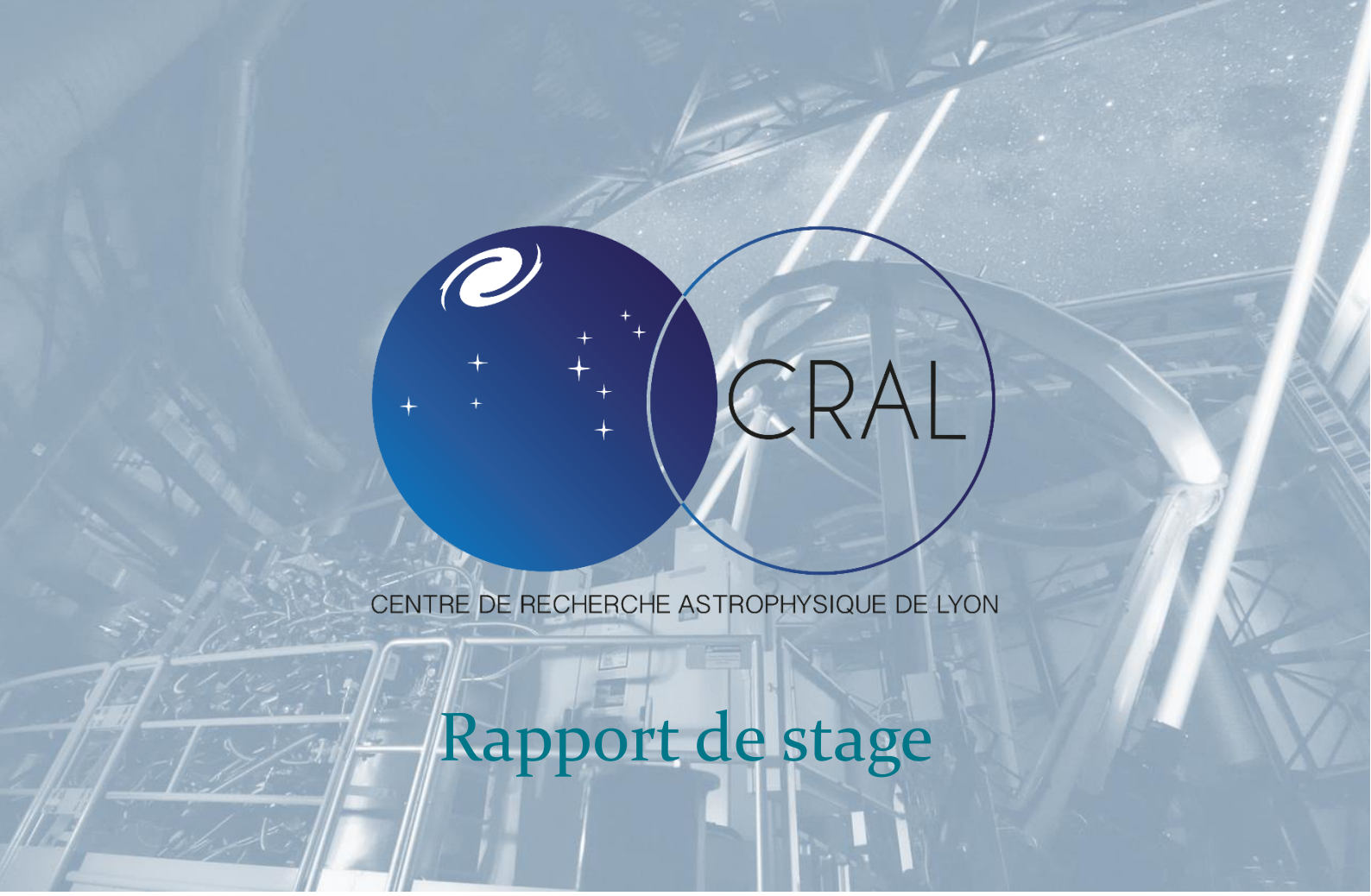


CENTRE DE RECHERCHE ASTROPHYSIQUE DE LYON

Rapport de stage

CODAGE DE LA STRUCTURE DU LOGICIEL DE REDUCTION DE DONNEES DE L'INSTRUMENT ELT/HARMONI

Lucien Burdet
DUT Informatique

08/04/2019 au 30/06/2019

-

Centre de Recherche Astrophysique de Lyon (CRAL)
9 avenue Charles André | 69230 | Saint Genis Laval

-

Maitre de stage : Laure Piqueras
Tuteur enseignant : Véronique Deslandres

-

Université Claude Bernard | Lyon 1 | Institut Universitaire de Technologie
Département informatique
43 Boulevard du 11 Novembre 1918| 69622| Villeurbanne
04-72-69-21-90

Fiche Technique

Nom de l'entreprise : Centre de Recherche Astrophysique de Lyon (CRAL).

Activité de l'entreprise : Recherche astronomique.

Intitulé du sujet : Codage de la structure du logiciel de réduction de données de l'instrument ELT/HARMONI.

Le sujet s'inscrit dans un projet plus large : ELT/HARMONI.

L'objectif principal est de coder la structure du logiciel notamment les entrées/sorties. Ce logiciel sera livré avec l'instrument HARMONI à l'ESO, il a donc des contraintes de développement fortes. En termes de contraintes de temps, il faut finir de développer l'architecture du logiciel avant la fin du stage. Le travail est basé sur des diagrammes d'architecture.

Il y a des informaticiens au CRAL notamment au service calcul scientifique où je suis affecté.

Je travaille seul sur le développement du logiciel mais avec l'aide de ma tutrice et de l'équipe du service calcul scientifique.

J'utilise un ordinateur sur environnement Debian. Le langage de programmation est le C et nous utilisons git afin de versionner le code. J'ai dû apprendre les librairies et logiciels de l'ESO : CPL, HDRL, EsoRex et EsoReflex. De plus, afin de générer de la documentation, j'ai dû me former à la librairie Doxygen.

J'ai eu la chance d'avoir quatre journées de travail consacrées à l'avancement de la rédaction de ce rapport.

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, j'adresse mes remerciements à ma tutrice enseignante, Mme Véronique Deslandres qui m'a suivi durant ce stage.

Je tiens à remercier vivement mon maitre de stage, Mme Laure Piqueras, pour son accueil, le temps passé ensemble et le partage de son expertise au quotidien. Grâce à sa confiance, j'ai pu m'accomplir totalement dans mes missions. Elle fut d'une aide précieuse dans les moments les plus délicats.

Je remercie également toute l'équipe du service calcul scientifique pour leur accueil et leur esprit d'équipe.

Sommaire

Table des matières

Glossaire

CRAL : Centre de Recherche Astrophysique de Lyon

CPL : Common Pipeline Library

CNRS : Centre national de la recherche scientifique

ELT : Extremely Large Telescope

ESO : European Southern Observatory

FITS : Flexible Image Transport System

HARMONI : High Angular Resolution Monolithic Optical and Near-infrated Integral field spectrograph

HDRL : High level Data Reduction Library

MUSE : Multi Unit Spectroscopic Explorer

NIR : Near Infra-red

OCA : Organisation, Classification, and Association

QC : Quality control

SOF : Set of Frame

SUTR : Sample-up-the-ramp

VLT : Very Large Telescope

VIS : Visible

4MOST : 4-metre Multi-Object Spectroscopic Telescope

Table des illustrations

Introduction

Etant passionné par le développement informatique, il était pour moi évident de trouver un stage dans ce domaine. Ainsi, j'ai eu l'immense joie de réaliser ce stage au Centre de Recherche Astrophysique de Lyon, unité mixte de recherche entre l'Université Claude-Bernard Lyon 1, l'Ecole Normale Supérieure de Lyon et le CNRS.

Le CRAL est un laboratoire de recherche fondamentale en astrophysique où sont développés des instruments destinés aux grands observatoires.

Fort d'une équipe d'une soixantaine de personnes, chercheurs et thésards, ingénieurs, techniciens et administratifs, le CRAL développe un fort potentiel de recherche théorique et instrumentale.

Ma mission est de développer la structure du logiciel de réduction de données de l'instrument ELT/HARMONI.

Ce stage va me permettre d'apprendre, me former afin de mieux appréhender mon futur parcours professionnel. Mais également de découvrir un centre de recherche où l'on travaille sur un sujet d'une portée internationale.

Nous verrons dans un premier temps une présentation de l'environnement de travail, puis de la mission et enfin un bilan de l'expérience du stage.

I. Présentation environnement du stage

La pensée maîtresse du CRAL est la suivante : « Seule science déifiée par sa muse Uranie, l'Astronomie est celle de l'humanité, ancrée dans son subconscient. Sa pratique et son développement n'est que l'accompagnement de toute société en marche. ».

1. Présentation générale de l'entreprise

1) Historique

L'astronomie s'est implantée relativement tôt dans la région lyonnaise, puisque c'est en 1684 qu'est fondé un observatoire dans le collège de la Trinité, actuellement lycée Ampère. En 1867, l'Observatoire déménagea dans une aile du palais Saint Pierre, mais se trouvait vite à l'étroit. C'est en 1878 que l'Observatoire de Lyon à Saint-Genis-Laval fut créé. Il était alors dirigé par Charles André. L'Observatoire comporte plusieurs instruments notamment l'Equatorial coudé ou encore un télescope de 1 mètre.

Figure I-2 : Equatorial coudé.

Figure I-1 : Observatoire de Lyon.

Mais avec le temps, à cause des lumières et des fumées dues à l'urbanisation, ces instruments ont de moins en moins été utilisés.

Les activités de recherche de l'Observatoire ont pris un nouvel élan avec la création du Centre de Recherche Astronomique de Lyon en 1995, sous la tutelle conjointe de l'Université Lyon I, l'Ecole Normale supérieure et le CNRS. Depuis 2007, le centre est renommé Centre de Recherche Astrophysique de Lyon.

L'évolution de l'Observatoire est loin d'être finie, car l'établissement s'adapte sans cesse pour répondre aux grandes orientations internationales dictées par les progrès de la science.

2) Activité

Devant la diversité et l'importance des thèmes de recherche en Astronomie, l'Observatoire se fixe des grands axes. Ces axes évoluent en fonction des découvertes, des techniques, des personnalités, et touchent, sinon à tout, à une majorité de sujets de recherche de la communauté internationale des astronomes.

Voici quelques thèmes centraux qui sont au cœur des recherches des astronomes du CRAL :

- **Formation et évolution des premières structures :** observer et comprendre la formation de l'Univers depuis son origine.
- **Les galaxies :** observer et modéliser les galaxies pour comprendre leur formation et leur évolution.
- **Les naines brunes et les planètes extrasolaires :** comprendre la formation des planètes et des étoiles, établir le lien entre étoiles et planètes.
- **Physique fondamentale, physique statistique, cosmologie théorique.**
- **Traitement des données astronomiques, science des données, traitement du signal.**
- **R&D imagerie à très haute dynamique.**

L'Observatoire est aussi connu internationalement pour ses réalisations instrumentales comme MUSE (Multi Unit Spectroscopic Explorer), spectrographe intégral de champ installé à Paranal au Chili sur le Très Grand Télescope (VLT) de l'Observatoire austral européen (ESO). L'ESO est une organisation intergouvernementale pour l'astronomie fondée en 1962 par cinq pays européens, afin de créer un observatoire astronomique de pointe au sol dans l'hémisphère austral à disposition des astronomes. La collaboration d'ingénieurs et de techniciens très qualifiés à ces recherches permet le développement et la réalisation d'instruments novateurs qui repoussent toujours plus loin les limites de l'Univers observable.

A la spectroscopie stellaire ont succédé la photométrie visible et infrarouge, puis l'imagerie infrarouge. Aujourd'hui, l'Observatoire est l'un des spécialistes mondiaux de l'étoile artificielle laser, de l'imagerie à haute résolution et de la spectroscopie 3D.

La quantité d'observations accumulées par les télescopes au sol et dans l'espace est aujourd'hui telle qu'il est nécessaire de recourir aux bases de données pour rendre cette information accessible et intelligible. L'Observatoire ouvre aussi à la communauté internationale ses bases de données de galaxies.

La passionnante recherche projette déjà l'Observatoire dans le futur. Il participe aux projets qui ne verront le jour que dans les 5 ou 10 ans à venir comme les spectrographes ELT/HARMONI ou encore 4MOST (4-metre Multi-Object Spectroscopic Telescope).

3) Nombre de salariés

Le CRAL comprend environ 30 chercheurs, 20 Ingénieurs, Techniciens et Administratifs, et 30 doctorants, post-doctorants et CDDs. Le CRAL héberge dans ses locaux le LabEx LIO, et les étudiants lyonnais M2 du Master Astrophysique Lyon-Montpellier.

2. L'entreprise comme organisation

1) Organigramme

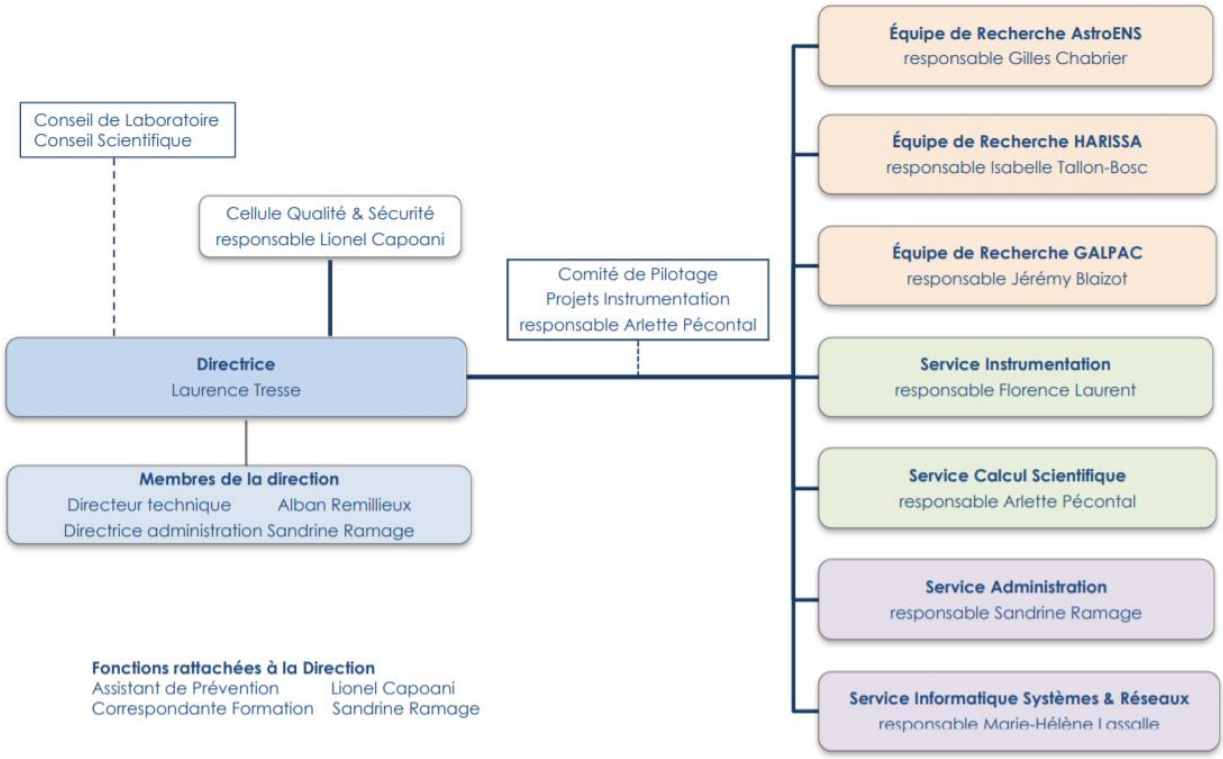


Figure I-3 : Organigramme du CRAL.

2) Ma place dans l'organisation

Je travaille dans le service calcul scientifique dont le responsable est Arlette PÉCONTAL, sur la partie réduction des données.

Le service Calcul Scientifique du CRAL regroupe les personnels techniques dont les compétences permettent de mener à bien les développements décrits ci-dessous, que ce soit en support aux développements instrumentaux, aux services d'observations ou aux activités de recherche.

- **Simulations astrophysiques** : Il s'agit de simulations numériques d'objets ou de systèmes astrophysiques complexes. Ces simulations ont pour but de tester des hypothèses théoriques et des scénarios d'évolution de ces systèmes. La simulation 3D de nombreux processus astrophysiques permet de comprendre l'importance et le rôle de ces processus pour reproduire un phénomène observé. Ces simulations peuvent aussi avoir un pouvoir prédictif et inspirer de nouvelles observations. Ce sont des simulations lourdes, adeptes du calcul intensif.

