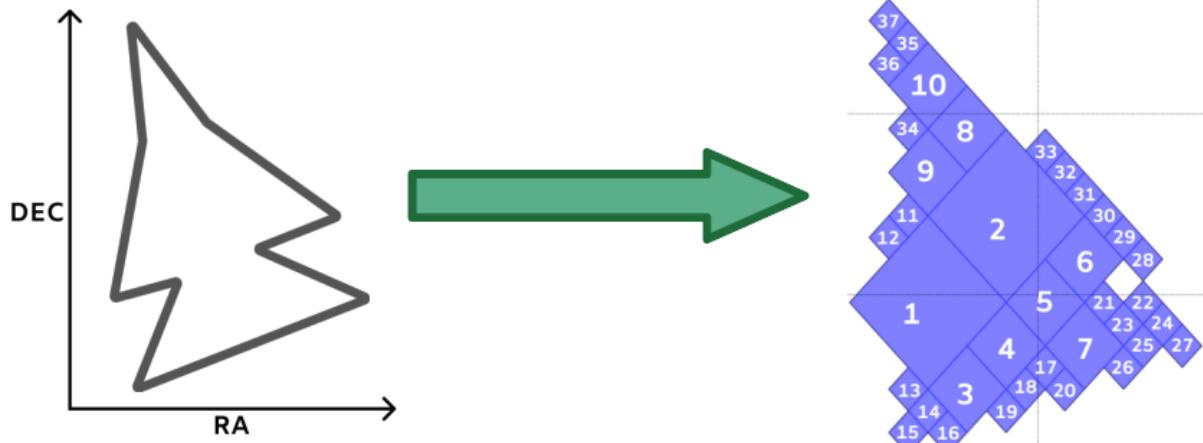
<sup>4</sup> Un script shell est une suite d'instructions pouvant être lancée sur un système d'exploitation utilisant un terminal shell. C'est notamment le cas de Linux et d'openBSD par exemple. On peut assimiler le langage shell à un langage de programmation classique avec une syntaxe.

<sup>5</sup> L'International Virtual Observatory Alliance (VO ou IVOA) est une organisation internationale ayant pour but d'établir des normes dans l'astronomie internationale.

<sup>6</sup> Une tessellation est le découpage d'une surface, souvent 3D, en éléments réguliers de base. C'est une technique souvent utilisée en informatique pour simplifier des calculs. On retrouve notamment dans les logiciels de conception assistée par ordinateur.

<sup>7</sup> Une cellule HEALPix est un standard de tessellation de sphère en losanges.

Ces cellules permettent de référencer de manière unique des pixels du ciel grâce à une numérotation conçue de telle manière que l'on peut aisément déduire la position d'une cellule à partir de son numéro. On peut donc construire des listes de pixels pour décrire le contour d'une forme dans le repère céleste. Cela permet de créer ce que l'on appelle des MOC : des simplifications de formes du ciel au moyen d'indexées de cellules HEALPix.



*Figure 10 : Simplification d'une forme par le passage en MOC (la numérotation est erronée)*

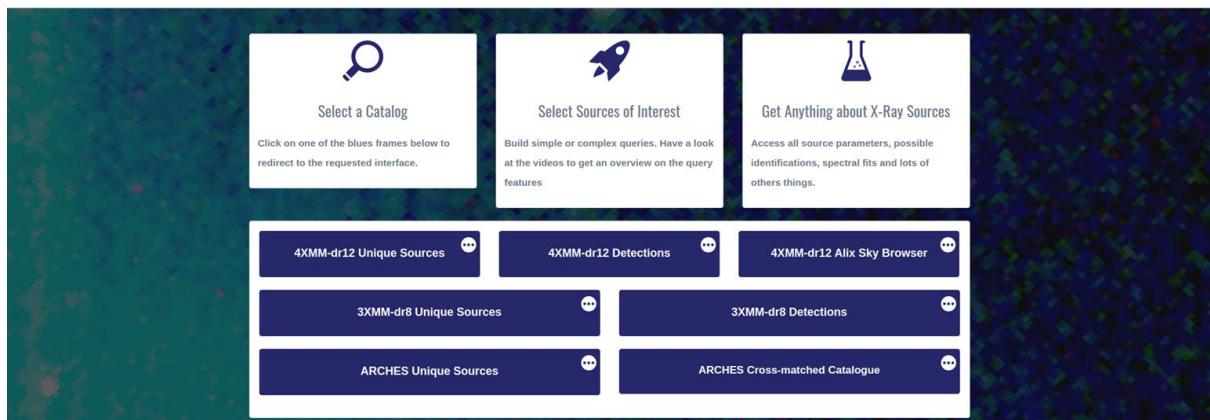
Ces MOC sont bien plus simples à traiter que des formes classiques présentant une multitude d'aspérités. Ils sont très utiles pour déterminer si deux formes se recoupent ou si l'une des formes inclut l'autre ou encore pour calculer l'union de deux formes.

## IV. Organisation du travail

### XCatDB

Le but de mon stage était d'ajouter des fonctionnalités à la XCatDB. Pour parcourir des données, XCatDB dispose de deux interfaces : l'une, dite “classique”, permet d'effectuer une requête sur les données des catalogues et l'autre est une carte du ciel interactive. Lors de mon stage, j'ai surtout été amené à collaborer sur la carte interactive du ciel.

XMM-Newton Catalogue Observatory of Strasbourg



*Figure 11 : Page d'accueil de XCatDB*

## Architecture & fonctionnalités

La XCatdb s'appuie sur une base de données Saada contenant les métadonnées des produits scientifiques, les liens qui les relient ainsi qu'un dépôt pour les fichiers bruts.

Cette base de données est dotée de deux interfaces, l'une dite classique sur laquelle je n'ai pas travaillé et l'autre qui permet une navigation plein ciel dans le style de Google map. Cette dernière permet d'afficher toutes les sources X détectées sur un fond de ciel choisi parmi 600 relevés couvrant tout le spectre électromagnétique. Il est également possible de combiner les sources X affichées avec des données issues de 22000 catalogues. Cette possibilité de mixer des données de toutes origines a été rendue possible grâce aux standards du VO et notamment grâce à la norme HiPS basée sur la HEALPix. Le module de visualisation plein ciel de la XCatDB est une adaptation d'un projet nommé Alix qui est une extension du widget Aladin-lite.

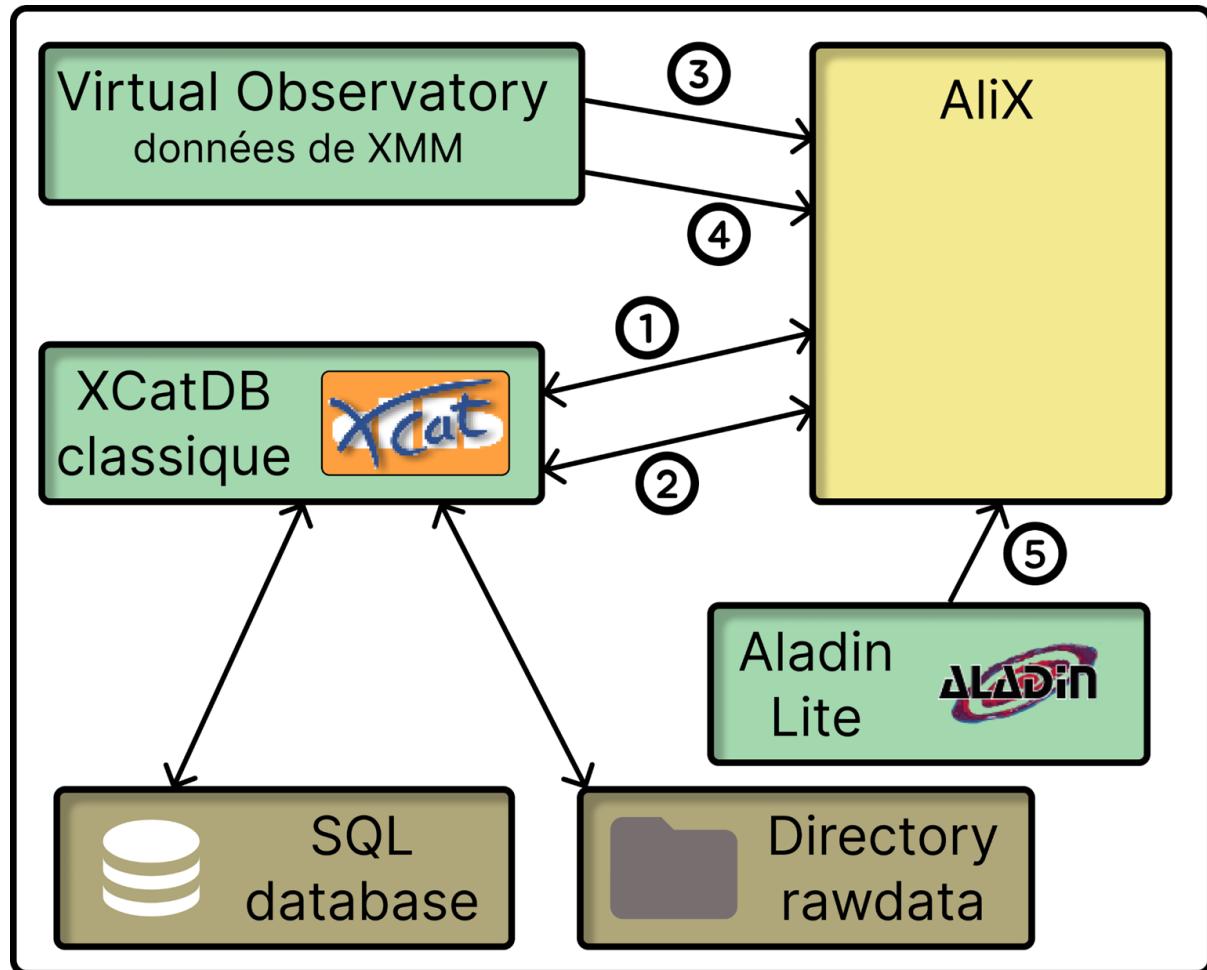


Figure 12 : Architecture de la XCatDB

La version classique de la XCatDB permet de réaliser des requêtes sur différents facteurs. On peut d'abord effectuer une recherche par rapport à une position du ciel grâce à ses coordonnées en degrés. On peut également rechercher des données en sélectionnant des particularités des sources ou des paramètres de détection.

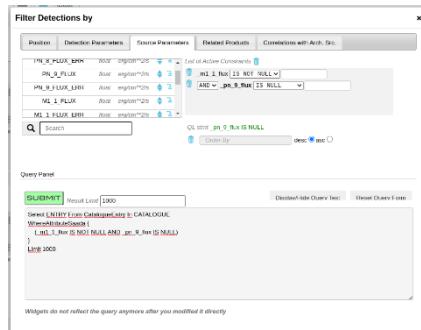


Figure 13 : Interface de requête

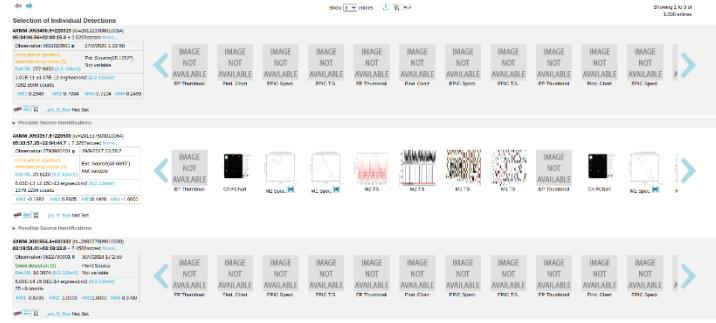


Figure 14 : Résultat de requête

## AliX

AliX est une surcouche à Aladin Lite. Elle dispose ainsi d'un explorateur de carte du ciel sur lequel il est possible d'afficher des sources provenant de divers catalogues. AliX dispose d'une barre d'outils latérale permettant de réaliser diverses actions.

Elle permet d'enregistrer une position du ciel dans un bookmark (**icône cœur orange**). Pour restaurer le bookmark, il suffit de le retrouver dans le carnet de bookmarks (**icône livre jaune**). On peut régler l'affichage des sources des différents catalogues (**icône liste violette**). On peut filtrer les observations affichées sur la carte en fonction de certains paramètres de la version classique de XCatDB (**icône entonnoir gris**). L'**icône d'image bleue** permet de changer le relevé (survey) du ciel utilisé comme fond. L'**icône viseur rouge** permet de centrer la fenêtre sur l'observation actuellement sélectionnée. Enfin l'**icône de dessin en vert** permet de tracer une région (un polygone) dans le ciel afin d'effectuer des opérations dessus. Cette dernière fonctionnalité est celle sur laquelle mon stage trouve son point d'ancre à la XCatDB. Cet éditeur de région devait être amélioré et sa sortie (un JSON) devait également se greffer sur le service que j'ai dû concevoir (AMORA).

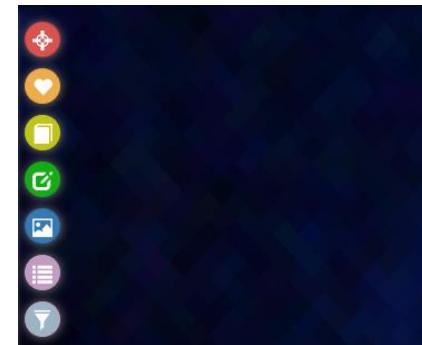


Figure 15 : Barre d'outils latérale d'AliX



Figure 16 : Interface de l'éditeur de région



Figure 17 : Interface du choix de catalogue

## Méthodologie

### Planning du stage



Figure 18 : Planning du stage

L'organisation de mon stage s'est déroulée en trois étapes bien distinctes. En premier lieu, j'ai eu le temps de m'acclimater à mon travail en réalisant un logiciel en lignes de commandes en Python. J'ai ainsi pu compléter mes connaissances en Python avant de devoir réaliser un logiciel plus complexe. J'ai également pu réaliser quelques tests sur les fonctionnalités implémentées. J'ai notamment pu déterminer des tailles de fichiers et réaliser du benchmarking<sup>8</sup> pour mesurer le temps de calcul des logiciels conçus.

En second lieu, j'ai travaillé longuement sur l'amélioration de la XCatDB par la conception d'un logiciel en ligne et l'amélioration de l'éditeur de région d'AliX. Cette tâche a été celle qui m'a pris le plus de temps sur l'intégralité de mon stage.

Enfin, à partir du 11 décembre, nous avons décidé de ne plus implémenter de nouvelles fonctionnalités. De cette manière, cela me laissait plus ou moins 2 mois pour finaliser les fonctionnalités développées.

## Co-encadrement scientifique

Ma mission au sein de l'observatoire de Strasbourg consistait à travailler sur la création d'outils destinés aux scientifiques du Consortium XMM. C'est pourquoi j'ai intégré l'équipe SSC-XMM de la division GALHECOS de l'observatoire. Au sein de cette équipe, j'ai été amené à collaborer avec des astrophysiciens et notamment à être co-encadré par M. Pierre MAGGI.

La collaboration avec les scientifiques est absolument nécessaire pour concevoir un logiciel leur étant destiné. En effet, en tant qu'ingénieur informatique, je ne dispose pas des connaissances et compétences nécessaires à la compréhension des phénomènes observables grâce aux rayons X. En outre, je ne suis pas capable d'utiliser correctement les logiciels utilisés par les astrophysiciens de l'équipe SSC-XMM (cf. § sur HEASoft & la SAS). Toutefois, grâce à mes compétences acquises au travers de mes études, je suis capable de concevoir des logiciels et des interfaces homme-machines.

Ainsi, mon rôle a été de comprendre la structure générale des scripts que concevaient les scientifiques et de les interfaçer avec le service en ligne que j'ai conçu. Puis, j'ai dû interfaçer ce service avec une page web en ligne sous la forme de fichiers html (ou jinja).

## Organisation au quotidien

Pendant mon stage, j'ai dû suivre une certaine routine sur mes tâches afin d'avoir un bon rythme. Mon tuteur de stage m'a grandement aidé à tenir ce rythme en me donnant régulièrement des tâches à réaliser sur les produits que je concevais. Ces tâches pouvaient être des rappels ou simplement de nouvelles idées qui venaient au fur et à mesure que l'on avançait sur la conception des logiciels.

Jusqu'au 11 décembre, j'effectuais chaque jour les tâches suivantes :

- Travailler sur les projets en autonomie
- Communiquer avec mon tuteur pour l'informer sur l'avancement général des fonctionnalités que j'implémentais. Ces moments d'échange permettaient également à mon tuteur de valider ou non le travail réalisé ainsi que de discuter des améliorations à apporter.

---

<sup>8</sup> Le benchmarking consiste à mesurer les performances d'une machine ou d'un logiciel en mesurant différents paramètres.

- Faire des tests sur les fonctionnalités réalisées. Ces tests étaient réalisés uniquement lorsqu'ils étaient vraiment nécessaires.
- Réaliser des pages de documentation pour les logiciels réalisés ou documenter le code.
- Prendre des notes sur les tâches réalisées dans la journée afin de pouvoir les consigner dans mon rapport de stage.

Après le 11 décembre, lorsque nous sommes rentrés dans la phase d'amélioration du service web, l'organisation de mes journées est plus ou moins restée la même mais je devais également consulter les suggestions de mes collaborateurs régulièrement par divers moyens (google doc et gitlab).

## Communication avec l'équipe

### Réunions hebdomadaires

Afin de tenir au courant régulièrement les scientifiques de l'équipe que j'ai intégrée, nous avons organisé des réunions à raison d'une fois par semaine le mercredi matin. Ces réunions ne sont arrivées qu'une fois ma phase d'acclimatation au sein de l'observatoire achevée. En effet, nous avons commencé à organiser ces réunions début octobre une fois que j'avais bien avancé sur mes tâches.

Ces réunions hebdomadaires étaient intégralement réalisées en anglais car l'un de mes collègues, Georgios VASILOPOULOS était grec. Ces réunions s'articulaient en deux étapes : Dans une première phase, je présentais le travail réalisé depuis la dernière réunion. Je prenais soin de rappeler les tâches réalisées depuis le début du stage en début de présentation, puis je présentais mon travail sur plusieurs transparents. Je profitais de ce moment pour expliquer également ce qu'était un MOC<sup>9</sup> à l'équipe. Dans un deuxième temps, une fois ma présentation terminée, nous passions à une phase de discussion qui me permettait de réexpliquer certains points si mes explications n'avaient pas été assez claires. Ces discussions permettaient également aux scientifiques de superviser mon travail afin de m'indiquer les besoins qu'ils avaient par rapport aux applications. Ces réunions permettaient également au reste de l'équipe de se mettre d'accord sur une marche à suivre.

### “peer programming” pour concevoir les scripts

À partir de fin novembre, le service que j'ai conçu commençait à être capable de faire tourner des scripts shell. J'ai donc réalisé quelques tests avec des scripts écrits par mes soins qui permettent de faire des opérations classiques (lister des fichiers, faire des copies, compresser des fichiers, etc.). Néanmoins, ces scripts ne correspondaient pas à une utilisation réelle du logiciel. Mes collègues, M. MAGGI et M. VASILOPOULOS, m'ont donc écrit quelques scripts permettant d'effectuer des tâches simples sur des observations.

Ces scripts simples m'ont permis de comprendre la structure globale que devait avoir mon logiciel pour pouvoir contrôler les paramètres des scripts. Nous avons alors commencé à travailler sur des scripts plus complets, réalisant des actions plus complexes. Sur ces scripts plus complexes, j'ai dû collaborer étroitement avec mon collègue M. VASILOPOULOS pour adapter ses scripts à mon logiciel et vice versa.

---

<sup>9</sup> Une Multi Order Coverage Map est un standard du VO. cf. Partie sur les tâches réalisées pour plus de détails.

Nous n'utilisions qu'un seul ordinateur sur lequel mon collègue me montrait d'abord la fonctionnalité à implémenter grâce à des scripts shell ou des classeurs jupyter<sup>10</sup>. Il écrivait ensuite les lignes dans le script final. J'en profitais alors pour commenter les fonctionnalités pour m'assurer de ce qui était nécessaire ou non dans le script. Je commentais également le nom des variables pour m'assurer que les noms étaient bien cohérents.

Ces sessions me permettaient de me rendre compte des fonctionnalités nécessaires pour les scientifiques. Je modifiais alors la structure du logiciel pour s'adapter au mieux aux besoins de mes collègues.

## Utilisation de gitlab pour les issues

Passé le 11 décembre, nous n'avons plus implémenté de nouvelle fonctionnalité majeure (breaking change). J'ai continué à améliorer le logiciel sur les conseils de mon tuteur de stage. Pour cela, nous avons utilisé le système d'issues de Gitlab pour déterminer les améliorations et bugs qu'il fallait apporter ou corriger.

Nous avons utilisé un système d'étiquette pour connaître le statut des issues :

- P1
- bug
- enhancement
- solved (to verify)

Figure 19 : Etiquettes des issues

- **Bug** : Correspond à un bug. Ce genre d'issues est en général prioritaire car des bugs peuvent détériorer l'expérience d'utilisation. Ils peuvent également rendre l'application inutilisable.
- **Enhancement** : Correspond aux améliorations.
- **P1** : Correspond aux issues de priorité 1. Ce sont les issues à régler en premier lieu. Cela peut se justifier car elle perturbe le fonctionnement normal de l'application ou parce que la fonctionnalité à implémenter bénéficierait beaucoup au confort d'utilisation.
- **solved (to verify)** : Correspond aux problèmes que je considère comme résolus. Il est possible de fermer des issues sur Gitlab mais par moment, il se peut qu'un problème que je considère comme résolu ne le soit pas pour mon tuteur. Afin d'indiquer que je pense avoir résolu le problème, je mettais donc cette étiquette sur l'issue.

Les issues de Gitlab sont extrêmement pratiques. On peut les consulter simplement sous la forme de liste où chaque issue est affichée avec son titre, le nombre de commentaires et les étiquettes par ordre de modification. Les issues peuvent également être consultées dans des tableaux de style Kanban où l'on peut passer un problème en résolu par un simple glissement.

---

<sup>10</sup> Jupyter notebook est un logiciel permettant de faire tourner des bouts de code python de manière interactive en ajoutant des commentaires ou des illustrations. Ce logiciel est en général utilisé pour apprendre ou pour tester des fonctionnalités de python.

Je me suis néanmoins efforcé de justifier le passage d'une issue en résolue. Pour cela je laissais un commentaire avec les changements effectués accompagnés de captures d'écran ou de bouts de code pour montrer la modification.

Alexandre VIALA @viala 1 week ago  
Integrated smart log messages using the specified label [STAGE].  
  
Solution  
I chose to integrate smart logs only in the client part of the software (the javascript part). I modified the `process.js` file of the AMORA project.  
  
Here the `processStageLog` element is an HTML `Element` that is located on the processing control interface shared with all the scripts :  
  
[STAGE] Process execution: generate\_spectrums.sh finished

```
async loadProcessLog(process) {
  let protocol = "", host = "", prefix = "", session = "", "": "";
  [protocol, host, prefix, session, ...] = this.getURLParameters();
  const fileContent = await (await fetch(`${protocol}//${host}/${prefix}/job/${session}/process-log`));
  let stagedLog = "";
  let stagedLogHTML = "";
  for (let line of fileContent) {
    if (line.match(`\n[STAGE]\n`)) {
      stagedLog += line;
      resultInnerHTML += `<li class="stage-log">${line}</li>`;
    } else {
      resultInnerHTML += `<li>${(line)}</li>`;
    }
  }
  this.processResult.innerHTML = resultInnerHTML;
  processStageLog.textContent = stagedLog;
  this.processResult.scrollTop = this.processResult.scrollHeight;
}
```

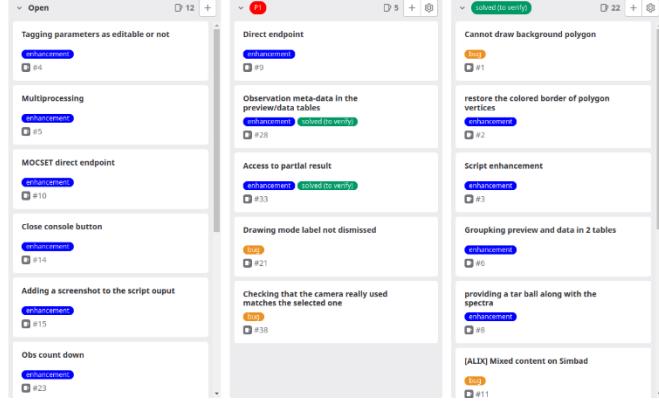


Figure 20 : Mode Kanban des issues Gitlab

Figure 21 : Réponse à une issue

## V. Tâches réalisées

### Indexation des observations

#### a. Préambule

Ma mission à l'observatoire astronomique de Strasbourg consistait à réaliser une analyse des données XMM sur des photons issus d'une zone du ciel tracée par l'utilisateur. L'éditeur de région intégré dans la XCatDB permet de renvoyer les positions des sommets d'un polygone dans le repère céleste sous la forme d'un message JSON. Néanmoins il ne réalise aucune action sur ces régions sinon de les afficher dans une boîte de dialogue. Le traitement de ces régions constitue le point d'entrée de mon travail.

