Rapport de Stage

Ajout d'interfaces scriptables à des applications scientifiques

GROS Jean-Philippe

Stage du 04/05/2015 au 10/07/2015

Tuteur en entreprise : Laurent Bourgès

Tuteur académique : Julie Peyre

Établissement : UFR IM^2AG

Entreprise d'accueil : Institut de Planétologie et d'Astrophysique de Grenoble
414, Rue de la Piscine

38400 Saint-Martin d'Hères

Remerciements

Tout d'abord, je tiens à remercier mon maître de stage Laurent Bourgès qui a toujours su se montrer disponible durant mon stage. Il m'a également appris énormément de choses qui me serviront dans ma poursuite d'études.

Je souhaite également remercier Guillaume Mella qui m'a accepté pour ce stage et qui a été de très bons conseils tout au long de cette période.

Je remercie aussi Sylvain Lafrasse, et Gilles Duvert, membres du JMMC qui n'ont jamais hésité à répondre à mes questions

Et enfin je remercie l'IPAG pour m'avoir accueilli ainsi que le LABEX qui a subventionné ma gratification de stage.

Table des matières

1 Introduction

Dans le cadre de ma troisième année de Licence en Mathématiques et Informatique, j'ai effectué un stage d'une durée de 10 semaines. Il s'est déroulé du 4 Mai au 10 Juillet à l'Institut de Planétologie et d'Astrophysique de Grenoble (IPAG), au sein de l'équipe du centre Jean-Marie Mariotti (JMMC).

Le sujet de mon stage était, tout d'abord d'ajouter une interface en ligne commande puis d'évaluer une interface scriptable dans un cas pratique, le logiciel OIFitsExplorer.

Dans la première partie de ce rapport, je présenterai le laboratoire de recherche ainsi que le contexte global du projet. Dans la seconde partie, j'expliquerai comment j'ai réalisé le travail qui m'a été demandé ainsi que les moyens mis en œuvre pour y parvenir . Dans la troisième partie je dresserai un bilan personnel de cette expérience.

Ma motivation a été, dès le début du stage, de me perfectionner en programmation, en effet je souhaite me réorienter en master Informatique et rattraper mon retard par rapport aux étudiants de licence Informatique pure voire prendre de l'avance sur certains points.
Ce stage m'a permis d'évoluer dans une équipe composée de professionnels en informatique.

2 Présentation

2.1 l’IPAG :

L'IPAG, est un laboratoire de recherche de l'Université Grenoble Alpes et CNRS, spécialisé en astrophysique et planétologie. Il a été créé le premier Janvier 2011 suite à la fusion du LAOG[1] et du LPG[2]. Cet institut est constitué de 52 chercheurs permanents, 36 ITA dont 6 administratifs, 29 doctorants et post-doctorants, L'IPAG est le deuxième plus grand laboratoire de l'OSUG[3], qui dirige plusieurs instituts en rapport avec les Sciences de la Terre, de l'Univers et de l'Environnement .
La majeure partie des activités de l'IPAG est l'exploitation scientifique d'instruments d'observation (NAOS, WIRCAM, SPHERE, GRAVITY, AMBER, PIONIER, etc.) dans le but d'équiper les plus puissants télescopes (imagerie directe ou interféromètres) dans le monde (VLT, VLTI, CFHT).

2.2 Présentation du JMMC

Crée en septembre 2000 par l'INSU[4], le JMMC est un réseau de laboratoires Français d'experts en interférométrie doté d'une équipe technique localisée à l'IPAG.

Le JMMC fournit des logiciels et des services afin d'aider les utilisateurs à obtenir et exploiter les données fournies par les interféromètres et leurs instruments.

Parmi les interféromètres, on retrouve le VLTI[5] composé de quatre télescopes principaux appelés UT[6] mais également quatre télescopes auxiliaires déplaçables appelés AT[7].

Le JMMC coordonne (centralise) les efforts des laboratoires partenaires afin de fournir aux utilisateurs un environnement logiciel complet pour l'exploitation des équipements interférométriques disponibles.
Le rôle du JMMC est divisé en 3 parties :

- Développer produire et documenter les programmes nécessaires à l'utilisation des équipements interférométriques, en particulier le VLTI.
- Former les utilisateurs extérieurs au JMMC.
- Participer aux éventuels nouveaux projets sur des équipements interférométriques.

La principale activité du JMMC est le développement logiciel.

Le JMMC est doté d'une équipe technique composée de 3 ingénieurs : G. Mella, L. Bourgès, S. Lafrasse. Cette équipe est localisée à Grenoble qui accueille également le directeur, Gilles Duvert. Elle est chargée du développement logiciel et assure le bon fonctionnement des services. C'est au sein de cette équipe que j'ai effectué mon stage.

Les principaux logiciels développés par l'équipe technique sont :
- ASPRO2 est un programme Java qui produit une interface graphique et dynamique afin de préparer les observations.
- SearchCal est un outil de recherche d'étoiles de calibration.
- LITpro est logiciel servant à ajuster des modèles sur les données interférométiques.

1 Laboratoire d'Astrophysique de Grenoble
2 Laboratoire de Planétologie de Grenoble
3 Observatoire des Sciences de l'Univers de Grenoble
4 Institut National des Sciences de l'Univers
5 Very Large Telescope Interferometer
6 Unit Telescope
7 Auxiliary Telescope

- OIFitsExplorer est un logiciel d'analyse et de visualisation des données interférométiques,
- OiDB[8] est un service créé dans le but de fournir aux astronomes un portail et une base données des observations mondiales.

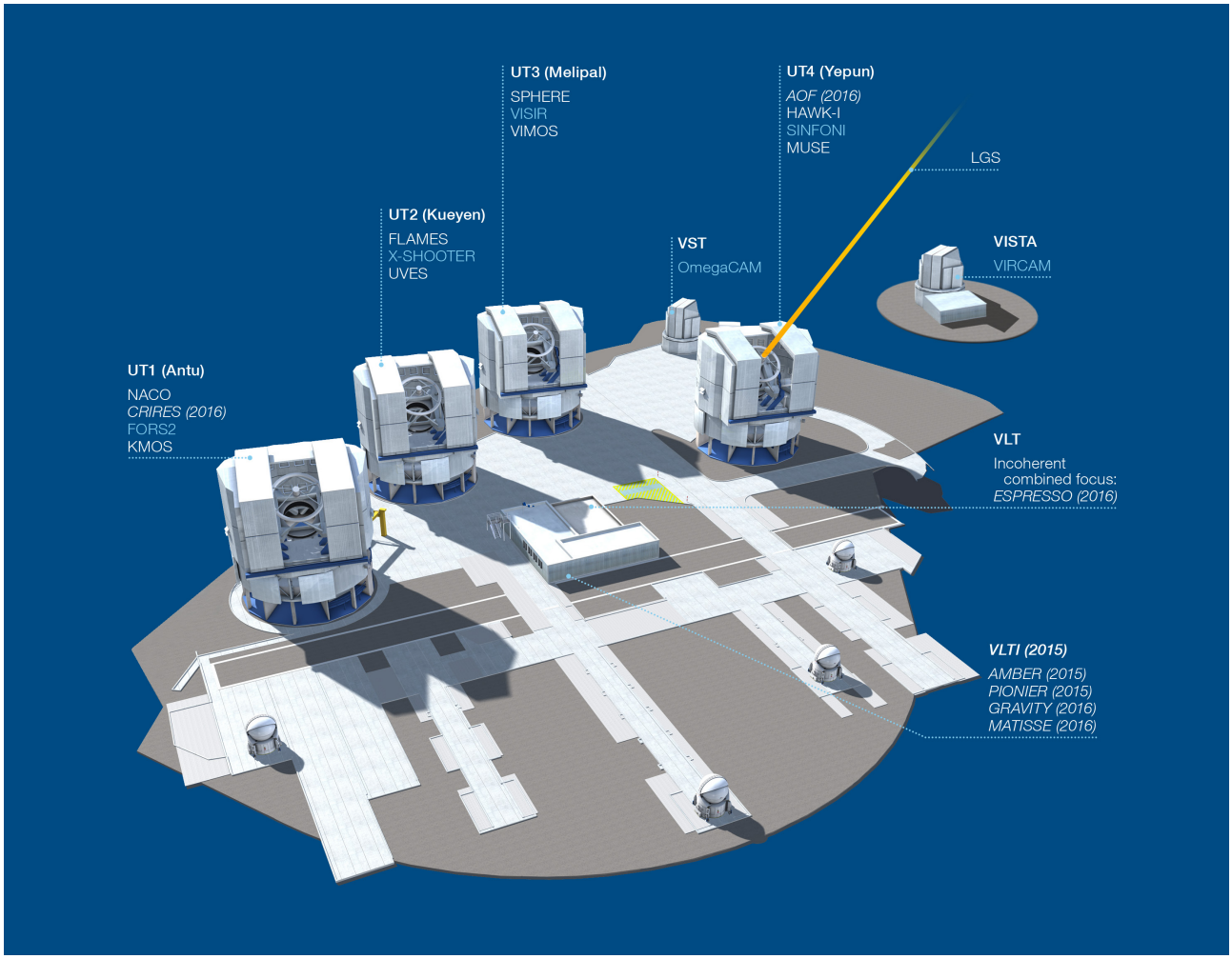


Schéma représentant le fonctionnement du VLTI avec ses 4 UT et ses 4 AT.