- **Simulateur d'instrument** : Un simulateur d'instrument est un logiciel basé sur l'optique de Fourrier qui reproduit le parcours de la lumière à travers l'instrument. Il remplace ainsi l'instrument pendant la phase de construction, créant à partir de scènes astrophysiques les images détecteurs telles que les produira l'instrument futur.

- **Réduction des données :** Les logiciels de réduction des données permettent d'extraire les données brutes issues d'instruments astrophysiques, et d'ôter la signature instrumentale de façon à les rendre comparables aux données acquises par d'autres instruments.

- **Analyse des données** : Les logiciels d'analyse permettent d'extraire les données physiques susceptibles de mettre en lumière les processus physiques en œuvre.

- **Support à la phase d'Assemblage, d'Intégration et de Vérification (AIV)** : Cette gamme de logiciel est principalement un support aux opticiens et à l'ingénierie système. Ils permettent en effet de valider des spécifications et/ou de faciliter l'alignement des optiques lors de la phase d'assemblage et d'intégration.

- **Ingénierie système/management de projet** : Le service assure des activités de management de projet et d'ingénierie système logiciel dans différents projets.

Le service gère de plus certains moyens de calcul comme le CCF (Common Computing Facility) un super-calculateur de 1200 cœurs.

3) Equipe de travail

L'équipe HARMONI est composée de plusieurs membres qui travaillent dans différents domaines :

- **Chef de projet :** Alban REMILLIEUX.
- **Scientifiques :** Roland BACON, Johan RICHARD et Nicolas BOUCHÉ (responsable scientifique).
- **Concepteur optique :** Alexandre JEANNEAU.
- **Designer optique :** Florence LAURENT, Magali LOUPIAS.
- **Concepteurs mécaniques :** Jean-Emmanuel MIGNIAU, Didier BOUDON.
- **Ingénieur en mécanique informatique :** Éric DAGUISÉ.
- **Ingénieure en systèmes informatiques :** Arlette PÉCONTAL.
- **Ingénieurs en logiciel scientifique :** Aurélien JARNO, Laure PIQUERAS.

De plus, des ingénieurs en CDD et d'autres stagiaires interviennent sur le projet, pour travailler sur le traitement d'image par exemple.

4) Organisation du travail

Le CRAL est ouvert de 7h à 19h, je dois effectuer 7 heures par jours avec en plus 1 heure de pause.

J'ai à ma disposition la documentation des différentes librairies et les spécifications du logiciel à développer. De plus, un ordinateur ainsi qu'un moniteur m'ont été fournis. J'ai été affecté dans un bureau du service calcul scientifique ; dans ce bureau se trouve également une post-doctorante de l'équipe GALPAC. En cas de questions ou de remarque sur le logiciel à développer, je peux solliciter ma tutrice ou d'autres membres du service calcul scientifique.

Nous faisons régulièrement le point avec ma tutrice afin de développer le code le plus propre possible, répondant aux contraintes de l'ESO. Ces entrevues me permettent aussi de mieux comprendre le projet et ses attentes en termes de développement.

Une fois par mois, toute l'équipe du projet effectue une réunion animée par le chef de projet afin de faire le point sur l'avancement du projet et sur les évènements à venir.

3. Présentation environnement technologique

1) Outils, logiciels, langages

J'utilise un ordinateur sur environnement Debian. En effet, le logiciel à développer requière différente librairies et modules de l'ESO (CPL, HDRL, EsoRex…, voir II.1.4) Les librairies et logiciels de l'ESO) qui sont utilisables sur des systèmes basés sur linux.

De plus, nous utilisons Doxygen qui est un générateur de documentation capable de produire une documentation logicielle à partir du code source du programme. Pour cela, il tient compte de la syntaxe du langage dans lequel est écrit le code source, ainsi que des commentaires qui sont écrits dans un format particulier.

Afin de versionner le code, nous utilisons GIT. Il permet, d'une part, de garder une archive de toutes les modifications et, d'autre part, de vérifier à chaque commit que les tests fonctionnent. En outre, nous devons écrire des tests unitaires pour les différentes fonctions que nous développons et GIT effectue tous les tests afin de vérifier que les nouvelles fonctionnalités n'influent pas sur les fonctions précédemment codées.

Le langage de programmation est le C.

Enfin, nous avons à notre disposition une machine très puissante nommée harmoni1 où l'on se connecte en SSH et où l'on peut exécuter ce qui demande des ressources que mon ordinateur ne peut pas fournir. Pour mieux s'imaginer la machine, voici quelques éléments de sa configuration : 192 Go de RAM, 24 cœurs, 15 To de disque.

2) Minimum de connaissance à avoir

Il faut forcément avoir des connaissances approfondies en C car les librairies et la structure du projet sont basées sur des notions du C assez avancées (pointeur, allocation dynamique, structures…). De plus, des connaissances en python sont nécessaires pour consulter des fichier FITS (voir II.1.2) Fichier FITS). Des connaissances de l'environnement Linux sont essentielles car nous sommes constamment amenés à en avoir besoin pour le développement.

Enfin, l'anglais est un minimum étant donné qu'il faut commenter le code en anglais et que toute la documentation est en anglais.

3) Familiarisation avec les outils

Au début du stage, il m'a fallu quelques jours de lectures et de test des différentes librairies afin de bien comprendre comment elles fonctionnaient.

Ensuite, durant le premier mois, je n'ai pas développé uniquement la structure, mais l'intégralité des deux recipes à l'aide des librairies CPL, HDRL et EsoRex afin de vraiment comprendre l'environnement de travail. Je me suis donc vraiment approprié les outils et j'ai pu me concentrer sur l'essentiel du sujet du stage, à savoir le développement de la structure du logiciel.

II. Présentation de la mission

L'objectif principal du stage est de développer la structure du logiciel de réduction de données de l'instrument ELT/HARMONI :

- Les structures de données.
- Les entrées/sorties de chaque fonction.
- Le code principal qui appelle toutes les fonctions.

Il faut suivre l'architecture prévue tout en l'améliorant (voir Annexes – Architecture prévue). Le logiciel doit suivre une architecture spécifique de l'ESO : les noms de variables, disposition des fichiers, etc... La structure élémentaire du projet avec quelques exemples de fonctions étaient déjà codés avant mon arrivée afin que je puisse vraiment comprendre le mécanisme de cette architecture.

De plus, il faut tester, d'une part avec des tests d'intégration avec des données factices et d'autre part coder des tests unitaires. Puis, s'il reste du temps, coder des fonctions, et une interface graphique à l'aide d'une autre librairie et logiciel.

S'imprégner des librairies et plus généralement du projet dans son ensemble est un enjeu primaire. En effet, il est primordial de comprendre le sujet afin de pouvoir développer le logiciel qui répond aux attentes et aux contraintes de l'ESO.

1. Découverte de l'environnement

1) HARMONI

HARMONI sera le premier instrument du futur Télescope Extrêmement Large (ELT) de l'ESO sur le site de Cerro Armazones au Chili.

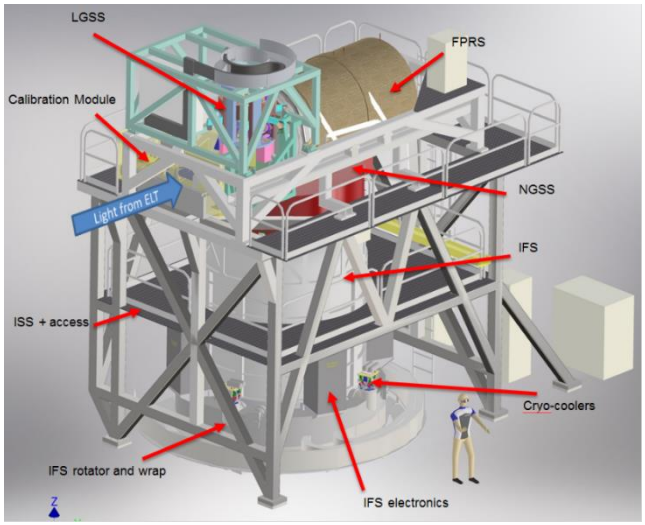


Figure II-1 : Représentation d'une vue d'ensemble d'HARMONI et de sa taille par rapport à une personne.

Plus précisément, c'est un spectrographe intégral de champ : il découpe la zone du ciel observée en un grand nombre de points afin d'en obtenir simultanément les spectres lumineux, qui sont enregistrés par des détecteurs. Il y a 8 détecteurs infrarouge (NIR) et 8 détecteurs visible (VIS). Un détecteur visible fournit une image 2D. Un détecteur infrarouge peut fournir soit une image 2D soit un cube 3D qui correspond à l'enregistrement de chaque lecture des pixels.

HARMONI comprend plus de 44 configurations possibles. Une configuration dépend de son réseau (*grating*), son échelle (*scale*) et du regroupement de pixel (*binning*).

HARMONI s'utilise soit en NIR, soit en VIS.

2) Fichier FITS

HARMONI produit en sortie des fichiers FITS.

Un fichier FITS est composé de plusieurs extensions. En effet, il comprend :

- Une extension primaire, appelé *primary header*. Ce *primary* contient de nombreux mots-clés (*keywords*) pouvant avoir une valeur. Ces mots-clés sont indispensables pour le bon fonctionnement du logiciel. En effet, ces mots-clés sont définis par l'ESO et doivent suivre des normes strictes. Ils permettent de connaitre la configuration du fichier, d'indiquer ce à quoi correspond le fichier… Par exemple le mot-clé permettant de savoir le réseau utilisé est : « INM INS GRATING ». En fonction des valeurs assignées à ce mot-clé, nous pouvons déterminer quel traitement appliquer au fichier.

- Des extensions pouvant contenir tous types de données : image, cube de données, tableau… De plus, ces extensions ont chacune un *header* qui leur est propre. Cela permet d'y mettre des mots clé propre à l'extension : nom de l'extension, résultat de calcul…

Les fichiers de sorties de HARMONI contiennent 8 extensions (pour chaque détecteur) que l'on appelle fichier FITS raw. Dans le cas où un des détecteurs serait défaillant, le fichier de sortie de l'instrument ne contiendrait pas 8 extensions mais le logiciel doit fonctionner quel que soit le nombre d'extensions du fichier.

Dans le logiciel que nous développons, il y a trois traitements de fichier :

- **Visible**, avec chaque extension contenant une image de 4224×4240.

- Infrarouge **INT**, avec chaque extension contenant une image de 4096×4096.

- Infrarouge **SUTR**, avec chaque extension contenant un cube de 64×4096×4096, soit 64 lectures dans le temps et on stocke une image par lecture.

Afin de mieux visualiser un fichier FITS voici un schéma de fichier avec une configuration VIS :

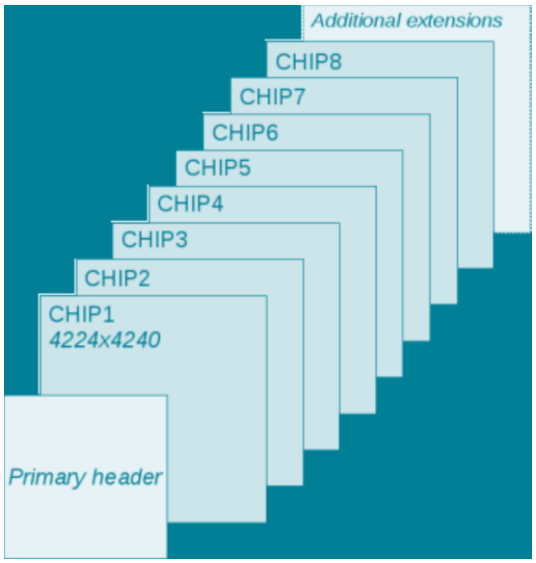


Figure II-2 : Fichier FITS VIS.

3) Le logiciel de réduction de données

Le logiciel de réduction des données applique sur les fichiers FITS des détecteurs des transformations et corrections que l'on découpe en différentes tâches appelées « *recipes* ». Il corrige au maximum les effets introduits par l'environnement terrestre, le télescope et l'instrument et permet d'ôter la signature instrumentale de façon à les rendre comparables aux données acquises par d'autres instruments.

Pour stocker les informations sur chaque pixel nous utilisons le format *pixel map*. Les *pixel map* sont des fichiers FITS contenant plusieurs extensions (6) :

- **Data :** valeurs de flux des pixels.
- **Stat :** valeurs de variance des pixels.
- **Dq :** état des pixels.
- **X :** coordonnées en X des pixels dans l'image d'entrée de l'instrument.
- **Y :** coordonnées en Y des pixels dans l'image d'entrée de l'instrument.
- **Lambda :** valeurs de longueur d'onde des pixels.

Les *full pixel map* sont composées de 8 *pixels map* soit un *pixel map* par détecteur. Il peut donc contenir 6×8 extensions.

En faisant une interpolation, nous obtenons, pour finir, un cube de données. Les trois dimensions d'un cube correspondent aux deux dimensions spatiales (**X** et **Y**) et une dimension spectrale (**Lambda**). Le fichier final contient trois extensions cube qui sont **Data**, **Stat** et **Dq**.

4) Les librairies et logiciels de l'ESO

Une partie non négligeable du stage fut de s'imprégner des différentes librairies. La documentation et le code source de celles-ci sont disponibles sur Internet (voir Sitographie).

CPL

CPL comprend un ensemble de bibliothèques ISO-C qui fournissent une boîte à outils logicielle complète pour développer des tâches astronomiques de réduction de données. Il y a des structures en C pour gérer les images, les fichiers FITS, les *headers* ainsi que toutes les fonctions d'entrées et de sorties liées à ces structures (enregistrer, charger en mémoire…).

HDRL

HDRL est basée sur CPL et reprend de nombreuse fonction de CPL avec certaines améliorations et fonctionnalités supplémentaires. Notamment des méthodes de calcul comme par exemple combiner des images.

EsoRex

L'une des fonctionnalités fournies par CPL est la possibilité de créer des algorithmes de réduction de données qui s'exécutent en tant que plugins (bibliothèques dynamiques). Celles-ci sont appelées *recipes* et constituent l'un des principaux aspects de l'environnement de développement de réduction de données. Comme ces *recipes* sont des bibliothèques dynamiques, il n'est pas possible de les exécuter directement à partir de la ligne de commande. C'est à cela que sert EsoRex.

EsoReflex

EsoReflex permet à l'utilisateur de traiter ses données scientifiques dans les étapes suivantes :

- Associer des fichiers scientifiques aux étalonnages requis.
- Choisir les jeux de données à traiter.
- Exécuter plusieurs *recipes* de pipeline.

C'est un logiciel permettant la représentation visuelle en temps réel d'une cascade de réduction de données, appelée flux de travail (*workflow*), qui peut être facilement compris par la plupart des astronomes. Il est construit en utilisant le moteur de flux de travail Kepler (https://kepler-project.org), qui lui-même utilise le Framework Ptolemy II (http://ptolemy.eecs.berkeley.edu/ptolemyII).

Le *workflow* est représenté visuellement par des boîtes interconnectées (acteurs) qui traitent des données. Cela permet à l'utilisateur de suivre la réduction des données, éventuellement en interaction avec lui. L'utilisateur peut visualiser l'association de données et les fichiers d'entrée et décider quelles données scientifiques il veut traiter.

Il est également possible de visualiser les produits intermédiaires en utilisant des composants fournis par EsoReflex, ou modifiez le *workflow* avec des composants personnalisés.

De plus, EsoReflex utilise des fichiers OCA afin de classifier et associer les fichier FITS. OCA signifie Organisation, Classification et Association.

La syntaxe OCA est semblable à SQL et consiste en un ensemble de :

- Règles de classification (if <conditions> then <classes>).
- Règles d'organisation (select ... from ... where <conditions> group by <règles de groupe>).
- Règles d'association (définissent des actions avec des ensembles d'associations à rechercher).

2. Codage de pixel map

Ma première mission était de coder les fonctions liées aux données de type *full pixel map*/*pixel map* car ce sont des fichiers que l'on va utiliser très souvent dans la suite du projet.

Les fonctions permettent de :

- Charger (load) une *full pixel map*.
- Créer un objet *full pixel map*/*pixel map* et donc allouer toute la mémoire nécessaire.
- Supprimer un objet *full pixel map*/*pixel map* et donc de libérer la mémoire qu'occupait cet objet.
- Enregistrer un objet *full pixel map*/*pixel map* en tant que fichier FITS.
- Dupliquer un objet *full pixel map*.

Premièrement, je devais ouvrir une image stockée dans une extension d'un fichier FITS. L'ouverture d'une image a été assez dure à mettre en œuvre au début, car il fallait utiliser la librairie CPL. Je n'avais pas encore bien compris son fonctionnement mais finalement cela m'a permis d'appréhender les mécanismes de CPL.