Ces positions peuvent aisément être envoyées à un autre service à l'aide d'une requête HTTP POST<sup>11</sup>, mais ça n'est pas suffisant pour initier un traitement. Il faut également pouvoir déterminer quelles observations incluent cette forme afin que l'utilisateur puisse choisir celles sur lesquelles portera le traitement. En effet, les observations astronomiques de XMM ne couvrent pas l'entièreté du ciel. En général, les scientifiques choisissent d'observer des zones bien particulières pour diverses raisons : des observations ont déjà été réalisées avec d'autres appareils de mesure ou alors on observe des bruits de fond sur certains spectres d'émission d'autres sources.

Ces raisons font que les mesures de XMM ne couvrent que 4% du ciel et que certains endroits ont été observés des dizaines de fois. Ainsi, il était nécessaire d'imaginer une solution

<sup>11</sup> Requête ayant un corps où il est possible de transmettre des données en “pièce jointe”. Ces données peuvent prendre la forme d'un JSON, d'un XML, d'un texte, d'une page html, etc.

afin de savoir quelles observations contiennent la forme tracée par l'utilisateur pour pouvoir réaliser les traitements sur seulement certaines observations bien particulières.

Pour récupérer ces observations nécessaires, mon tuteur m'a proposé d'utiliser un outil développé par F.-X. Pinault du Centre de Données astronomiques de Strasbourg : les MOCSet.

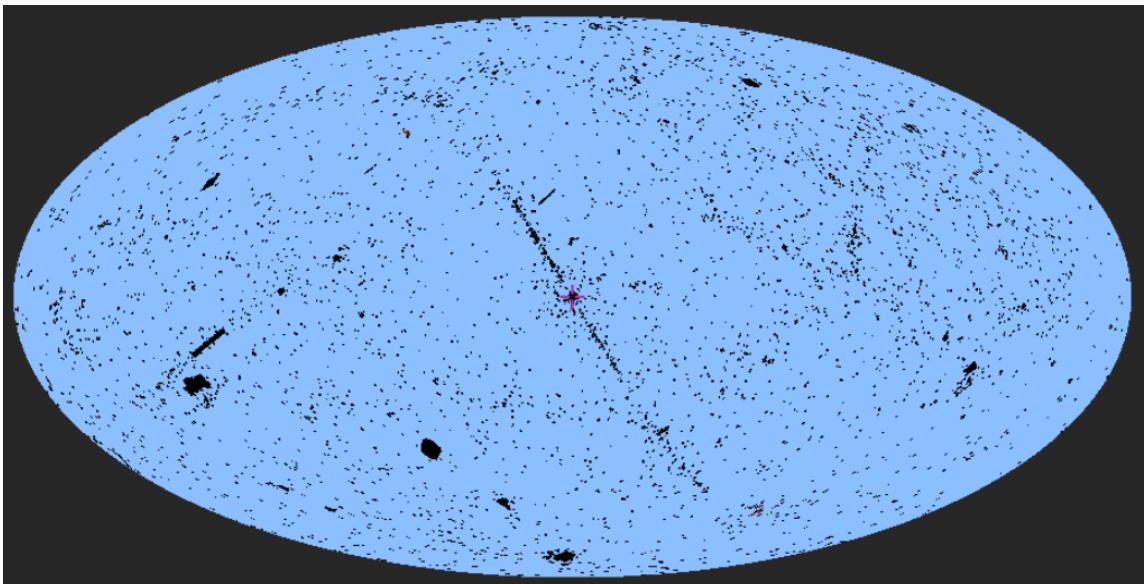


Figure 22 : Observations prises par la Caméra EPN du télescope XMM-Newton visualisées dans Aladin Lite (les zones en bleu sont les zones sans mesure effectuée par cette caméra).

### b. Les MOCSet

Les MOCSet sont des ensembles de MOC (cf. § Multi Order Coverage map). Ces ensembles de MOC sont stockés sous la forme d'un fichier binaire qu'il est possible de requêter à l'aide de commandes shell. En argument de ces requêtes, il est possible d'utiliser différentes structures. On peut notamment requêter ce fichier à l'aide d'un cercle, d'un polygone ou d'un point. L'outil va alors effectuer une opération d'intersection ou d'inclusion, selon les paramètres indiqués par l'utilisateur. Si l'outil est réglé pour réaliser des intersections alors le logiciel va renvoyer les MOC comprenant au moins une partie de la forme de requête. Si l'outil est réglé pour réaliser des inclusions alors le logiciel ne va que renvoyer les MOC comprenant entièrement la forme de requête. Considérons que l'ensemble des cercles dans l'illustration ci-contre est un MOCSet et que la forme en gris est la forme utilisée pour la requête. En mode inclusion, le logiciel ne va renvoyer que le cercle orange alors qu'en mode intersection, le logiciel renverra les cercles orange, bleu, rouge et violet.

Les MOCSet ont été conçus pour être des fichiers requêtables rapidement. Dans ce but, un minimum d'informations sont stockées à l'intérieur du fichier binaire. Ainsi, lors de la création du MOCSet, il est nécessaire de renseigner un identifiant unique (sous forme de nombre) qui sera utilisé

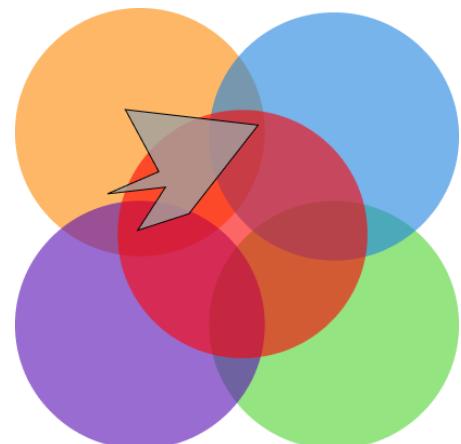
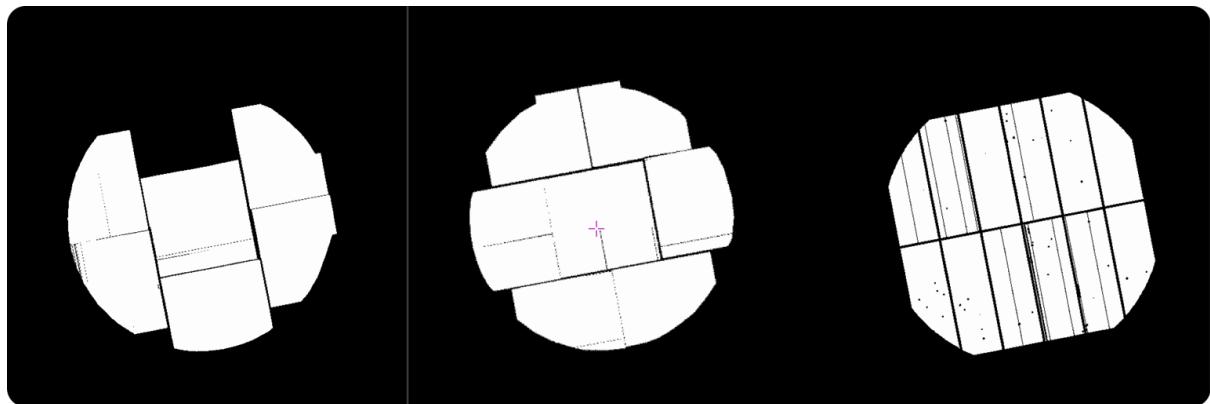


Figure 23 : Illustration d'un MOCSet et d'une requête

pour identifier chaque MOC de façon unique. Lorsque l'utilisateur réalise une requête sur le MOCSet, les données sont renvoyées sous la forme d'une liste d'identifiants (à la manière d'un fichier csv). La charge est donc à l'utilisateur de consigner dans un autre fichier les informations complémentaires qu'il voudrait stocker sur un MOC.

Lors de cette mission, nous nous sommes servi des MOCSet afin de stocker les différentes observations du satellite Newton-XMM. Les fichiers d'images des caméras du télescope Newton-XMM (fichiers fits) comportent une description de la forme de l'observation. Nous avons donc transformé ces observations en MOC puis nous les avons stockées dans des MOCSet.



*Figure 24 : De gauche à droite, masques des caméras EMOS-1, EMOS-2, EPN*

### c. Analyse

Il faut savoir que le satellite Newton-XMM dispose de trois caméras et que chacune de ces caméras a réalisé 14000 observations depuis le lancement du télescope. Afin de construire l'architecture du logiciel, nous avons imaginé les fonctionnalités que l'utilisateur souhaiteraient utiliser. Ces fonctionnalités sont résumées dans le diagramme de cas d'utilisation ci-dessous. Nous voulions qu'avant toute chose l'utilisateur ait un logiciel correspondant le plus à ses besoins pour assurer une durabilité du service.

Nous avions besoin d'un outil de sélection des observations, le MOCSet et d'un outil capable de fournir les méta-données d'observation manquantes au MOCSet. Pour ces dernières, nous avons choisi de produire une base de données SQL car cela permet de facilement réaliser des sélections de données. Toutefois, la construction de cette base est bien plus efficace si l'on génère d'abord un fichier CSV qui contient toutes les données. De la sorte la base peut être abondée en une seule transaction.

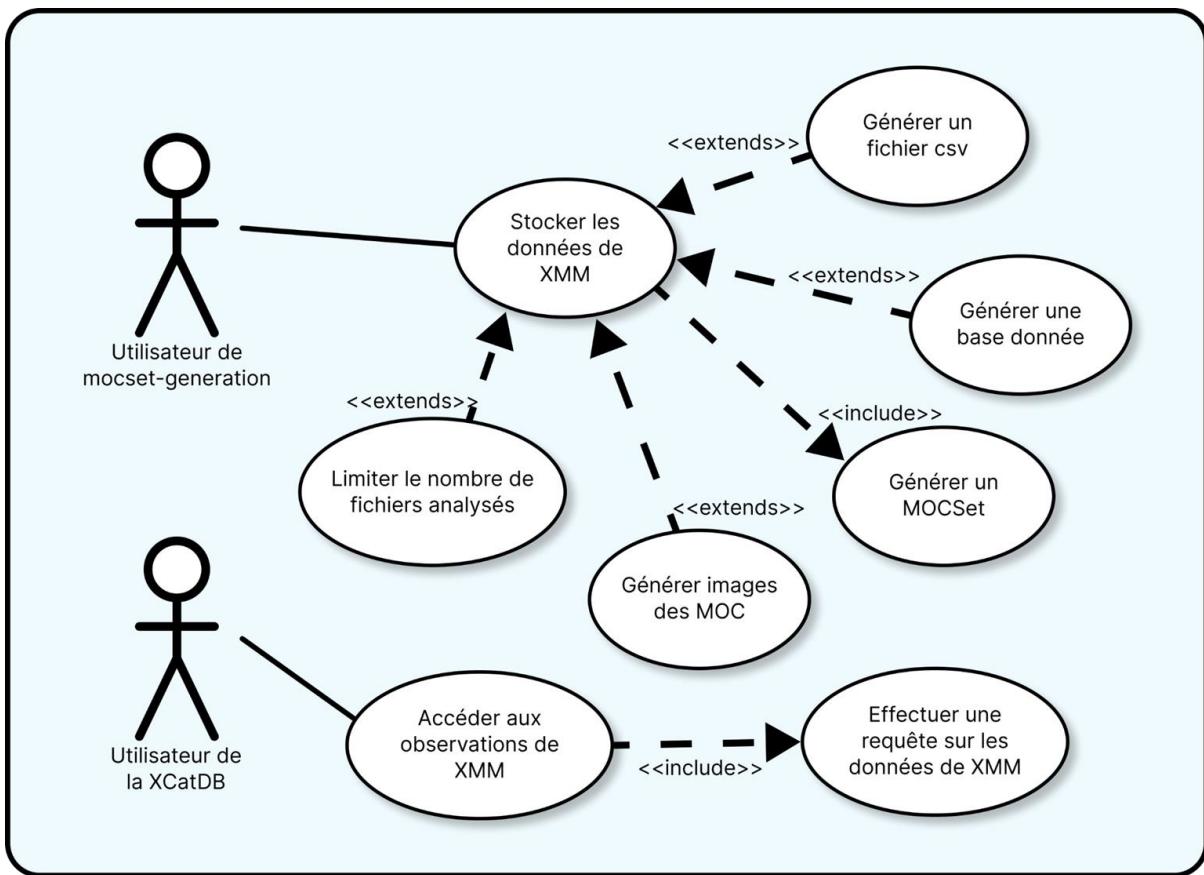


Figure 25 : Diagramme de cas d'utilisations du logiciel *mocset-generation*

Nous avons également imaginé le fonctionnement classique d'une telle application en imaginant que l'utilisateur souhaite générer un fichier CSV, un MOCSet et la base de données SQLite associée. Dans un tel cas d'utilisation l'utilisateur démarrera le logiciel au moyen d'une ligne de commande dans son terminal Shell en précisant les options qu'il souhaite utiliser. Ici, nous générerons un MOCSet pour les observations de la caméra EPN. Le logiciel prend en compte ces paramètres et va directement interroger les données du satellite XMM, les fichiers image plus précisément.

Une fois que le logiciel a obtenu toutes les observations correspondant à la demande de l'utilisateur, il va générer un MOC par observation et ajouter ses données dans un fichier CSV. Les données sont ensuite sauvegardées dans la base de données SQLite puis le logiciel *mocset*<sup>12</sup> génère le MOCSet à partir du fichier CSV généré. Le programme s'achève alors en renvoyant un message d'exécution réussie.

<sup>12</sup> Logiciel en ligne de commande permettant de requêter et de créer des MOCSet.

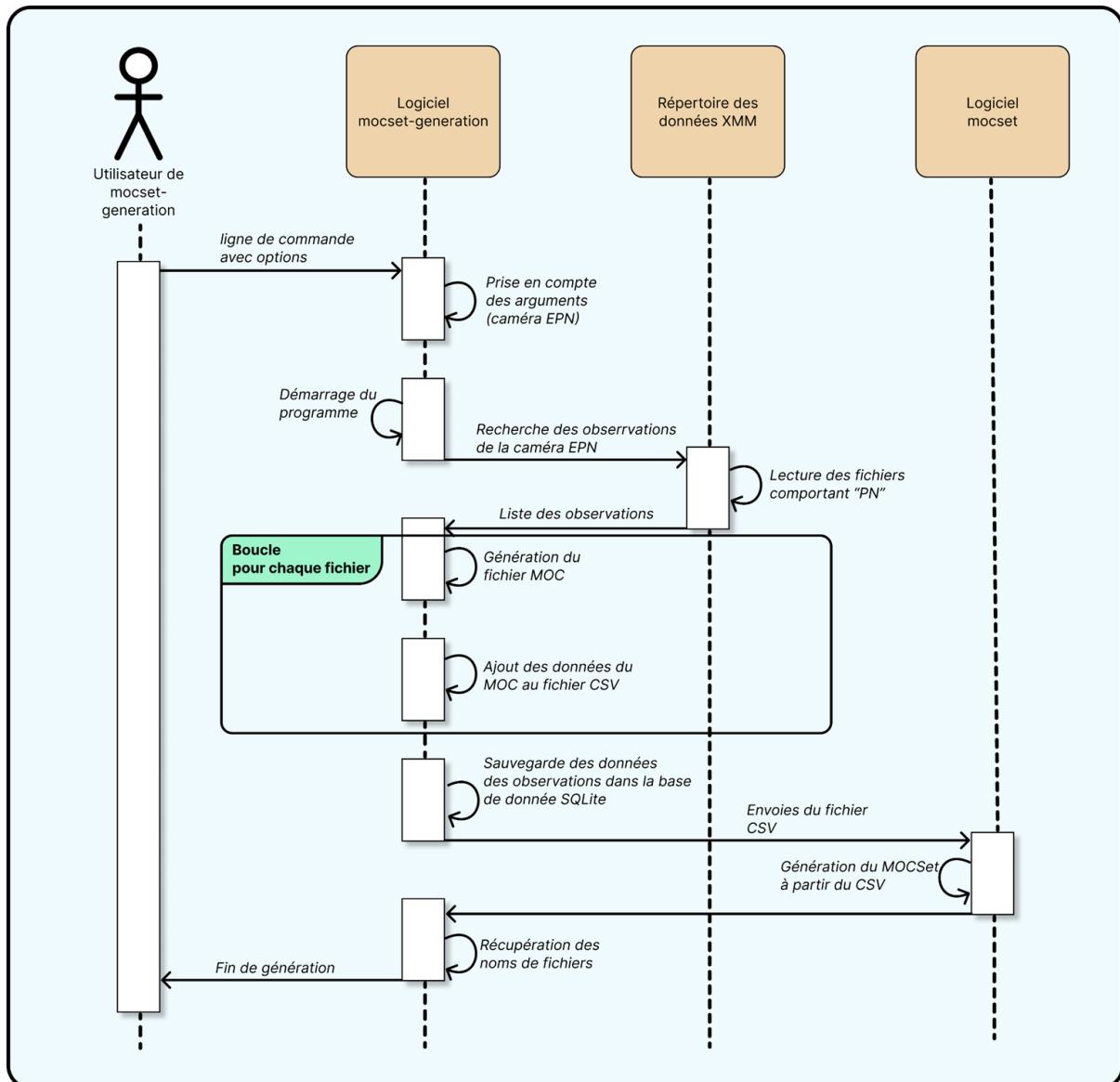


Figure 26 : Diagramme de séquence d'une utilisation classique du logiciel

#### d. Conception

J'avais pour contrainte de réaliser le logiciel en Python. Pour la phase de conception, j'ai donc choisi de suivre le paradigme de la programmation orientée objet. Cette approche m'a permis de concevoir un logiciel extensible.

Le logiciel a été conçu en implémentant un certain nombre de classes statiques<sup>13</sup> qui m'ont par la suite servi à avoir des fonctionnalités connexes. Ce logiciel étant en lignes de commande comme pourrait l'être un paquet linux, le choix a été fait d'utiliser une classe main (un module main dans les faits) afin de réaliser de manière séquentielle les opérations du logiciel.

<sup>13</sup> Une classe statique est une classe ne disposant que d'une seule instance qui est initialisée par défaut.

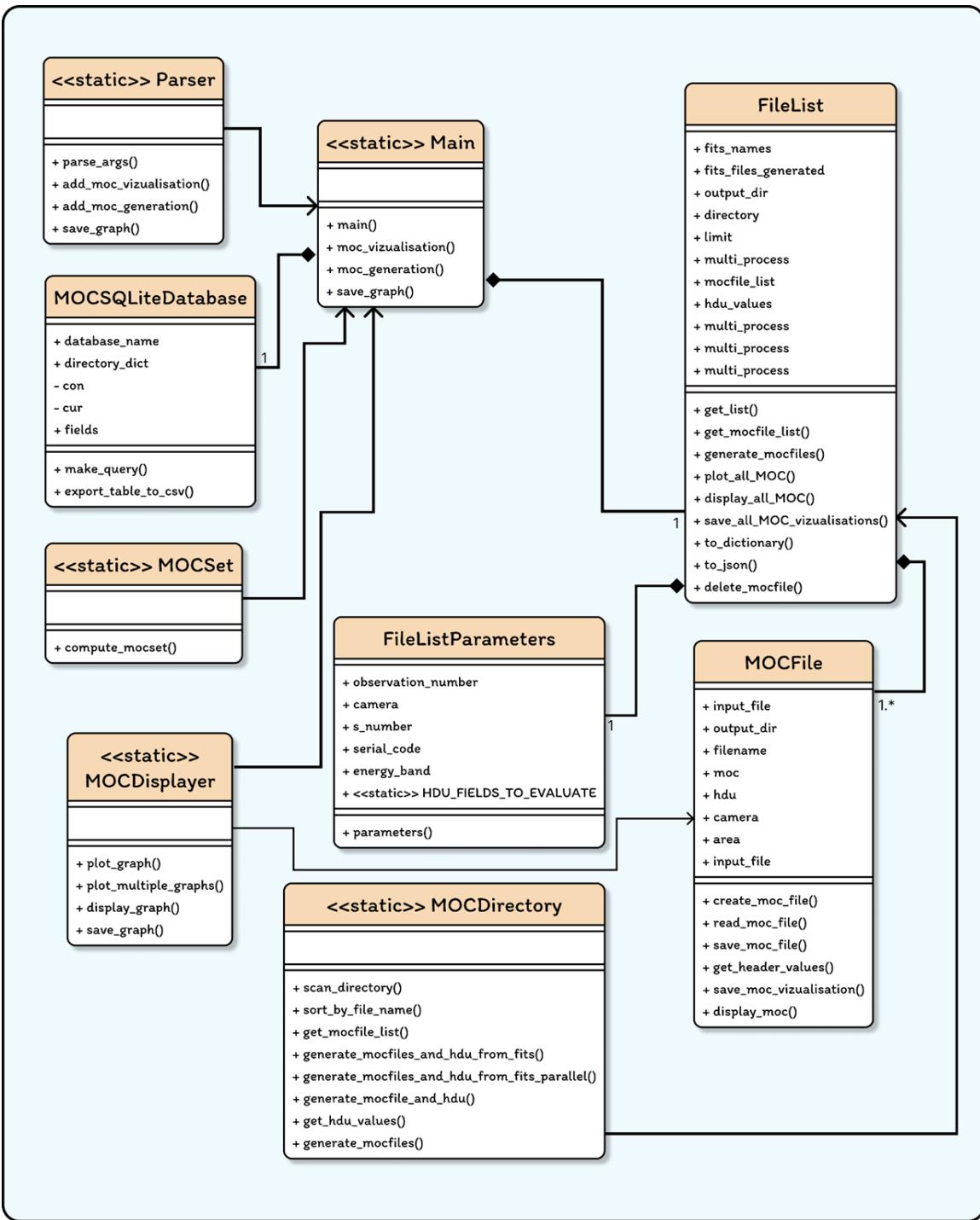


Figure 27 : Diagramme de classes du logiciel `mocset-generation`

Lors de la conception, il a été considéré utile de diviser le logiciel en deux sous commandes permettant dans un premier lieu de créer le MOCSet et tous les fichiers, puis dans un second temps d'avoir une fonction permettant de générer des visualisations de MOC. Cette deuxième fonctionnalité permettait de voir quelle apparence ont les MOC correspondant aux observations à l'intérieur du MOCSet. En effet, le MOC va, en quelque sorte, dégrader l'image et il peut être nécessaire, notamment pour réaliser des tests, de visualiser ces images dégradées pour être sûr que la précision des MOC est conforme à celle attendue par l'utilisateur.

## e. Implémentation

Pendant la phase de programmation, de nombreuses lignes de commentaires ont été ajoutées afin de documenter les modules<sup>14</sup>, les classes et les fonctions implémentées. La réalisation s'est faite relativement rapidement en l'espace de quelques semaines.