2.2.1 L'interférométrie

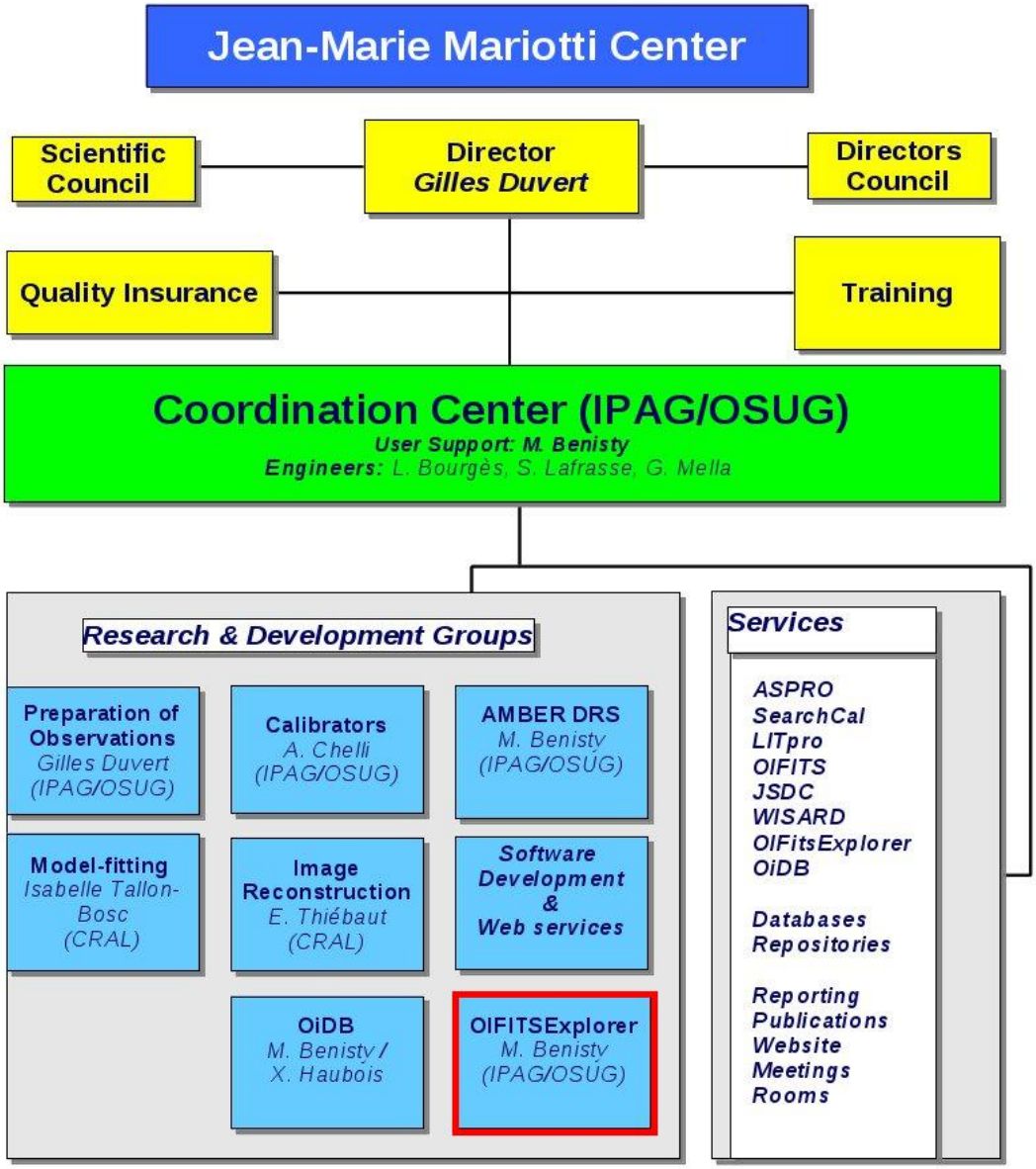
C'est une technique d'observation différente des méthodes d'imagerie directe. L'interférométrie permet d'obtenir des informations et caractéristiques des étoiles observés avec les meilleures résolutions angulaires disponibles à ce jour.
Cette technique utilise plusieurs télescopes simultanément pour combiner la lumière reçue et obtenir des franges d'interférence dans les instruments. Les observables sont des visibilités et des clôtures de phases (transformée de Fourier de l'objet).

2.2.2 Calibrateurs

Lors de l'observation interférométrique d'un objet astronomique, les mesures doivent êtres corrigées des effets perturbateurs produits par la turbulence atmosphérique ou instrumentale. L'observation de l'objet de science doit être accompagné de l'observation d'une étoile de référence voisine, appelée calibrateur avec des caractéristiques connues (diametre angulaire).

8 Optical inferferometry Database

Organigramme du JMMC ainsi que les logiciels développés :

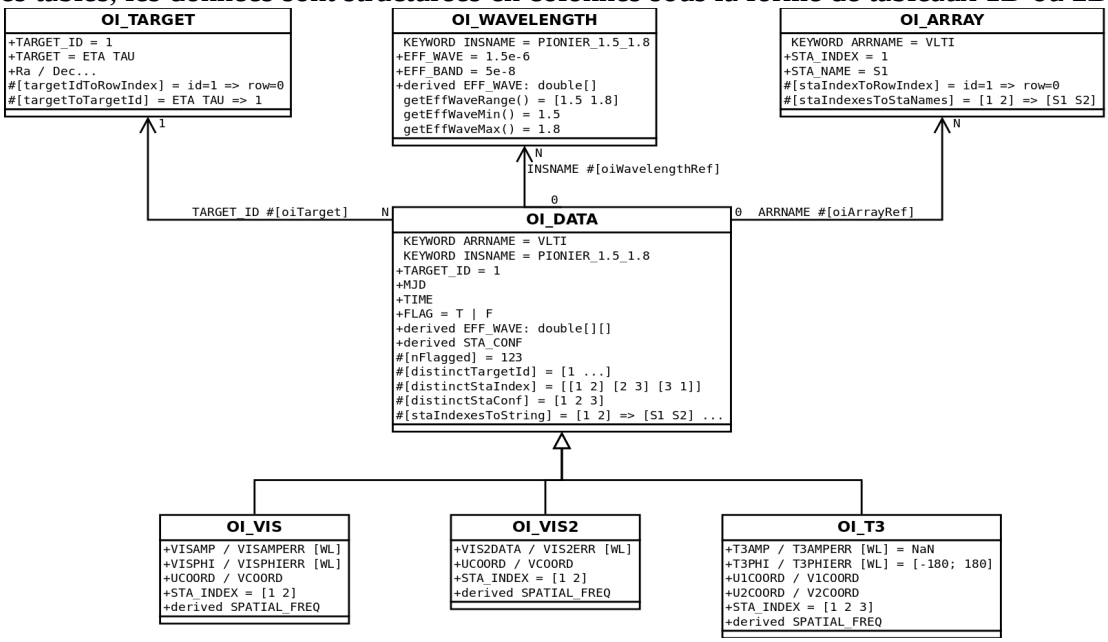


Encadré en rouge OIFitsExplorer le logiciel qui a fait l'objet de mon stage.