La gestion de la mémoire est l'élément le plus complexe, car il faut penser à désallouer la mémoire pour chaque objet que l'on traite, puis ne pas accéder à des espaces de mémoires non autorisés. J'ai trouvé qu'EsoRex permettait d'avoir un retour sur le nombre de pointeur actif (mémoire non désallouée) en activant le mode debug. Cela m'a été très utile.

C'est-à-dire qu'à la fin de l'exécution de la *recipe*, EsoRex affiche dans la console ceci :

```
#----- Memory Diagnostics -----
Maximum number of pointers (approximate): 5493
#----- Memory Currently Allocated -----
Number of active pointers:    0
```

Figure II-3 : Console lors de l'exécution d'une recipe en mode debug.

Dans le cas ci-dessus, nous constatons qu'il n'y a aucun pointeur actif, ce qui est une bonne chose. De plus, nous pouvons savoir approximativement le nombre maximal de pointeurs, 5493 dans cet exemple.

La formation de l'IUT m'a beaucoup aidée car les notions de C et d'architecture d'ordinateur ont joué un rôle déterminant dans la compréhension des librairies et de la gestion de la mémoire.

Par la suite, j'ai codé les tests unitaires des fonctions. Il a fallu penser à tous les cas possibles afin de les tester. Un exemple est de créer un objet *full pixel map*, l'enregistrer, le charger, le comparer, le supprimer… CPL propose de nombreuses fonctions pour gérer les tests unitaires comme la comparaison d'images, tester si un pointeur est NULL… Utiliser les fonctions de CPL pour gérer les tests est un grand avantage car si une erreur survient, CPL peut nous indiquer quel test à échoué et nous l'indiquer dans un fichier texte (*log*). De plus, les fonctions ont été développées spécialement pour répondre aux besoins liés à ce type de logiciel et permettent donc un gain de temps dans le développement.

3. Gérer les configurations

Les fichiers de l'instrument peuvent avoir plusieurs configurations : visible, infrarouge… Etant donné que nous allions souvent être amenés à comparer les configurations des fichiers, j'ai codé toutes les fonctions liées à l'instrument.

La fonction de chargement d'une configuration d'une image doit ouvrir le *header* de l'image, récupérer les valeurs des mots clés, renvoyer une erreur si un des mots clés est manquant, et créer un objet configuration.

Un exemple de mot clé : « INM INS GRATING » : cela correspond au réseau d'un fichier et nous permet de savoir si le fichier est visible ou infrarouge.

Par rapport à l'architecture initialement prévue, nous nous sommes rendu compte qu'il manquait des fonctions notamment une permettant d'ajouter tous les mots clés et valeurs d'un objet configuration dans un objet *header*.

4. Codage d'une première recipe

Une fois les principales fonctions codées, j'ai pu développer la première *recipe* : la *recipe* bias. C'est la *recipe* la plus simple car elle traite seulement le cas visible.

Bias est un objet qui correspond à la première lecture des pixels. Il est composé de 8 extensions contenant chacune une image.

Cette *recipe* permet de créer un fichier master bias à partir de plusieurs fichier bias en entrée. Je n'ai pas codé seulement la structure, mais j'ai codé cette *recipe* entièrement afin de vraiment comprendre le fonctionnement des librairies.

Au commencement, j'ai eu énormément de mal à comprendre comment exécuter une *recipe*, comment lui donner des paramètres… C'est en étudiant le logiciel de l'instrument MUSE et le code source de la librairie que j'ai pu comprendre comment l'utiliser (voir Sitographie).

Voici l'algorithme de cette *recipe* :

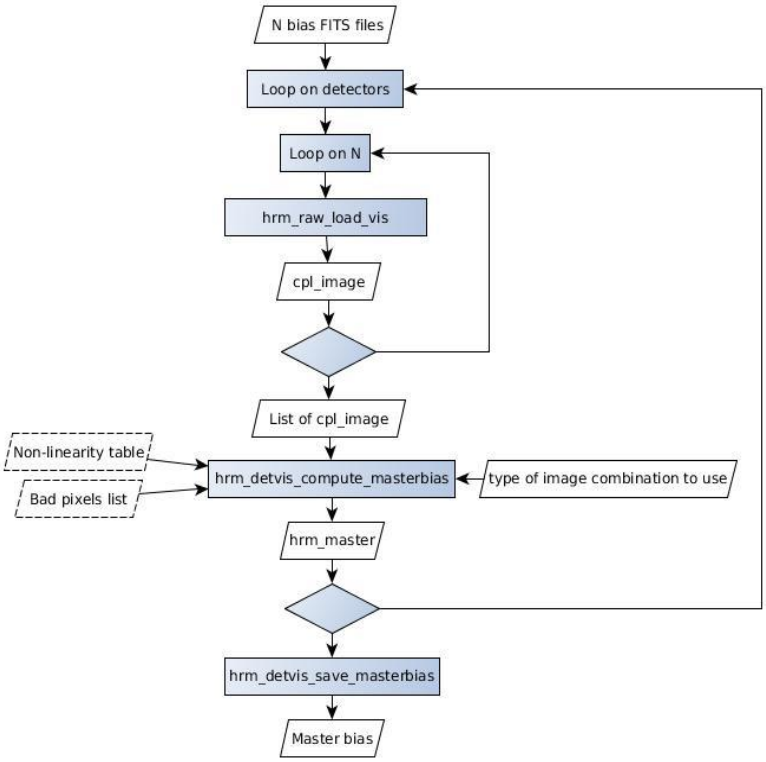


Figure II-4 : Algorithme de la recipe bias.

Nous pouvons constater qu'elle prend en entrée d'autres types de fichiers comme *bad pixel* et fait appel à des fonctions que je n'avais pas encore codées. Il m'a donc fallu coder les fonctions relatives à ces fichiers et les autres fonctions.

Son fonctionnement est simple :

1. Pour chaque détecteur puis pour chaque fichier bias, nous les chargeons en mémoire. Afin de charger une image d'une extension en mémoire, il faut utiliser le type de la librairie CPL *cpl_image*.
2. Ensuite, nous avons une liste de *cpl_image*.
3. Puis, nous les combinons grâce à une fonction de HDRL et créons un objet master.
4. Enfin, nous avons une liste de 8 objets master que nous enregistrons.

J'ai dû ajouter de nombreuses fonctions qui n'étaient pas prévues afin d'avoir le code le plus global et réutilisable possible.

J'ai testé avec des fichiers factices afin de vérifier que tout fonctionnait bien.

Au commencement, nous avions fait volontairement une approximation sur la fonction de chargement d'un fichier raw car elle était complexe à mettre en œuvre. Mais après avoir testé et validé cette *recipe*, j'ai corrigé l'approximation que nous avions fait (voir II.7. Point technique, découpage d'une image).

Voici une capture d'écran de la console lors de l'exécution de cette *recipe* :

```
lucien@harmoni1:~/sof$ esorex harmoni_bias bias_bin1.sof

     ***** ESO Recipe Execution Tool, version 3.13.1  *****

[ INFO  ] harmoni_bias: The mode is 1
[ INFO  ] harmoni_bias: 5 BIAS.
[ INFO  ] harmoni_bias: 1 DET_BADPIX_TABLE.
[ INFO  ] harmoni_bias: The type is VR.
[ INFO  ] harmoni_bias: Loading files CHIP1...
[ INFO  ] harmoni_bias: Loading files CHIP2...
[ INFO  ] harmoni_bias: Loading files CHIP3...
[ INFO  ] harmoni_bias: Loading files CHIP4...
[ INFO  ] harmoni_bias: Loading files CHIP5...
[ INFO  ] harmoni_bias: Loading files CHIP6...
[ INFO  ] harmoni_bias: Loading files CHIP7...
[ INFO  ] harmoni_bias: Loading files CHIP8...
[ INFO  ] harmoni_bias: Writing FITS propertylist product(MASTER_BIAS): master_bias.fits
[ INFO  ] esorex: Calculating product checksums
[ INFO  ] esorex: Created product master_bias.fits (in place)
[ INFO  ] esorex: 1 product created
[ INFO  ] esorex: Recipe operation(s) took          47.4 seconds to complete.
[ INFO  ] esorex: Total size of 5 raw input frames  =  1433.10 MB

[ INFO  ] esorex: => processing rate of    30.24 MB/sec

#----- Memory Diagnostics -----
Maximum number of pointers (approximate): 5493
#----- Memory Currently Allocated -----
Number of active pointers:    0
```

Figure II-5 : Console lors de l'exécution de la recipe bias.

Tous les messages avec le nom de la *recipe* devant (harmoni_bias) ont été généré par moi afin d'informer l'utilisateur sur le déroulement de la *recipe*.

Nous pouvons voir « The mode is 1 », soit « Le mode est 1 ». L'utilisateur peut passer des paramètres à la *recipe*. J'ai créé un paramètre mode qui permet de choisir ce que l'on veut calculer :

- Moyenne : 0.
- Médiane : 1, valeur par défaut.
- Minmax : 2.
- Sigclip : 3.

D'autre paramètres sont modifiables. Afin de changer un paramètre, il suffit de faire comme ci-dessous :

```
lucien@harmoni1:~/sof$ esorex harmoni_dark --mode=2 dark_vis.sof
```

Figure II-6 : Paramètre applicable à l'appel d'une recipe.

De plus, nous pouvons constater que quand on exécute la *recipe*, il faut préciser un fichier SOF, ce fichier contient le chemin où se trouvent les fichiers à ouvrir avec un *tag* afin de savoir à quel type de fichier nous avons à faire. Voici un exemple de fichier SOF :

bias_bin1.sof 354 Bytes

```
1  /home/piqueras/lucien/bias_vis/raw_bias1_bin1.fits  BIAS
2  /home/piqueras/lucien/bias_vis/raw_bias2_bin1.fits  BIAS
3  /home/piqueras/lucien/bias_vis/raw_bias3_bin1.fits  BIAS
4  /home/piqueras/lucien/bias_vis/raw_bias4_bin1.fits  BIAS
5  /home/piqueras/lucien/bias_vis/raw_bias5_bin1.fits  BIAS
6  /home/piqueras/lucien/bias_vis/badpixtab.fits       DET_BADPIX_TABLE
```

Figure II-7 : Fichier SOF pour la recipe bias.

5. Codage de la recipe dark

Puis, j'ai codé la *recipe* dark entièrement également car elle ressemble fortement à la bias, mais elle prend en compte deux cas :

- Visible.
- Infrarouge SUTR.

Elle prend en entrée des fichiers dark, qui sont des fichiers de sortie de l'instrument quand il n'y a pas de lumière en entrée.

Cette *recipe* permet de créer un fichier master dark.

Après de nombreux tests, j'ai pu détecter une erreur dans la façon de procéder. En effet, si les fichiers en entrée n'avaient pas la même configuration, ils n'avaient pas forcément la même taille, et cela pouvait engendrer des erreurs. Il fallait donc une fonction permettant de vérifier cela.

Afin de mieux voir la différence entre visible et infrarouge SUTR voici les deux algorithmes :

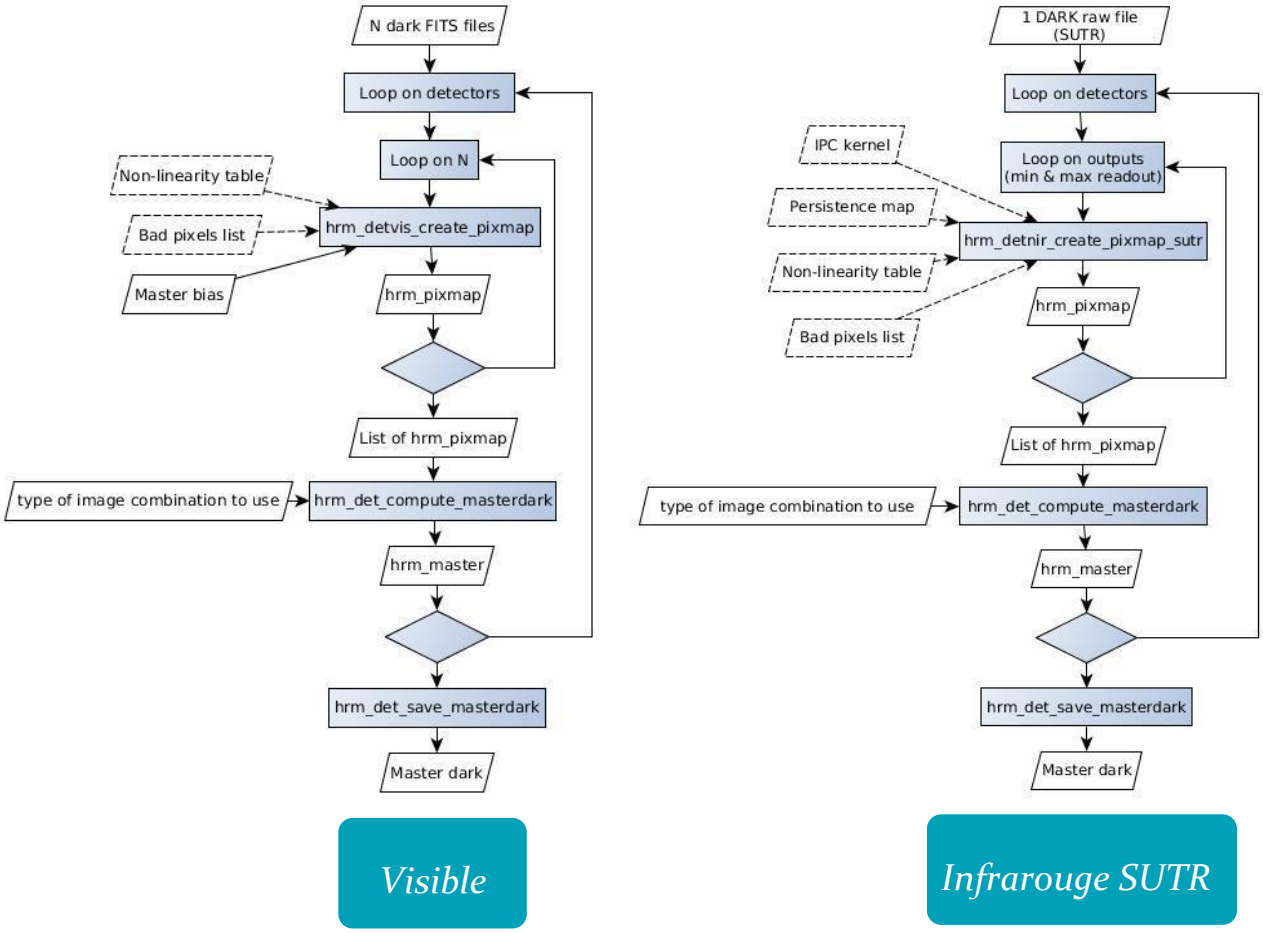


Figure II-8 : Les deux algorithmes de la recipe dark.

Nous pouvons voir que l'itération est la même que sur la *recipe* bias excepté pour le cas SUTR ou l'on va boucler sur les images du cube de données. En revanche, cette *recipe* fonctionne avec des objets *pixel map* et à encore d'autres types de fichiers en entrée.

Son fonctionnement reste relativement simple :

1. Pour chaque détecteur puis pour chaque fichier dark/image du cube, nous les chargeons en mémoire. Afin de charger une image d'une extension en mémoire, il faut utiliser le type de la librairie CPL *cpl_image* que nous mettons dans un objet *pixel map*.
2. Ensuite, nous avons une liste de *pixel map*.
3. Puis, nous les combinons grâce à une fonction de HDRL et créons un objet master.
4. Enfin, nous avons une liste de 8 objets master que nous enregistrons.

Voici deux captures d'écran de la console après l'exécution de cette *recipe* :

Visible

```
lucien@harmoni1:~/sof$ esorex harmoni_dark --mode=2 dark_vis.sof

     ***** ESO Recipe Execution Tool, version 3.13.1  *****

[ INFO  ] harmoni_dark: The mode is 2
[ INFO  ] harmoni_dark: 5 DARK.
[ INFO  ] harmoni_dark: 1 DET_LINEARITY_GAIN_TABLE.
[ INFO  ] harmoni_dark: 1 DET_BADPIX_TABLE.
[ INFO  ] harmoni_dark: The type is VR.
[ INFO  ] harmoni_dark: Loading files CHIP1...
[ INFO  ] harmoni_dark: Loading files CHIP2...
[ INFO  ] harmoni_dark: Loading files CHIP3...
[ INFO  ] harmoni_dark: Loading files CHIP4...
[ INFO  ] harmoni_dark: Loading files CHIP5...
[ INFO  ] harmoni_dark: Loading files CHIP6...
[ INFO  ] harmoni_dark: Loading files CHIP7...
[ INFO  ] harmoni_dark: Loading files CHIP8...
[ INFO  ] harmoni_dark: Writing FITS propertylist product(MASTER_DARK): master_dark.fits
[ INFO  ] esorex: Calculating product checksums
[ INFO  ] esorex: Created product master_dark.fits (in place)
[ INFO  ] esorex: 1 product created
[ INFO  ] esorex: Recipe operation(s) took            241 seconds to complete.
[ INFO  ] esorex: Total size of 5 raw input frames  =  1433.10 MB

[ INFO  ] esorex: => processing rate of     5.95 MB/sec
```

Figure II-10 : Console lors de l'exécution de la recipe dark VIS.