Au cours de l'implémentation, mon tuteur m'a suggéré d'utiliser la fonctionnalité de "multi-processing" présente dans le langage Python. Cette fonctionnalité permet de démarrer plusieurs tâches en même temps sous le contrôle du processus parent. Je m'en suis notamment servi pour générer les MOC et les enregistrer dans le fichier CSV. J'ai suivi ces étapes :

- On crée une fonction qui permet de faire les traitements sur une seule donnée (une seule image d'observation).
- On crée une "Pool", une structure de données qui peut traiter plusieurs opérations en même temps
- On lance la fonction précédemment créée en précisant sur quelles données effectuer les traitements.
- On récupère le résultat en fermant la "Pool"

Cette méthode permet de réaliser une parallélisation en utilisant principalement la puissance de calcul du ou des processeurs. En cas de processeur sans cœurs multiples, les actions parallélisées seront traitées dans un seul flux de données.

Dans cette phase de réalisation, nous avons choisi de générer trois MOCSet, un pour chaque caméra, plutôt qu'un seul MOCSet. De cette manière, il est plus facile d'accéder aux données d'une seule caméra. Sans cela, il aurait été nécessaire de compliquer nos requêtes SQL sur la base de données.

Une fois le logiciel réalisé, il a été utilisé pour générer les MOCSet nécessaires au fonctionnement des autres outils conçus durant mon stage. J'ai donc rentré la commande suivante dans mon terminal, successivement pour chaque MOCSet (un MOCSet par caméra) :

```
python3 moc_generation/launcher/main.py moc-generate  
/rawdata/4XMMdr12/ready_to_load/DetMsk -o data/subdata_m1/ -t  
data/m1/db_m1.sqlite3 -c data/m1/db_m1.csv -m data/m1/mocset_m1.bin -O  
50 -d -g M1 -w 2000
```

Cette commande permet de générer le MOCSet lié à la caméra EMOS-1. L'exécution est très longue, j'ai donc réalisé quelques tests de benchmarking pour évaluer le temps moyen qu'un MOCSet prend à être généré. Je me suis servi de la commande **time** qui permet de mesurer le temps d'exécution d'un programme. J'ai également regardé la taille des fichiers générés et j'ai finalement regardé le temps d'exécution d'une commande de requête du paquet **MOCSet**.

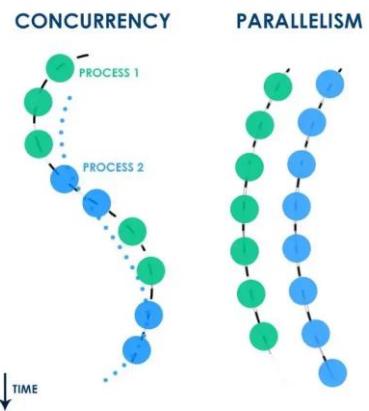


Figure 28 : Différence entre le multi-processing et le multi-threading

<sup>14</sup> Un module est le nom que porte les fichiers en Python

Les résultats de ces différentes opérations ont été consignés dans un graphique récapitulatif. Ces graphiques expriment le temps de génération de MOCSet, le temps de requête sur le MOCSet et la taille des MOCSet en fonction du nombre de fichiers traités.

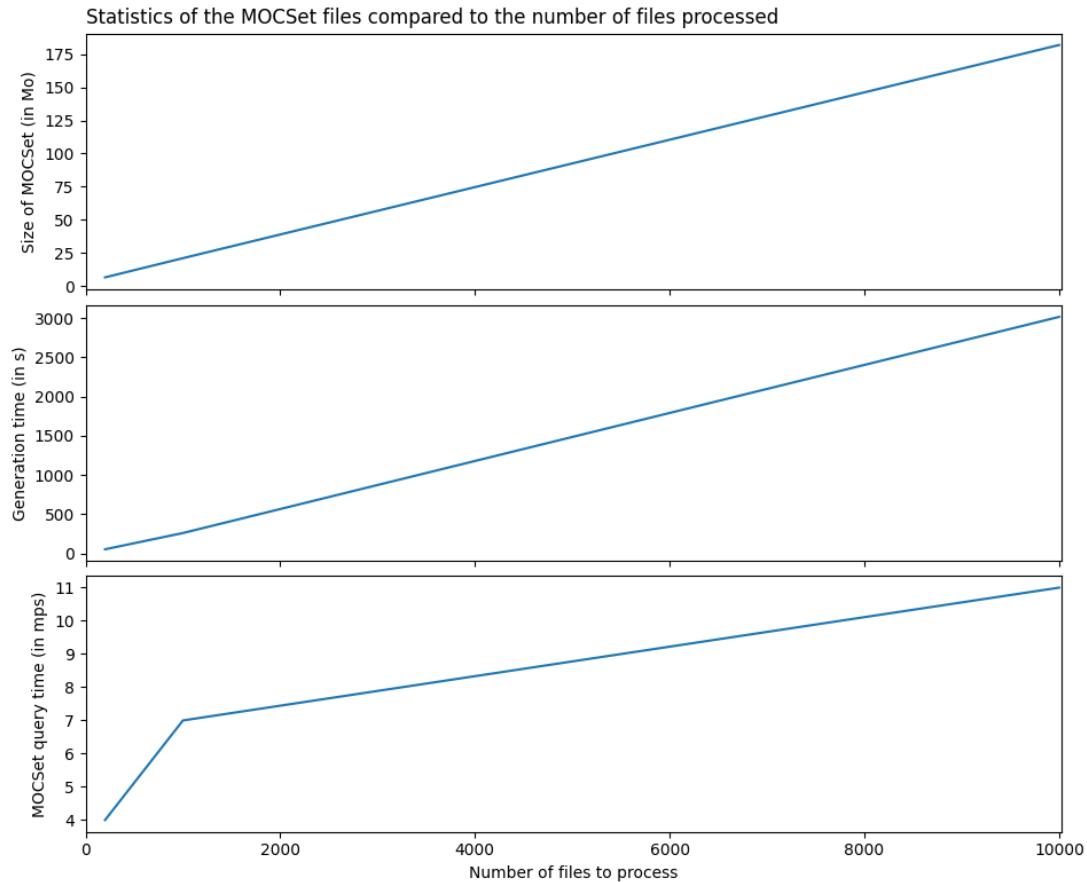


Figure 29 : Mesures sur les MOCSet

```
time mocset query input/mocset.bin cone 19.3013745 -73.464333 1000

id
893400101000
601212301000
601212201000
11450201000
784570301000
11450201001
11450201002
7846990701000
601212401000
893400101001
601212301001
11450201003
893400301000

real    0m0.011s
```

Figure 31 : Mesure du temps d'exécution de la fonction de requête de MOCSet

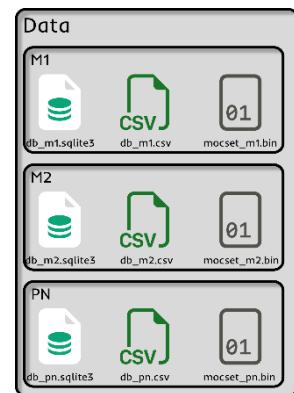


Figure 30 : Arborescence de fichiers du répertoire data après la génération des fichiers de MOCSet

# Adaptation de l'outil de tracé de région d'AliX

## a. Préambule

Un autre composant de mon projet était de modifier profondément deux composants d'AliX: l'éditeur de régions et le gestionnaire de bookmarks. Tout d'abord, l'éditeur de région devait être amélioré car ce dernier ne permettait que de tracer des polygones dans le ciel. Or les membres de l'équipe SSC-XMM de l'observatoire trouvaient un réel intérêt dans l'implémentation d'un éditeur de cônes ( cercle sur le ciel) en plus de l'éditeur de polygones. De plus, il s'est avéré nécessaire d'implémenter un deuxième éditeur de région dans le but de tracer des zones ne contenant que du bruit de fond. Cette fonctionnalité a été demandée par les scientifiques de l'équipe. L'implémentation de ces nouvelles fonctionnalités ont été pour la plupart décidées au cours de réunions, en ma présence, qui m'ont permis de comprendre aux mieux les attentes des membres de mon équipe. Le bruit de fond est en fait utilisé dans certains traitements afin de pouvoir filtrer des produits, il correspond à une zone du ciel en général proche de la source qui permet de diminuer certains effets de l'échantillonnage.

En second lieu, le fonctionnement des bookmarks (signets) a dû être amélioré. Un signet est, sur une application informatique, un élément de l'interface qui permet de retrouver une ressource ou un état de l'application qui a été enregistrée auparavant. Dans AliX, les signets permettent d'enregistrer une position du ciel, d'y retourner en cliquant sur l'image enregistrée et de stocker une région. Les signets d'AliX utilisent une fonctionnalité des navigateurs modernes : l'espace de stockage local. Cet espace est similaire aux cookies mais contrairement à ces derniers, ces données ne sont pas renvoyées au serveur à chaque requête http. Le but n'est pas de tracer l'activité de l'utilisateur mais simplement de lui donner la possibilité d'enregistrer des données localement et de les retrouver lors d'une connexion ultérieure.

Avant les modifications que j'ai apportées sur l'éditeur de région d'AliX, ces derniers étaient capables de stocker des régions, mais pas les contextes de traitements liés à mon projet. L'éditeur de région devait enfin être lié aux traitements qui seraient réalisés par la suite.



Figure 33 : Éditeur de régions d'AliX

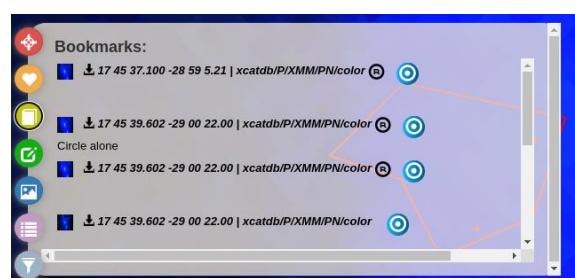


Figure 32 : Signets d'AliX

## b. Analyse

J'ai dû réfléchir à la manière dont les deux fonctionnalités d'ALIX allaient être utilisées. Pour cela, les différents cas d'utilisation listés par mes collègues ont été repris. L'utilisateur s'attend donc à pouvoir tracer une région pour la zone de données et une autre pour un bruit de fond. Ces deux types de régions auront chacune un éditeur avec la possibilité de dessiner un cercle ou un polygone. Jusqu'alors, une fois que la région était validée par l'utilisateur, AliX

renvoyait une alerte contenant un JSON décrivant la région tracée par l'utilisateur. Désormais, il fallait ouvrir une page pour démarrer le service avec la région validée.

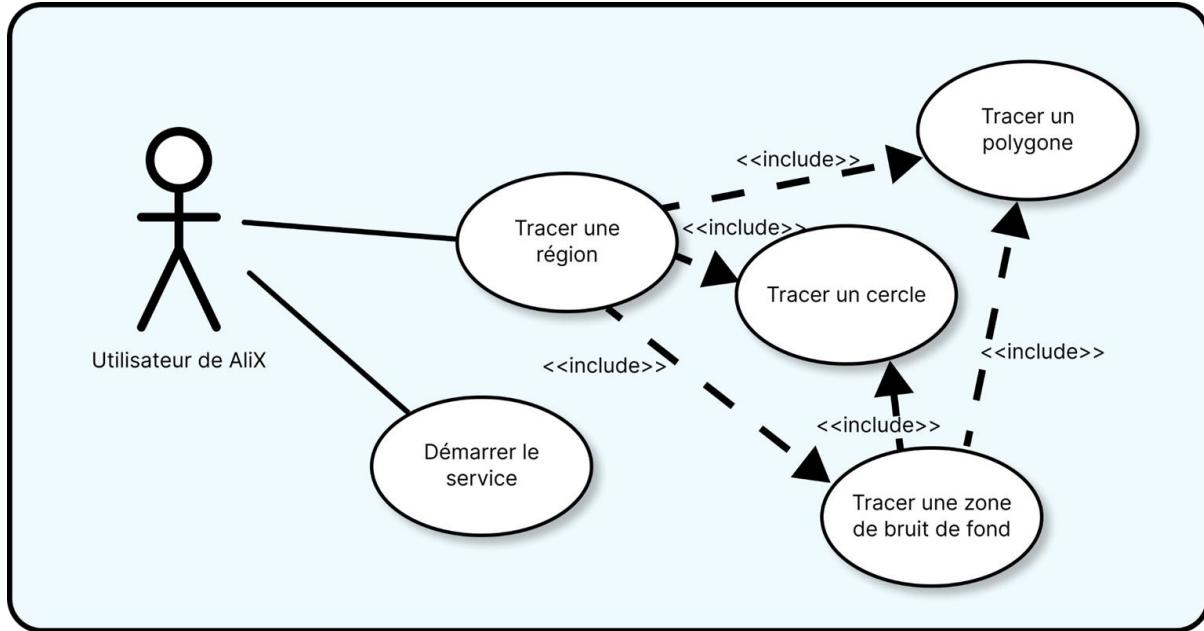


Figure 34 : Diagramme de cas d'utilisation de l'éditeur de régions

Les signets ont également dû être modifiés. Avant, ils ne stockaient une région que sous la forme d'un polygone. Désormais, il fallait également qu'ils puissent stocker des régions possédant un bruit de fond et des régions ayant la forme de cercle. Il fallait également ajouter la possibilité d'enregistrer une session du service AMORA dans les bookmarks ainsi que la possibilité de les restaurer à l'aide d'un bouton.

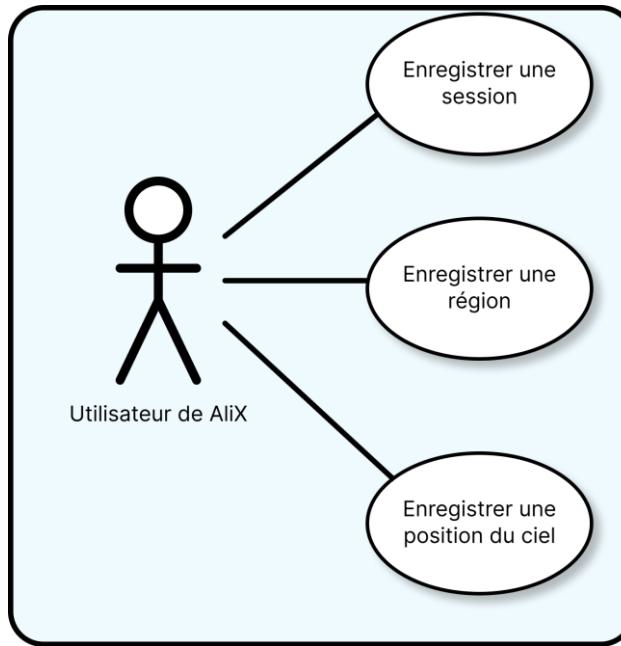


Figure 35 : Diagramme de cas d'utilisation des signets

Afin de mieux comprendre le déroulement d'une fonctionnalité, j'ai parfois eu recours à des diagrammes de séquence. Cela m'a permis de construire des scénarios d'utilisation bien spécifiques du logiciel. Ci-dessous, sont décrits les scénarios où l'utilisateur utiliserait AliX pour tracer une région et démarrer une session AMORA ou alors le scénario où l'utilisateur souhaite quitter la fenêtre pour décrire le fonctionnement des signets.

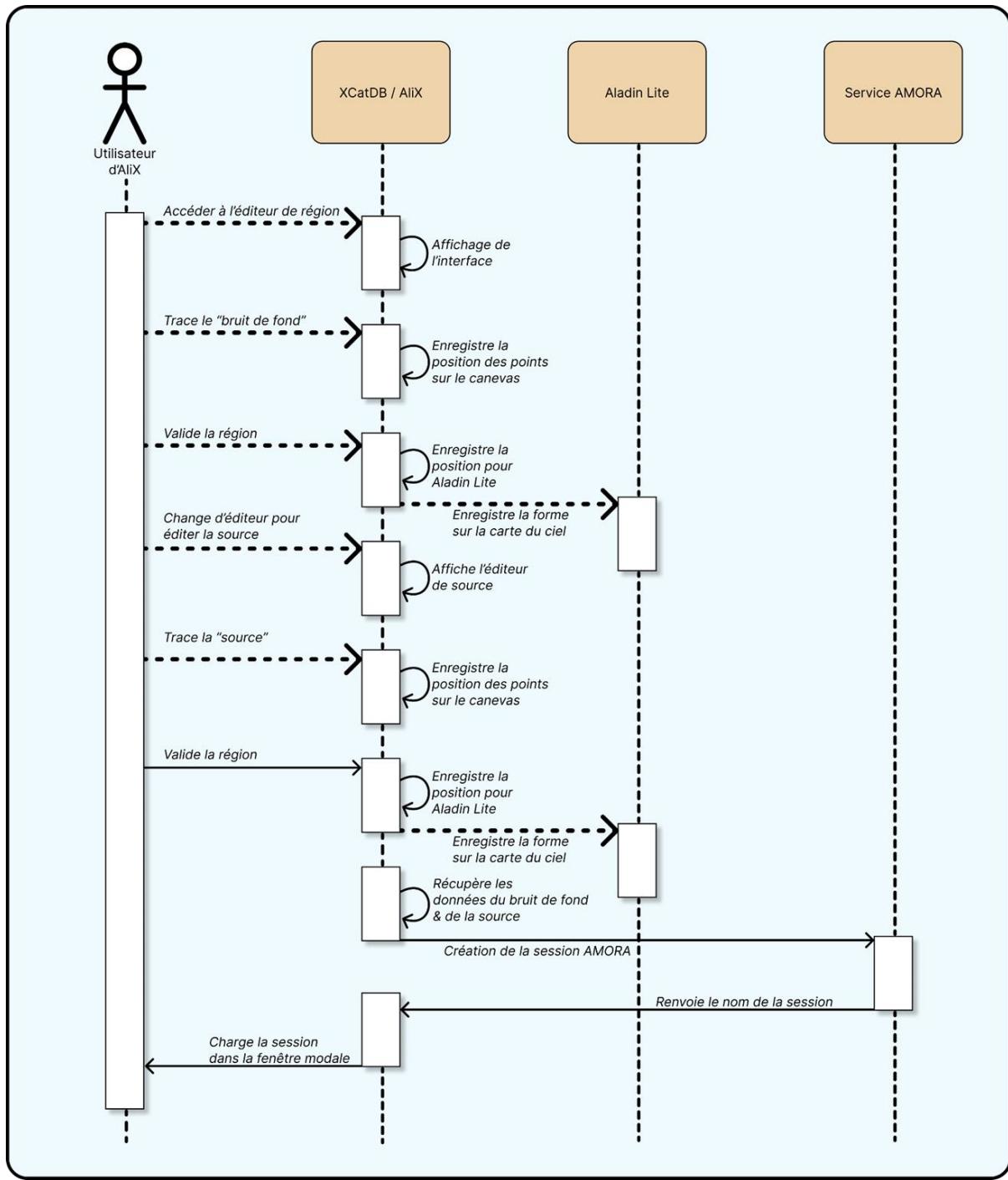


Figure 36 : Diagramme de séquence de l'utilisation de l'éditeur de région

Dans ce premier scénario, l'utilisateur va vouloir tracer une région sur l'éditeur de région. Il va donc se placer sur une région du ciel comportant une source qu'il souhaite étudier. Il commence alors par accéder à l'éditeur de région (en cliquant sur le bouton vert) puis il commence par tracer un bruit de fond. Une fois qu'il a fini de tracer son polygone ou son cercle, il valide sa sélection pour que les positions soient enregistrées dans l'instance d'Aladin Lite. Il change ensuite d'éditeur pour construire la région de la source. Il trace comme auparavant la forme qu'il souhaite sur le canevas puis il valide pour une nouvelle fois enregistrer la position de sa forme dans Aladin Lite.

Une fois que l'utilisateur appuie sur le bouton "valider" de la source, AliX va récupérer les données des points enregistrées sous la forme d'un JSON. Ces données vont être envoyées au service AMORA afin de générer une session pour cette donnée particulière. En retour, AMORA renvoie le nom de la session générée, ce qui permet à XCatDB de générer une fenêtre modale qui va permettre d'afficher la session générée par AMORA.

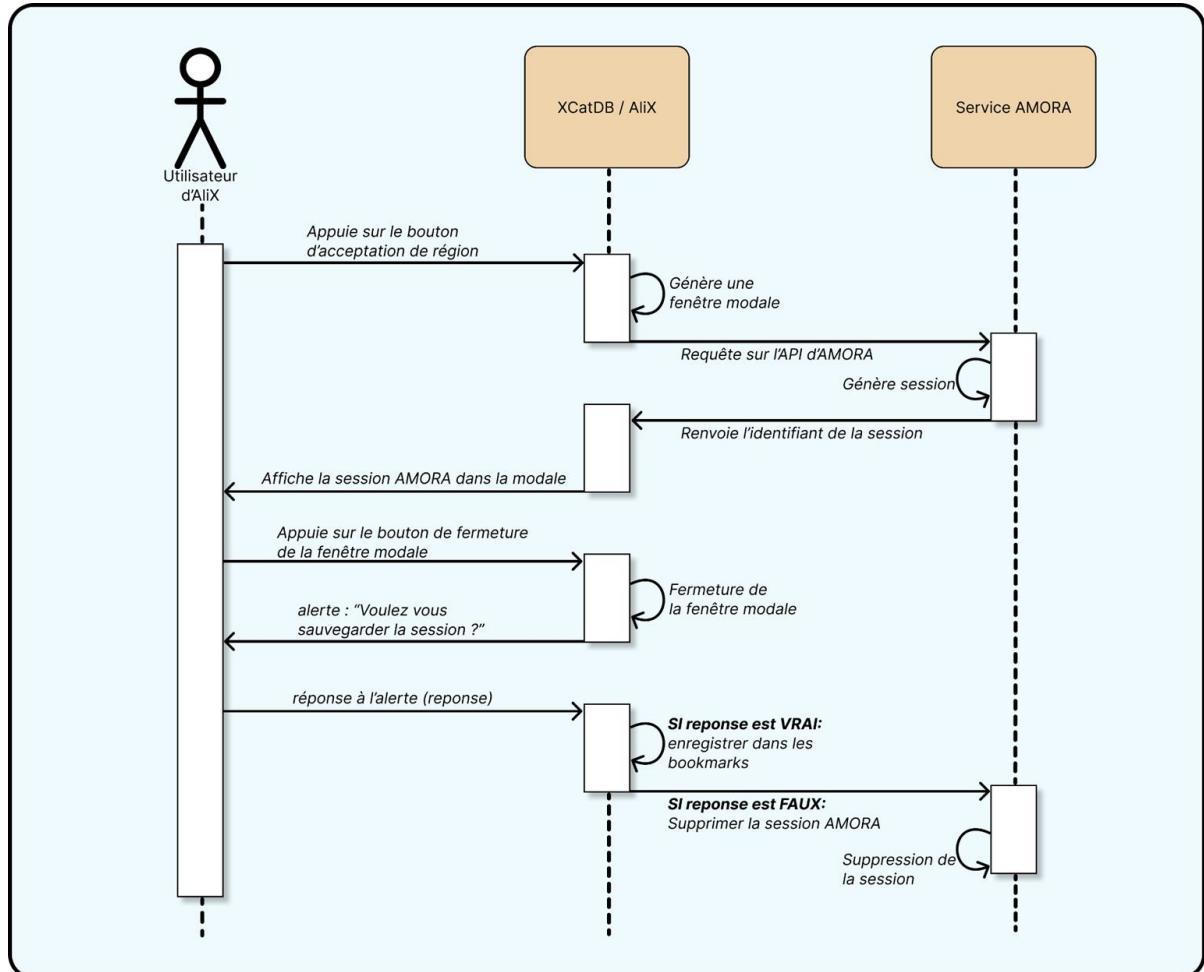


Figure 37 : Diagramme de séquence de l'enregistrement de la session en signet

Dans ce second scénario, on considère que l'utilisateur a fini d'utiliser sa session AMORA et qu'il souhaite fermer la fenêtre modale de sa session. L'utilisateur génère donc une session comme précédemment, puis il appuie sur le bouton de fermeture de la fenêtre en haut à droite de la fenêtre. La fenêtre modale se ferme mais un dialogue s'ouvre pour permettre à l'utilisateur de sauvegarder sa session dans les bookmarks. Si l'utilisateur appuie sur le bouton "Oui", la session sera enregistrée dans les bookmarks. Sinon, la session n'est pas sauvegardée et elle sera également supprimée du service AMORA pour libérer de l'espace.

### c. Conception

Sur AliX, j'ai surtout été chargé de concevoir une interface graphique permettant de réaliser des actions pour la plupart déjà existantes dans le code de la XCatDB. Une interface graphique selon deux principes essentiels : l'expérience utilisateur (UX) et l'interface (UI).