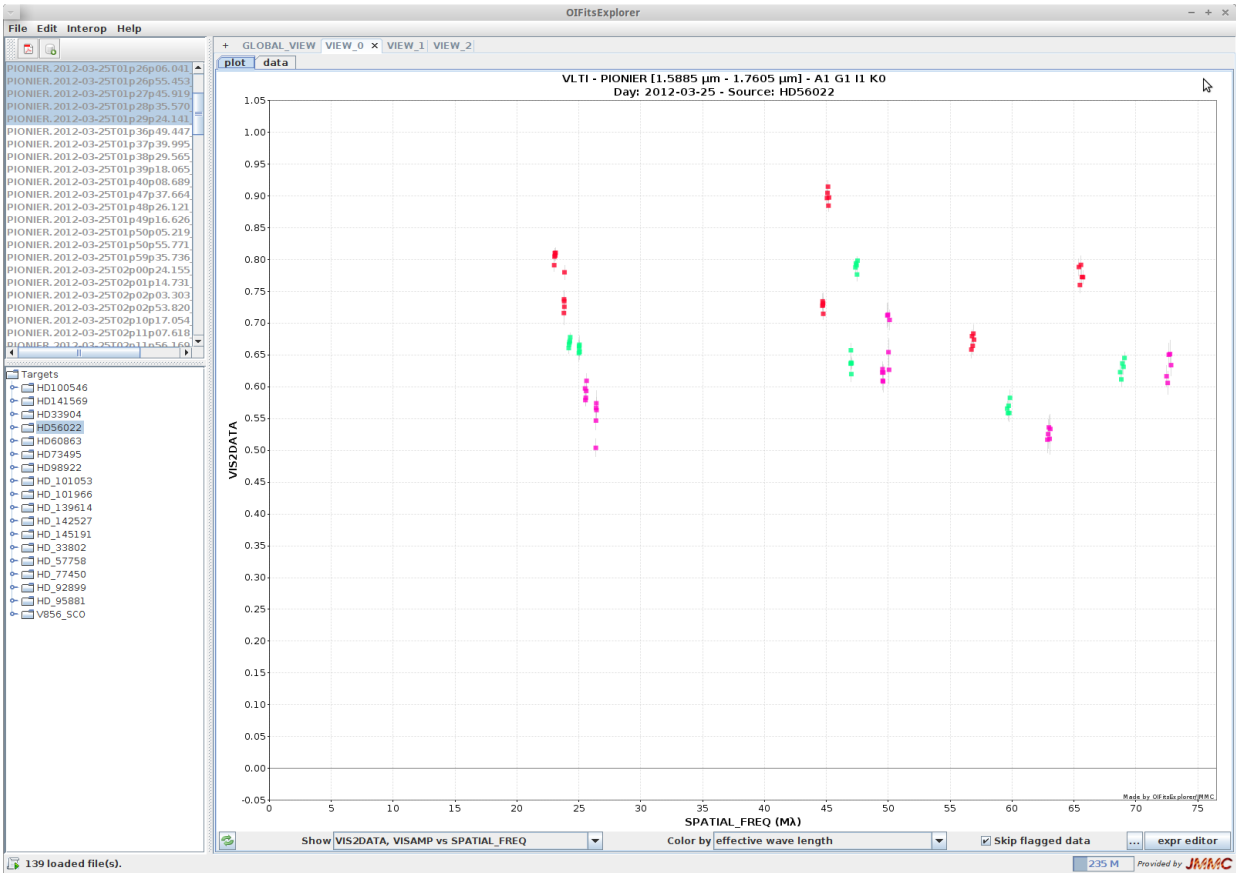
2.3 OIFitsExplorer

Les données réduites et calibrées sont stockées sous la forme de fichiers binaires FITS[9] utilisant le standard OIFits[10]. Ce standard décrit les tables de données en interférométrie optique dans le diagramme ci-dessous. Pour le moment, il existe 3 tables de mesures : OIVis, OIVis2 et OIT3.

Dans ces tables, les données sont structurées en colonnes sous la forme de tableaux 1D ou 2D.



Impression d'écran du logiciel OIFitsExplorer.



9 Flexible Image Transport System
10 OIFITS: A Data Exchange Standard for Optical (Visible/IR) Interferometry

OIFits Explorer est un outil du JMMC qui permet de visualiser les données issues des observations interférométriques au format OIFITS.
Les grandes fonctions permettent à l'utilisateur de charger plusieurs fichiers OIFITS, créer plusieurs graphiques et sélectionner les données à représenter et également d'exporter les graphiques en divers formats.
La session courante peut être sauvegardée sous forme de fichiers XML au format OIFitsExplorer collection (extension ,oixp), appelée par la suite collection.

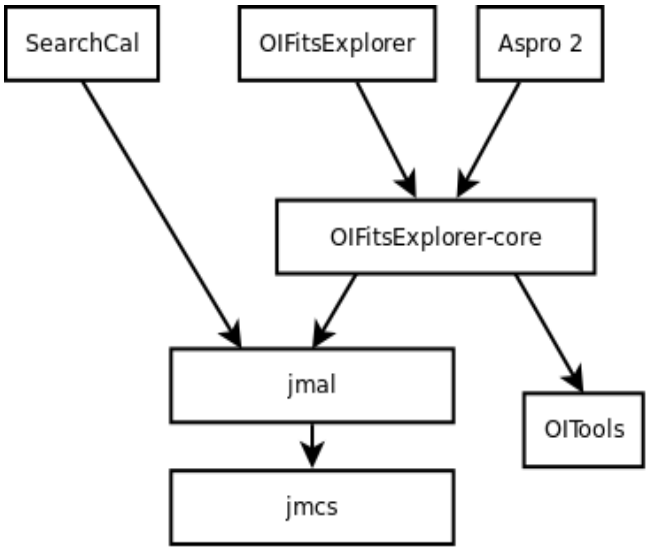
2.3.1 Existant/ Fonctionnalités :

Jusqu'à présent, l'utilisateur chargeait la collection souhaitée, et pouvait afficher plusieurs graphiques gérés par des vues (VIEW) organisées en onglets. Il pouvait sélectionner les données de graphiques (plot) mais également choisir les axes du graphique ainsi que les couleurs des points affichés et également voir les données tabulaires brutes. L'export des graphiques était limité au seul format PDF et uniquement vue par vue.

Une autre fonctionnalité du logiciel est d'effectuer des calculs sur les colonnes des tables de données (tel que les fréquences spatiales, ...) et les afficher dans les graphiques. Ces calculs sont effectués en java (non modifiable) et les résultats sont disponibles sous la forme de colonnes calculées.

2.4 Architecture logicielle

Les logiciels du JMMC ont une architecture commune et utilisent des librairies tierces et modules JMMC.



jMCS[11] est un framework qui a pour but de centraliser l'ensemble des fonctions générique aux applications GUI (interfaces graphiques, préférences, fonctions utilitaires...) il gére l'interface bas niveau pour que les développeurs puissent la réutiliser sans avoir à repartir de zéro. Il s'intègre le

11 Java Mariotti Common Software

plus possible aux différents systèmes d'exploitation (Linux, Windows, Mac OS X)

Jmal est une librairie Mathématique qui regroupe du code pour l'utilisation de modèles interférométriques ou plus généralement des traitements spécialisés en astronomie.

OITools est une librairie dédiée à la lecture de fichiers OIFits et génère des structures de données qui sont utilisés par OIFitsExplorer.

OIFitsExplorerCore est le module commun permettant de gérer les graphiques, utilisé par OIFitsExplorer et Aspro 2.

OIFitsExplorer est le module principal du logiciel OIFitsExplorer avec ses composants spécifiques.

J'ai dû intervenir en particulier sur jMCS afin de modifier le cycle de vie de l'application, et les modules OIFitsExplorerCore et OIFitsExplorer.

2.5 Description de l'environnement de travail :

Pendant toute la durée du stage j'ai participé aux activités du JMMC, à savoir la programmation, mais également la participation aux réunions pour faire le point sur l'avancement des différents projets en cours au sein du JMMC.