Infrarouge SUTR

```
lucien@harmoni1:~/sof$ esorex harmoni_dark dark_nir.sof

     ***** ESO Recipe Execution Tool, version 3.13.1  *****

[ INFO  ] harmoni_dark: The mode is 1
[ INFO  ] harmoni_dark: 1 DARK.
[ INFO  ] harmoni_dark: 1 DET_IPC_MAP.
[ INFO  ] harmoni_dark: 1 DET_PERSISTENCE_MAP.
[ INFO  ] harmoni_dark: 1 DET_LINEARITY_GAIN_TABLE.
[ INFO  ] harmoni_dark: The type is HK.
[ INFO  ] harmoni_dark: Loading files CHIP1...
[ INFO  ] harmoni_dark: Loading files CHIP2...
[ INFO  ] harmoni_dark: Loading files CHIP3...
[ INFO  ] harmoni_dark: Loading files CHIP4...
[ INFO  ] harmoni_dark: Loading files CHIP5...
[ INFO  ] harmoni_dark: Loading files CHIP6...
[ INFO  ] harmoni_dark: Loading files CHIP7...
[ INFO  ] harmoni_dark: Loading files CHIP8...
[ INFO  ] harmoni_dark: Writing FITS propertylist product(MASTER_DARK): master_dark.fits
[ INFO  ] esorex: Calculating product checksums
[ INFO  ] esorex: Created product master_dark.fits (in place)
[ INFO  ] esorex: 1 product created
[ INFO  ] esorex: Recipe operation(s) took            337 seconds to complete.
[ INFO  ] esorex: Size of single raw input frame  = 17179.92 MB

[ INFO  ] esorex: => processing rate of    50.95 MB/sec
```

Figure II-9 : Console lors de l'exécution de la recipe dark NIR SUTR.

Vous pouvez voir que dans les deux cas, la *recipe* écrit le même fichier en sortie : le master dark.

6. Codage du squelette de toutes les recipes

Par la suite, je me suis vraiment concentré sur le sujet du stage, c'est-à-dire le codage de l'architecture du logiciel. La façon de coder ressemble fortement aux premières missions. En effet, ensuite j'ai codé toutes les autres *recipes* et c'est ainsi que j'étais amené à coder toutes les entrées/sorties dont nous avons besoin.

Voici par ordre chronologique les autres *recipes* codés :

- *Recipe* **flat**
- *Recipe* **wavecal**
- *Recipe* **geometry**, avant cette *recipe*, nous avions supposé que les fichiers auraient toujours les mêmes noms d'extensions. Or, ce nom peut changer, donc j'ai dû revoir la gestion du nom des extensions afin qu'il soit géré dynamiquement.
- *Recipe* **illum**, nous avons remarqué que les valeurs associées aux mots-clés d'un *header* que nous calculions avaient trop de nombres après la virgule. J'ai ajouté une fonction qui permet d'arrondir le chiffre à deux nombres après la virgule.
- *Recipe* **flux**, en testant les *recipes*, nous avons remarqué que bien que nous ne prenions pas en compte les fichiers n'ayant pas la même configuration, ils apparaissaient quand même dans les mots-clés du fichier de sortie.
  Ceci était problématique car ces fichiers n'avaient pas servi au calcul du fichier de sortie. J'ai donc dû supprimer ces fichiers de l'objet comptant les fichiers d'entrée que l'on appelle « *frameset* ».
  En somme, au début de chaque *recipe* j'ai un tableau contenant tous les types de fichiers possibles. Puis, avec une fonction, je vérifie que les fichiers d'entrées de la *recipe* sont tous compatibles (même configuration, géré par le *recipe*...). Dans le cas contraire, le fichier ne sera pas pris en compte et supprimé de la *frameset*.
- *Recipe* **telluric**
- *Recipe* **astrometry**
- *Recipe* **pixmap ins**
- *Recipe* **pixmap env**
- *Recipe* **cube**

J'avais à ma disposition des exemples de fichiers afin d'avoir un format de référence me permettant de coder les différents formats de fichiers. D'une part, cela me permet de savoir comment lire le fichier (le charger en mémoire), et d'autre part, comment créer un fichier (fichier de sortie des *recipes* par exemple).

De plus, chaque fichier de sortie doit contenir des mots-clés spécifiques dans son header appelés *QC parameters*. Il peut parfois y en avoir plus de 60 par extension dans une image. Ces *QC* contiennent des résultats de calcul sur l'image comme la moyenne, la médiane...

Certaines *recipe* ont en entrée les fichiers de sortie d'autres *recipe*. Comme le montre le schéma suivant, afin de créer le cube de données, nous avons besoin de toutes les sorties des *recipes*. J'ai pu tester la compatibilité des fichiers que je génère avec les autres *recipes*. Voici le schéma des flux (*workflow*) :

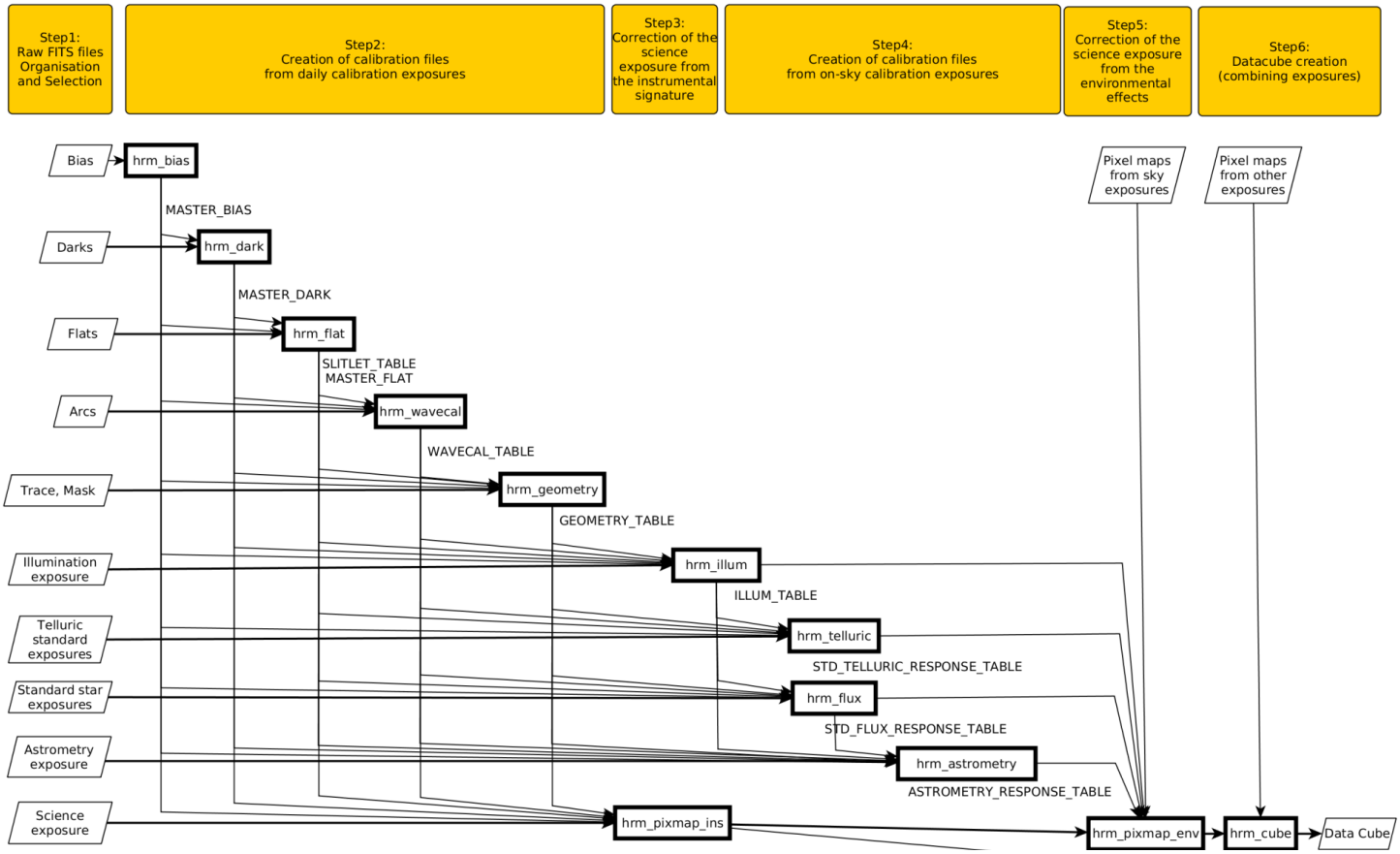


Figure II-11 : Workflow du logiciel.

Enfin, j'ai pu coder l'interface graphique du logiciel.

7. Point technique, découpage d'une image VIS

Quand nous chargeons une image visible en mémoire, nous devons la redécouper afin d'enlever certains pixels qui ne doivent pas être pris en compte.

Dans le header de l'extension, il y a certains mots-clés permettant de savoir combien de pixels nous devons retirer à gauche/en bas (pre-scan), à droite/en haut (over-scan) et la taille de l'image.

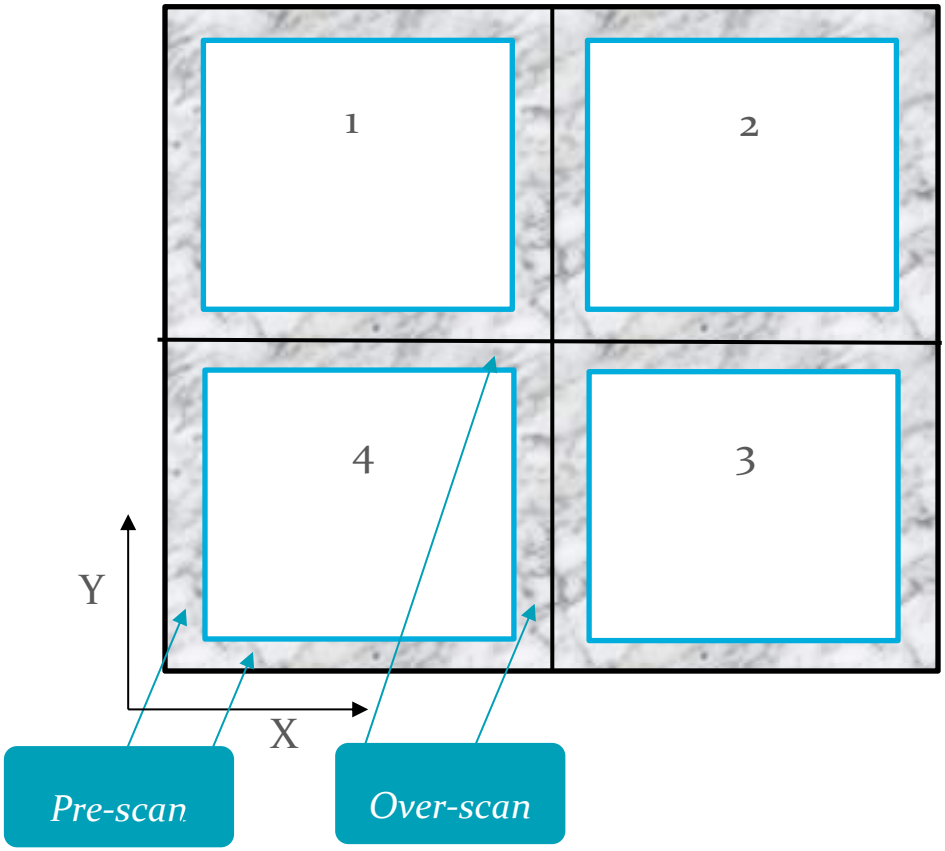


Figure II-12 : Image avant le découpage.

En résumé, l'image entière doit être recoupée en quatre, puis nous enlevons le pre/over-scan sur chaque image et enfin nous recollons les quatre images afin d'en former une seule.

Il m'a d'abord fallu récupérer les valeurs des pre-scan, taille de l'image…

J'ai créé une fonction permettant d'attribuer aux variables passées en paramètres, les valeurs équivalentes.

Afin de faire ceci, il faut déjà charger le *header* de l'image :

```
/* Load header */
header_load = cpl_propertylist_load(filename, extension_place);
```

Figure II-13 : Charger le header d'une image.

Le *header* d'une image est de type *cpl_propertylist*. Afin de mieux comprendre comment fonctionne CPL, voici un extrait de la documentation :

cpl_propertylist* cpl_propertylist_load (const char * name,
cpl_size position
)

Create a property list from a file.

Parameters

name Name of the input file.

position Index of the data set to read.

Returns

The function returns the newly created property list or `NULL` if an error occurred.

Figure II-14 : Documentation CPL sur le chargement d'un header.

Nous pouvons constater qu'il faut lui donner en paramètres le nom du fichier et le numéro de l'extension (nombre variant de 1 à 8).

Une fois que nous avons récupéré le *header*, je vérifie qu'il n'est pas NULL afin d'être certain qu'aucune erreur n'est survenue.

Lorsque nous avons fait tout cela, nous pouvons récupérer les valeurs souhaitées dans le *header*.

Voici un exemple de récupération du pre-scan sur l'axe x (à gauche) :

```
/* Get prscx */
if(prscx){
    tmp     = harmoni_raw_get_extension_parameter_name(HARMONI_DET_PRSCX, output);
    *prscx  = cpl_propertylist_get_int(header, tmp);
    cpl_free(tmp);
}
```

Figure II-15 : Récupération de la valeur du pre-scan sur l'axe x.

Je récupère d'abord le nom du mot-clé dans une variable temporaire, puis, je récupère la valeur de ce paramètre dans la variable prscx et enfin je libère la mémoire du nom précédemment récupéré.

Je récupère l'image à découper :

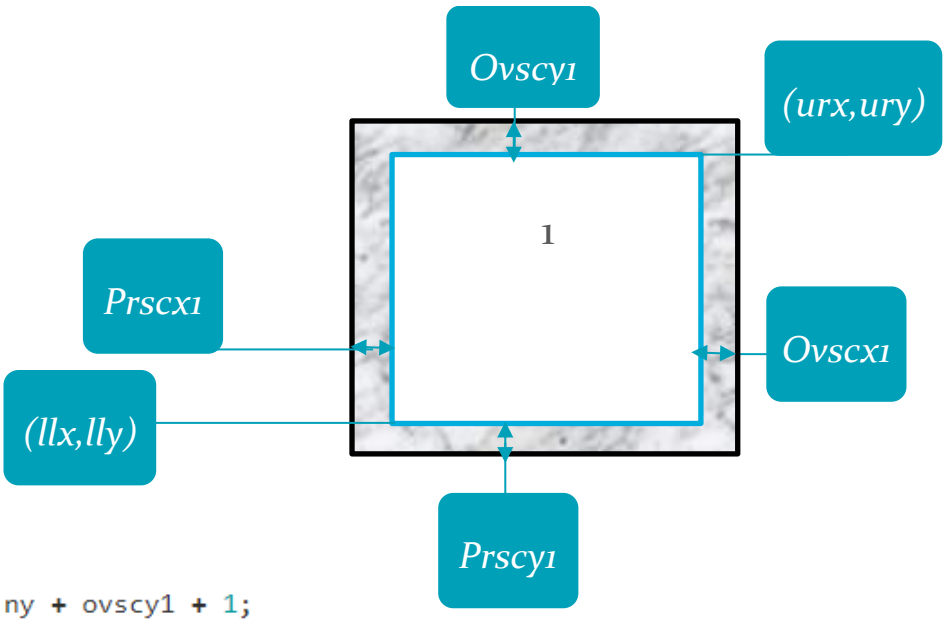
```
/* Get extname without INT01 */
harmoni_get_extname(chipnames, filename, detnum);
harmoni_remove_sub(chipnames[detnum], HARMONI_INT_NAME);
/* Load image */
harmoni_raw = cpl_image_load(filename,CPL_TYPE_UNSPECIFIED,0,extension_place);
```

Figure II-16 : Charger une image.