L'interface est la partie la plus souvent mise en lumière par les concepteurs car elle correspond à la partie visible du travail. Il s'agit de tous les éléments qui vont être visibles à l'écran avec leur forme, leur couleur, leur police de caractère, leur icône et parfois leur animation. Il ne faut toutefois pas négliger l'expérience utilisateur dans une conception d'une interface. Cette dernière régit les actions qui vont être produites à chaque fois que l'utilisateur va interagir avec des éléments visuels. Pour concevoir ce logiciel, il a d'abord fallu concevoir l'expérience utilisateur pour ne réfléchir à l'interface visuelle uniquement dans la phase d'implémentation.

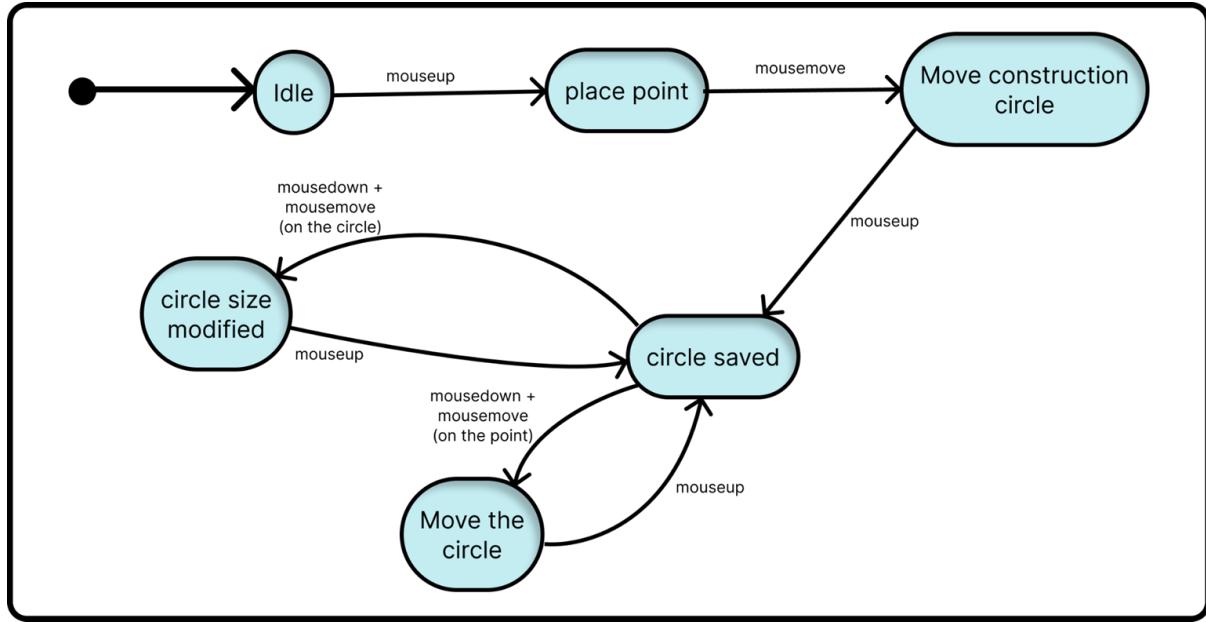


Figure 38 : Diagramme de contrôle de la construction d'un cercle

Dans un premier temps, j'ai travaillé sur l'implémentation de l'éditeur de région. Il a été nécessaire de réfléchir à une manière intuitive de tracer un cercle. Afin de concevoir des interactions logiques, je me suis inspiré de l'utilisation de l'ancien éditeur de région par mes collègues. En effet, en les regardant utiliser le logiciel je me suis rendu compte que certaines interactions pour le traçage de polygones n'étaient pas parfaites :

- Lorsque l'utilisateur trace un premier point d'un polygone, le trait attaché à ce point n'est pas directement construit. Il doit cliquer à nouveau sur le point placé pour tracer un trait.
- Lorsque l'on clique sur un point, il est nécessaire de laisser le bouton de sa souris enfoncé pour pouvoir placer le point suivant.
- Lorsque l'utilisateur veut insérer un point dans le polygone, il doit d'abord appuyer sur un trait puis maintenir le doigt appuyé sur le point créé.

Ces interactions rendent plus difficiles le traçage d'une zone du ciel car maintenir une pression sur un bouton de la souris entraîne une tension dans la main de l'utilisateur.

Ma mission n'avait pas pour but d'améliorer l'éditeur de polygone, je me suis donc servi de ces éléments remarqués pour concevoir un éditeur de cercle plus simple à appréhender. Toutefois, il fallait prendre en compte que les utilisateurs de l'interface actuelle sont habitués à devoir maintenir le bouton de la souris appuyé pour tracer un polygone. Il fallait donc faire en sorte de laisser la possibilité à l'utilisateur de pouvoir tracer un cercle en maintenant le bouton de la souris enfoncé ou en le relâchant.

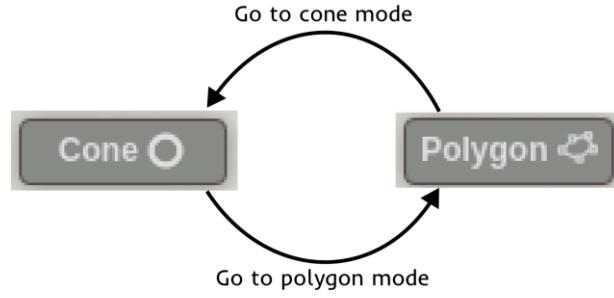


Figure 39 : Diagramme présentant le bouton permettant de changer de forme

Il était ensuite nécessaire de réfléchir à la manière dont il fallait intégrer l'éditeur de cercle avec l'éditeur de polygone actuel. Pour cela, un bouton “switch” a été ajouté sur l’interface, ce qui permet à l’utilisateur de changer facilement d’éditeur. Chaque éditeur de forme est par ailleurs contrôlé par une classe Javascript particulière de l’éditeur de région (cf. Diagramme de classe ci-dessous). Ils ont des méthodes relativement similaires qui leur permettent d’être contrôlé par un seul contrôleur d’une manière assez similaire.

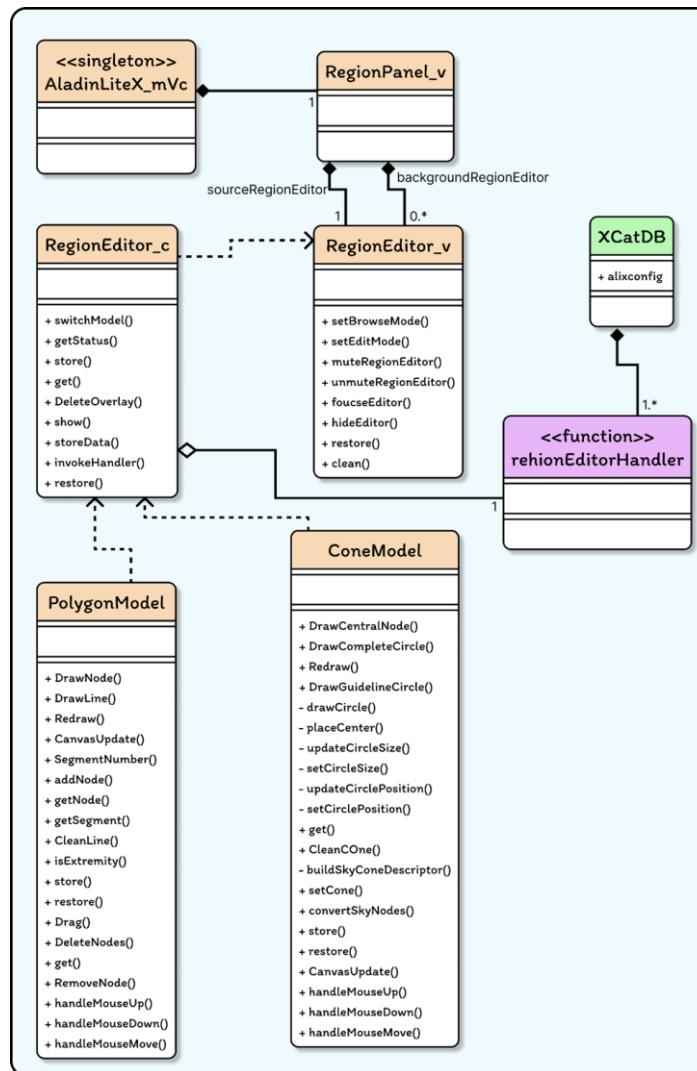


Figure 40 : Diagramme de classe de l’éditeur de région

Après avoir travaillé sur l’éditeur de région, il fallait ensuite travailler sur le fonctionnement des signets et de l’enregistrement des sessions. J’ai essayé de concevoir

avec mon tuteur un système de signet qui soit simple et intuitif. Les signets étaient particulièrement liés avec l'enregistrement des sessions dans AMORA. Nous avons, pour cela, imaginé le fonctionnement de la création de sessions. Cette fonctionnalité permet de limiter le nombre de sessions créées ainsi que de simplifier la compréhension de l'interface pour l'utilisateur. En effet, en évitant de générer une session à chaque fois que l'utilisateur clique sur le bouton de validation, on permet à l'utilisateur de quitter temporairement la fenêtre modale d'AMORA pour pouvoir naviguer dans la carte du ciel puis de retourner sur la session en cliquant à nouveau sur le bouton d'acceptation.

Cet enregistrement de session couplé au dialogue permettant de sauvegarder une session dans les signets à la fermeture d'une fenêtre modale permet à l'utilisateur d'être toujours sûr que les traitements qu'il a démarrés seront à nouveau accessibles.

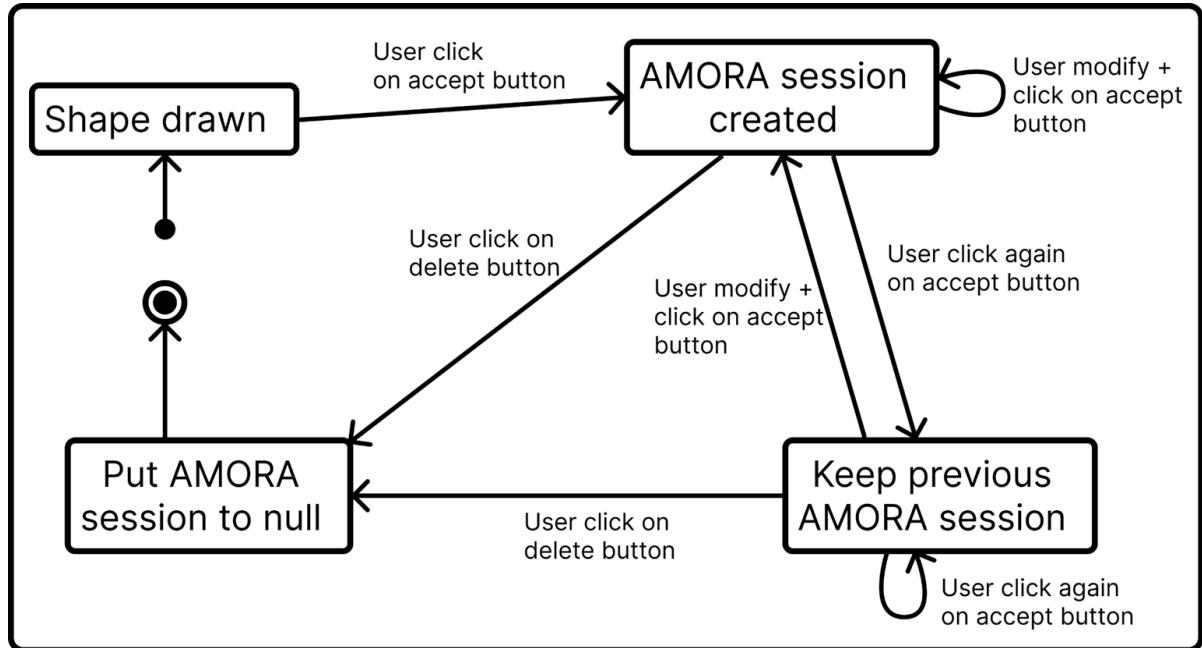


Figure 41 : Diagramme de contrôle de l'enregistrement des sessions

Le fonctionnement des bookmarks à proprement parler est quant à lui relativement simple. Appuyer sur le bouton orange permet d'enregistrer une session déjà créée avec toutes les données de position et de région associées. La position et la région peuvent être restaurés en appuyant sur la zone du ciel sélectionnée et la session est restaurée en cliquant sur le bouton à droite du bookmark.

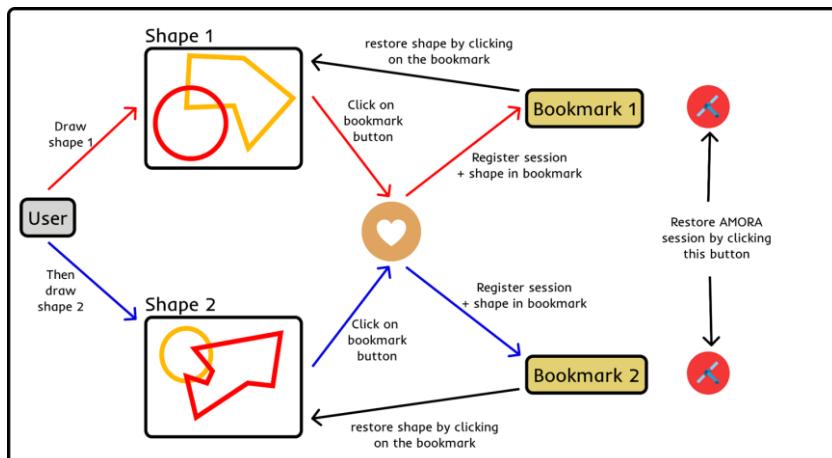


Figure 42 : Diagramme expliquant le principe du bouton de bookmark

## d. Implémentation

Comme énoncé précédemment, nous avons choisi de concevoir l'interface utilisateur au début de la phase d'implémentation. Pour cela j'ai réalisé des maquettes pour faire coexister l'éditeur de région du bruit de fond et l'éditeur de région de la source. Trois solutions différentes ont été envisagées : des éditeurs accessibles via des onglets, des éditeurs accessibles via un accordéon<sup>15</sup> ou simplement des éditeurs affichés en même temps sur la fenêtre des éditeurs de région.



Figure 43 : Choix de l'interface à réaliser

La solution retenue a été celle utilisant des onglets. Cette dernière était une des plus simples à mettre en place (contrairement à l'accordéon) tout en n'affichant uniquement que les informations nécessaires à l'utilisateur. De plus, cette solution permet d'être modulable en ajoutant un nombre indéfini d'onglets sur l'interface.

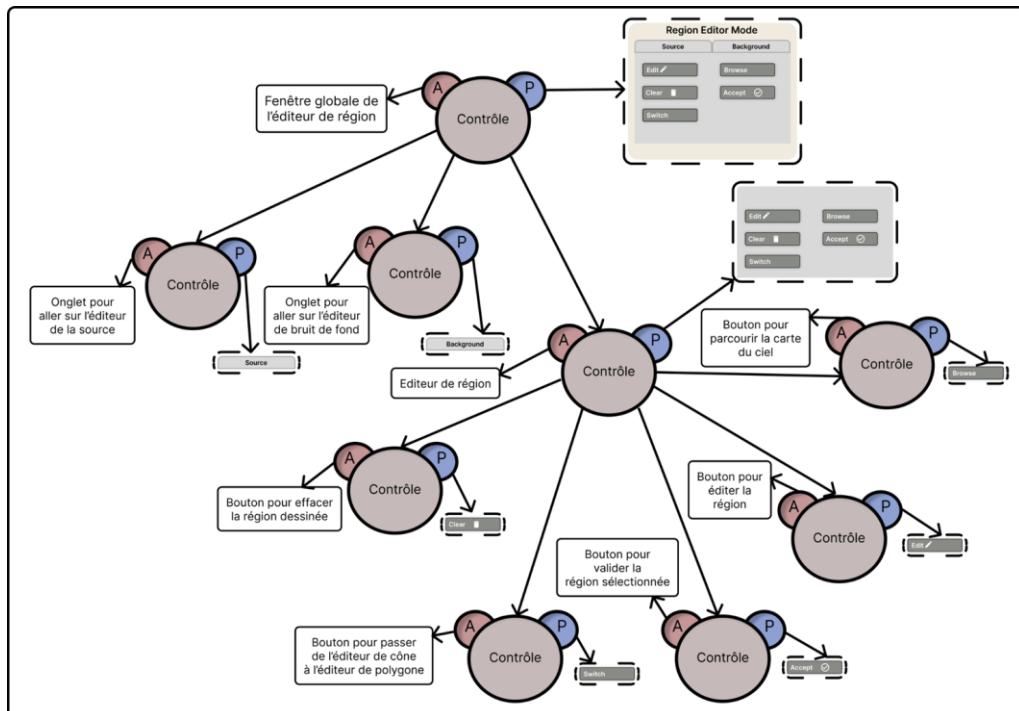


Figure 44 : Diagramme PAC de l'interface

<sup>15</sup> Un accordéon est un élément d'interface qui permet de dérouler des rubriques en cliquant sur le titre de cette dernière.

Finalement, cet éditeur de région a été implémenté en deux étapes. Tout d'abord, le CSS de l'éditeur de région actuel a été modifié pour utiliser les fonctionnalités de layout<sup>16</sup> du langage. J'ai ensuite ajouté à ce layout mes propres définitions de style. Une fois que la forme de l'interface était réalisée, les deux éditeurs de région ont été implémentés.

Lors de la modification du code d'AliX, un certain nombre de bugs ont été rencontrés. Certains étaient dus à certaines modifications apportées au logiciel mais d'autres étaient simplement liées à des oubli de ma part lors de la conception. J'ai par exemple rencontré un bug dans certains cas d'utilisation lorsque l'utilisateur traçait un cercle puis l'effaçait et retournait en mode édition par la suite, le cercle était encore conservé. Pour régler ces bugs, j'ai souvent dû me plonger dans le code d'AliX pour comprendre les interactions qui avaient été brisées. J'ai beaucoup utilisé le débogueur du navigateur chrome qui permet de pouvoir effectuer des arrêts dans l'exécution du code pour bien saisir quel était le cheminement des interactions. Ce débogueur m'a également permis de surveiller l'état des variables pour repérer certains changements qui n'avait pas eu d'effet.



Figure 46 : Interface finale

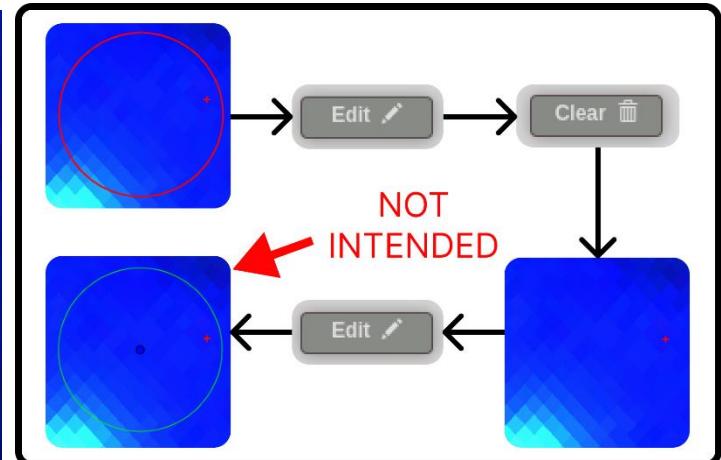


Figure 45 : Illustration de l'un des bugs rencontrés

## Le Service AMORA

### a. Préambule

Pendant mes six mois passés à l'observatoire astronomique de Strasbourg, j'ai été amené à travailler sur plusieurs logiciels. Néanmoins, tous ces logiciels modifiés ou conçus ont gravité autour de l'intégration d'un service bien particulier. Ce service, l'Asynchronous Multi Observation Region-based Analysis (AMORA), a constitué le fil rouge de mon stage.

Ce logiciel est un service en ligne utilisant le langage de programmation Python pour son backend grâce au Framework Flask et le Javascript pour son frontend. Le but d'AMORA était de pouvoir effectuer des traitements sur des observations bien spécifiques du satellite Newton-XMM grâce à une région tracée depuis l'éditeur de région d'AliX.

<sup>16</sup> Un layout est un élément de d'interface implémenté dans bon nombre de langages de programmation. Ils permettent d'afficher les objets selon des contraintes (marge, espacement entre les objets, etc.) et non seulement selon des positions absolues.

## b. Analyse

Grâce au document rédigé par mon tuteur au début de mon stage et aux remarques accumulées au fil des réunions hebdomadaires, j'ai pu constituer une liste de fonctionnalités attendues dans AMORA. Trois de ces fonctionnalités étaient absolument nécessaires. Tout d'abord, il fallait que l'utilisateur puisse filtrer l'ensemble des observations du Newton-XMM à l'aide de la forme tracée par l'utilisateur. Il fallait ensuite que l'utilisateur puisse encore une fois filtrer les observations de la zone de recherche à l'aide d'une sélection manuelle. Par ailleurs, cette sélection ne devait se faire que sur les données d'une seule caméra à la fois. C'est-à-dire qu'il ne devait pas être possible de sélectionner en même temps des observations de la caméra EMOS-1 et la caméra EMOS-2 par exemple. Cette limitation a été demandée par l'équipe scientifique pour éviter de générer des résultats incohérents avec les processus.

Enfin, le service devait permettre à l'utilisateur de démarrer un processus en utilisant les données d'observation sélectionnées ainsi que des paramètres définis par l'utilisateur. Le but de cette fonctionnalité était de pouvoir démarrer un script utilisant des logiciels tels que HEASoft ou SAS sans avoir à les installer sur sa machine.

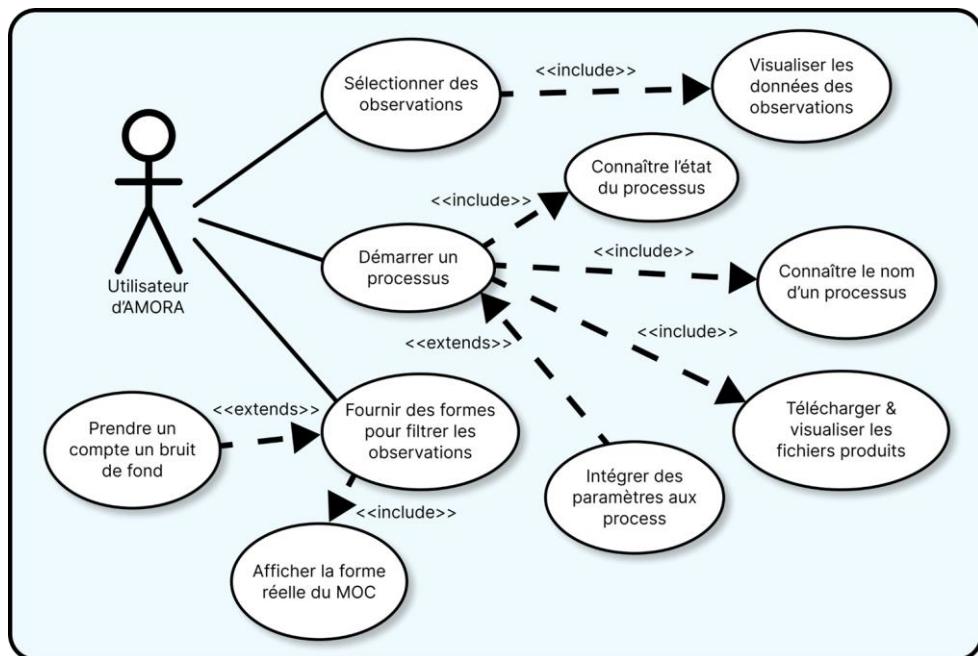


Figure 47: Diagramme de cas d'utilisations du service AMORA

Le projet était assez conséquent. J'ai donc commencé à travailler dessus indirectement en testant les fonctionnalités du Framework Flask que je ne connaissais pas. Au bout de quelques semaines, lorsque j'ai mieux compris comment fonctionnait Flask et la mission sur laquelle je devais travailler, il a été possible d'imaginer le déroulement attendu d'une utilisation classique de mon service par un utilisateur scientifique.