2.5.1 Outils utilisés :

Le premier outil mis à ma disposition a été Netbeans qui est un IDE[12], il m'a servi à programmer en Java. Netbeans propose des fonctionnalités qui m'ont également beaucoup servi tel que la gestion de code en version avec subversion (SVN).
SVN permet de développer du code en équipe, il est possible de récupérer le travail effectué (update) mais également de 'pousser' les modifications apportées (commit). J'ai moi-même été amené à utiliser l'update au tout début de mon stage afin de récupérer le code source. Lorsque j'avais finalisé et testé certaines parties du projet j'ai soumis mon code (commit) pour que tous les membres de l'équipe puissent le récupérer. Cette étape de commit peut paraître anodine mais elle simplifie beaucoup la vie des programmeurs puisque SVN stocke et renseigne l'ensemble des modifications en version. Elle demande également de la rigueur et un esprit de synthèse car il faut être capable faire un résumé très bref sur les modifications apportées. Subversion permet également de voir les différences apportées dans le code et si besoin est, de revenir à une version précédente ou annuler les modifications.
Netbeans supporte Maven, un outil crée par apache permettant la gestion et l'automatisation des projets logiciels Java (compilation, déploiement). Maven utilise pour chaque projet un fichier POM.xml qui est un fichier de configuration .
Un outil très utile et également fournit par Netbeans est le générateur de diagramme de classes (EasyUML). Il permet de voir rapidement l'architecture, les liens entres les différents objets et méthodes.
L'éditeur fournit une assistance soutenue et permet de naviguer très rapidement entre les fichiers sources. Par exemple l'outil de recherche FIND USAGE, accessible directement en effectuant un clic droit m'a fait gagner beaucoup de temps. Lorsque on a du mal à se repérer dans l'arborescence, et que l'on veut trouver où se situe un fichier ouvert, on peut utiliser l'outil Select in projects.
Il y a également de nombreux raccourcis disponibles sous Netbeans tel que le très connu Ctrl+f qui permet de faire une recherche selon un champ texte. Mais un raccourci qui m'a particulièrement

12 Integrated Development Environment (Environnement de développement)

aidé, est Ctrl + O, il permet d'accéder directement à un type désigné.
Un autre outil très pratique est le Design GUI SWING, il permet d'ajouter des composants en les sélectionnant directement.
Cette liste des fonctions proposées par Netbeans est non exhaustive et il en existe encore beaucoup à découvrir.

Tout au long du stage j'ai mis à jour une page wiki. Ces pages sont très utilisées dans l'équipe car ellls permettent de prendre de notes rapidement et sont visibles par tous les membres du JMMC.

L'équipe utilise très régulièrement l'outil Trac. Cet outil permet de poster des courtes notes appelés tickets qui sont des taches plus ou moins longues à accomplir, il peut s'agir de bugs à fixer ou d'améliorations à effectuer. Ces tickets sont visibles par toute l'équipe. Une fois qu'un des membres de l'équipe a résolu un ticket, il modifie son état. Ce changement ferme l'incident dans la base de données et prévient automatiquement les autres développeurs du changement apporté.

L'organisation de l'équipe technique est différente de l'organisation utilisant le cycle en V.
L'équipe s'organise de manière agile, car il n'y a pas de dates butoirs définies pour l'ensemble des sous-taches. Cela s'explique par le fait que les projets répondent directement aux besoins des chercheurs et utilisateurs. Le découpage en sous-taches se fait au fur et à mesure des discussions et de l'étude des besoins. Il n'y a donc pas de diagramme de Gantt, mais des cycles itératifs. Cette manière de procéder implique une très forte communication entre les membres de l'équipe, pour se faire ils ont à leur disposition différents outils tel que les mails, réunions ou le wiki ainsi que trac.

2.5.2 Rappel du sujet du stage

Les missions confiées ont été :
- Ajout d'interfaces en ligne de commande permettant d'exécuter un traitement spécifique sans utiliser l'interface graphique. Le cas pratique de ce stage a été de pouvoir exporter des graphiques dans un document.

- Ajout d'interfaces scriptables à des applications scientifiques. L'objectif est de permettre à l'utilisateur d'adapter le logiciel à ses besoins. Ce point a fait l'objet d'une étude pour utiliser le langage Python, connu des scientifiques. Le cas pratique de ce stage a été de pouvoir calculer de nouvelles quantités physiques à partir d'une expression donnée par l'utilisateur.

3 Réalisation :

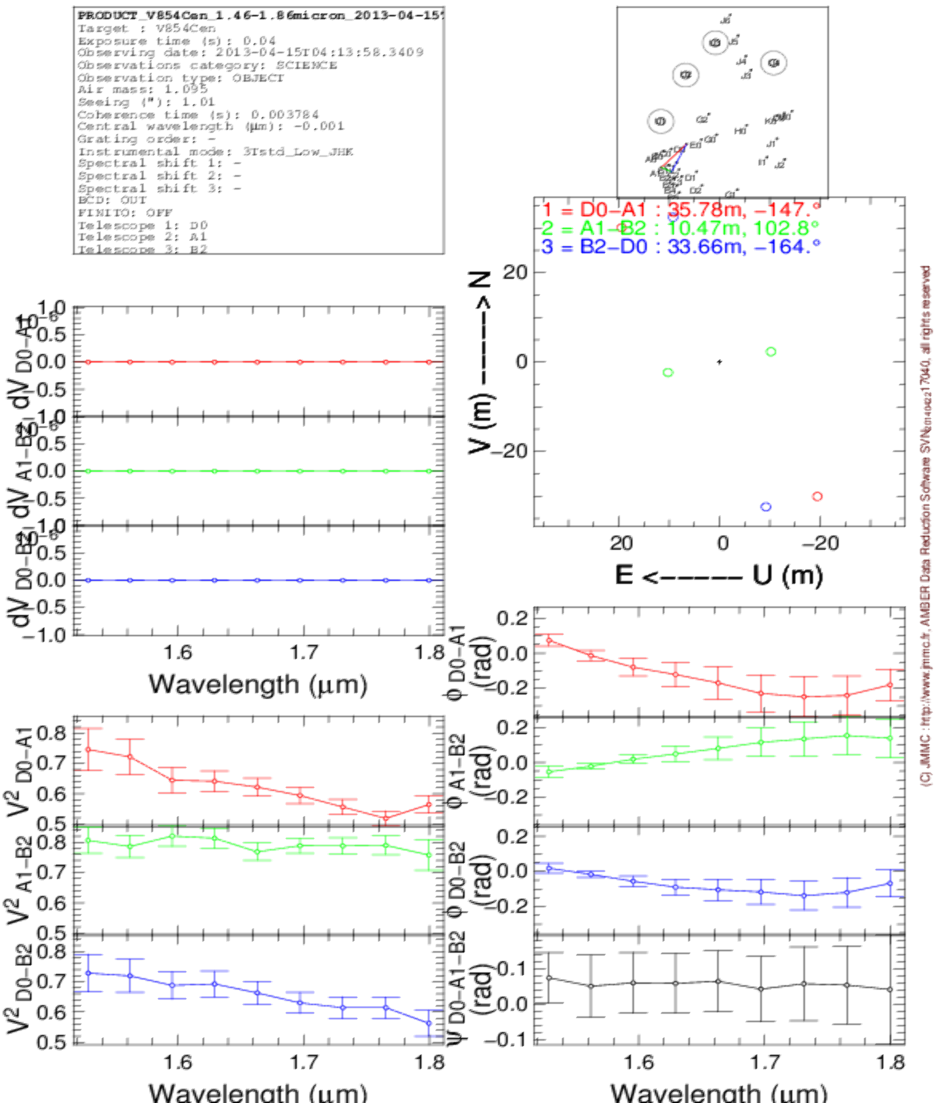
Les deux parties ont volontairement été séparées en deux parties distinctes. Je vais tout d'abord présenter la partie CLI[13] puis la partie interprétation d'expressions.

3.1 Solutions apportées :

Le besoin des chercheurs était de pouvoir exporter sur la même page tous les graphiques à l'aide de la ligne de commande, voir l'image ci-dessous.
La solution apportée a été d'ajouter une vue globale permettant de positionner tous les composants graphiques sur une même page.
Les utilisateurs voulaient également pouvoir exporter aussi en format image (PNG ou JPG).
Cet aspect devait comporter des options facultatives tel que le mode d'exportation ou le choix des

13 Commande Line Interpretation

dimensions dans le cas d'une exportation en format image (PNG ou JPG).



3.2 Exportation des graphiques :

3.2.1 Phase de faisabilité

Cette étape a débuté par une phase de tests, en effet il fallait vérifier si l'exportation en ligne de commande serait possible, étant donné que l'on utilise des interfaces graphiques dans ce logiciel. Ce test s'est effectué avec l'exportation d'un fichier PDF vide en ligne de commande sans affichage graphique (mode console TTY). Ensuite j'ai testé l'exportation d'un fichier PDF de plusieurs pages à partir de plusieurs graphiques. Puis, j'ai effectué le test d'export de la vue globale seule.
Le résultat de ce test s'est révélé être positif, nous avons donc pu continuer l'exportation en ligne de commande avec davantage d'options.

3.2.2 Vue globale

La solution trouvée est de créer une vue globale (classe GlobalView). Une fois les données chargées, cette vue globale a pour but de réunir tous les graphiques des différentes vues sur cette même vue globale.
Cette classe est constituée d'une méthode addChart(JFreeChart) et removeChart(JFreeChart) qui permettent d'ajouter ou supprimer un graphique de cette vue globale. Ces deux méthodes ont besoin de la méthode refreshJPanel() qui a pour but de mettre en page les graphiques dans la vue globale.
Cette classe est fournie à la fin du rapport dans l'annexe.