Je récupère le nom de l'extension correspondant au numéro de l'extension, puis je vais charger l'image. Je ne spécifie pas de type afin qu'il soit mis automatiquement.

Puis, j'extrais chaque image sans le pre/over scan :

Figure II-17 : Une des quatre parties de l'image.




```
/* Get out1 */
llx = prscx1 + 1;
lly = 2 * prscy1 + ny + ovscy1 + 1;

urx = prscx1 + size;
ury = 2 * prscy1 + 2 * ny + ovscy1;

cpl_image *out1 = cpl_image_extract(harmoni_raw, llx, lly, urx, ury);
```

Figure II-18 : Récupération d'une des quatre parties de l'image.

Ny correspond à la taille sur l'axe y de l'image souhaitée, et size en x. Je calcule les coordonnées du coin inférieur gauche et coin supérieur droit, puis, grâce à ces coordonnées, je peux extraire l'image.

Enfin, je recolle les images dans une nouvelle, et je libère la mémoire :

```
/* Now we have to stick the images */
cpl_image *harmoni_image = cpl_image_new(2 * nx, 2 * ny, cpl_image_get_type(harmoni_raw));

cpl_image_copy(harmoni_image, out4, x1, y1);
cpl_image_copy(harmoni_image, out3, size + 1, y1);
cpl_image_copy(harmoni_image, out1, x1, ny + 1);
cpl_image_copy(harmoni_image, out2, size + 1, ny + 1);

cpl_image_delete(out1);
cpl_image_delete(out2);
cpl_image_delete(out3);
cpl_image_delete(out4);
cpl_image_delete(harmoni_raw);
cpl_propertylist_delete(header_load);
```

Figure II-19 : Recollage des quatre images et libération de la mémoire.

Je crée une nouvelle image faisant deux fois la taille d'une image recoupée car les images sont collées comme ci-dessous :

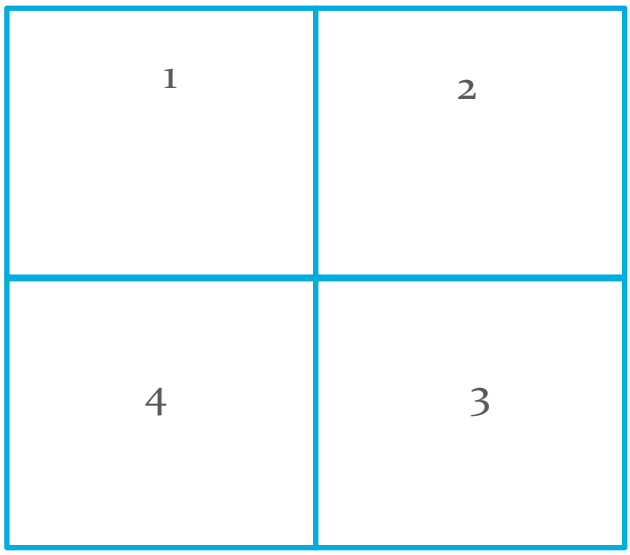


Figure II-20 : Nouvelle image recadrée.

Puis, je copie chaque image coupée dans la nouvelle image au bon emplacement, je donne en paramètre le coin inférieur gauche. Enfin je libère la mémoire des images dont je n'ai plus besoin.

8. Documentation à partir du document design

Afin de décrire le mieux possible le logiciel, j'ai utilisé le document design des spécifications de la librairie. Pour des raisons de confidentialité, je n'ai pas pu le mettre en annexe de ce rapport.

Après avoir codé toutes les *recipes*, j'ai repris le document design afin de commenter tout le code le plus précisément possible. La génération de la documentation à l'aide de Doxygen permet maintenant d'avoir une documentation assez complète du logiciel. Voici par exemple la documentation de la *recipe* bias :

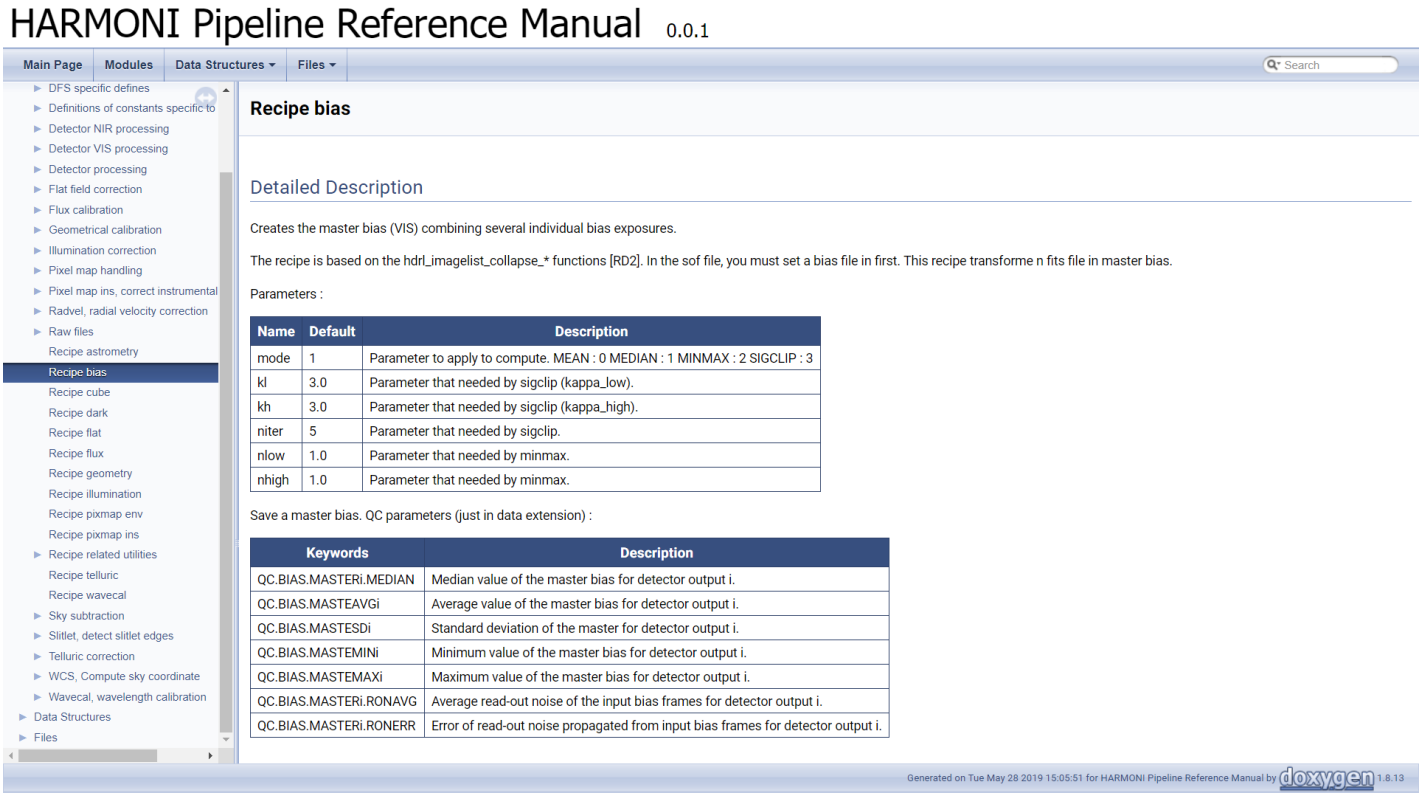


Figure II-21 : Documentation de la recipe bias.

D'autres exemples de la documentation que j'ai faite sont disponibles en annexe (voir Annexes – Documentation du logiciel).

Afin de générer cette documentation, il faut commenter le code d'une certaine manière. Par exemple, avant chaque fonction, il faut expliquer les entrées/sorties et ce que fait la fonction.

```
/*-----------------------------------------------------------------------------*/
/**
  @brief    Compute a master bias.
  @param    imlist              List of images.
  @param    params              Parameters to apply.
  @param    detnum              Number of detector.
  @param    list_header         List of header.
  @return   A harmoni detmaster : harmoni_detmaster.

  Create a new harmoni_detmaster and set it his parameters. To do that,
  We use the imagelist to determine the data and dq. If a badpixtab is passed
  as parameter, we use it to mask the result.
  It's based on the hdrl_imagelist_collapse_* functions [RD2].

 */
/*-----------------------------------------------------------------------------*/
```

Figure II-22 : Commentaire d'une fonction.

9. Interface graphique

L'ESO demande que les logiciels de réduction de données soient livrés avec une interface graphique basé sur leur logiciel EsoReflex. EsoReflex est lui-même basé sur Kepler.

Kepler représente visuellement un *workflow* sous la forme d'une séquence d'acteurs (actors) avec un seul réalisateur (director). Ce dernier planifie l'exécution des acteurs et le premier manipule les données.

Les principaux composants d'un *workflow* sont les suivants :

- **Director**, détermine l'ordre d'exécution des acteurs et leur indique quand ils peuvent agir. Il y a plusieurs types de directeur, le paramètre par défaut pour EsoReflex est le Director Dynamic Dataflow (DDF). Il doit seulement y avoir un directeur par *workflow*.

- **Actor**, représente une seule étape d'exécution. L'acteur prend des données des ports d'entrée, effectue des traitements et envoie les résultats aux ports de sortie. Chaque acteur peut avoir un certain nombre de paramètres qui contrôlent son exécution.

- **Port**, chaque acteur peut posséder un ou plusieurs ports, ce qui lui permet d'échanger des données avec d'autres acteurs.

- **Jeton**, un objet qui encapsule des données. Les acteurs échangent des données sous forme de jetons via des ports.

Voici à quoi ressemble Kepler :

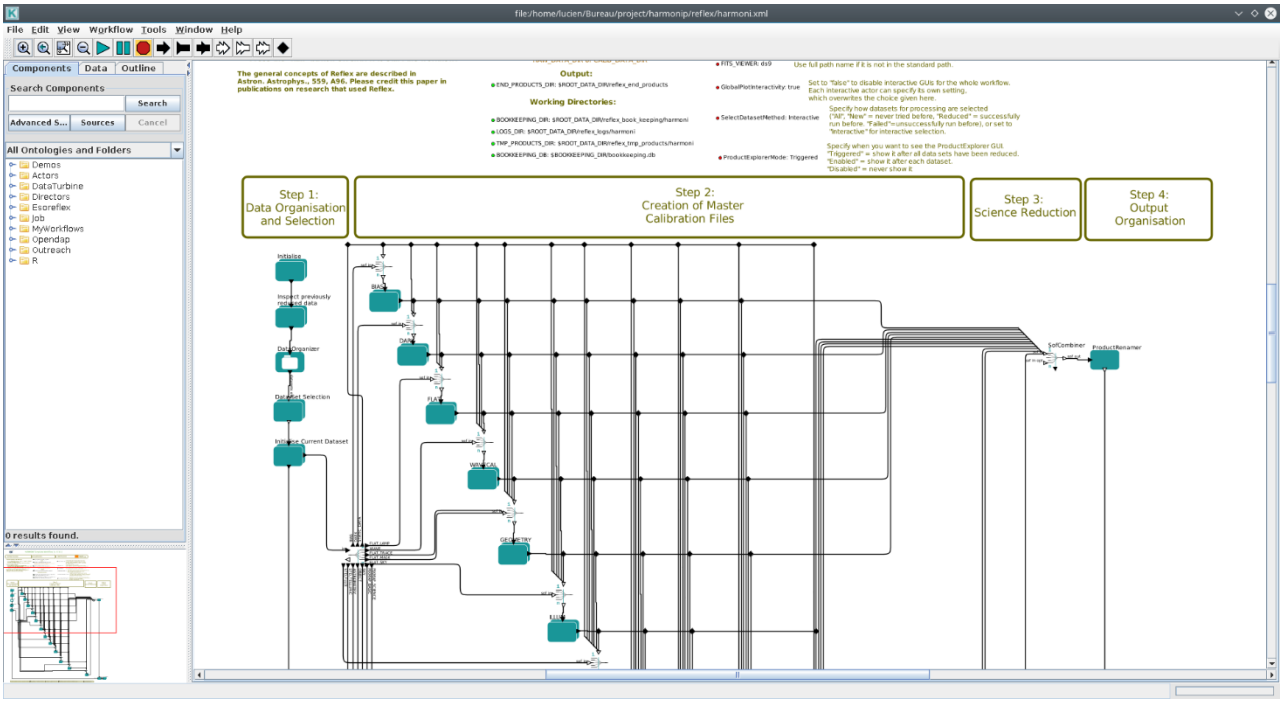


Figure II-23 : Capture d'écran de EsoReflex (Kepler)

Nous pouvons constater au milieu le *workflow*, à gauche les composants et en haut la bar de gestion qui permet notamment d'exécuter le *workflow*.

J'ai d'abord dû installer EsoReflex sur mon ordinateur. Par la suite, j'ai lu la documentation afin de comprendre comment faire fonctionner celui-ci.

Mon premier but était d'avoir un *workflow* qui exécute la *recipe* bias et prenne en entrée des fichiers FITS. J'ai vainement suivi le tutoriel de la documentation permettant de créer un workflow simple. Je ne comprenais pas d'où venait l'erreur. J'ai donc cherché sur le site de l'ESO un *workflow* Template afin de l'exécuter et trouver comment faire fonctionner le mien. J'ai donc dû installer les *recipes* de test, récupérer le *workflow* de test et configurer EsoReflex pour qu'il prenne en compte les bonnes *recipes* (pas les miennes). Finalement, le *workflow* de test a fini par s'exécuter correctement.

En refaisant étape par étape mon *workflow* et en comparant ce que je faisais à celui de test, j'ai fini par trouver mon erreur. Cela provenait du fichier OCA. Il manquait une règle dans mon fichier, ce qui fait que rien ne fonctionnait. Voici la ligne manquante :

```
73        REFLEX.TARGET = "T";
```

Figure II-24 : OCA règle

Cette règle précise à quel endroit se termine le *workflow* et est donc essentielle.

Par la suite, j'ai pu rajouter les autres *recipes*. En testant le *workflow*, j'ai pu trouver et corriger des bugs dans mon code. En effet, je n'avais pas pu tester tous les cas précédemment.

Ci-dessous le composant qui permet d'exécuter une *recipe*. Au-dessus du rectangle vert (le composant de la recipe) se trouve ce que l'on appel un *combiner*, il permet de combiner différents fichier SOF qu'il a en entrée.

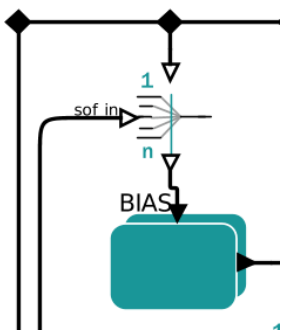


Figure II-25 : Composant recipe bias

Ce composant est lui-même composé de sous composants :

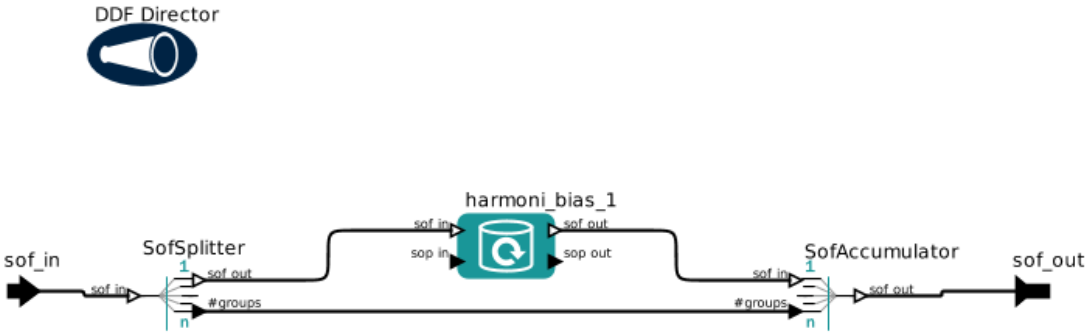


Figure II-26 : Sous composants recipe bias

Nous pouvons constater au milieu ce qui représente la *recipe* bias. De plus, on donne en entrée à la recipe un fichier SOF contenant les bons fichiers à traiter. En sortie de la *recipe* se trouve un fichier SOF avec les fichiers FITS créés par la recipe.