Lorsqu'un utilisateur veut utiliser mon service, il se rend sur le site de XCatDB puis sur l'éditeur de région. Il trace sa région puis appuie sur le bouton de validation. Une fenêtre modale contenant une page web correspondant à la page principale de mon service apparaît. Cette page est en fait la page principale de la session créée par l'utilisateur. Une session est un objet identifié par un nom qui va tracer toutes les actions réalisées par l'utilisateur. Cela va permettre au service de se souvenir des avancées réalisées sur la page web si l'utilisateur retourne sur la session après avoir fermé la fenêtre de son navigateur.

Une fois sur la page internet d'AMORA, l'utilisateur va en général sélectionner un certain nombre d'observations puis il va sélectionner un processus. Le backend de

l'application prend en compte les actions de l'utilisateur grâce au frontend qui communique par le biais de requêtes HTTP. L'utilisateur va ensuite configurer des paramètres ou non puis il va démarrer le processus. Le service va alors démarrer une tâche en fond pour faire tourner un processus sur la machine distante. L'avancée du processus est renseignée dans un fichier. Pour que l'utilisateur puisse suivre l'avancée du processus, le frontend va demander régulièrement au backend l'avancée du processus.

Lorsque le processus est fini, le backend l'écrit dans le fichier. À la prochaine requête du frontend, il pourra renvoyer le statut fini à l'utilisateur. Il demandera ensuite l'accès aux fichiers générés par le processus, s'il y en a et créera un accès pour l'utilisateur afin qu'il puisse récupérer les données calculées.

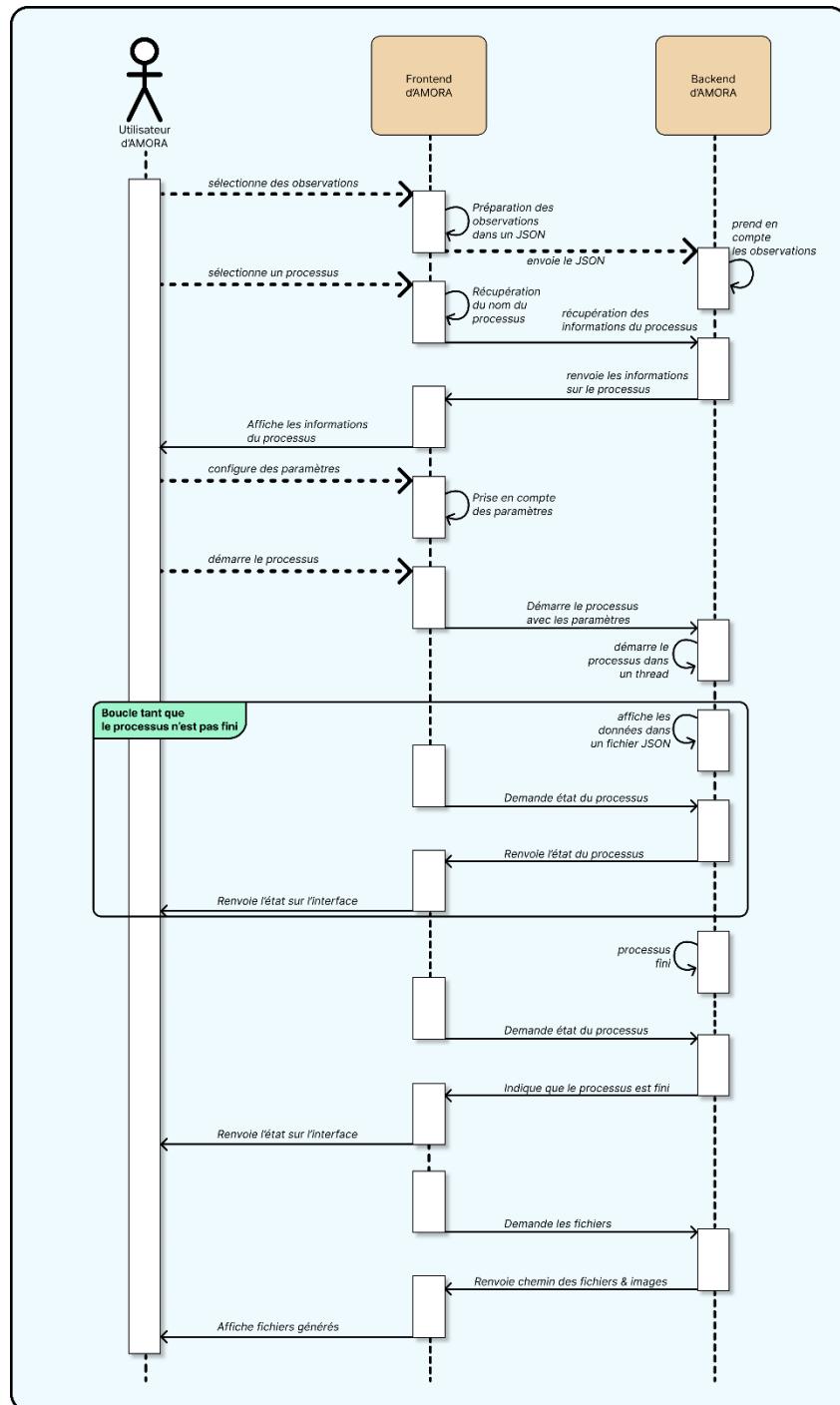


Figure 48 : Diagramme de séquence d'une utilisation classique d'AMORA

### c. Conception

Ce service en ligne reposait sur deux composantes : le backend et le frontend. Le backend avait pour but de faire tous les traitements lourds et le frontend de communiquer à l'utilisateur les fonctionnalités réalisées par le backend. Le backend de l'application avait également pour but de servir les pages HTML et d'afficher certaines données sur les pages web.

### La REST API

Pour afficher les différentes pages du service en ligne, séparé l'application AMORA a été séparée en deux parties distinctes : l'application web, qui sert les pages principales de l'application, et la REST API. Cette API avait pour but de proposer une liste de endpoints permettant la communication entre le frontend et le backend. Typiquement, l'application sert à présenter toutes les données statiques qui ne changeront pas pendant l'exécution du service, tandis que la REST API sert à présenter toutes les données qui vont venir modifier la page actuelle.

Pour construire la REST API, une liste de endpoints nécessaire au bon fonctionnement de logiciel a été répertoriée. Chaque endpoint a une fonctionnalité particulière. Un endpoint créé, par exemple, la session, un autre renvoie le statut d'un processus, un autre encore permet de supprimer la session, etc. Je suis venu régulièrement enrichir cette liste de endpoints au cours de la phase d'implémentation car il arrive régulièrement qu'une fonctionnalité non anticipée soit en fait nécessaire.

Endpoints	HTTP request type	Goal	Return	Parameters
<a href="http://&lt;service&gt;/job/">http://&lt;service&gt;/job/</a>	POST	Create session with the given shape if it was specified	Name of the session created	json string containing the shape
<a href="http://&lt;service&gt;/job/&lt;session-name&gt;/shape/">http://&lt;service&gt;/job/&lt;session-name&gt;/shape/</a>	PUT	Register shape in session	json file path containing the shape	json string containing the shape
<a href="http://&lt;service&gt;/job/&lt;session-name&gt;/get-shape/">http://&lt;service&gt;/job/&lt;session-name&gt;/get-shape/</a>	GET	Get shape registered in session	json string containing the shape	
<a href="http://&lt;service&gt;/job/&lt;session-name&gt;/&lt;camera&gt;/obs">http://&lt;service&gt;/job/&lt;session-name&gt;/&lt;camera&gt;/obs</a>	GET	Get all Obs matching the shape	A json result with all the obs matching the shape	
<a href="http://&lt;service&gt;/job/&lt;session-name&gt;/select-obs">http://&lt;service&gt;/job/&lt;session-name&gt;/select-obs</a>	POST	Select set of Obs	The location of the file storing the selection	A json containing the observations selected *
<a href="http://&lt;service&gt;/job/&lt;session-name&gt;/&lt;camera&gt;/select-obs">http://&lt;service&gt;/job/&lt;session-name&gt;/&lt;camera&gt;/select-obs</a>	GET	Get selected set of Obs	A json response as a list of all the observations	

Figure 49 : Extrait du tableau des endpoints (disponible en annexe)

### La gestion de l'asynchronisme

Le fonctionnement des processus a été une des parties centrales de la conception de l'interface. En termes d'expérience utilisateur, il fallait que l'utilisateur ait constamment une

mise à jour sur le fonctionnement du processus. Il n'existait cependant aucun moyen simple de surveiller l'état d'une donnée d'une API Rest sans utiliser une bibliothèque externe. Certains langages de programmation utilisent un design pattern<sup>17</sup> appelé observer. Ce dernier permet de surveiller l'état d'une donnée en étant notifié lorsque la valeur de ladite donnée est modifiée. Néanmoins, étant donné que seul le backend pouvait initier une requête sur le backend, il n'était pas possible d'utiliser une telle solution dans ce cas précis.

Il a été nécessaire de concevoir une solution permettant de pouvoir connaître le statut du backend. Il a été décidé d'utiliser une boucle cyclique qui effectue des requêtes au backend pour connaître l'état de la donnée. Toutes les x périodes, le frontend connaît l'état de la donnée et peut vérifier si le processus est encore en train de tourner. Si c'est le cas, la boucle continue,

sinon la boucle est arrêtée et le frontend demande au serveur les données générées.

Pendant que le frontend cherche à récupérer les données, le backend lui démarre le processus dans un thread afin de ne pas bloquer la boucle principale. Le thread va alors démarrer le script sélectionné depuis le frontend avec les paramètres choisis par l'utilisateur s'il y en a. Les paramètres sont envoyés sous la forme de paramètres optionnels ou positionnels selon la configuration renseignée dans le fichier de configuration **scripts.json**.

Le thread fait tourner le processus dans un sous-processus, que l'on peut considérer comme un autre thread, qui permet de ne pas bloquer le thread principal et de renseigner régulièrement l'état de la donnée dans un fichier de statut (**status.json**). Les logs du script, c'est-à-dire les caractères affichés à chaque exécution de commande, sont également écrits dans un fichier à chaque actualisation. Le processus peut être interrompu par l'utilisateur à tout moment. Dans ce cas, l'information est retransmise au thread qui va exécuter un signal SIGINT afin d'interrompre l'exécution du sous-processus. Que le processus ait été interrompu ou non, le thread va finir son exécution en écrivant la manière dont il s'est achevé dans le fichier de statut. Il va enfin réordonner les données pour séparer les images (png, jpeg, etc.) des autres fichiers. Ces résultats sont accessibles par le frontend qui peut alors les afficher sur l'interface.

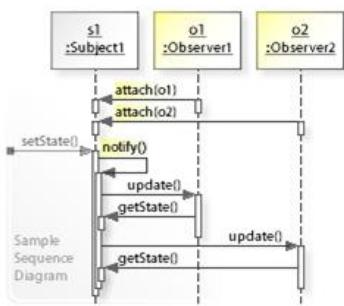


Figure 50 : Diagramme de séquence d'un observer

<sup>17</sup> Un design pattern ou patron de conception est une solution classique à un problème récurrent. Il consiste en général en un assemblage particulier de classes et de méthodes pour obtenir une interaction particulière entre deux classes.

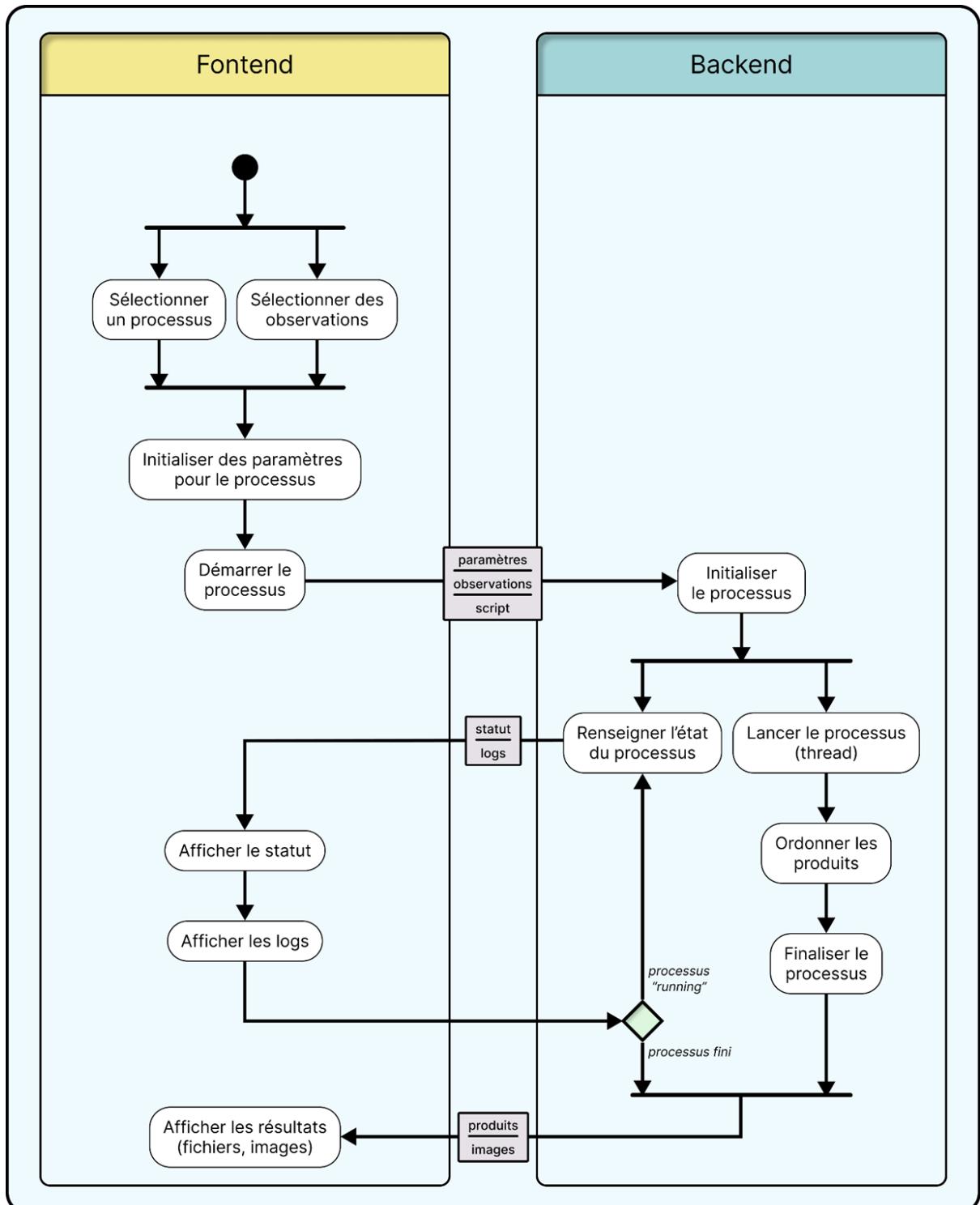


Figure 51 : Diagramme d'activité du fonctionnement des processus

## Les sessions d'AMORA

Le backend de l'application est la partie centrale de cette dernière. Comme dit précédemment, le backend a été conçu sur le principe d'une REST API. Cette REST API permet de contrôler entièrement les actions réalisées dans la session sur le serveur. En

théorie, il serait donc possible de manipuler les données d'une session simplement en se servant des endpoints.

Afin de faciliter la lecture des endpoints de l'API dans le code Python, mon tuteur m'a suggéré de séparer la logique, c'est-à-dire tous les traitements relevant de la manipulation de données ou de calculs, et la déclaration des endpoints. Cela permet également de rendre certaines actions beaucoup plus rapides. En effet, il est parfois nécessaire, au sein même du backend d'utiliser des fonctionnalités spécifiques à un endpoint dans un autre. Sans cette séparation de la logique, le backend doit lui-même passer par un requête HTTP pour accéder à une donnée alors que cette donnée est localement présente.

Chaque session de l'application correspond à un répertoire dans les fichiers de l'application. Ces répertoires ont une architecture qui est manipulée par un "session manager". Cette session manager va donc pouvoir ajouter, supprimer, déplacer ou accéder des fichiers ou répertoire à l'intérieur du répertoire de la session. Chaque session est également liée à un thread. Ce dernier est généré lorsque l'utilisateur démarre un processus. Le rôle du thread est de faire tourner le processus sur la machine tout en surveillant les entrées, sorties et l'avancement de celui-ci. A l'intérieur de la session, on retrouve un fichier stockant la forme envoyée depuis Alix. Afin de pouvoir la lire, la session possède une classe permettant de la lire puis de l'afficher par divers moyens ( la classe GeomtricMOC). Cette classe permet notamment de générer l'affichage du MOC correspondant à la forme enregistrée.

Finalement, la session stocke un répertoire pour chaque script à l'intérieur de la session. Chacun de ces scripts est défini par une liste de paramètres qui définit avec quels types de paramètres le script doit être démarré. Les types de paramètre doivent être extensibles afin de pouvoir en rajouter d'autres dans le futur. C'est pourquoi chaque paramètre hérite d'une classe abstraite qui définit quelques propriétés et méthodes d'un paramètre. Chaque paramètre est également doté d'un "template" qui permet de définir le code HTML lié à ce type de paramètre particulier.

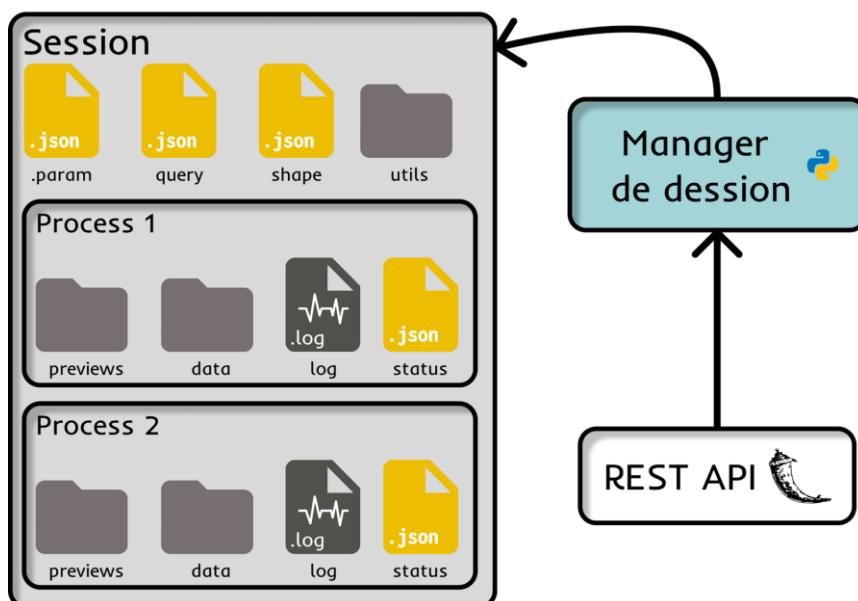


Figure 52 : Structure des sessions AMORA

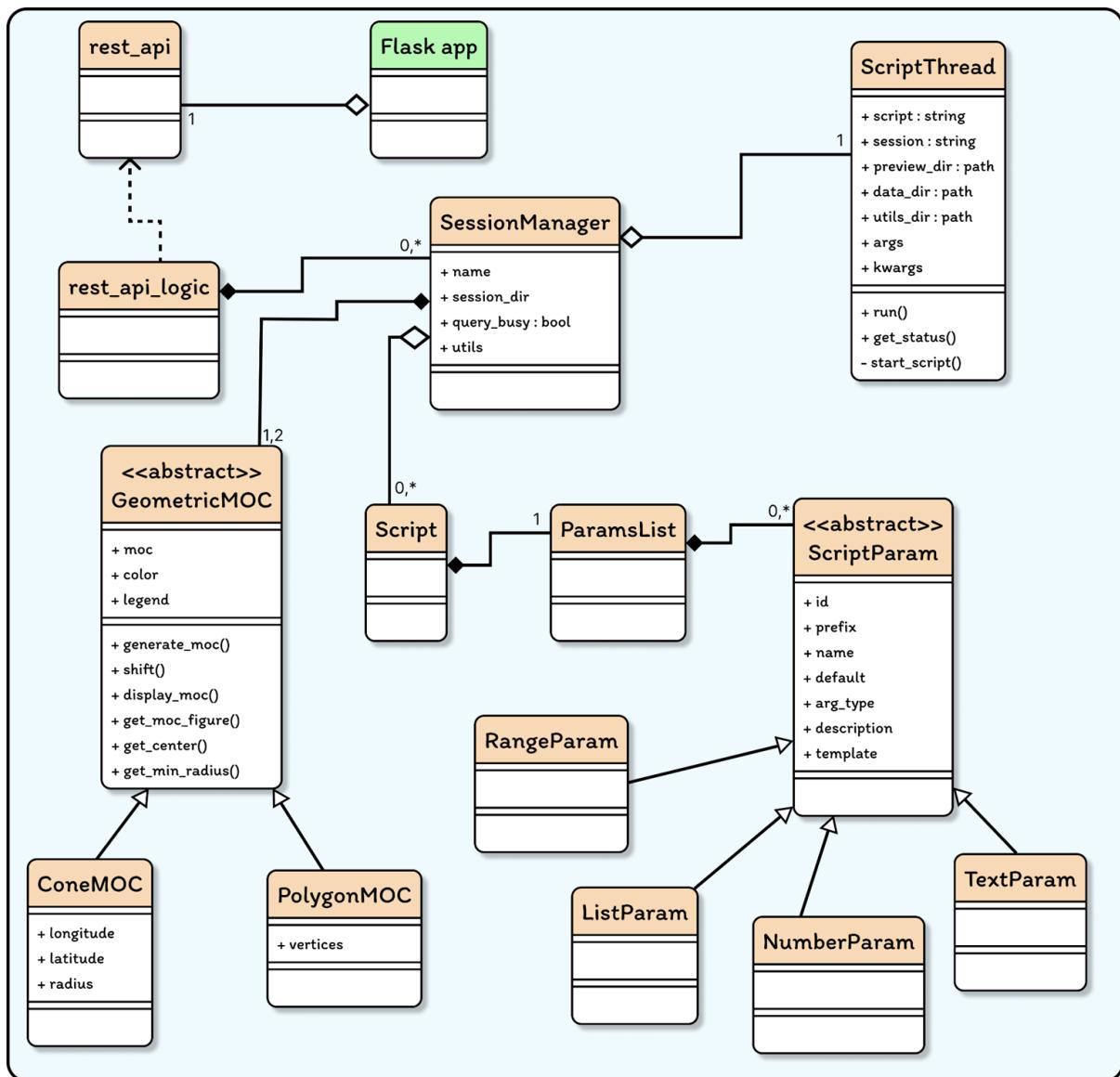


Figure 53 : Diagramme de classe du backend d'AMORA

## d. Implémentation

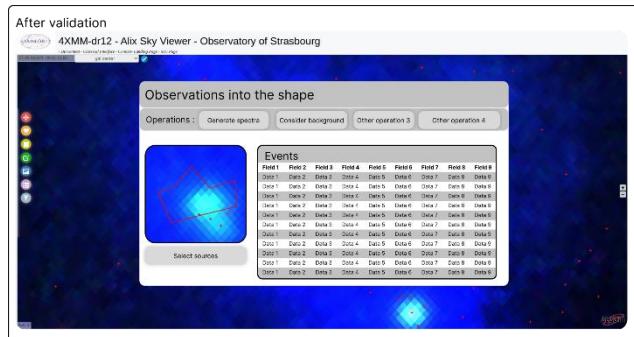


Figure 55 : Première maquette d'AMORA

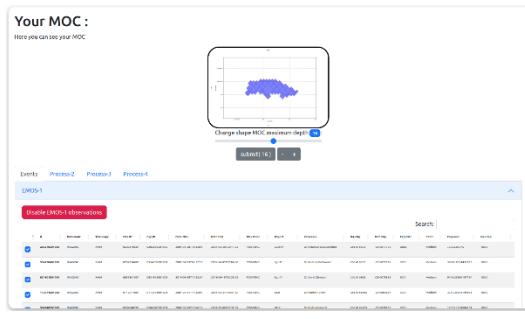


Figure 54 : Première approche de l'interface d'AMORA

Pour concevoir l'interface de mon service, je me suis basé sur l'interface existante du traitement de données liée à une source unique dans XCatDB. J'ai ensuite utilisé les composants de Bootstrap pour réaliser une interface plus ou moins proche de la maquette. Cette première approche m'a permis de concevoir une interface sommaire qui permettait de tester les fonctionnalités décidées lors de la phase de conception.