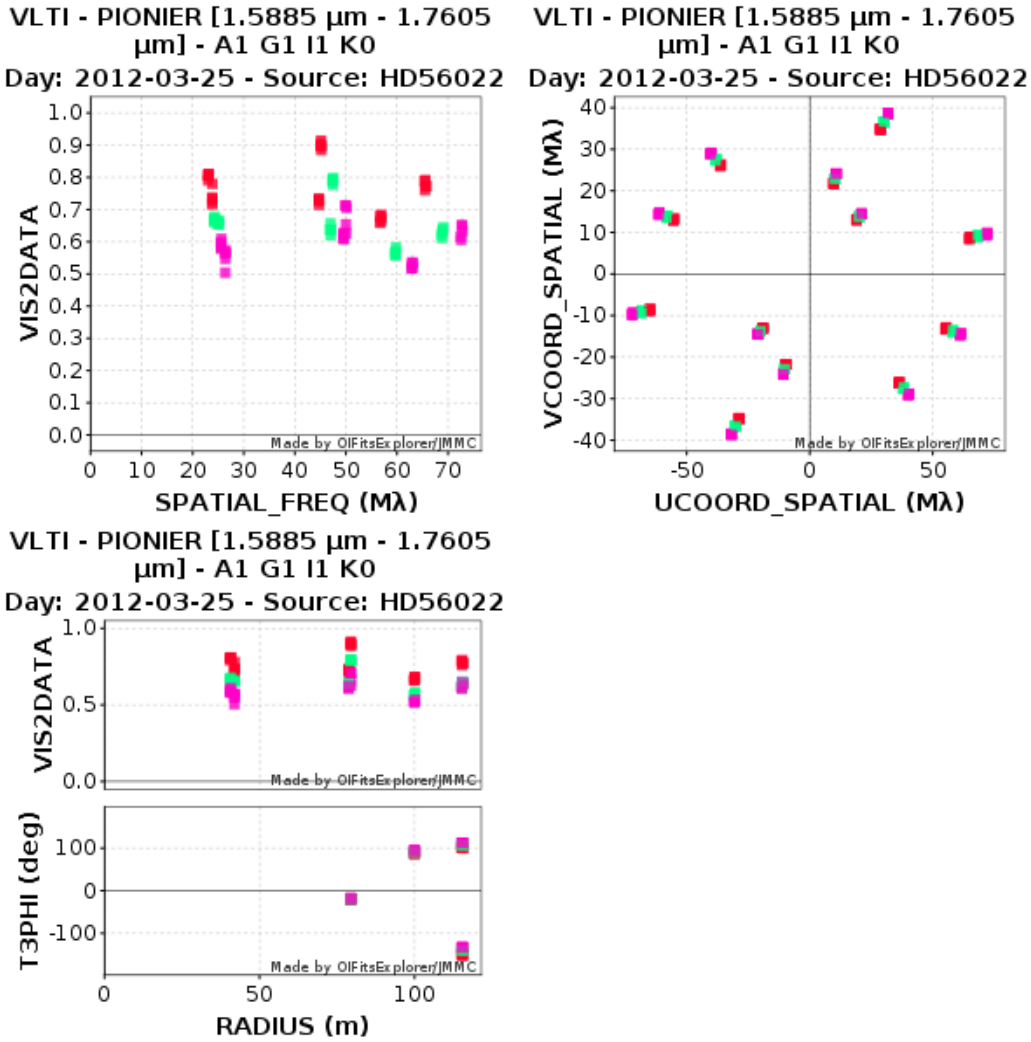


Image représentant le résultat d'un export.

3.2.3 Interfaces génériques :

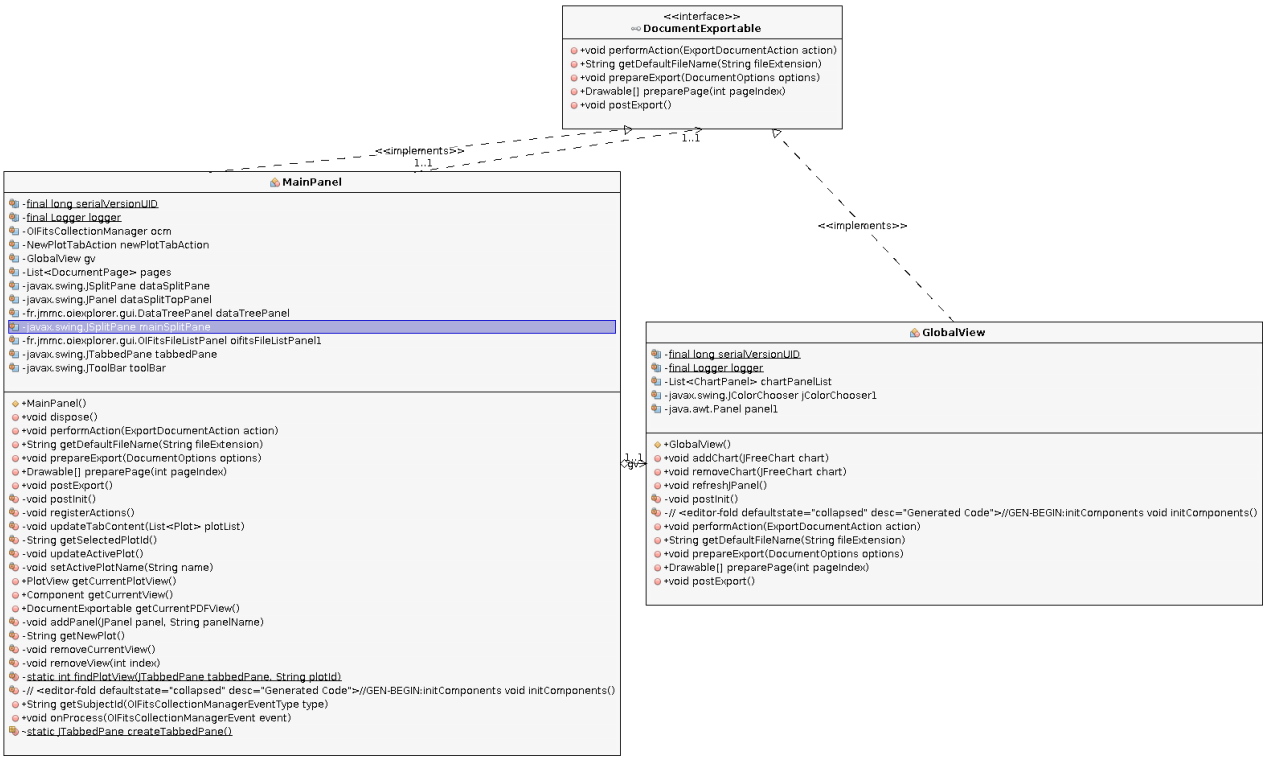
Les chercheurs ont également émis le besoin de pouvoir exporter d'autre composants que des graphiques sur la vue globale. Il était donc nécessaire d'apporter plus de généricité au logiciel afin de laisser la possibilité d'exporter ces autre types de composants.

Au moment de l'exportation de la vue globale, GlobalView implémente la méthode preparePage() de l'interface DocumentExportable qui fournit la liste des graphiques (type JFreeChart) en un tableau de Drawable (interface parente).

Cela permet d'exporter la même chose que ce qui est affiché à l'écran. (approche WYSIWYG).

```
public Drawable[] preparePage(final int pageIndex) {
        final int nCharts = chartPanelList.size();
        final JFreeChart[] charts = new JFreeChart[nCharts];

        for (int i = 0; i < nCharts; i++) {
            charts[i] = chartPanelList.get(i).getChart();
        }
        return charts;
    }
```

Code expliquant le fonctionnement de la méthode preparePage()



Ce diagramme, crée avec easyUML de NetBeans montre les dépendances de GlobalView. Elle implémente l'interface DocumentExportable. GlobalView est également utilisée dans la classe MainPanel.

3.2.4 Ajout options exportation

Afin de me familiariser avec le logiciel OIFits Explorer, j’ai amélioré l’interface graphique du programme, c'est-à-dire ajouter des options d’exportation dans les menus.
Parmi ces améliorations, il y a désormais le mode d'exportation, ce mode offre 3 possibilités différentes. Le mode ‘multi-page’ permet d'exporter sur plusieurs pages les différents graphiques (plus la page de la vue globale) et le mode 'single-page' permet d’exporter tous les graphiques sur une seule page.
Dans le cas d'une exportation en format image (PNG ou JPG) une option de dimensions était à

prévoir dans le but de pouvoir gérer soi-même la hauteur ainsi que la largeur en pixels de son image.