Voici une capture d'écran du workflow entier :

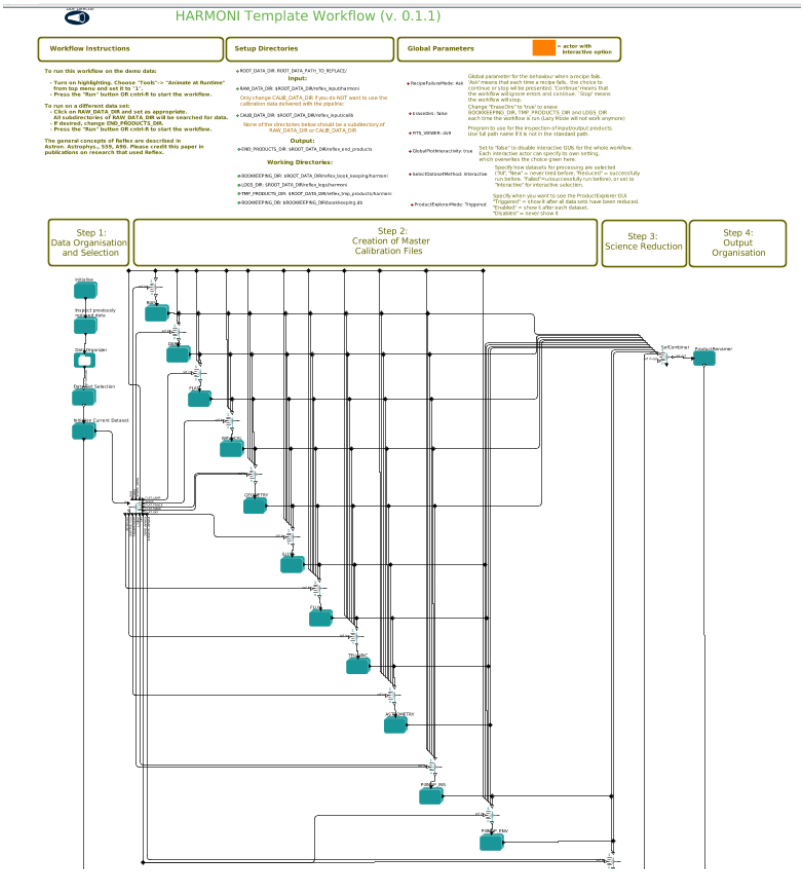


Figure II-27 : Workflow HARMONI

10. Planning des tâches

Voici le déroulement des tâches durant le stage :

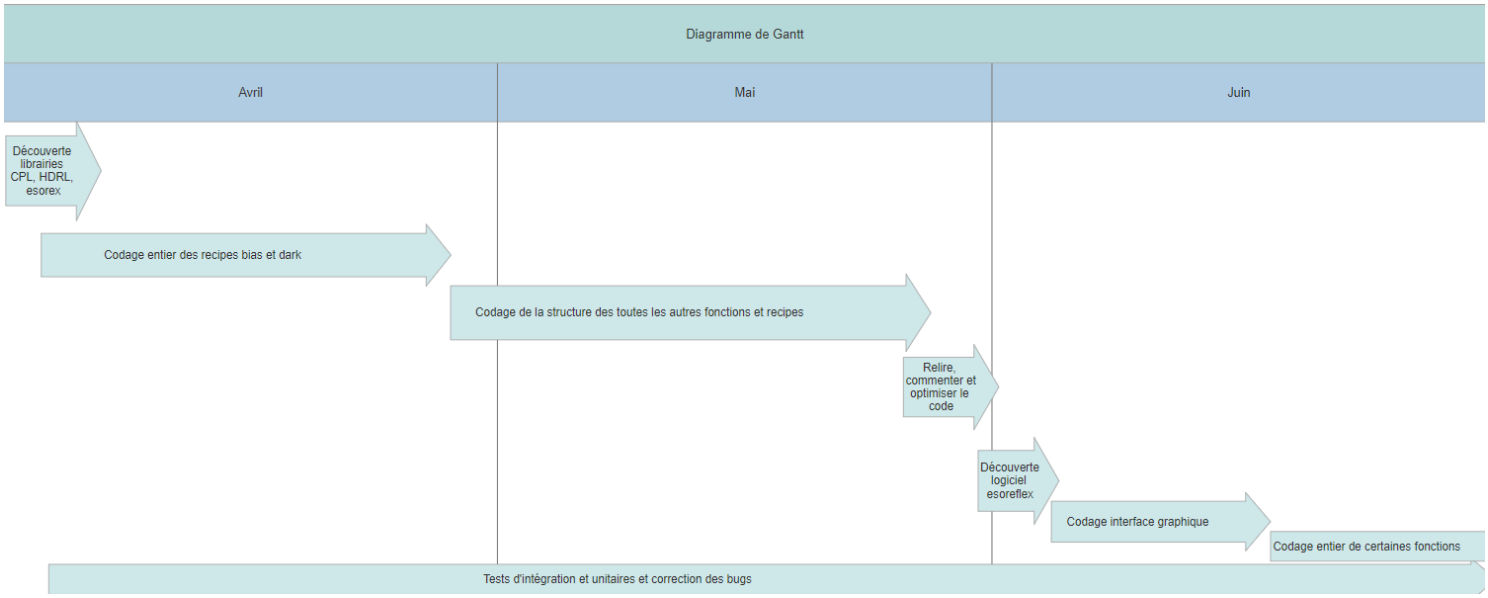


Figure II-28 : Diagramme de Gantt.

Nous pouvons constater que certaines tâches continuent après juin. En effet, je vais travailler en CDD afin d'avancer le projet durant le mois de juillet. De plus, j'ai effectué des tests d'intégration et codé des tests unitaires durant tout le stage car cela m'a permis de détecter et corriger des bugs dans mon code.

III. Bilan de l'expérience de stage

Ce stage a été très enrichissant pour moi car il m'a permis de découvrir de manière approfondie le secteur du développement informatique dans l'astronomie.

1. Bilan du travail effectué

J'ai pu développer un logiciel fonctionnel, qui répond aux attentes de ma tutrice, mais plus généralement aux fortes contraintes de l'ESO. Il m'a fallu être rigoureux, afin de respecter les nombreuses normes de développement, notamment en commentant et créant la documentation de tout le code en anglais. J'ai mis en place un environnement graphique.

2. Bilan des apprentissages techniques

Ce stage m'a permis d'approfondir mes connaissances dans le langage de programmation C, dans l'environnement Linux et dans l'utilisation de l'anglais technique. J'ai pu découvrir et appréhender de nouvelles librairies et logiciels de l'ESO mais également des librairies moins spécifiques à l'astronomie comme Doxygen qui permet de générer la documentation.

3. Bilan humain

Ce stage dans un milieu professionnel tourné vers la recherche a été particulièrement instructif. En effet, j'ai pu développer mes aptitudes personnelles et techniques grâce à cet environnement spécifique. J'ai eu par exemple l'opportunité d'assister à une soutenance de thèse. J'ai pu aussi côtoyer des chercheurs passionnés par leur domaine de compétence qui ont partagé leurs savoirs avec moi. Je me suis senti parfaitement intégré dans cette équipe.

La découverte de la vie au sein d'une entreprise a été très enrichissante sur le plan relationnel. Le fait de se référer à une tutrice constitue une aide précieuse dont je n'aurais pu me passer.

4. Bilan professionnel

Ce stage m'a conforté dans mon choix de travailler dans le développement informatique. Il m'a vraiment motivé à évoluer dans ce domaine qui me passionne. Cette expérience professionnelle m'a beaucoup plu et m'encourage dans ma poursuite d'études dans une école d'ingénieur en alternance.

Conclusion

Figure III-1 : ELT/HARMONI

Pour conclure, le CRAL m'a beaucoup apporté. Lors de ce stage de 3 mois, j'ai pu mettre en pratique mes connaissances théoriques acquises durant ma formation, ce qui m'a permis d'atteindre le but qui m'avait été fixé. J'ai pu développer un logiciel fonctionnel qui s'inscrit dans un projet international et tourné vers l'avenir comme le montre la figure ci-dessus : le plus grand télescope optique et proche infrarouge au monde : "l'œil le plus grand au monde pointé sur le ciel". Et le fait d'avoir pu contribuer à un tel projet est très formateur. J'ai énormément appris sur l'astronomie et sur les méthodes de travail de l'ESO. HARMONI regroupe un consortium de plusieurs laboratoires européens ce qui nous montre encore l'échelle de ce projet.

Mon travail a permis de valider et d'améliorer le design du logiciel. Ce logiciel est un élément clé du projet HARMONI, car sans logiciel de réduction de données, les données des détecteurs sont inexploitables par les chercheurs. Ce projet permettra donc dans le futur de faire avancer la recherche dans le domaine de l'astronomie. En effet, ce logiciel ne représente peut-être qu'une petite partie du projet HARMONI, mais il est essentiel. Grâce à ce travail, le CRAL va mettre à jour et avoir un document de design plus détaillé qui sera livré à l'ESO.

De plus, il est nécessaire de rappeler qu'à la date de rendu de ce rapport, le stage n'est pas encore fini, puisqu'il se termine le 30 juin 2019. Par ailleurs, je vais travailler en CDD au CRAL le mois de juillet 2019. Ainsi, pendant les mois restants, d'autres fonctionnalités du logiciel seront codées.

La formation de l'IUT à été essentielle au bon déroulement et à la réussite de mes missions. En effet, j'ai pu mettre en pratique certain cours comme les cours de C, de GIT, de programmation système…

Enfin, je tiens à exprimer ma satisfaction d'avoir pu travailler dans de bonnes conditions matérielles et un environnement agréable.

Sitographie

Centre de Recherche Astrophysique de Lyon, CRAL, consulté le 27/05/19 :
https://cral.univ-lyon1.fr/

Observatoire de Lyon, Wikipedia, consulté le 25/05/19 :
https://fr.wikipedia.org/wiki/Observatoire_de_Lyon

European southern observatory, ESO, consulté le 20/05/19 :
https://www.eso.org/public/

HARMONI - Home, HARMONI, consulté le 27/05/19 :
http://astroweb1.physics.ox.ac.uk/instr/HARMONI/homepage.html

CPL documentation, ESO, consulté le 28/05/19 :
https://www.eso.org/sci/software/cpl/reference/
https://www.eso.org/sci/software/cpl/documentation.html

EsoReflex documentation, ESO, consulté le 12/06/19 :
https://www.eso.org/sci/software/esoreflex/
ftp://ftp.eso.org/pub/dfs/reflex/EsoReflexUserManual-3.9.pdf
ftp://ftp.eso.org/pub/dfs/reflex/reflex_dev_guide-1.0.pdf

EsoRex documentation, ESO, consulté le 28/05/19 :
https://www.eso.org/sci/software/cpl/esorex.html

HDRL documentation, ESO, consulté le 28/05/19 :
https://www.eso.org/sdd/pub/INSCS14/INSCS2014_pipeline_slides/03b_HDRL.pdf

MUSE pipeline documentation, ESO, consulté le 28/05/19 :
https://www.eso.org/sci/software/pipelines/muse/

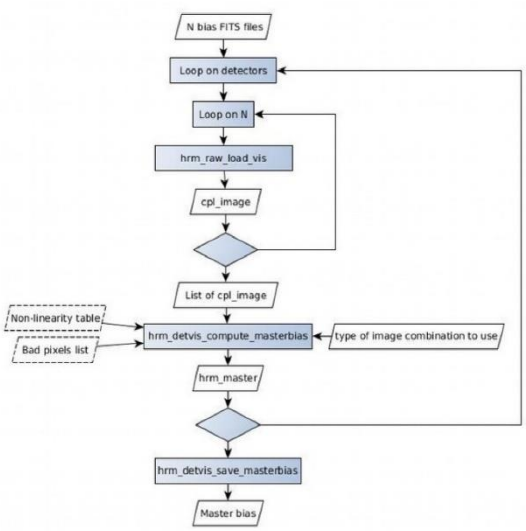
MUSE EsoReflex documentation, ESO, consulté le 12/06/19 :
ftp://ftp.eso.org/pub/dfs/pipelines/muse/muse-reflex-tutorial-11.0.pdf

OCA documentation, ESO, consulté le 12/06/19 :
http://www.eso.org/~qc/dfos/OCA_syntax.html