L'interface a changé tout au long de l'implémentation du logiciel. Le plus gros changement a sûrement été le menu de contrôle des processus. En effet, il a été décidé dès le début de la conception du logiciel que mon logiciel ne pourrait faire tourner qu'un seul script à la fois par session. Ce choix a permis de simplifier d'une part le fonctionnement interne du logiciel mais également la représentation visuelle du statut d'un processus.



Figure 56 : Maquette et première implémentation de la barre de contrôle des processus

Un certain nombre d'éléments visuels ont dû être ajoutés sur l'interface comme le bouton de démarrage, une description du processus et une fonctionnalité de "staging log". Lors de l'exécution d'un processus, de nombreuses informations sont affichées dans le terminal de l'utilisateur. Notre but était de retransmettre ce fonctionnement sur le service conçu en proposant un cadre semblable à une console qui affiche le résultat des actions exécutées par le script. Ces résultats sont récupérés, comme le statut, à intervalle de temps régulier. Certaines de ces lignes commencent par un mot-clé entre crochets ([STAGE] par exemple). Elles sont mises en surbrillance dans la fenêtre des logs et sont également affichées dans la barre de staging log.



Figure 57 : Maquette de l'interface de contrôle des scripts

```
[STAGE] bash /home/aviala/service_host/xsasd/xcatdb-online-processing/scripts/generate_previews_from_obs.sh
[STAGE] Processing file P0764191201M25002IMAGE_8000.FIT.gz
Scan the Observation directory
[STAGE] WARNING: FITSfixedWarning: RADECSYS= 'FK5' / World coord. system for this file
the RADECSYS keyword is deprecated, use RADESYSa [astropy.wcs.wcs]
[STAGE] WARNING: FITSfixedWarning: 'datfix' made the change Set MJD-OBS to 57452.978785 from DATE-OBS.
Set MJD-END to 57453.344826 from DATE-END [astropy.wcs.wcs]
[STAGE] Generating picture from P0764191201M25002IMAGE_8000.FIT.gz
[INT] Process execution: generate_previews_from_obs.sh interrupted
```

Figure 58 : Console intégrée dans AMORA

**Observation selector**

Select camera

**Process control**

Generate multiple previews

Shell script that generate preview from selected observations

[INT] Process execution: generate\_previews\_from\_obs.sh interrupted

2023-01-26 23:59:46 00:00:03 interrupted

**Searched region**

order 16

**Logs**

**Previews**

Preview	Obs ID	Camera	Date obs	Exposure time	Creation date	File size	Download
	0764191201	EMOS2	2016-03-05T23:29:27	31619	01/26/2023, 23:59:49	62.2KiB	<input type="button" value="Download"/>

Figure 59 : Interface finale d'AMORA

# VI. Conclusion

## 1. Travail accompli

Durant mon stage, j'ai eu le temps d'accomplir un certain nombre de missions. Parmi ces projets, il y a notamment la création d'un service en ligne, l'Asynchronous Multi-Observation Region based Analysis (AMORA). Ce service en ligne est couplé à une REST API qui permet à un utilisateur d'effectuer des traitements sur des observations du satellite Newton-XMM.

L'éditeur de région d'AliX a également été amélioré en lui intégrant une fonctionnalité de traçage de cercle et en ajoutant la possibilité de tracer une région de bruit de fond pour être mis en perspective avec la région de source.

Enfin j'ai créé un logiciel en ligne de commande permettant de générer d'indexer les observations des trois caméras du satellite Newton-XMM.

Lors de ce stage, j'ai utilisé nombre de mes compétences acquises en cours. J'ai notamment utilisé mes connaissances en programmation orientée objet afin de réaliser des logiciels extensibles. J'ai également mis à profit mes connaissances en conception d'interfaces centrées utilisateurs pour rendre mes solutions intuitives pour un utilisateur. J'ai également beaucoup appris, notamment sur le fonctionnement des technologies web. En effet, je n'avais jamais utilisé de framework de développement web avant ce stage. J'ai beaucoup appris sur leur fonctionnement en me documentant sur le sujet et en réalisant une REST API.

Le service AMORA et les modifications sur l'éditeur de région vont être prochainement déployés dans XCatDB, un logiciel accessible au public. Un article sur la page facebook du projet XMM2ATHENA va également être publié afin de présenter le travail que j'ai réalisé durant mes 6 mois à l'observatoire astronomique de Strasbourg.

A l'achèvement de mon stage, j'ai laissé un certain nombre de documentation et de scripts pour faciliter la reprise de mon projet. Certains de ces fichiers de documentation sont disponibles en annexe. J'ai également documenté une grande partie du code que j'ai écrit, toujours dans l'optique de faciliter la passation.

## 2. Prospectives

Bien que mon stage arrive à sa fin, le logiciel que j'ai conçu n'est pas parfait. Aussi, un grand nombre de fonctionnalités sont encore manquantes. Certaines seraient nécessaires afin d'améliorer l'expérience des utilisateurs ainsi que de multiplier les cas d'utilisations de l'interface. Ces améliorations sont les suivantes :

- La création d'une Interface administration. Elle permettrait de pouvoir purger des sessions qui ne sont plus actives depuis une certaine période. Cette interface permettrait également d'établir des quotas sur le nombre d'actions réalisables par les utilisateurs. Elle permettrait également directement de téléverser les scripts sans avoir à passer par git. Cette interface serait évidemment réservée aux administrateurs et devrait donc bénéficier d'un système d'authentification.
- L'amélioration des bookmarks. Le fonctionnement actuel des signets est parfois contre-intuitif. Il faudrait pouvoir rendre ce fonctionnement plus simple. On pourrait notamment implémenter un système de partage de session qui permettrait de

récupérer la région d'un autre autre utilisateur avec la session AMORA qui lui est attribuée.

- La correction d'un bug concernant des conversions de coordonnées. Nous utilisons une conversion permettant de convertir des coordonnées d'un plan cartésien local en coordonnées célestes. Toutefois la précision du plan cartésien local n'est pas suffisante et la conversion est donc souvent approximative. Cela peut créer des problèmes d'offset (décalage) des formes tracées lors d'un passage d'un système de coordonnées à un autre.
- L'ajout de la possibilité de lancer plusieurs processus en parallèle dans le service AMORA. On pourrait utiliser plusieurs threads pour cela.
- L'ajout d'une possibilité de connecter les résultats d'un script en entrée d'un autre dans le service AMORA. On pourrait imaginer des enchaînements d'opérations élémentaires sur les observations.
- L'intégration d'un fonctionnement en mode REST au travers d'un endpoint unique dans l'API. Ainsi, on pourrait démarrer un script simplement en exécutant une requête sur un endpoint qui créerait une session, exécuterait le script et récupérerait les résultats dans une archive compressée à partir d'une position et d'un rayon.
- La modification des scripts pour qu'ils puissent traiter chaque observation indépendamment en utilisant le multi-processing.

## VII. Bibliographie

### Documents consultés

- OBSERVATOIRE ASTRONOMIQUE DE STRASBOURG. Observatoire Astronomique de Strasbourg [En ligne]. Disponible sur : <https://astro.unistra.fr/fr/> (Consulté le 11/01/2023).
- WIKIPEDIA. Framework [En ligne]. Disponible sur : <https://fr.wikipedia.org/wiki/Framework> (Consulté le 18/01/2023).
- FONDATION MOZILLA. Méthodes de requêtes HTTP [En ligne] <https://developer.mozilla.org/fr/docs/Web/HTTP/Methods> (Consulté le 18/01/2023).
- WIKIPEDIA. JQuery [En ligne]. Disponible sur : <https://fr.wikipedia.org/wiki/JQuery> (Consulté le 18/01/2023).
- WIKIPEDIA. Eclipse (projet) [En ligne]. Disponible sur : [https://fr.wikipedia.org/wiki/Eclipse\\_\(projet\)](https://fr.wikipedia.org/wiki/Eclipse_(projet)) (Consulté le 18/01/2023).
- SIDDHY. Les boucles (for, foreach, each) en javascript [En ligne]. Disponible sur : <https://www.zendevs.xyz/les-boucles-for-foreach-each-en-javascript/#javascript-each-jquery> (Consulté le 19/01/2023).
- WIKIPEDIA. ECMAScript [En ligne]. Disponible sur : [https://fr.wikipedia.org/wiki/ECMAScript#ECMAScript\\_Edition\\_6\\_\(ES6\)](https://fr.wikipedia.org/wiki/ECMAScript#ECMAScript_Edition_6_(ES6)) (Consulté le 19/01/2023).
- WIKIPEDIA. Forge (informatique) [En ligne]. Disponible sur : [https://fr.wikipedia.org/wiki/Forge\\_\(informatique\)](https://fr.wikipedia.org/wiki/Forge_(informatique)) (Consulté le 19/01/2023)
- WIKIPEDIA. Flexible Image Transports System [En ligne]. Disponible sur : [https://fr.wikipedia.org/wiki/Flexible\\_Image\\_Transport\\_System](https://fr.wikipedia.org/wiki/Flexible_Image_Transport_System) (Consulté le 19/01/2023).
- AGENCE SPATIALE EUROPÉENNE. WHat is SAS - XMM Newton - Cosmos [En ligne]. Disponible sur : <https://www.cosmos.esa.int/web/xmm-newton/what-is-sas> (Consulté le 19/01/2023).
- OBSERVATOIRE ASTRONOMIQUE DE STRASBOURG. Patrimoine historique - Observatoire Astronomique de Strasbourg [En ligne]. Disponible sur : <https://astro.unistra.fr/fr/tout-public/patrimoine-historique/#~:text=Le%20premier%20observatoire%20de%20la,de%20sciences%20et%20de%20m%C3%A9decine> (Consulté le 19/01/2023).
- ARCHIVES DE LA VILLE ET DE L'EUROMÉTROPOLE DE STRASBOURG. Les bâtiments remarquables de Strasbourg [En ligne]. Disponible sur : <https://archives.strasbourg.eu/chronologie/liste/les-batiments-remarquables-de-strasbourg-2/n:334> (Consulté le 19/01/2023).
- WIKIPEDIA. Observatoire astronomique de Strasbourg [En ligne]. Disponible sur : [https://fr.wikipedia.org/wiki/Observatoire\\_astronomique\\_de\\_Strasbourg](https://fr.wikipedia.org/wiki/Observatoire_astronomique_de_Strasbourg) (Consulté le 19/01/2023).
- WIKIPEDIA. Histoire de Strasbourg [En ligne]. Disponible sur : [https://fr.wikipedia.org/wiki/Histoire\\_de\\_Strasbourg](https://fr.wikipedia.org/wiki/Histoire_de_Strasbourg) (Consulté le 19/01/2023).
- WIKIPEDIA. André Danjon [En ligne]. Disponible sur : [https://fr.wikipedia.org/wiki/Andr%C3%A9\\_Danjon](https://fr.wikipedia.org/wiki/Andr%C3%A9_Danjon) (Consulté le 19/01/2023).
- XMM-NEWTON SURVEY SCIENCE CENTRE. Le projet XMM-Newton [En ligne]. Disponible sur: <http://xmm-ssc.irap.omp.eu/fr/le-projet/> (Consulté le 22/01/2023).

- INTERNATIONAL VIRTUAL OBSERVATORY ALLIANCE. IVOA Recommendation - HiPS - Hierarchical Progressive Survey [En ligne]. REC-HIPS-1.0-20170519. IVOA, 2017, 32 p. Disponible sur : <https://www.ivoa.net/documents/HiPS/> (Consulté le 24/01/2023).
- WIKIPEDIA. Jinja (moteur de template) [En ligne]. Disponible sur : [https://fr.wikipedia.org/wiki/Jinja\\_\(moteur\\_de\\_template\)](https://fr.wikipedia.org/wiki/Jinja_(moteur_de_template)) (Consulté le 26/01/2023).
- BOOTSTRAP TEAM. Get started with Bootstrap [En ligne]. Disponible sur : <https://getbootstrap.com/docs/5.3/getting-started/introduction/> (Consulté le 26/01/2023).
- SPRYMEDIA LTD. DataTables - Table plug-in for JQuery [En ligne]. Disponible sur : <https://datatables.net/> (Consulté le 26/01/2023).
- WIKIPEDIA. SQLite [En ligne]. Disponible sur : <https://fr.wikipedia.org/wiki/SQLite> (Consulté le 26/01/2023).
- INTERNATIONAL VIRTUAL OBSERVATORY ALLIANCE. IVOA Recommendation - MOC: Multi-Order Coverage map [En ligne]. REC-moc-2.0-20220727. IVOA, 2022, 33 p. Disponible sur : <https://ivoa.net/documents/MOC/> (Consulté le 26/01/2023).
- WIKIPEDIA. tessellation [En ligne]. Disponible sur : <https://fr.wiktionary.org/wiki/tessellation> (Consulté le 26/01/2023).
- Górski & Hivon et al. HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere [En ligne]. The Astrophysical Journal 622(2), 759-771, arXiv:astro-ph/0409513. Disponible sur : <http://doi.org/10.1086/427976> (Consulté le 26/01/2023).
- REFACTORYING.GURU. Patrons de Conception / Design Patterns [En ligne]. Disponible sur : <https://refactoring.guru/fr/design-patterns> (Consulté le 29/01/2023).

## Illustrations utilisées

- CENTRE DE DONNEES ASTRONOMIQUE DE STRASBOURG. Logo du CDS. Disponible sur : [https://cortecs.unistra.fr/plateforme/?tx\\_ameosplatforms\\_platformviewerdetail%5Bplatform%5D=172&cHash=cdde73cf884808b85990e907cb29c804](https://cortecs.unistra.fr/plateforme/?tx_ameosplatforms_platformviewerdetail%5Bplatform%5D=172&cHash=cdde73cf884808b85990e907cb29c804) (Consulté le 13/01/2023).
- OBSERVATOIRE ASTRONOMIQUE DE STRASBOURG. Logo de l'Observatoire Astronomique de Strasbourg. Disponible sur : <https://astro.unistra.fr/> (Consulté le 13/01/2023).
- AGENCE SPATIALE EUROPEENNE. XMM-Newton mission logo. Disponible sur : [https://www.esa.int/ESA\\_Multimedia/Images/2013/06/XMM-Newton\\_mission\\_logo](https://www.esa.int/ESA_Multimedia/Images/2013/06/XMM-Newton_mission_logo) (Consulté le 13/01/2023).
- Propriétés du spectre électro-magnétique. Disponible sur : [https://fr.wikipedia.org/wiki/Spectre\\_%C3%A9lectromagn%C3%A9tique](https://fr.wikipedia.org/wiki/Spectre_%C3%A9lectromagn%C3%A9tique) (Consulté le 17/01/2023).
- PLINE. Schéma optique d'un télescope Wolter de type I. Disponible sur : [https://fr.wikipedia.org/wiki/Hitomi\\_\(satellite\)](https://fr.wikipedia.org/wiki/Hitomi_(satellite)) (Consulté le 17/01/2023).
- NASA/JPL-CALTECH. Une optique Wolter du télescope X Nustar similaire à celles embarquées sur XMM-Newton. Disponible sur : <https://fr.wikipedia.org/wiki/XMM-Newton> (Consulté le 17/01/2023).

- ECLIPSE FOUNDATION. Eclipse IDE Logo. Disponible sur : <https://www.eclipse.org/org/artwork/> (Consulté le 19/01/2023).
- GITLAB. Logo Gitlab. Disponible sur : [https://gitlab.com/gitlab-org/gitlab-foss-/blob/master/app/assets/images/logo\\_wordmark.svg](https://gitlab.com/gitlab-org/gitlab-foss-/blob/master/app/assets/images/logo_wordmark.svg) (Consulté le 19/01/2023).
- JQUERY FOUNDATION. Logo de Jquery. Disponible sur : <http://brand.jquery.org/logos/#the-mark> (Consulté le 19/01/2023).
- CHRIS WILLIAMS. Logo non officiel de Javascript. Disponible sur : <https://github.com/voodootikigod/logo.js/blob/1544bdeed/js.svg> (Consulté le 19/01/2023).
- PALLETS FOUNDATION. Logo de Flask. Disponible sur : <https://flask.palletsprojects.com/en/2.2.x/> (Consulté le 19/01/2023).
- OBSERVATOIRE ASTRONOMIQUE DE STRASBOURG. Tour de l'Hôpital civil – Strasbourg. Disponible sur : <https://astro.unistra.fr/fr/tout-public/patrimoine-historique/#:~:text=Le%20premier%20observatoire%20de%20la,de%20sciences%20et%20de%20m%C3%A9decine> (Consulté le 19/01/2023).
- OBSERVATOIRE ASTRONOMIQUE DE STRASBOURG. Vue aérienne du bâtiment Est de la grande coupole de l'observatoire. Disponible sur : <https://astro.unistra.fr/fr/tout-public/patrimoine-historique/#:~:text=Le%20premier%20observatoire%20de%20la,de%20sciences%20et%20de%20m%C3%A9decine> (Consulté le 19/01/2023).
- SURVEY SCIENCE CENTRE. Logo de XcatDB. Disponible sur : <https://github.com/lmichel/alix/blob/master/images/Xcat.gif> (Consulté le 24/01/2023).
- CENTRE DE DONNEES ASTRONOMIQUES DE STRASBOURG. Logo de Aladin. Disponible sur : <https://aladin.u-strasbg.fr/AladinLite/> (Consulté le 24/01/2023).
- JASON LONG. Logo de git. Disponible sur : <https://git-scm.com/downloads/logos> (Consulté le 26/01/2023).
- PALLETS FOUNDATION. Logo de Jinja. Disponible sur : <https://github.com/pallets/jinja/blob/master/artwork/jinjalogo.svg> (Consulté le 26/01/2023).
- NEW ZEALAND eSCIENCE INFRASTRUCTURE. Git Diagram. Disponible sur : <https://support.nesi.org.nz/hc/en-gb/articles/360001508515-Git-Reference-Sheet> (Consulté le 26/01/2023).
- BOOTSTRAP TEAM. Logo de Bootstrap. Disponible sur : <https://blog.getbootstrap.com/> (Consulté le 26/01/223).
- MICROSOFT CORPORATION. Logo de Visual Studio Code. Disponible sur : <https://code.visualstudio.com/brand> (Consulté le 26/01/2023).
- DWAYNE RICHARD HIPP. Logo de SQLite. Disponible sur : <https://commons.wikimedia.org/wiki/File:SQLite370.svg> (Consulté le 26/01/2023).
- NIKHIL VERMA. Concurrency of code. Disponible sur : <https://lih-verma.medium.com/multi-processing-in-python-process-vs-pool-5caf0f67eb2b> (Consulté le 28/01/2023).
- VANDERJOE. A sample UML class and sequence diagram for the observer design pattern. Disponible sur : [https://en.wikipedia.org/wiki/Observer\\_pattern](https://en.wikipedia.org/wiki/Observer_pattern) (Consulté le 29/01/2023).

## Mots clefs

Aéronautique, Espace – Fonction publique - Informatique – Recherche - Architecture logicielle – IHM – Logiciel scientifique

**VIALA Alexandre**

**Rapport de stage ST42 – A2022**

## Résumé

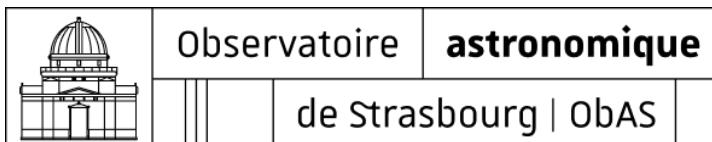
L'Observatoire Astronomique de Strasbourg (ObAS) est un Observatoire des sciences de l'Univers (OSU). A ce titre, ses objectifs sont la recherche et l'enseignement dans le domaine de l'astronomie. Cet observatoire est divisé en deux équipes : l'équipe du Centre de Données astronomique de Strasbourg (CDS) et l'équipe « Galaxies, High Energy, Cosmology, Compact Objects & Stars » (GALHECOS).

Au cours de mon stage, j'ai intégré l'équipe GALHECOS au sein d'une unité de recherche du Survey Science Centre XMM (SSC-XMM). L'intégration d'une équipe de recherche m'a permis de collaborer avec des astrophysiciens. J'ai été amené à ajouter des fonctionnalités dans le logiciel XCatDB développé par mon maître de stage M. Laurent MICHEL, ingénieur de recherche. Ma mission consistait à permettre aux utilisateurs de cette application en ligne de pouvoir effectuer des traitements scientifiques (création de spectres, courbes de lumières, périodogrammes, etc.) à partir de zones du ciel définies par l'utilisateur. Pour cela, j'ai créé un service en ligne, l'Asynchronous Multi Observation Region-based Analysis (AMORA), que j'ai interfacé avec l'éditeur de région disponible dans XcatDB. J'ai également été amené à modifier cet éditeur de région en lui ajoutant des fonctionnalités.

Pour l'ensemble du stage, j'ai dû mobiliser mes connaissances acquises à l'UTBM en concevant des interfaces homme-machine. Ces dernières devaient être adaptées aux demandes des utilisateurs scientifiques. J'ai également utilisé mes connaissances en technologies web et en programmation orientée objet pour l'implémentation de l'ensemble des logiciels que j'ai été amené à concevoir.

# Observatoire Astronomique de Strasbourg

11 rue de L'Université  
67 000 Strasbourg  
[astro.unistra.fr](http://astro.unistra.fr)





# Annexes

## Table des matières

Annexes.....	2
Annexe 1 : Descriptions détaillée des outils et protocoles.....	3
a. Python .....	3
b. Javascript et CSS .....	3
c. JQuery .....	4
d. Eclipse .....	4
e. Le Gitlab du CDS.....	4
f. Git.....	5
g. Flask & Jinja .....	6
h. Bootstrap .....	6
i. DataTable .....	6
j. Visual Studio Code .....	7
k. SQLite.....	8
Annexe 2 : Procédure d'installation .....	8
Step 1: Logging.....	8
Step 2: Install HEASoft & the SAS .....	8
Step 3 : Install Conda.....	8
Step 4 : Install & initialize the Web Service .....	9
Step 5 : Generate MOCSet .....	9
Step 6 : Launch the service.....	11
Annexe 4 : Ecriture de scripts scientifiques dans AMORA.....	12
Preamble .....	12
Environment variables.....	12
Shell util scripts .....	13
background_session.py .....	13
shape_session.py .....	15
list_event.py .....	17
list_obs.py .....	17
compression.py.....	17
obs_countdown.py .....	18
xspec_plt.py.....	19
Annexe 5 : Descriptions des scripts et des paramètres de scripts .....	20
Define scripts in the web service.....	20

Location of the scripts.json file .....	21
Annexe 6 : Utilisation de MOCSet Generation .....	24
Options .....	24
Filtering options.....	24
Some example .....	25
Annexe 7 : Tableau des endpoints de la REST-API.....	25
Annexe 8 : Script de minification d'AliX et XCatDB.....	29

## Annexe 1 : Descriptions détaillée des outils et protocoles

### a. Python

Afin de réaliser le backend de l'application sur laquelle j'ai travaillé, on m'a imposé l'utilisation du langage de programmation Python. Ce backend a été réalisé avec le framework<sup>1</sup> Flask. Ce dernier permet de facilement servir les points d'accès (ou endpoints) d'API REST<sup>2</sup>. Ce framework est très léger et permet aux développeurs de définir comme bon leur semble ces endpoints. Il est ainsi possible, très facilement de définir le type de requête<sup>3</sup> traitées sur les points d'accès.



*Figure 1: Logo de Flask*

### b. Javascript et CSS

Couplé au backend, j'ai dû également travailler sur un frontend. Pour cela, on m'a imposé l'utilisation de Javascript et de CSS. Il s'agit des outils traditionnels pour concevoir un frontend de sites internet. Javascript est un langage de programmation permettant de modifier en temps réel le contenu d'une page internet. CSS est un langage informatique permettant de styliser des balises HTML<sup>4</sup>.