La plus grosse partie de cette approche en ligne de commande a été la gestion des arguments, c'est une partie qui a demandé de modifier du code dans jMCS.

La première méthode abstraite, ajoutée dans la classe App que je vais vous présenter est defineCustomCommandLineArgumentsAndHelp().
Elle est implémentée dans la classe OIFitsExplorer, pour définir les nouveaux arguments disponibles pour l'exportation en ligne de commande. Pour se faire elle renseigne les attributs de la classe ArgumentDefinition pour toutes les options possibles à savoir PDF, PNG, JPG, MODE, DIMS :

```
protected void defineCustomCommandLineArgumentsAndHelp() {
         addCustomCommandLineArgument(ARG_PDF, true, "export plots to the given
file (PDF format)",
                App.ExecMode.TTY);
         addCustomCommandLineArgument(ARG_PNG, true, "export plots to the given
file (PNG format)",
                App.ExecMode.TTY);
         addCustomCommandLineArgument(ARG_JPG, true, "export plots to the given
file (JPG format)",
                App.ExecMode.TTY);
           addCustomCommandLineArgument(ARG_MODE, true, " export mode [multi|
single] page");
         addCustomCommandLineArgument(ARG_DIMS, true, " export image dimensions
[width,height]",
                App.ExecMode.TTY);
    }
```

La deuxième modification a été de créer ces nouveaux arguments dans une classe appelée ArgumentDefinition :

```
public final class ArgumentDefinition {

    private final String name;
    private final boolean hasArgument;
    private final App.ExecMode mode;
    private final String help;
…
}
```

L'attribut 'name' renseigne le nom de chaque argument.
hasArgument permet de déterminer si un argument a besoin d'une valeur.
App.ExecMode détermine le mode d'exécution, en ligne de commande, on sera en mode TTY sinon on sera en mode graphique (GUI).
'help' permet de donner une description de chaque argument disponible, cette description sera affichée à l'utilisateur s'il demande à voir l'aide ou s'il entre une mauvaise commande.

3.2.5 Modification du cycle de vie

Une autre grosse partie de ce travail a concerné le cycle de vie de l 'application.
Cette gestion s'opère toujours dans jMCS et est gérée par Bootstrapper, cette classe implémente des méthodes d'initialisation qui s'appuient sur App et ses classes dérivées.

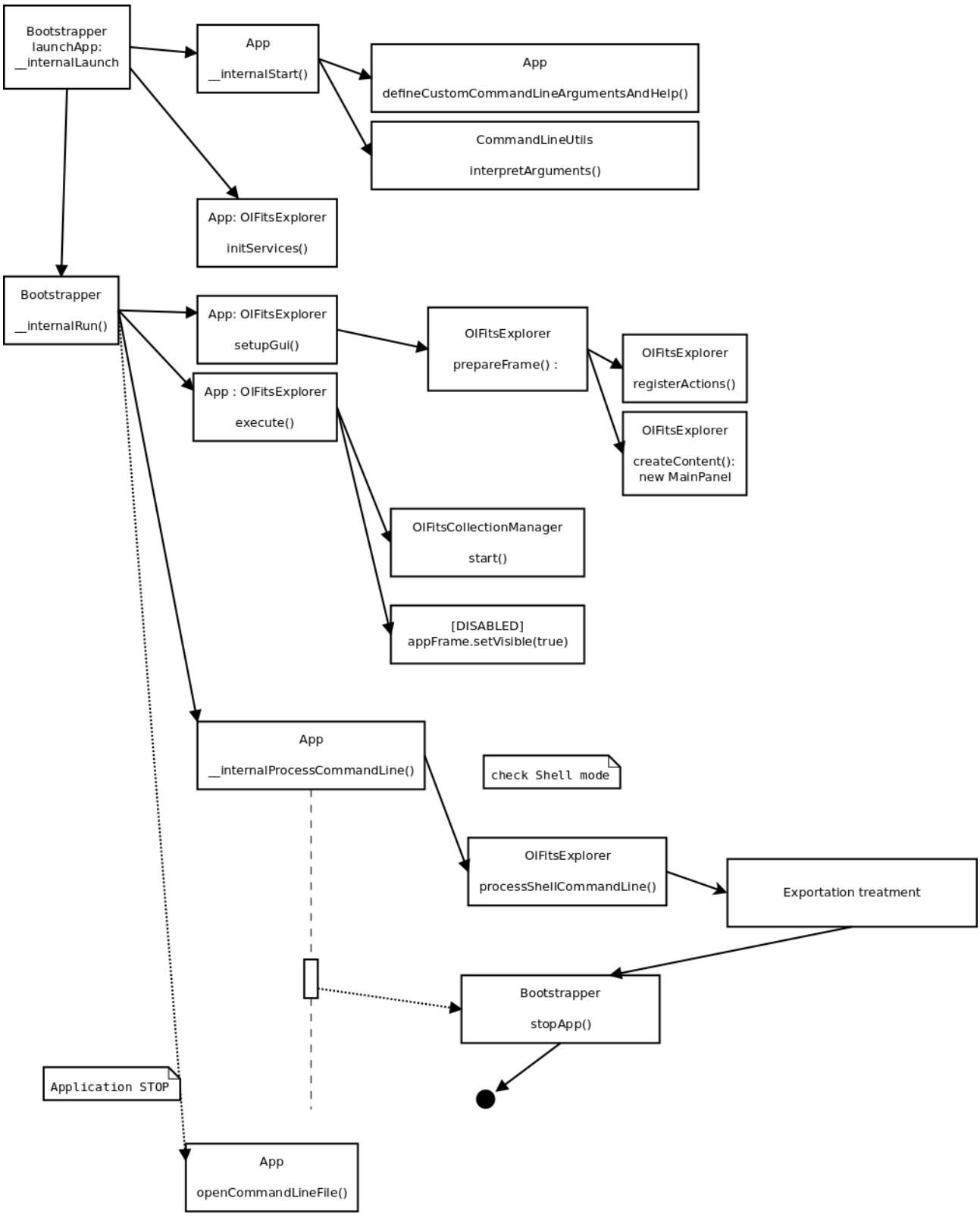
Tout d'abord, Bootstrapper va lancer __internalRun(), cette méthode va utiliser __internalStart() qui est une méthode de App dont l'utilité est de définir et interpréter les arguments de la ligne de commande.
Ensuite, __internalLaunch() va appeler __internalRun(), une autre méthode de Bootstrapper.
Cette méthode a trois utilités principales, elle permet de lancer l'initialisation de l'interface Gui grâce à setupGui().
Et enfin, __internalRun() utilise __internalProcessCommandLine() qui permet le traitement de la ligne de commande grâce à processShellCommandLine().
Dans le cas d'une erreur dans __internalProcessCommandLine(), une exception est lancée et le programme est arrêté de manière sûre.

La méthode abstraite processShellCommandLine() est implémentée dans la classe OIFitsExplorer. Cette méthode permet de récupérer les potentiels arguments et de lancer le chargement des données puis réaliser l'export si les arguments sont corrects.

```
protected void processShellCommandLine() throws IllegalArgumentException {
        final Map<String, String> argValues = getCommandLineArguments() ;
 //Récupération des arguments
       [...]

        final File fileOpen = new File(fileArgument);

        // same checks than LoadOIDataCollectionAction:
        if (!fileOpen.exists() || !fileOpen.isFile()) {
              throw new IllegalArgumentException("Could not load the file: " +
fileOpen.getAbsolutePath());
        }

        final String pdfFile = argValues.get(ARG_PDF);
        // Pour toutes les valeurs d'arguments disponibles, on récupère la
//valeur si elle existe
        [...]

        try {
            boolean doLoad = false;

            if (pdfFile != null) {
                doLoad = initializeExport(pdfFile, MimeType.PDF, mode, dims);
 //si l'argument pdf existe, on initalise l'exportation et doLoad vaut vrai
            }
[...]
            if (doLoad) {
                ExportUtils.loadDataAndWaitUntilExportDone();
            }
// si doLoad est vrai, c'est qu'au moins un des arguments disponibles a
//été entré et on peut donc charger les données.
       [...]
```

Voici le diagramme expliquant la partie cycle de vie de l'application du point de vue du

Bootstrapper.