Annexes

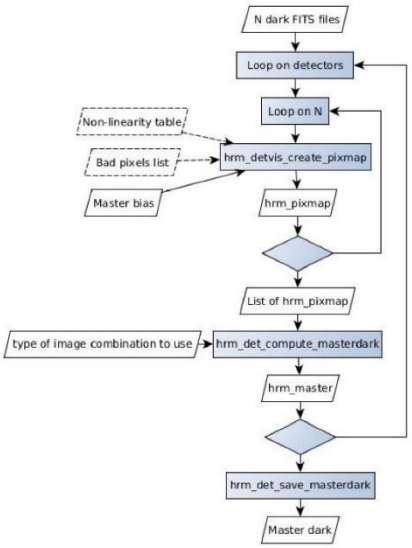
Architecture prévue

hrm_bias

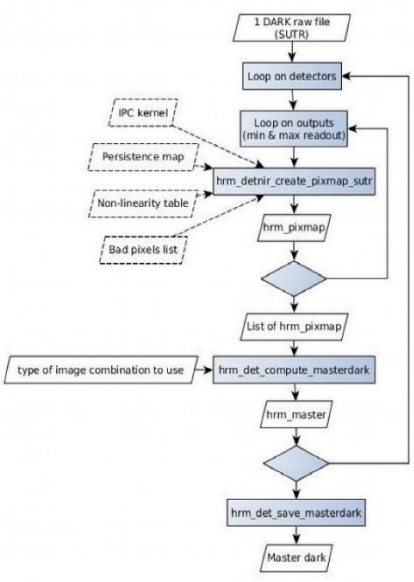


hrm_dark

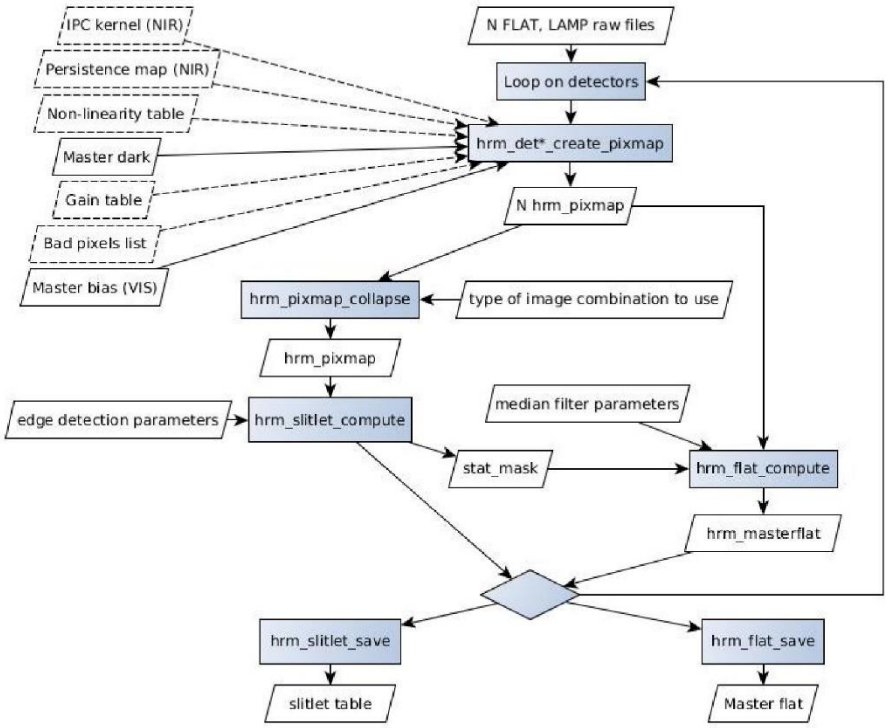
VIS



NIR

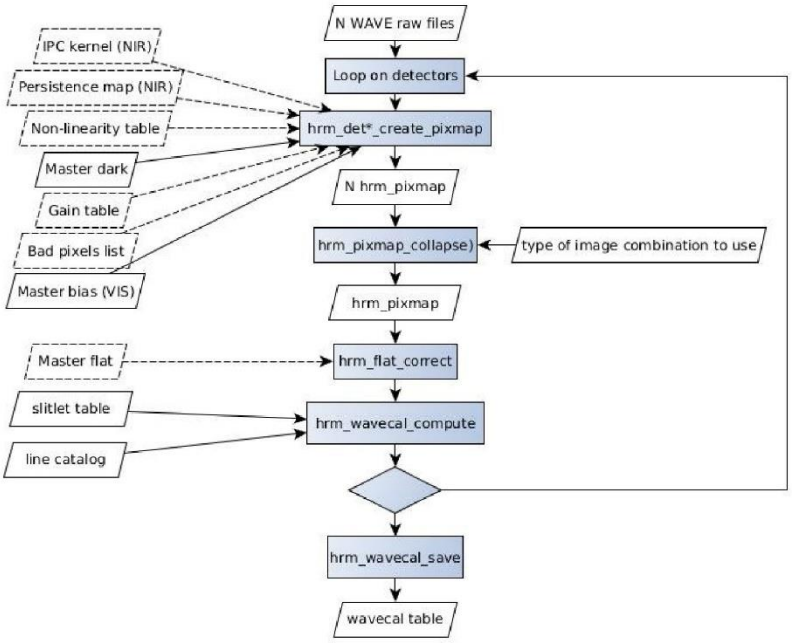


hrm_flat

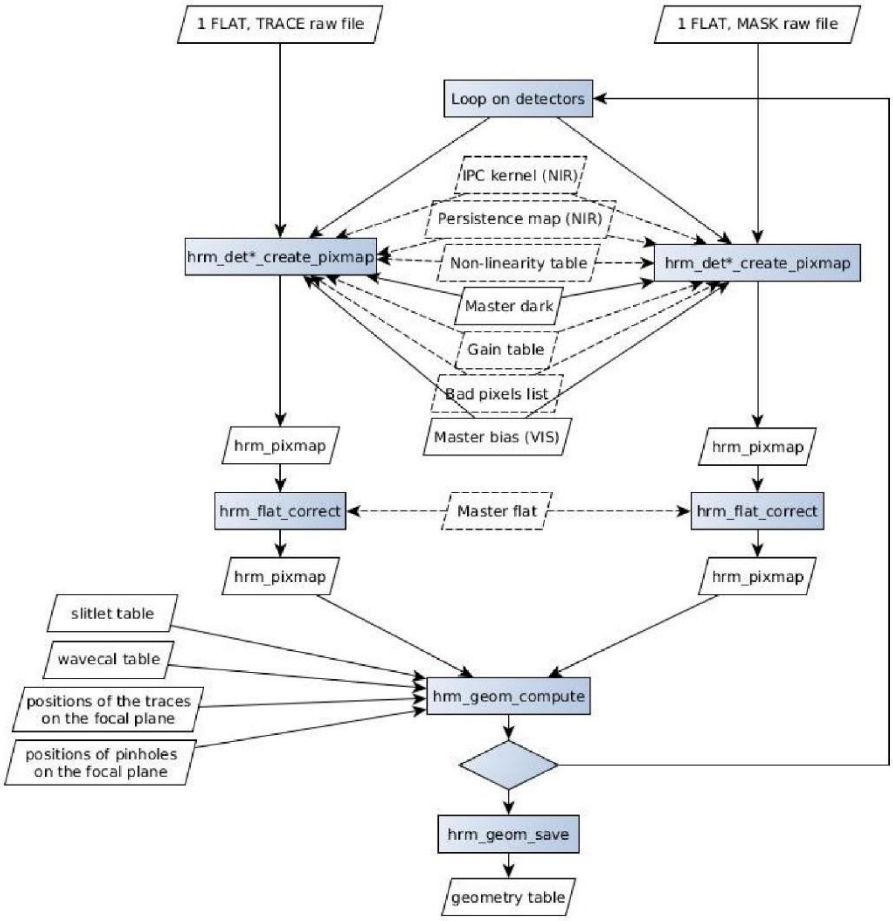


Split into hrm_flat and hrm_slitlet ???

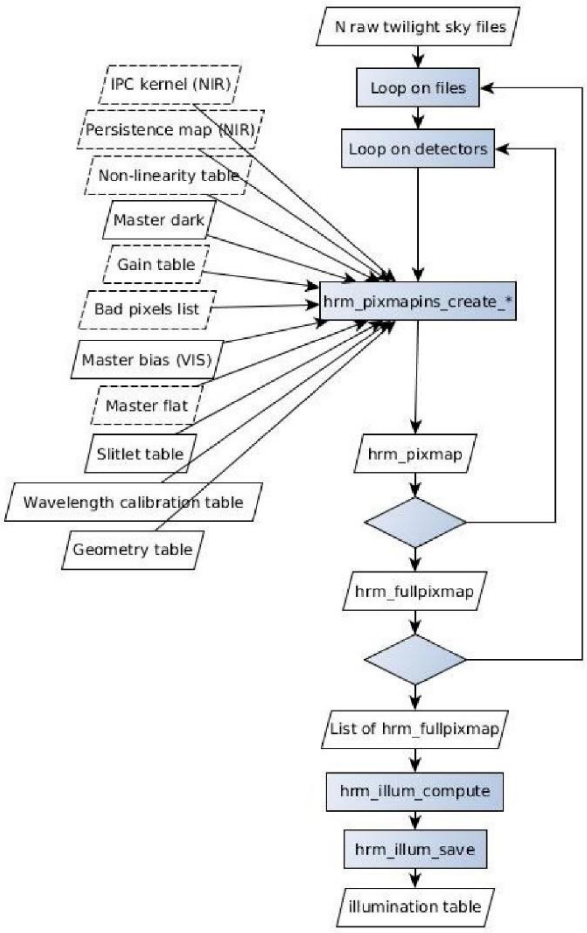
hrm_wavecal


N WAVE raw files
Loop on detectors
IPC kernel (NIR)
Persistence map (NIR)
Non-linearity table
Master dark
Gain table
Bad pixels list
Master bias (VIS)
hrm_det*_create_pixmap
N hrm_pixmap
hrm_pixmap_collapse
type of image combination to use
hrm_pixmap
Master flat
hrm_flat_correct
slitlet table
line catalog
hrm_wavecal_compute
hrm_wavecal_save
wavecal table

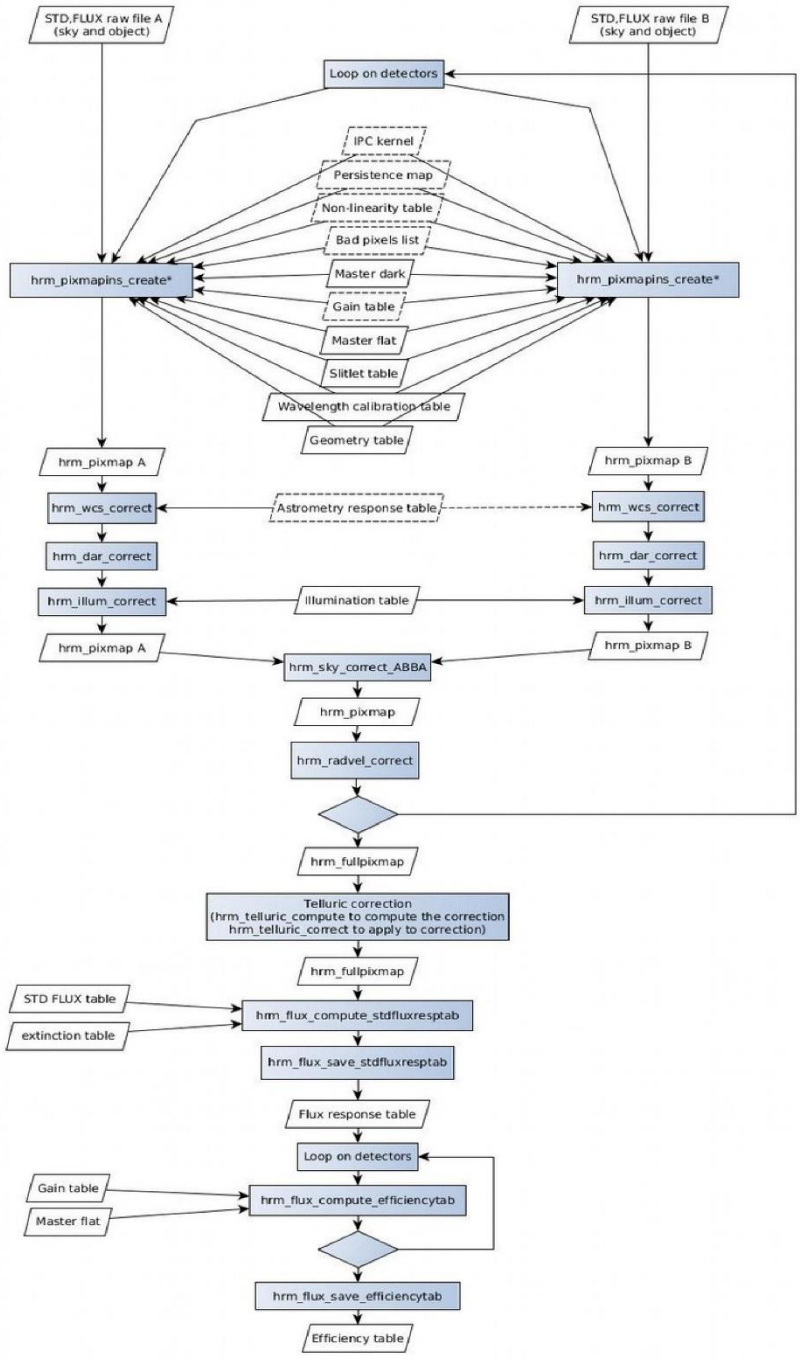
hrm_geometry



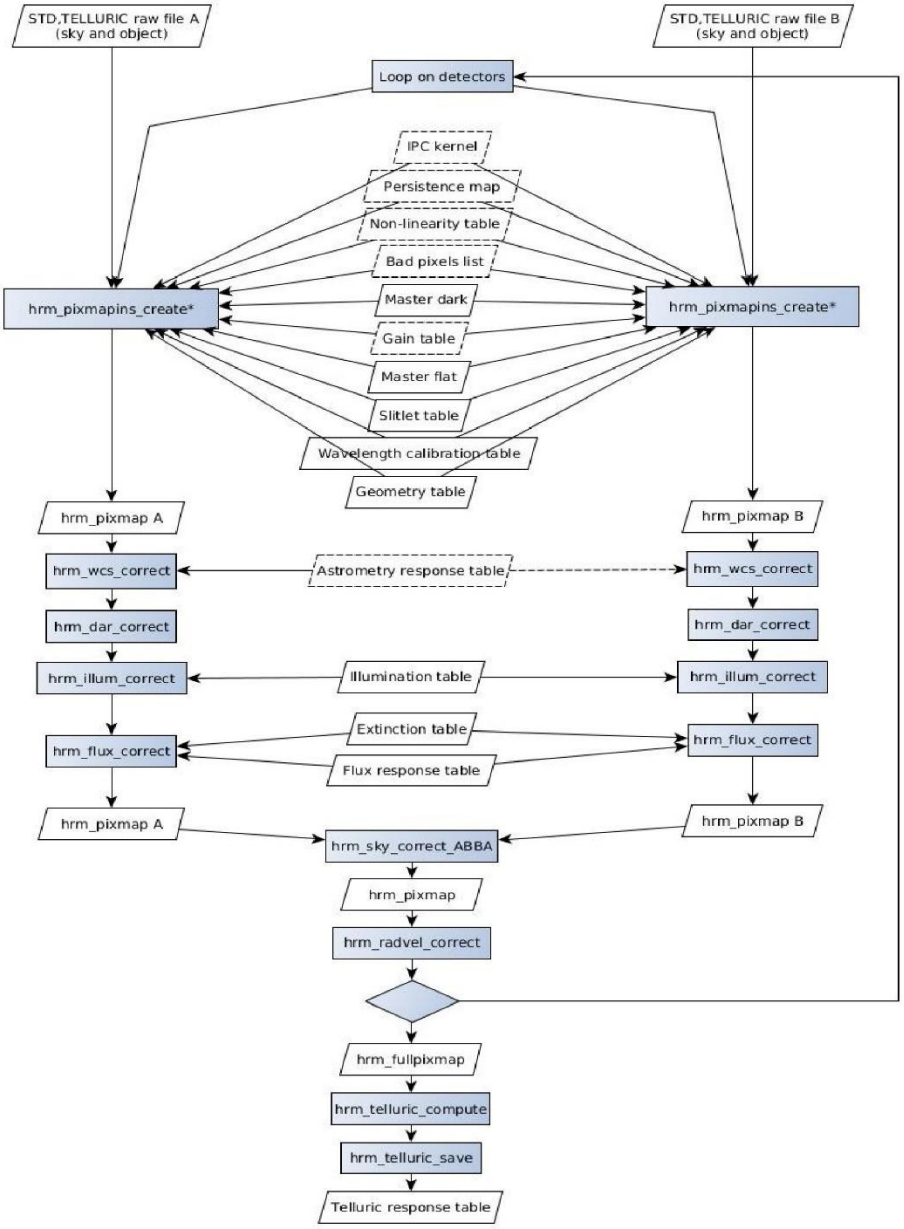
hrm_illum


N raw twilight sky files
Loop on files
Loop on detectors
IPC kernel (NIR)
Persistence map (NIR)
Non-linearity table
Master dark
Gain table
Bad pixels list
Master bias (VIS)
Master flat
Slitlet table
Wavelength calibration table
Geometry table
hrm_pixmapins_create_*
hrm_pixmap
hrm_fullpixmap
List of hrm_fullpixmap
hrm_illum_compute
hrm_illum_save
illumination table

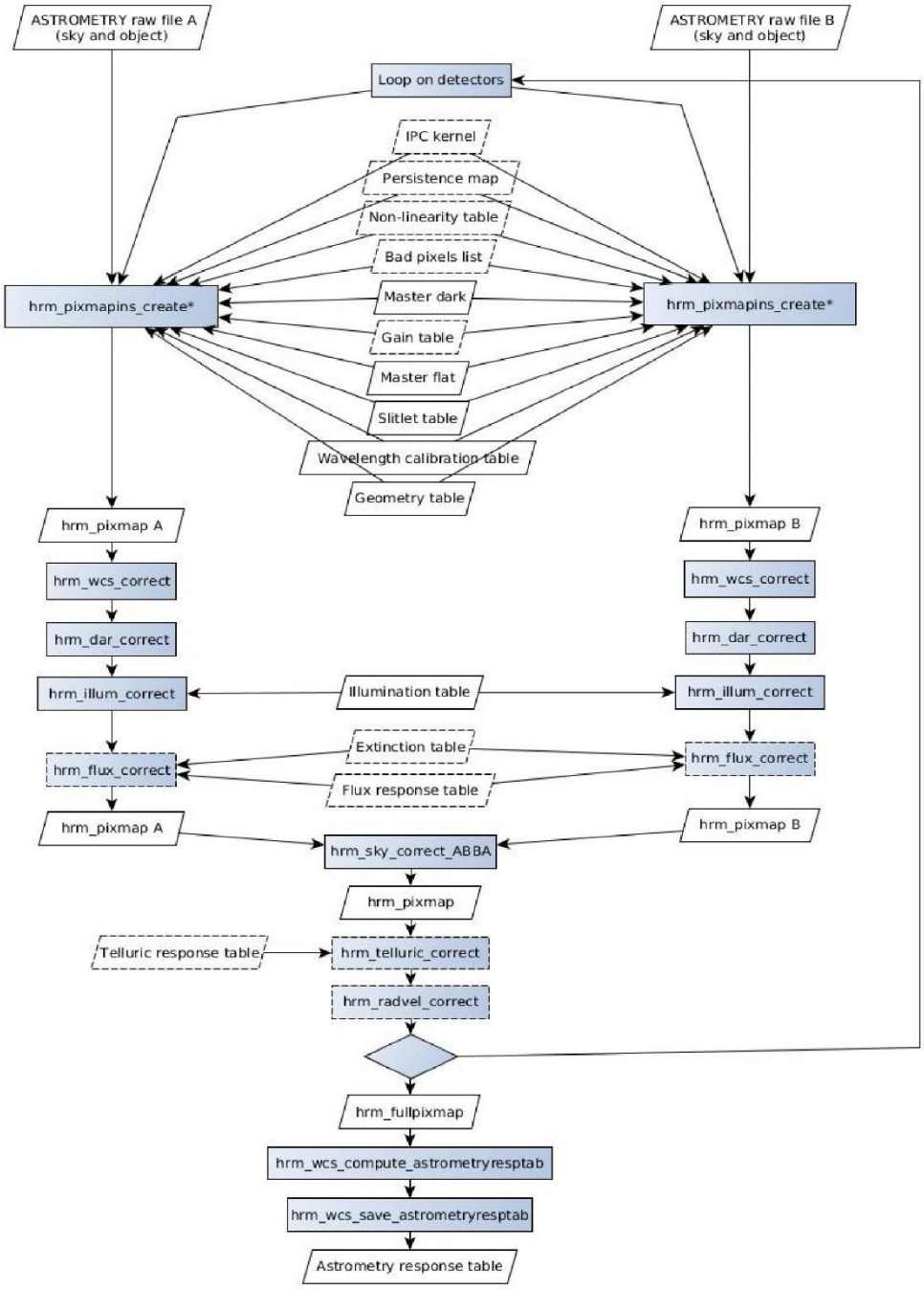
hrm_flux


STD,FLUX raw file A (sky and object)
STD,FLUX raw file B (sky and object)
Loop on detectors
IPC kernel
Persistence map
Non-linearity table
Bad pixels list
Master dark
Gain table
Master flat
Slitlet table
Wavelength calibration table
Geometry table
hrm_pixmapins_create*
hrm_pixmapins_create*
hrm_pixmap A
hrm_pixmap B
hrm_wcs_correct
Astrometry response table
hrm_wcs_correct
hrm_dar_correct
hrm_dar_correct
hrm_illum_correct
Illumination table
hrm_illum_correct
hrm_pixmap A
hrm_pixmap B
hrm_sky_correct_ABBA
hrm_pixmap
hrm_radvel_correct
hrm_fullpixmap
Telluric correction
(hrm_telluric_compute to compute the correction
hrm_telluric_correct to apply to correction)
hrm_fullpixmap
STD FLUX table
extinction table
hrm_flux_compute_stdfluxresptab
hrm_flux_save_stdfluxresptab
Flux response table
Loop on detectors
Gain table
Master flat
hrm_flux_compute_efficiencytab
hrm_flux_save_efficiencytab
Efficiency table