*Figure 2 : Logo de Javascript*

Les alternatives auraient pu être le typescript ou l'utilisation d'un framework tel que Svelte, Vue ou React pour le javascript. Des alternatives au CSS auraient pu être SCSS, SASS ou bien encore tailwind. Ces solutions sont cependant relativement lourdes et nécessitent l'ajout de surcouches logicielles sans garantie de respect de la compatibilité ascendante

---

<sup>1</sup> Un framework est un ensemble d'outils logiciels cohérent. Un framework peu spécialisé contrairement à une bibliothèque.

<sup>2</sup> Une API REST est une interface pour accéder à une application depuis le web. Elle permet de fournir l'état d'une donnée sans connaître le traitement sous-jacent.

<sup>3</sup> On parle ici des requêtes POST, GET, PUT et autres requêtes définies par le standard HTTP.

<sup>4</sup> HTML (HyperText Markup Language) utilise des balises pour définir sémantiquement les éléments d'une page internet (comme un titre, un article, une barre de navigation, etc.).

### c. JQuery

En travaillant directement sur le code de la XCatDB, j'ai été amené à utiliser la bibliothèque javascript JQuery. En effet, la partie frontend de la XCatDB a entièrement été réalisée en HTML, CSS et JQuery.

JQuery est une surcouche de Javascript créée en 2006. Son utilisation est largement répandue car elle permet de simplifier la syntaxe parfois lourde du javascript. Sa popularité au sein de la communauté l'a amené à être considéré sérieusement par Ecma International<sup>5</sup>. Ainsi, nombre de fonctions de JQuery ont été implémentées nativement dans Javascript à partir du standard ES6 (ou ES2015).



Figure 3 : Logo de JQuery

### d. Eclipse

Afin de travailler sur la XCatDB, j'ai dû utiliser l'environnement de développement (IDE) Eclipse. La XCatDB est scindée en trois projets distincts requérant une installation particulière afin de fonctionner ensemble (notamment un backend J2EE). Eclipse facilite cette installation et propose des outils permettant de lancer le service en local.

Cet environnement de développement a été créé en 2001. Il avait alors pour but de créer un ensemble de solutions propice à la création de logiciels. Il se basait alors sur les principes du développement Java. Depuis, le logiciel a bien évolué et de nombreuses versions plus ou moins spécialisées ont émergé. Il est maintenant possible d'avoir par exemple des aides contextuelles pour réaliser des pages HTML, des feuilles style en cascade (CSS) ou bien des codes en Javascript.



Figure 4 : Logo d'Eclipse

### e. Le Gitlab du CDS

Gitlab est un logiciel de forge<sup>6</sup> basé sur le gestionnaire de source git<sup>7</sup>. La particularité de cette forge est qu'elle est en sources ouvertes. Il est donc possible d'accéder à l'entièreté de son code source en libre accès sans aucune restriction. Cette forge dispose de plusieurs versions, dont notamment une en ligne hébergée sur la Google Cloud Platform et d'une autre pouvant être déployée localement par l'utilisateur.

Le CDS a fait le choix d'héberger sa propre instance de Gitlab sur ses serveurs. En effet, cette équipe de l'observatoire dispose de plusieurs machines hébergées au sein de l'observatoire permettant de faire tourner des processus et de stocker des données. C'est sur l'une de ces machines qu'est hébergé le gitlab. Pour des raisons de sécurité, la forge n'est accessible que depuis le réseau interne de l'observatoire.

J'ai été amené à utiliser cet outil car bon nombre des projets des équipes GALHECOS et CDS sont stockés sur ces serveurs. C'est un moyen simple de collaborer sur du code de



Figure 5 : Logo de Gitlab

<sup>5</sup> ECMA International est l'organisation ayant pour but de normer une version de Javascript : ECMA.

<sup>6</sup> Un logiciel de forge permet de collaborer sur des documents textes, et notamment des programmes informatiques. Ce sont souvent des sites internet.

<sup>7</sup> git est un gestionnaire de source en ligne de commande. Un gestionnaire de source est un logiciel permettant de tracer localement l'historique des versions d'un logiciel. git est le gestionnaire de source le plus répandu.

manière privée. Certains projets du GitLab CDS ont ensuite été déployés sur GitHub afin de s'ouvrir au public.

## f. Git

Comme énoncé précédemment, j'ai dû utiliser un logiciel de forge en ligne : Gitlab. Pour utiliser ces logiciels, il est nécessaire d'utiliser un gestionnaire de source côté client. Nous avons choisi d'utiliser le plus populaire de ces gestionnaires de sources : Git.

Git est un gestionnaire de sources puissant qui permet de nombreuses opérations. Il facilite par exemple le travail en équipe en permettant de créer ce que l'on appelle des "branches". Il s'agit d'un historique de versions séparées des versions principales. Ces branches permettent de travailler sur une fonctionnalité sans affecter la branche principale. Une fois que les fonctionnalités sont parfaitement au point dans la branche de travail, il est alors possible de fusionner le travail réalisé avec celui de la branche principale. Cette fusion est en général automatiquement réalisée par git. Lorsque l'on travaille sur une fonctionnalité avancée qui change quelque peu la structure du projet, il se peut que la fusion ne puisse pas être réalisée automatiquement. Il faut alors utiliser le gestionnaire de fusion manuel de git afin de fusionner les deux branches.



Figure 6 : Logo de git

git s'utilise en général avec une forge en ligne. La forge peut être vue comme un utilisateur de git qui copie toutes les actions de l'utilisateur. On dit que la forge possède la branche en ligne ou "en amont" (upstream) et que l'utilisateur possède la branche locale. Pour communiquer avec la branche en ligne, il est nécessaire d'effectuer régulièrement des contacts avec le serveur en "poussant" ou en "tirant" le code.

Au travers des projets sur lesquels j'ai travaillé, j'ai dû utiliser deux logiciels de forge : Github pour le projet AliX (cf. § Organisation du travail/XCatDB) et le Gitlab du CDS.

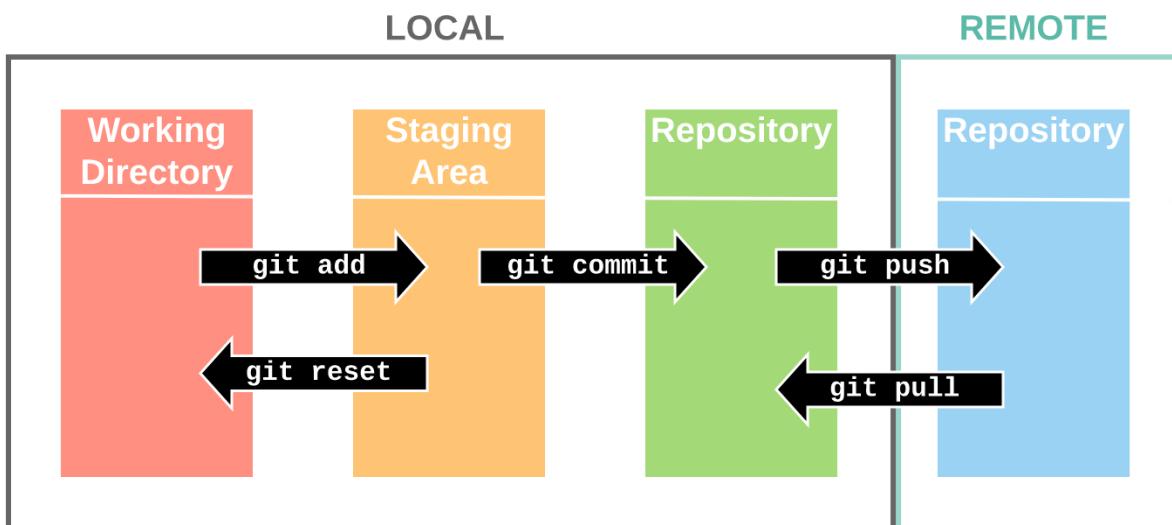


Figure 7 : Fonctionnement de git

## g. Flask & Jinja

Au cours de ma mission, j'ai dû réaliser un service web. On m'a demandé de réaliser ce service en Python avec le framework Flask (cf. § Les contraintes/Python). Flask peut-être utilisé afin de communiquer des données sous toutes les formes. On peut par exemple choisir de communiquer toutes les données sous la forme de JSON, ou bien de texte. On peut finalement s'en servir uniquement comme un backend simple qui ne communique que les données aux pages web de notre service.

Il est toutefois possible de faire des rendus de page html disposant déjà de certaines valeurs au moyen de template Jinja. Jinja est un moteur de template utilisé par le langage Python qui permet d'exécuter du code Python dans ces templates avant que la page ne soit générée.

Cet outil s'est avéré fort utile lorsque j'ai dû concevoir le service. Certaines données sont disponibles dès le lancement de la page. Les template Jinja permettent alors de directement chercher ces données et de les afficher sous la forme précisée par l'utilisateur. Ces templates m'ont permis de gagner un temps précieux sur la réalisation de mes projets.



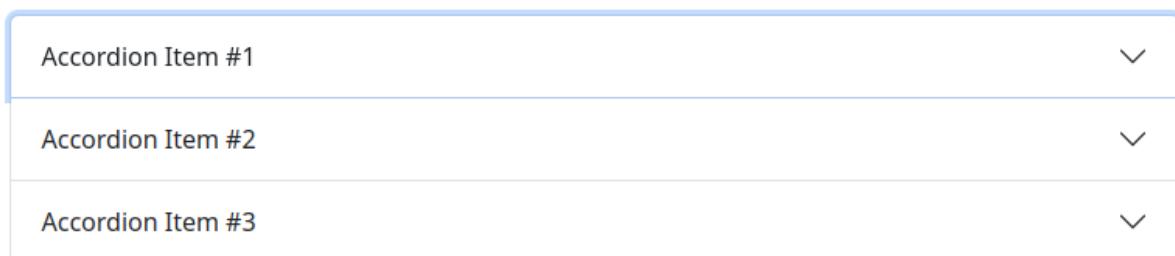
Figure 8 : Logo de Jinja

## h. Bootstrap

En complément de HTML, CSS et Javascript pour la partie front end de mon service, j'ai également utilisé Bootstrap. Bootstrap est une bibliothèque CSS et Javascript proposant un grand nombre de classes CSS permettant de créer des composants que l'on retrouve communément dans toutes sortes d'applications. Bootstrap permet simplement de rendre le contenu d'un site internet plus esthétique tout en évitant au programmeur de devoir réaliser le style complet de son site.



Figure 9 : Logo de Bootstrap



The image shows a screenshot of a Bootstrap accordion component. It consists of three items, each with a title and a downward arrow icon indicating it can be expanded. The titles are "Accordion Item #1", "Accordion Item #2", and "Accordion Item #3".

Accordion Item #1
Content for Accordion Item #1

Accordion Item #2
Content for Accordion Item #2

Accordion Item #3
Content for Accordion Item #3

Figure 10 : exemple de composant Bootstrap (un accordéon)

## i. DataTable

DataTable est une bibliothèque Javascript et CSS permettant de présenter des tableaux sur une page internet. Ces tables peuvent être entièrement configurables pour que le rendu des colonnes et des lignes soit à la convenance de l'utilisateur. Il est notamment possible de choisir le nombre de données à afficher par page d'un tableau, de choisir si l'affichage de la table se fait sur plusieurs pages ou encore d'afficher ou non une barre de recherche permettant de rechercher des mots clefs dans la table.

Cette bibliothèque possède une forte compatibilité avec Bootstrap qui m'a permis d'implémenter ces tables sans avoir à faire de modifications de style dans les tables. Cela s'est avéré être un gain de temps considérable.

Show 10 entries		Search:			
Name	Position	Office	Age	Start date	Salary
Airi Satou	Accountant	Tokyo	33	2008-11-28	\$162,700
Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009-10-09	\$1,200,000
Ashton Cox	Junior Technical Author	San Francisco	66	2009-01-12	\$86,000
Bradley Greer	Software Engineer	London	41	2012-10-13	\$132,000
Brenden Wagner	Software Engineer	San Francisco	28	2011-06-07	\$206,850
Brielle Williamson	Integration Specialist	New York	61	2012-12-02	\$372,000
Bruno Nash	Software Engineer	London	38	2011-05-03	\$163,500
Caesar Vance	Pre-Sales Support	New York	21	2011-12-12	\$106,450
Cara Stevens	Sales Assistant	New York	46	2011-12-06	\$145,600
Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012-03-29	\$433,060
Name	Position	Office	Age	Start date	Salary

Showing 1 to 10 of 57 entries

Previous 1 2 3 4 5 6 Next

Figure 11 : exemple de DataTable utilisant des styles Bootstrap

## j. Visual Studio Code

J'ai dû utiliser J2EE lorsque j'ai dû modifier directement la XCatDB. Néanmoins, mon projet n'étant pas directement intégré dans la XCatDB, j'ai pu utiliser l'environnement de développement de mon choix pour travailler sur les autres parties de l'application. Mon dévolu s'est porté sur Visual Studio Code, un environnement de développement que j'utilise régulièrement en semestre d'étude. Il s'agit d'un éditeur de texte personnalisables doté d'une grande communauté de développeurs. Ces derniers conçoivent régulièrement toutes sortes de fonctionnalités pour faciliter le travail des développeurs. Cet éditeur de texte possède notamment des fonctionnalités permettant de détecter les erreurs de syntaxe dans la plupart des langages de programmation. Il possède également des fonctionnalités permettant de formater le code source selon certaines normes ou encore de déboguer directement depuis l'environnement de développement.

Je me suis servi de cet éditeur surtout pour programmer dans les langages Python, Javascript et Shell.



Figure 12 : Logo de VS Code

## k. SQLite

Pour référencer les métadonnées des observations indexées, j'ai été amené à utiliser une base de données relationnelle. Mon tuteur m'a donc proposé d'utiliser le moteur de base de données SQLite. Ce moteur, écrit en langage C, est léger mais ne propose pas toutes les fonctionnalités de moteurs classiques comme MySQL ou PostgreSQL.



Figure 13 : Logo de SQLite

L'avantage principal de ce moteur est de ne pas forcer l'utilisateur à travailler avec un serveur hébergeant la base de données. Les données sont en fait enregistrées dans un fichier binaire SQLite qui peut être stocké dans l'arborescence de fichiers du projet. Les bases de données que j'ai conçues au moyen de SQLite ont été réalisées par l'intermédiaire d'une bibliothèque Python intégrant les fonctionnalités de SQLite. J'ai ainsi pu enregistrer et questionner les données grâce au langage de requêtes SQL. L'utilisation que j'ai eu de cette bibliothèque est réellement sommaire et je n'ai utilisé qu'une seule table par base de données.

## Annexe 2 : Procédure d'installation

### Step 1: Logging

- Start by logging you on the remote machine (serendib in my case):

```
ssh serendib
```

- Create an heasoft & sas directory bash

```
mkdir heasoft
```

```
mkdir sas
```

### Step 2: Install HEASoft & the SAS

- Start by downloading [HEASoft Source code](#)
- Then install HEASoft (by following this guide : [heasoft-installation](#))
- Download the [SAS](#)
- Make sure to initialize the SAS\_PERL environmenet variable :

```
export SAS_PERL=`which perl`
```

- Install the SAS (by following this guide : [sas-installation](#))
- Set aliases in your ~/.bashrc file :

```
export HEADAS="$HOME/heasoft/heasoft-6.30.1/x86_64-pc-linux-gnu-libc2.31"
alias heainit=". $HEADAS/headas-init.sh"
alias sasinit="source $HOME/sas/xmmsas_20211130_0941/setsas.sh"
```

- restart your shell or source your ~/.bashrc :

```
source ~/.bashrc
```

- Try the 2 aliases (heainit & sasinit) & if you get no error, you're done.

### Step 3 : Install Conda

- Download the Miniconda installation script:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

- Execute the script in one or other way:

```
chmod +x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3-latest-Linux-x86_64.sh -p $HOME/miniconda3
```

OR

```
bash Miniconda3-latest-Linux-x86_64.sh -p $HOME/miniconda3
```

- Initialize Conda:

```
source $HOME/miniconda3/bin/activate
conda init
```

- Try to run the conda -V command. If it sends back the version, conda has been correctly installed.

## Step 4 : Install & initialize the Web Service

Install the project

- Clone the project using SSH (or not):

```
git clone git@gitlab.cds.unistra.fr:lmichel/xcatdb-online-processing.git
```

OR

```
git clone https://gitlab.cds.unistra.fr:lmichel/xcatdb-online-processing.git
```

- Go to the project directory.

```
cd xcatdb-online-processing
```

Initialize the conda environment

- Create the environment using conda :

```
conda create -n <name>
```

- Activate the environment :

```
source activate <name>
```

❖ This action should be executed again each time you log to the machine

- Install pip using conda :

```
conda install pip
```

- Install the project dependencies:

```
$HOME/miniconda3/envs/<name>/bin/pip install -r requirements.txt
```

- Initialize PYTHONPATH :

```
export PYTHONPATH=$PWD
```

❖ This action should be executed again each time you log to the machine

## Step 5 : Generate MOCSet

To work, the project must use some data & some special files called MOCSet. These MOCSets are light database designed to be queried. They present a lack of informations, so the user must make sure to generate SQLite database also.

The data necessary to generate these files are on the hosting machine, usually in the /rawdata/<XMM-release>/ready\_to\_load/ file.

To generate the MOCSet, you must be in the xcatdb-online-processing directory and follow these instructions:

- create data directory:

```
mkdir data
```

- create necessary directories inside data directory :

```
mkdir data/m1
mkdir data/m2
mkdir data/pn
mkdir data/subdata_m1
mkdir data/subdata_m2
mkdir data/subdata_pn
```

- Initialize the MOCSET environment variable
- Then generate MOCSet M1 :

```
tmux new -s mocset_m1

time python3 moc_generation/launcher/main.py moc-generate
/rawdata/4XMMdr12/ready_to_load/DetMsk -o data/subdata_m1/ -t
data/m1/db_m1.sqlite3 -c data/m1/db_m1.csv -m data/m1/mocset_m1.bin -O 50 -d -
g M1 -w 2000
Press ctrl + B + D to exit
```

- Generate MOCSet M2 :

```
tmux new -s mocset_m2

time python3 moc_generation/launcher/main.py moc-generate
/rawdata/4XMMdr12/ready_to_load/DetMsk -o data/subdata_m2/ -t
data/m2/db_m2.sqlite3 -c data/m2/db_m2.csv -m data/m2/mocset_m2.bin -O 50 -d -
g M2 -w 2000
Press ctrl + B + D to exit
```

- Generate MOCSet PN :

```
tmux new -s mocset_pn

time python3 moc_generation/launcher/main.py moc-generate
/rawdata/4XMMdr12/ready_to_load/DetMsk -o data/subdata_pn/ -t
data/pn/db_pn.sqlite3 -c data/pn/db_pn.csv -m data/pn/mocset_pn.bin -O 50 -d -
g PN -w 2000
Press ctrl + B + D to exit
```

- Once you have done that, you should wait for approximatively 45 min ~ 1 hour

- ❖ You can verify that it is finished by running the following commands :

```
tmux attach -t mocset_m1
```

```
tmux attach -t mocset_m2
```

```
tmux attach -t mocset_pn
```

- When the mocset were created without error, you can kill all the sessions with the following commands:

```
tmux kill-server
```

## Step 6 : Launch the service

- Modify the start\_project.sh file if necessary:

```
export HEADAS=""  
export SAS_DIR=""  
  
export PYTHONPATH=`find $HOME -name "xcatdb-online-processing" | head -1`  
source activate flask  
# Setup the environment variables :  
export PROJECT=`find $HOME -name "xcatdb-online-processing" | head -1`  
export MOCSET=`find $HOME -name "mocset" | head -1`  
  
alias start-app="python3 $PROJECT/photon_extractor/evt_ext_service/app.py"
```

- HEADAS should point on the HEASOFT soft (something like /path/to/the/file/x86\_64-pc-linux-gnu-libc2.31)
  - SAS\_DIR should point on the SAS Directory (something like /path/to/the/dir/xmmsas\_20211130\_0941/)
  - PYTHONPATH should point on the place you installed xcatdb-online-processing repository
  - source activate flask initialize the flask conda environment you have setup
  - PROJECT should be te same as PYTHONPATH
  - MOCSET is the binary file location of mocset program
  - start-app is an alias to start the project, you can change it to something else if you prefer
- Start a session with a terminal multiplexer (tmux by example) :

```
tmux new -s flask_server
```

- Initialize the project:

```
source start_project.sh
```

- Start the project with the start-app alias command (or anything else if you changed the alias command) :

```
start-app
```

- Type **ctrl + b + d** to exit the session
- Your service is now perfectly working remotely. You can exit ssh and come to it later, it should work except if the remote server has been restarted.
- You can access to the application commandline at anytime by using the following command :

```
tmux attach -t flask_server
```

## Annexe 4 : Ecriture de scripts scientifiques dans AMORA

### Preamble

AMORA is a software designed to run tasks without having to install any software on his own machine. The only goal of AMORA is to ease the process of repeating a same script on different observations. Therefore, AMORA do not provide lots of script and one needs to create some in order to have a complete tool.

This part will give you the keys to write your own script that could run on AMORA.

### Environment variables

AMORA provides some environment variable to the scripts that you can use in your scripts :

#### Variables specific to the AMORA environment

- **HEADAS** : Point on the HEASOFT initialization script
- **SAS\_DIR** : Point on the SAS directory
- **MOCSET** : Point on the MOCSET binaries used in the project ##

#### Variables specific to AMORA

- **MOCSET\_LOCATION** : Point on the directory storing the mocsets.
- **DB\_LOCATION** : Point on the directory storing the databases linked with the mocsets.
- **GLOBAL\_INPUT** : Point to the test directory `input/` that you can use to test some data.
- **SCRIPTS\_DIR** : Point to the directory containing the scripts. Useful if you want to run a script into an other.
- **PPS\_PRODUCT\_DIR** : Point to the directory containing the data from XMM.
- **UTIL\_PATH** : Point to the directory containing the shell util scripts

#### Variables specific to the session

- **SESSION\_UTILS** : Point on a directory specific to the sessions containing already computed files (by previous uses). By example, one can put arf and rmf files in this directory.

## Shell util scripts

Scripts need to use data from AMORA to compute data according to the selected observation and/or shape. For this purpose, one created a bunch of scripts used to compute specific & re-usable functions into other scripts. These re-usable scripts are called the **shell util scripts**.

These scripts are all located in the directory shell\_utils/ at the root of the project.

You can consult these scripts whenever you want.

### ❖ Modification

You can modify these scripts but at your own risks. Some of these scripts have a specific purpose and make sure to understand well the script before modifying it.

Here you have complementary informations on the already existing shell utils scripts :

### background\_session.py

The only goal of this script is to get the background of the region stored in the session. Of course, if the region does not contain a region, it will return anything.

Way to use

Check if background exist

If you use this util script, you should verify that a background is present. To check if there is a background in the region, you can proceed like this :

```
SHAPE_BACKGROUND=$(python3 $UTIL_PATH/background_session.py)

if [[ $SHAPE_BACKGROUND ]] ; then
    echo "Background exists: $SHAPE_BACKGROUND"
fi
```

Check if the region is a circle or a polygon

Another difficulty with this script is to verify if the script is a circle or a polygon.

To overpass this difficulty, you can see how a polygon and a circle is encoded :

Circle encoding

*Encoding pattern*

```
ra dec radius
```

*Example*

```
266.39876678986116 -28.98751591821719 0.00738863640229788
```

## Polygon encoding

*Encoding pattern*

```
ra_1,dec_1 ra_2,dec_2 ra_3,dec_3 ..... ra_n,dec_n
```

*Example*

```
266.40925260134463,-28.99319972268723 266.39329348428254,-28.998382546139965  
266.3861135440521,-28.992897652320735 266.38189345846,-29.001072156704613  
266.3983080850932,-29.004764115827708 266.41153212515644,-28.998982451990887
```

*Check*

To check whether the shape is a circle or a polygon, the user can use a condition like the following :

```
if [[ $SHAPE_BACKGROUND =~ , ]] ; then  
    echo "Background is a polygon"  
else  
    echo "Background is a circle"  
fi
```

## Parse shape for SAS

To parse the shape, you have different options. I will present a way to do it for a polygon and for a circle.