Le point délicat de cette partie a été la réutilisation de l'interface graphique. En effet pour exporter les graphiques, on utilise les composants Swing de l'interface graphique même en ligne de commande. Il y a beaucoup de vérifications pour que l'affichage des composants Swing ne se fasse jamais (setVisible(true)) sinon le programme aura une erreur et s'arrêtera.

Remarque : Une version complète du diagramme est disponible dans les annexes.

Ces modifications d'architecture ont été nécessaires pour contrôler plus précisément l'application . La gestion du cycle de vie en est devenu plus complexe.

3.2.6 Processus d'export

Un autre point important du code est le moment où l'action d'export est lancée dans la classe ExportDocumentAction :

```
    public void process(final DocumentExportable exportable, final File
proposedFile,
            final DocumentOptions cmdOptions) {

        final File file;
        if (proposedFile == null) {
            // GUI mode [...]
        } else {
            // command line mode [...]
        }

            // prepare file export (layout and options):
            final DocumentOptions docOptions =
DocumentOptions.createInstance(mimeType);
          [...]

            // Perform Layout and get options from component:
            exportable.prepareExport(docOptions);
            try {
               [...]
                    if (cmdOptions != null) {
                        // overwrite options with given command line params:
                        docOptions.merge(cmdOptions);
                    }

            Writer.getInstance(mimeType).write(file, exportable, docOptions);
           [...]
            } catch (IOException ioe) {
                MessagePane.showErrorMessage("Could not write to file : " +
file.getAbsolutePath(), ioe);
            } finally {
                // post file export: restore Chart state if modified:
        exportable.postExport();
            }
        }
    }
```

prepareExport() a pour but de préparer les composants à exporter en fonction du mode sélectionné (SINGLE, MULTI ou DEFAULT).

merge() est une méthode permettant d'écraser les options mises par défaut dans la variable docOptions à partir de cmdOptions. merge() s'occupe de la gestion d'erreur dans le cas où

cmdOptions manque d'options (par exemple s’il n'y a pas de format d'exportation). Cette façon de fonctionner permet d'avoir des arguments par défaut en cas d'options non renseignées.

Writer.getInstance(mimeType).write(file, exportable, docOptions);
Cette ligne permet de récupérer le mimeType du format d'exportation (PDF, PNG ou JPG) et d'appeler la méthode write() en fonction de ce mimeType. Or write() est une méthode abstraite de la classe Writer. En fonction du mimeType trouvé, c'est la méthode write() d'une implémentation de Writer qui sera appelée (PDFWriter, ImageWriter).

postExport() est mise dans un bloc finally car on veut qu'elle soit appelée même en cas d'exception.
Cette méthode permet de réinitialiser les composants.

3.3 Refactoring

3.3.1 Intérêt de cette mise en place

Pour clarifier le code, il a été nécessaire de changer la hiérarchie à certains endroits, en effet auparavant l'exportation ne se faisait qu'en PDF, il a fallu réajuster les classes afin que l'exportation se fasse en fonction des options de chaque format de fichier. Par exemple l'exportation en PDF ne prend pas en compte la largeur ou la hauteur entrée par l'utilisateur.

3.3.2 Exemples d'utilisation

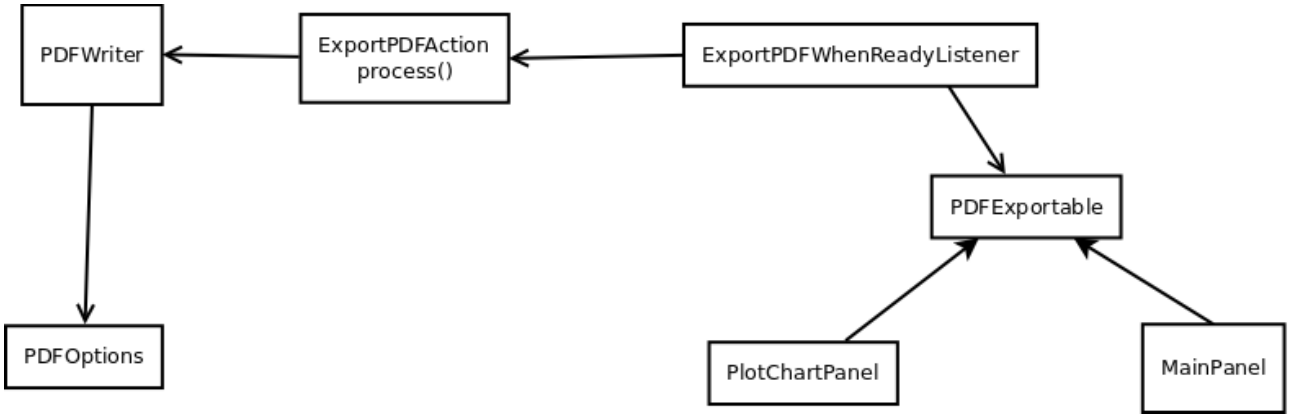
La solution a donc été de créer une classe regroupant les options communes des deux modes d'exportation (PDF ou image) et deux classes filles qui possèdent les options propres à leur format. Le tout en implémentant le plus possible les méthodes dans la classe mère afin d'éviter de dupliquer du code. Idem pour la classe qui se chargeait d'exporter le fichier, elle a été repensée de la même façon que pour les options avec une classe mère et deux classes filles.

3.3.3 Différences apportées

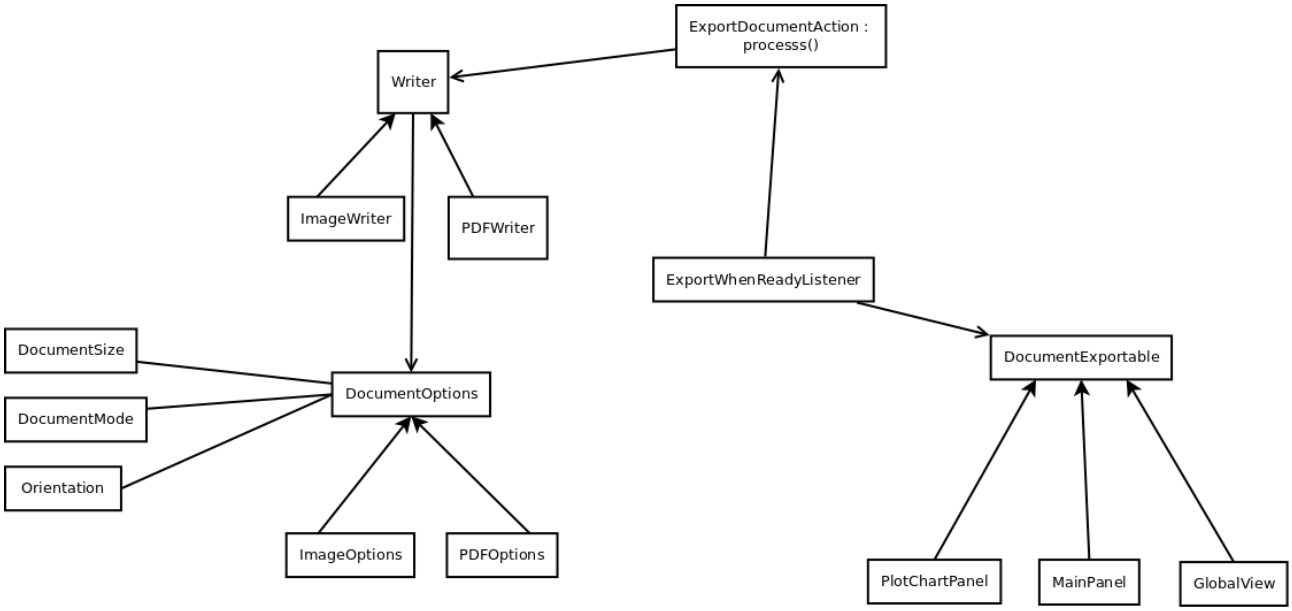
Au départ, les classes et méthodes étaient prévues uniquement pour exporter en PDF comme en témoigne leurs noms attribués.

Afin de pouvoir exporter en PNG (et en JPG) il a fallu penser à un modèle à partir de l'architecture déjà existante.

3.3.3.1 Diagramme représentant l'ancien fonctionnement :



3.3.3.2 Fonctionnement actuel :



Après la modification, on se rend compte grâce au diagramme ci-dessus que l'architecture a beaucoup changé. On remarque d'une part qu'il est dorénavant possible d'exporter en différents formats (image ou PDF). Il est également plus facile d'ajouter d'éventuels autres formats.