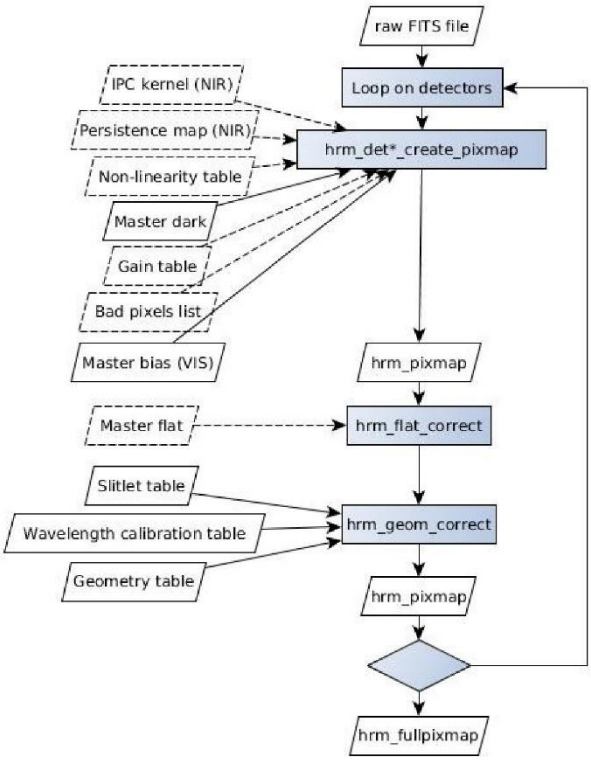
hrm_telluric



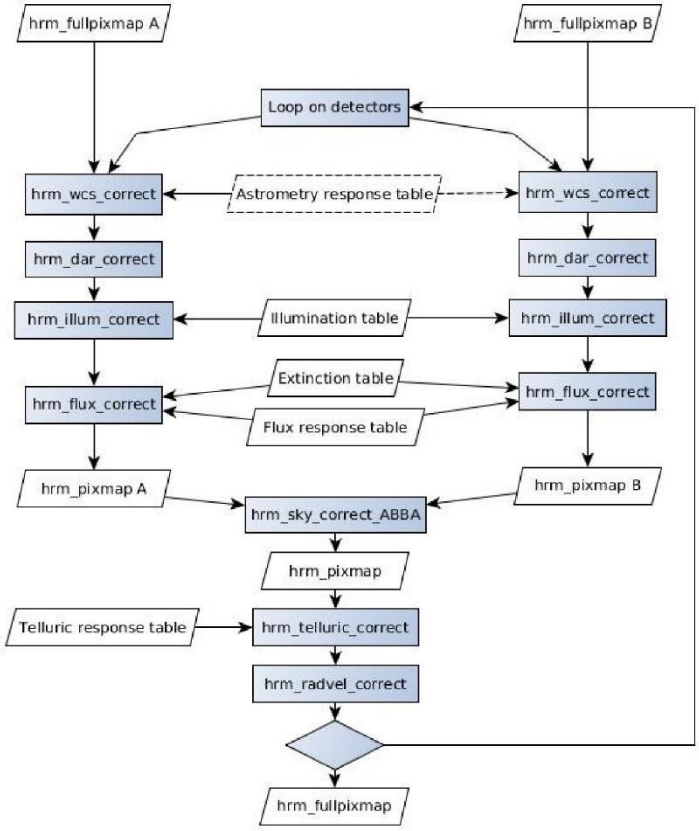
hrm_astrometry



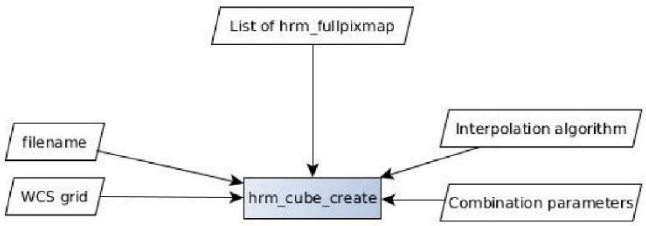
hrm_pixmap_ins



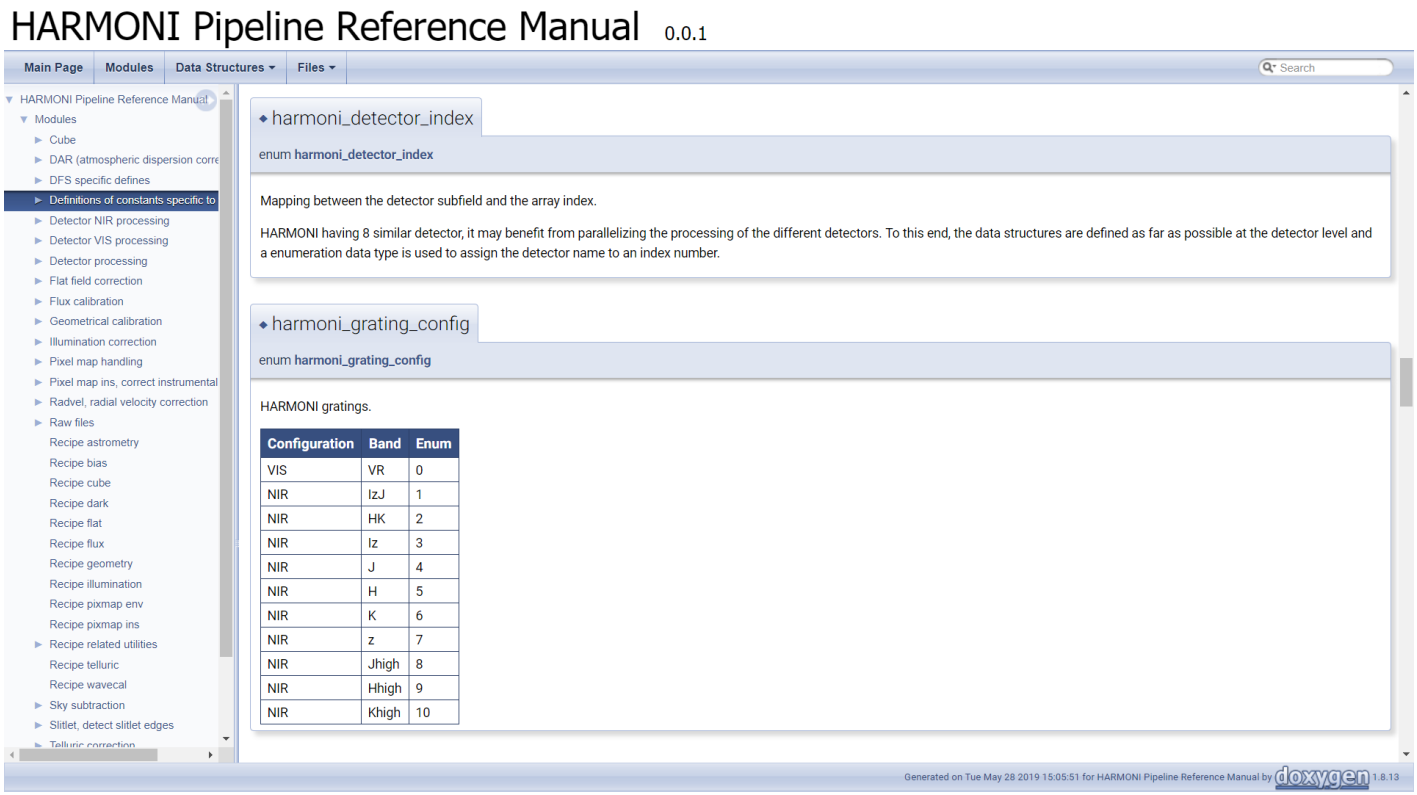
hrm_pixmap_env

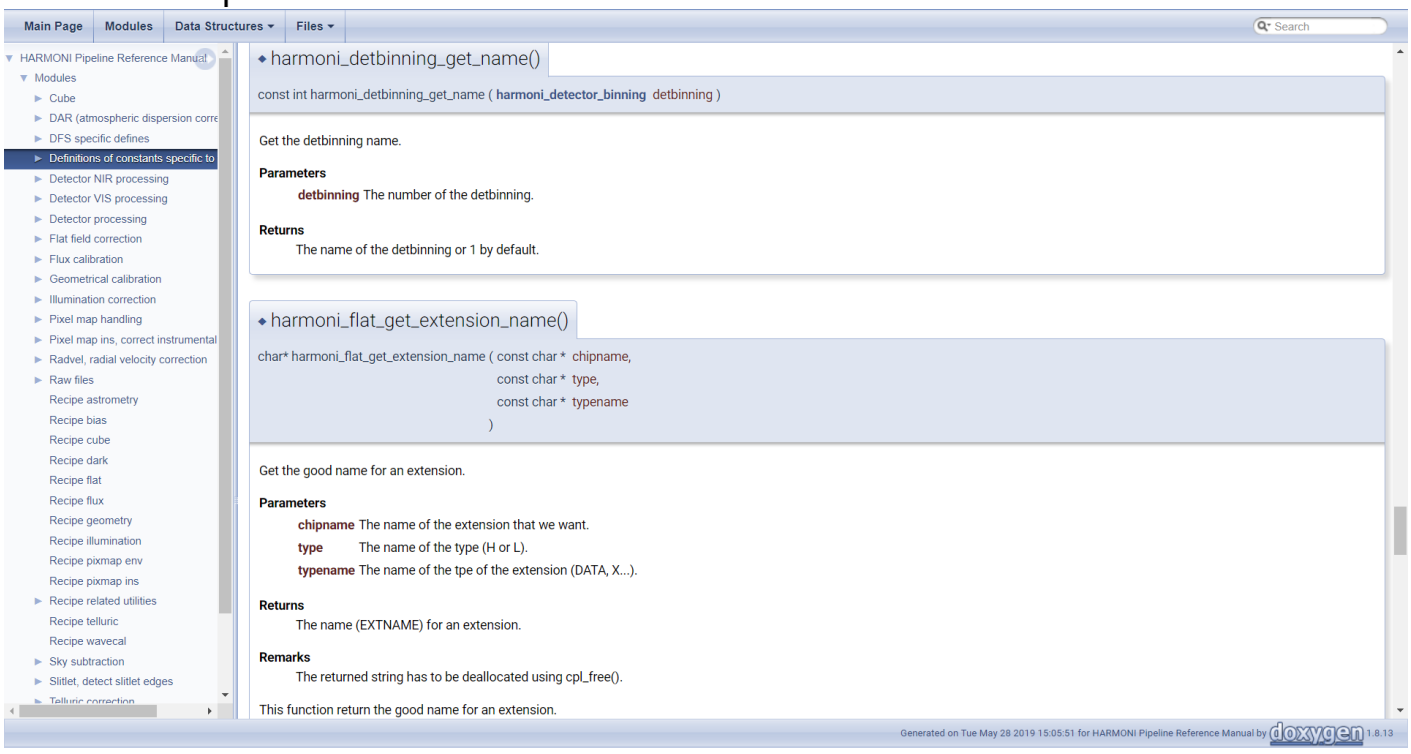


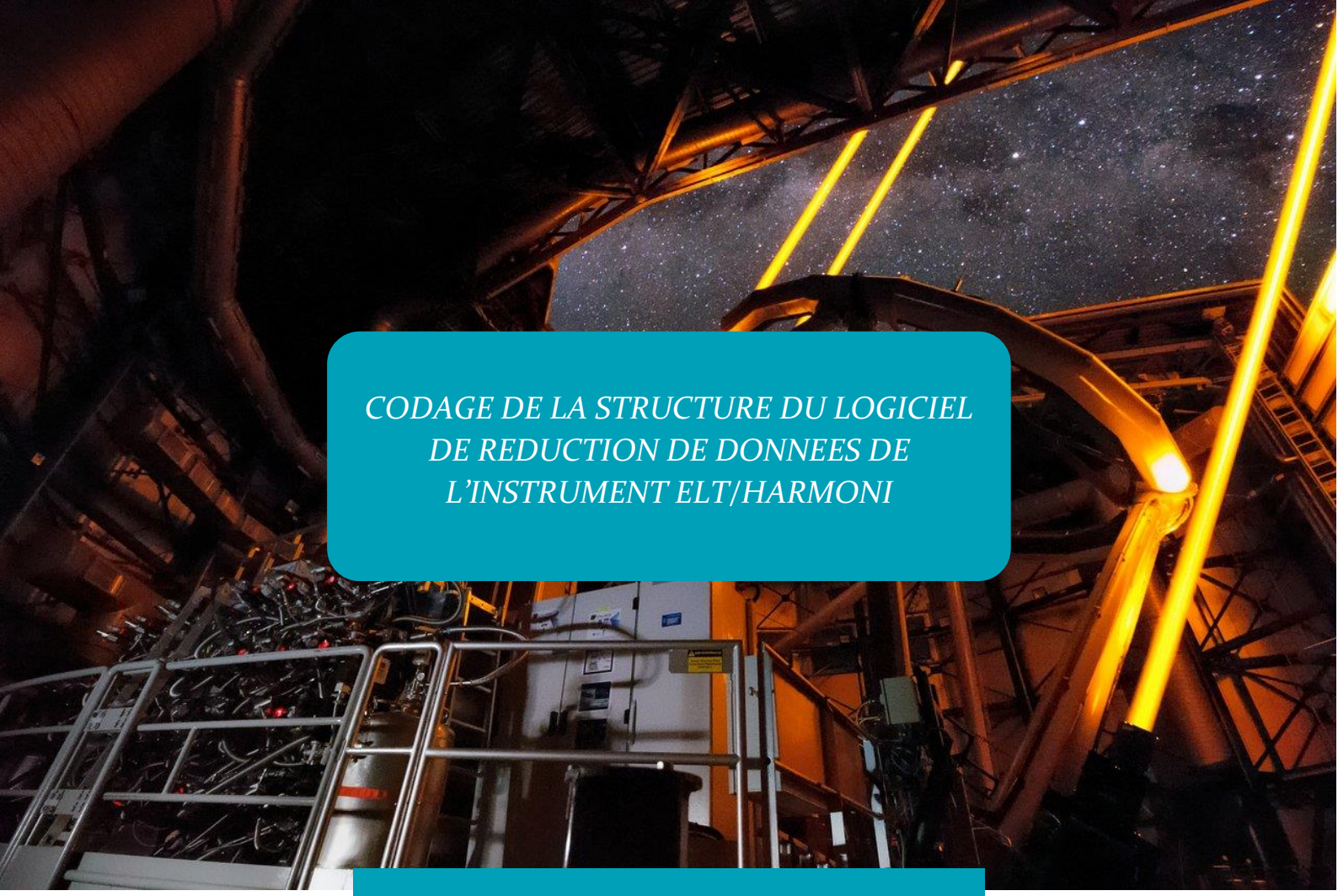
hrm_cube



Documentation du logiciel







CODAGE DE LA STRUCTURE DU LOGICIEL DE REDUCTION DE DONNEES DE L'INSTRUMENT ELT/HARMONI

L'OBJECTIF PRINCIPAL EST DE CODER LA STRUCTURE DU LOGICIEL NOTAMMENT LES ENTREES/SORTIES. CE LOGICIEL SERA LIVRE AVEC L'INSTRUMENT HARMONI A L'ESO ET IL A DONC DES CONTRAINTES DE DEVELOPPEMENT FORTES. LE TRAVAIL EST BASE SUR DES DIAGRAMMES D'ARCHITECTURE.

Lucien Burdet

DUT Informatique

08/04/2019 au 30/06/2019

UNIVERSITÉ DE LYON

IUT Lyon 1

l'excellence technologique

CRAL

CENTRE DE RECHERCHE ASTROPHYSIQUE DE LYON