Circle parsing

```
# We get the coordinate form the shape array  
RA=$(echo $SHAPE_BACKGROUND | cut -d' ' -f 1)  
DEC=$(echo $SHAPE_BACKGROUND | cut -d' ' -f 2)  
RADIUS=$(echo $SHAPE_BACKGROUND | cut -d' ' -f 3)  
  
# We basically convert the radius in arcsec and then in physical coordinates  
RADIUS=$(bc -l <<< "$RADIUS*3600*20")  
  
output=$(ecoordconv imageset=$image x=$RA y=$DEC coordtype=eqpos)  
  
grep_var=$(echo $output | grep -o -e "X: Y: [0-9\.\ ]*")  
X_i=$(echo $grep_var | cut -d' ' -f 3)  
Y_i=$(echo $grep_var | cut -d' ' -f 4)  
background="circle($X_i,$Y_i,$RADIUS,X,Y)"
```

Polygon parsing

```
function join_by {  
    local d=${1-} f=${2-}  
    if shift 2; then  
        printf %s "$f" "${@/#/$d}"  
    fi  
}
```

```

for node in $SHAPE_BACKGROUND ; do
    RA=$(echo $node | cut -d, -f 1)
    DEC=$(echo $node | cut -d, -f 2)
    output=$(ecoordconv imageset=$image x=$RA y=$DEC coordtype=eqpos)

    grep_var=$(echo $output | grep -o -e "X: [0-9.\. ]*" )
    X_i=$(echo $grep_var | cut -d' ' -f 3)
    Y_i=$(echo $grep_var | cut -d' ' -f 4)

    converted_coords+=($X_i)
    converted_coords+=($Y_i)
done

STRING_ARRAY=$(join_by , "${converted_coords[@]}")
background="polygon($STRING_ARRAY,X,Y)"

```

## shape\_session.py

The only goal of this script is to get the source region stored in the session.

Way to use

Check if the region is a circle or a polygon

A difficulty with this script is to verify if the script is a circle or a polygon.

To overpass this difficulty, you can see how a polygon and a circle is encoded :

Circle encoding

*Encoding pattern*

```
ra dec radius
```

*Example*

```
266.39876678986116 -28.98751591821719 0.00738863640229788
```

Polygon encoding

*Encoding pattern*

```
ra_1,dec_1 ra_2,dec_2 ra_3,dec_3 ..... ra_n,dec_n
```

*Example*

```
266.40925260134463, -28.99319972268723 266.39329348428254, -28.998382546139965
266.3861135440521, -28.992897652320735 266.38189345846, -29.001072156704613
266.3983080850932, -29.004764115827708 266.41153212515644, -28.998982451990887
```

### *Check*

To check whether the shape is a circle or a polygon, the user can use a condition like the following :

```
if [[ $SHAPE_ARRAY =~ , ]] ; then
    echo "Background is a polygon"
else
    echo "Background is a circle"
fi
```

### Parse shape for SAS

To parse the shape, you have different options. I will present a way to do it for a polygon and for a circle.

#### Circle parsing

```
# We get the coordinate form the shape array
RA=$(echo $SHAPE_ARRAY | cut -d' ' -f 1)
DEC=$(echo $SHAPE_ARRAY | cut -d' ' -f 2)
RADIUS=$(echo $SHAPE_ARRAY | cut -d' ' -f 3)

# We bascially convert the radius in arcsec and then in physical coordinates
RADIUS=$(bc -l <<< "$RADIUS*3600*20")

output=$(ecoordconv imageset=$image x=$RA y=$DEC coordtype=eqpos)

grep_var=$(echo $output | grep -o -e "X: Y: [0-9\.\ ]*")
X_i=$(echo $grep_var | cut -d' ' -f 3)
Y_i=$(echo $grep_var | cut -d' ' -f 4)
shape="circle($X_i,$Y_i,$RADIUS,X,Y)"
```

#### Polygon parsing

```
function join_by {
    local d=${1-} f=${2-}
    if shift 2; then
        printf %s "$f" "${@/#/$d}"
    fi
}

for node in $SHAPE_ARRAY ; do
    RA=$(echo $node | cut -d, -f 1)
    DEC=$(echo $node | cut -d, -f 2)
    output=$(ecoordconv imageset=$image x=$RA y=$DEC coordtype=eqpos)

    grep_var=$(echo $output | grep -o -e "X: Y: [0-9\.\ ]*")
    X_i=$(echo $grep_var | cut -d' ' -f 3)
    Y_i=$(echo $grep_var | cut -d' ' -f 4)
```

```

converted_coords+=($X_i)
converted_coords+=($Y_i)
done

STRING_ARRAY=$(join_by , "${converted_coords[@]}")
shape="polygon($STRING_ARRAY,X,Y)"

```

## list\_event.py

The goal of this script is to let the user access to the event list available in XMM data according to the observations he selected in the query.json file.

Way to use

The script will just display a list of event list into the standard output of the process.

So you can juste use it like that :

```

for event_list in $(python3 $UTIL_PATH/list_event.py) ; do
    echo "Make operations on event list $event_list"
done

```

or you can define a variable :

```
EVENT_LIST_LIST=$(python3 $UTIL_PATH/list_event.py)
```

## list\_obs.py

The goal of this script is to let the user access to the observations available in XMM data according to the observations he selected in the query.json file.

Way to use

The script will just display a list of observations into the standard output of the process.

So you can juste use it like that :

```

for obs in $(python3 $UTIL_PATH/list_obs.py) ; do
    echo "Make operations on obs $obs"
done

```

or you can define a variable:

```
OBS_LIST=$(python3 $UTIL_PATH/list_obs.py)
```

## compression.py

Compression shell util script was created to pack data selected by the user into a single compressed archive.

Way to use it

Options

This shell util script use some options :

- **-d, --dir DIR1 [DIR2 ...]** : Directory(ies) that should be scanned. . by default.
- **-r, --regexp REGEXP** : Regular expression that should be used to identify files that should be compressed.
- **-f, --files FILE1 [FILE2 ...]** : Filenames that should be compressed. It can be combined with regexp to add other files.
- **-e, --extensions EXT1 [EXT2 ...]** : Extensions of the files that should be compressed.
- **-o, --output OUTPUT** : Path to the tar archive that will be generated. **./result.tar** by default.

By default, the directory will take all the files in the current directory and compress them in the **result.tar** archive.

Example

The script can be used like that :

```
observation="P0674600701"
python3 $UTIL_PATH/compression.py -r "${observation}.*" -o
"${observation}.tar.gz"
```

Here it will take all the files starting with **P0674600701** and put them in **P0674600701.tar.gz**.

**obs\_countdown.py**

The goal of this script is to count the observations processed by the scripts. Then, the number of observations processed can be displayed on the screen.

Functions available

This shell utils script is composed of multiple python functions :

- **get\_files\_count\_path(countdown\_dir: str)** : A function to get the path of the file storing the countdown. > :warning: **Access** > You should never access to this file directly. It could create problem in other countdown functions.
- **write\_processed\_files(files\_processed: list[str], countdown\_dir: str = os.getcwd())** : A function to initialize a countdown in a script.
  - **Use in other shell utils scripts**  
This function is normally used by shell utils scripts that display observations. You should avoid setting the countdown directly from the script.
- **clean\_files\_count(countdown\_dir: str = os.getcwd())** : A function that clears a previous countdown by deleting the file containing this countdown.

- `count_down(countdown_dir: str = os.getcwd())`: A function to update count down. Typically, this function should be used when an observation is processed.
- `get_total_files(countdown_dir: str = os.getcwd())`: A function to get the total number of files managed by the script. It should be called after initialization.
- `get_countdown(countdown_dir: str = os.getcwd())`: A function for getting the number of observations already processed.

Way to use

This utils script can be used either in a python script or a bash script.

Bash usage

To use a function from this shell utils script, you just need to start the python script and give the targeted function with parameters as positional arguments of the command.

*Example*

```
python3 $UTIL_PATH/obs_countdown_manager.py count_down
```

Python usage

You just need to import the shell utils script module and then call the targeted function.

*Example*

```
import shell_utils.obs_countdown_manager as ocm
ocm.write_processed_files(file_path_list)
```

## xspec\_plt.py

The goal of this shell util script is to let user access to various functions that compute scientific products plot.

### ❖ Specific script

This shell util script is very different from the other one because it is written by scientists.

Way to use it

Options

This shell util script use some options:

- `-l, --lomb_scargle`: Enables computation of Lomb-Scargle curves.
- `-t, --power_spectrum`: Enables computation of power spectra.
- `-c, --light_curves`: Enables computation of light curves.

- **-e,--event\_list EVENT\_LIST** : Event list path used to generate some curves. It is required for generating Lomb-Scargle, Light Curves & Power Spectra.
- **-s,--grp\_spectrum GRP\_SPECTRUM** : Grouped spectrum fits file path used to generate spectra with BlackBody & Power Law models.
- **-n,--nh NH** : Set the value of nh to a specific value.  $10^{(22)}$  by default.
- **-f,--nh\_frozen** : Freeze the nh value in spectra generation if set.

Example:

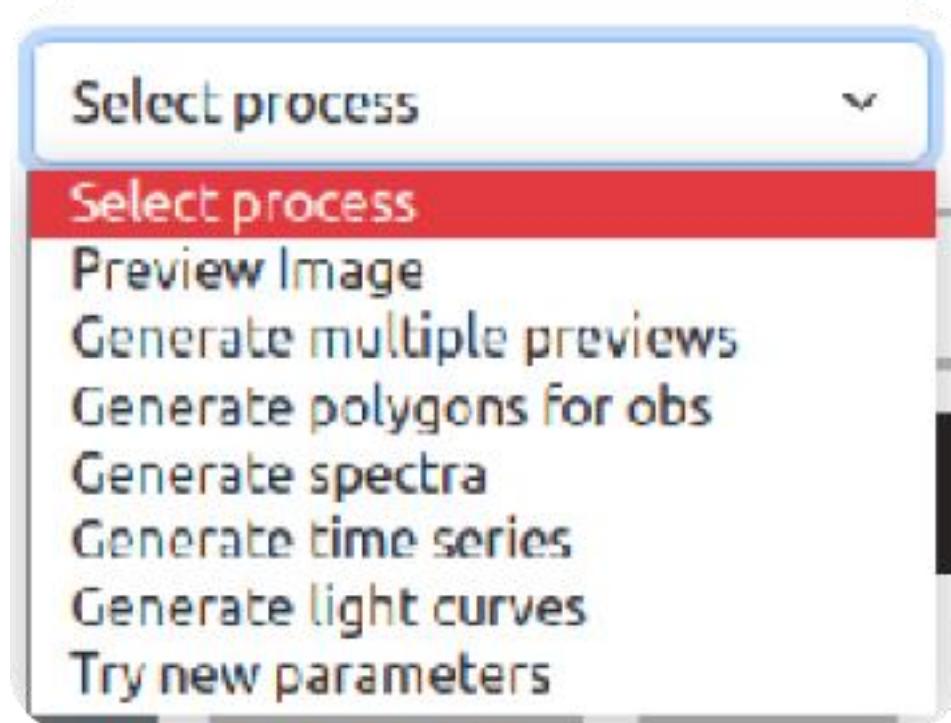
If one wants to generate Lomb-Scargle curves and spectra with modified nh, one needs to run the following command:

```
python3 shell_utils/xspec_plt.py --event_list ${path_to_event_list} --
lomb_scargle --nh 1e+19 --grp_spectrum ${path_to_grp_spectrum}
```

## Annexe 5 : Descriptions des scripts et des paramètres de scripts

Define scripts in the web service

In the service we can define multiple processes :



*Services available example*

To define them, we only need to modify the `scripts.json` file.

## Location of the scripts.json file

The file is located in the folder : xcatdb-online-processing

### Example

For defining our scripts, you can use the following example :

```
{
  "label": "Try new parameters",
  "name": "test_params.py",
  "desc": "Some placeholder script to tests new inputs",
  "params": [
    {
      "id": "range0",
      "cmd_prefix": "",
      "display_name": "Test range pos",
      "type": "number",
      "values": {
        "range": {
          "min": 0,
          "max": 100,
          "step": 0.5
        }
      },
      "default": 50,
      "arg_type": "positional",
      "desc": "Some test range"
    },
    {
      "id": "range1",
      "cmd_prefix": "--range",
      "display_name": "Test range opt",
      "type": "number",
      "values": {
        "range": {
          "min": 0,
          "max": 1e+23,
          "step": 0.5
        }
      },
      "default": 50,
      "arg_type": "optional",
      "desc": "Some test range"
    },
    {
      "id": "dropdown0",
      "cmd_prefix": "",
      "display_name": "Test dropdown pos",
      "type": "dropdown"
    }
  ]
}
```

```

    "type": "string",
    "values": {
        "list": [
            "choice 1",
            "choice 2",
            "choice 3"
        ]
    },
    "default": "take a choice",
    "arg_type": "positional",
    "desc": "Some test dropdown"
},
{
    "id": "dropdown1",
    "cmd_prefix": "--list",
    "display_name": "Test dropdown opt",
    "type": "string",
    "values": {
        "list": [
            "choice 1",
            "choice 2",
            "choice 3"
        ]
    },
    "default": "take a choice",
    "arg_type": "optional",
    "desc": "Some test dropdown"
},
{
    "id": "text0",
    "cmd_prefix": "",
    "display_name": "Test Text pos",
    "type": "string",
    "default": "nothing",
    "arg_type": "positional",
    "desc": "Some text test"
},
{
    "id": "text1",
    "cmd_prefix": "--text",
    "display_name": "Test Text opt",
    "type": "string",
    "default": "nothing",
    "arg_type": "optional",
    "desc": "Some text test"
},
{
    "id": "number0",

```

```

        "cmd_prefix":"",
        "display_name":"Test Number pos",
        "type":"number",
        "default":7,
        "arg_type":"positional",
        "desc":"Some number test"
    },
{
    "id":"number1",
    "cmd_prefix": "--number",
    "display_name":"Test Number opt",
    "type":"number",
    "default":2,
    "arg_type":"optional",
    "desc":"Some number test"
},
{
    "id":"bool1",
    "cmd_prefix": "--bool",
    "display_name":"Test Bool opt",
    "type":"bool",
    "default":false,
    "arg_type":"optional",
    "desc":"Some boolean test"
}
]
}

```

## Description

### Script

**label** : Label to complete the script name

**name** : Name of the script

**desc** : Description of the script

**params** : List of parameters mandatory we want to start the process with (we can add any number of parameters)

## Script Parameters

**id** : The name of the parameter (please use names with underscores and no special character like é,à,@,&,è,ø,£,etc.).

**cmd\_prefix** : Prefix of the command. Necessary if it should be considered as an optional parameter in the command line.

**display\_name** : Name of the option as it will be displayed on the web page (You can use the characters you want).

**type** : The type of input. It can be number, bool or string.

**default** : Default value taken by the parameter.

**arg\_type** : arg\_type can be either positional or optional.

**desc** : description of the parameter.

**values** : Values is a special parameter used to display more complex options.

- **list** : Use to display the elements in a list to choose only one option.
- **range** : Use to display numerical elements with minimum and a maximum value. You can also define a step.

## Annexe 6 : Utilisation de MOCSet Generation

To generate a MOCSet, use the moc-generate sub-command from the moc\_generation sub-project. You must have all your observations in only one directory. Then, you must specify this directory in the command or the command will not run.

### Options

To generate your MOCSet, you can specify a various number of options. The main options, that might be interesting for you, are :

- **-o, --output\_dir** : The directory where to put the resulting MOC.
- **-d, --clean** : When active, delete subsidiary files after use.
- **-t [filename.db], --database [filename.db]** : When active, save the database in a specific file.
- **-c [filename.csv], --csv [filename.csv]** : When active, export useful data to generate moc-set in a csv file.
- **-m [filename.bin], --mocset [filename.bin]** : When active, generate a mocset based on the source\_dir.
- **-O [nb\_processes], --optimized [nb\_processes]** : When active, Run without storing moc in the RAM, it should be used with large sets of data. You can also choose a number of multi process to start.

### Filtering options

You can filter your files if you want to generate a specific set of data with other options :

- **-b observation\_number, --observation\_number observation\_number** : The observation number of the set of files we want to observe (1 letter + 10 numbers).
- **-g camera, --camera camera** : The camera used for the set of files we want to observe (2 characters).
- **-n s\_number, --s\_number s\_number** : The s number for the set of files we want to observe (4 characters).
- **-s serial\_code, --serial\_code serial\_code** : The serial code for the set of files we want to observe (6 characters).
- **-w energy\_band, --energy\_band energy\_band** : The energy band used for the set of files we want to observe (4 numbers).

## Some example

To generate a MOCSet with this options : - from the directory /rawdata/4XMMdr12/DetMsk - put all the MOC files in data/subdata/ - then delete the MOC files - filtered with the camera EPN - filtered with energy band 2 - use 50 processes to generate MOC files & run in optimized mode - generate an SQLite file db\_pn.sqlite3 in the directory data/pn/ - generate a CSV file db\_pn.csv in the directory data/pn/ - generate a MOCSet mocset\_pn.bin in the directory data/pn/

You must run the following command:

```
python3 moc_generation/launcher/main.py moc-generate /rawdata/4XMMdr12/DetMsk  
-o data/subdata/ -t data/pn/db_pn.sqlite3 -c data/pn/db_pn.csv -m  
data/pn/mocset_pn.bin -O 50 -d -g PN -w 2000
```

## Annexe 7 : Tableau des endpoints de la REST-API

Endpoints	HTTP request type	Goal	Return	Parameters
http://<service>/job/	POST	Create session with the given shape if it was specified	Name of the session created	json string containing the shape
http://<service>/job/<session-name>/shape/	PUT	Register shape in session	json file path containing the shape	json string containing the shape
http://<service>/job/<session-name>/get-shape/	GET	Get shape registered in session	json string containing the shape	
http://<service>/job/<session-name>/<camera>/obs	GET	Get all Obs matching the shape	A json result with all the obs matching the shape	
http://<service>/job/<session-name>/select-obs	POST	Select set of Obs	The location of the file	A json containing the

			storing the selection	observations selected *
http://<service>/job/<session-name>/<camera>/select-obs	GET	Get selected set of Obs	A json reponse as a list of all the observations	
http://<service>/job/<session-name>/param	PUT	Put some params in the params from a session	A json object representing all the parameters	A json object
http://<service>/job/<session-name>/param/<key>	GET	Get a particular parameter from the param file	the value matching the key given in url	
http://<service>/job/<session-name>/select-process/<process>	POST	Select process and put it in the session	A message showing which process has been selected	
http://<service>/job/<session-name>/run-process	POST	Run the process with the given parameters and put the result in the session	A message telling which thread is running	A json object**
http://<service>/job/<session>/status/	GET	Get a json list of all the processes currently running in the session	A json object representing the first element of the json list	
http://<service>/job/<session>/status/<process>	GET	Get the status of a particular process	a json object representing the status of the selected process	

http://<service>/job/<session>/status/<process>	DELETE	Delete result of a process	The status json object of the finished process	
http://<service>/job/<session>/process-result/<process>	GET	Get the result of a particular process if the process is finished. Otherwise, it sends a 503 error.	A log of the process	
http://<service>/job/<session>/process-previews/<process>	GET	Get the previews generated by a finished process	A json list of objects***	
http://<service>/job/<session>/process-data/<process>	GET	Get the data generated by a finished process	A json list of objects***	
http://<service>/job/<session>/interrupt-process/	DELETE	Interrupt a running process. Send a 403 error if no thread is running	Message indicating which thread has been interrupted	
http://<service>/job/<session>/get-available-processes/	GET	Get a json list of all the scripts currently available in the session	A json list of objects****	
http://<service>/job/<session-name>/delete	DELETE	Delete <session-name>	Success or failure message	

http://<service>/job/sessions	GET	Display all active sessions	JSON list of sessions names	
http://<service>/mocset-query	GET	Query the MOCSet stored with a cone described in argument.	JSON containing the data requested	ra, dec and radius (written with the ?/& HTTP syntax)
http://<service>/script-help/<script>	GET	Get the help page of a specific script	An HTML page containing the documentation of the script	
http://<service>/job/<session>/process-log/<process>	GET	Get the log of a running process	A text response containing the log or an error if no script is running on the session	
http://<service>/job/<session>/process-previews/<process>	GET	Get a list of the previews generated	A JSON listing each preview by a json object	
http://<service>/job/<session>/process-data/<process>	GET	Get a list of the data generated	A JSON listing each data by a json object	
http://<service>/<session>/<process>/preview/<picture>	GET	Get a base64 presentation of a picture generated by a process	A base64 PNG descriptor	
http://<service>/<session>/<process>/data/<file>	GET	Download a data generated	A download link for a data generated by the process	

http://<service>/<session>/<process>/logs/download/	GET	Download the current log of the targeted process	A download link for the log file of the process	
http://<service>/<process>/download	GET	Download a script available on AMORA	A download link for a script	
http://<service>/obs/<file>	GET	Get teh metadata of an observation	A JSON corresponding to the metadata of the the observation	

## Annexe 8 : Script de minification d'AliX et XCatDB

```
#!/bin/sh

PROJECTS_ROOT=~/git # Root of AliX and XCatDB repository

LMPACK=$PROJECTS_ROOT/alix/packager/lmpack
WEBPACK=$PROJECTS_ROOT/alix/packager/webpack

# Start by minifying Javascript

cd $LMPACK/minifier

ln -s ~/git/4XMM-WEB/minifier/yuicompressor-2.4.8.jar yuicompressor-2.4.8.jar

for script in $(find $LMPACK/minifier -maxdepth 1 -name "*.bash"); do
    bash $script
done

rm yuicompressor-2.4.8.jar

# Then minify the CSS with webpack

cd $WEBPACK/node_modules

npm start

# Copy all the generated files in 4XMM-WEB

cd $PROJECTS_ROOT

# Start by javascript files
```

```

if [ ! -d $PROJECTS_ROOT/4XMM-WEB/WebContent/xcatweb/packedjs/ ]; then
    mkdir $PROJECTS_ROOT/4XMM-WEB/WebContent/xcatweb/packedjs/
fi

cp $PROJECTS_ROOT/alix/packager/lmpack/alix_packed/*
$PROJECTS_ROOT/4XMM-WEB/WebContent/xcatweb/packedjs/

# Then copy CSS files

if [ ! -d $PROJECTS_ROOT/4XMM-WEB/WebContent/xcatweb/packedcss ]; then
    mkdir $PROJECTS_ROOT/4XMM-WEB/WebContent/xcatweb/packedcss
fi
if [ ! -d $PROJECTS_ROOT/4XMM-WEB/WebContent/xcatweb/packedcss/alix ]; then
    mkdir $PROJECTS_ROOT/4XMM-WEB/WebContent/xcatweb/packedcss/alix
fi

cp -r $PROJECTS_ROOT/alix/packager/webpack/icons $PROJECTS_ROOT/4XMM-
WEB/WebContent/xcatweb/packedcss/alix/
cp -r $PROJECTS_ROOT/alix/packager/webpack/images $PROJECTS_ROOT/4XMM-
WEB/WebContent/xcatweb/packedcss/alix/
cp -r $PROJECTS_ROOT/alix/packager/webpack/fonts $PROJECTS_ROOT/4XMM-
WEB/WebContent/xcatweb/packedcss/alix/
cp -r $PROJECTS_ROOT/alix/packager/webpack/public/style_bundle.css
$PROJECTS_ROOT/4XMM-WEB/WebContent/xcatweb/packedcss/alix/
cp $PROJECTS_ROOT/alix/styles/aladinliteX.css $PROJECTS_ROOT/4XMM-
WEB/WebContent/xcatweb/packedcss/alix/

# Copy all the necessary images

cp $PROJECTS_ROOT/alix/images/* $PROJECTS_ROOT/4XMM-
WEB/WebContent/xcatweb/packedcss/alix/images/
cp $PROJECTS_ROOT/alix/images/* $PROJECTS_ROOT/4XMM-
WEB/WebContent/xcatweb/packedcss/images/
# Minify 4XMM-WEB

cd $PROJECTS_ROOT/4XMM-WEB/minifier
for script in $(find . -maxdepth 1 -name "*.bash"); do
    bash $script
done

```