En effet, cette partie du code est implémentée en objet, une modification ne modifierai que DocumentOptions et Writer. Pour Document, il faudrait créer une classe qui hérite de DocumentOptions et qui utilise ses attributs. Attributs qu'il sera peut-être nécessaire de mettre à jour afin de traiter le nouveau format.

Pour Writer, il faudra également créer une classe dérivée de Writer et implémenter dans cette nouvelle classe les méthodes d'exportations associées au nouveau format.

Pour ce qui est de la partie export, c'est l'interface DocumentExportable qui s'en occupe.

Étant donné que DocumentExportable est une interface, ce sont les implémentations (PlotChartPanel, GlobalView...) qui implémentent chaque méthode en fonction du contexte. Ce qui montre encore une fois l'aspect générique de cette partie du logiciel.

Pour ce qui est des méthodes déclarées par DocumentExportatble, on trouve principalement prepareExport() qui sert à préparer les composants à exporter en fonction des modes indiqués (Single, Multi ou Default). postExport() permet de signaler que l'export est fini et que le composant peut être réinitialisé. Et enfin preparePage() sert à récupérer le tableau de Drawable nécessaire pour l'exportation des composants. Exemples de code :

```
public enum DocumentMode {

    SINGLE_PAGE, MULTI_PAGE, DEFAULT;

    public static DocumentMode parse(final String value) {
        if ("multi".equals(value)) {
            return MULTI_PAGE;
        }
        if ("single".equals(value)) {
            return SINGLE_PAGE;
        }
        if ("default".equals(value)) {
            return DEFAULT;
        }
        throw new IllegalArgumentException("Invalid mode: " + value);
    }
```

Cette classe permet de renseigner les différents modes d'exportation à savoir SINGLE MULTI et DEFAULT.

```
public interface DocumentExportable {

    /**
     * Export the component as a document using the given action:
     * the component should check if there is something to export ?
     * @param action export action to perform the export action
     */
    public void performAction(final ExportDocumentAction action);

    /**
     * Return the default file name
     * @param fileExtension  document's file extension
     * @return default file name
     */
    public String getDefaultFileName(final String fileExtension);

    /**
     * Prepare the page layout before doing the export:
     * Performs layout and modifies the given options
     * @param options document options used to prepare the document
     */
    public void prepareExport(final DocumentOptions options);

    /**
     * Return the page to export given its page index
     * @param pageIndex page index (1..n)
     * @return Drawable array to export on this page
     */
    public Drawable[] preparePage(final int pageIndex);

    /**
     * Callback indicating the export is done to reset the component's state
     */
    public void postExport();
}
```

Cette interface gère l'aspect exportation.

3.4 Améliorations possibles

Actuellement le logiciel ne donne pas la possibilité d'exporter autre chose que des plots, il serait possible d'exporter d'autre types d'objets tel que des extractions de données ou statistiques grâce à l'implémentation de l'interface Drawable.

Il serait intéressant de pouvoir laisser la possibilité à l'utilisateur de charger directement les fichiers OIFits qu'il souhaite traiter.

Ajout d'un nouveau format d'exportation, le SVG[14]. Ce format est utilisé pour les images vectorielles. Grâce au refactoring, ce format s’ajouterait facilement en créant une classe SVGWriter dérivée de Writer.

14 Scalable Vector Graphics

3.5 Création de scripts en Jython

3.5.1 Qu'est ce que le Jython et pourquoi l'utiliser

Jython est un interpréteur de commandes Python écrit en Java.
Python est un langage que les scientifiques peuvent connaître ou maîtriser, c'est pour cette raison que nous avons choisi d'utiliser la librairie Jython pour nos logiciels écrits en Java .

L'objet de cette étude a été de vérifier la possibilité d'utiliser ce moteur Jython afin d'interpréter des expressions mathématiques. En effet, l'expression entrée par l'utilisateur sera directement exécutée par l'interpréteur Jython qui se chargera d'effectuer le calcul, une fois le résultat obtenu, le logiciel reprendra la main et affichera graphiquement les nouvelles données.

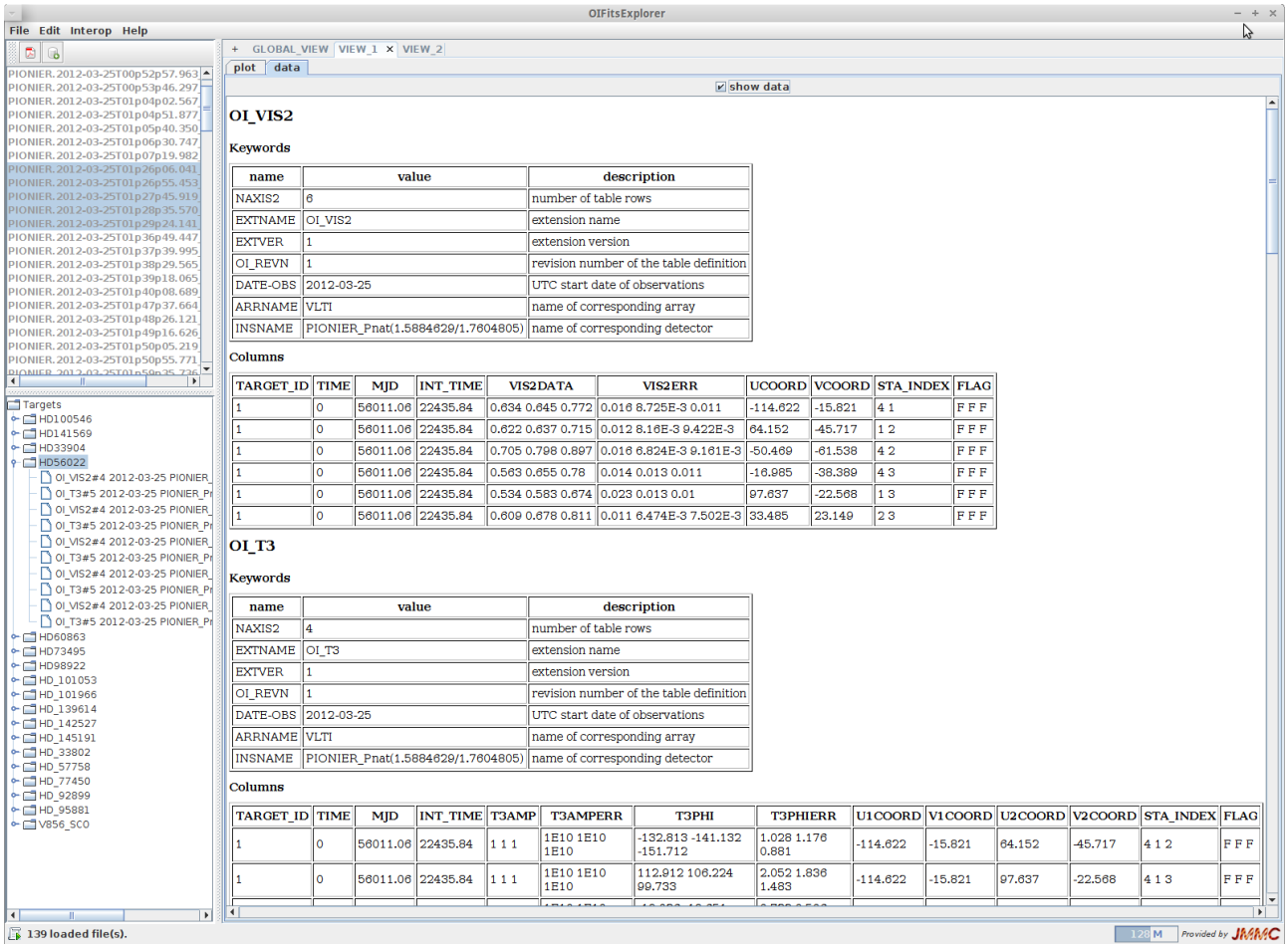
3.5.2 Phase de tests

Avant de pouvoir être sûr de pouvoir utiliser Jython dans notre contexte, il a fallu effectuer quelques tests.
Dans un premier temps il était nécessaire d'étudier si l'installation de packages optionnels était capitale. La question s'est posée notamment pour Numpy qui permet entre autres de faire des calculs directement sur des tableaux plutôt que sur des scalaires. Finalement l'installation de ce package a été écartée car il nécessitait l'installation de JyNI[15] qui est compatible avec peu d'OS existant et qui aurait posé un problème de portabilité. Or cette portabilité est l'un des avantages du langage Java car une fois le code écrit, il peut être utilisé partout.

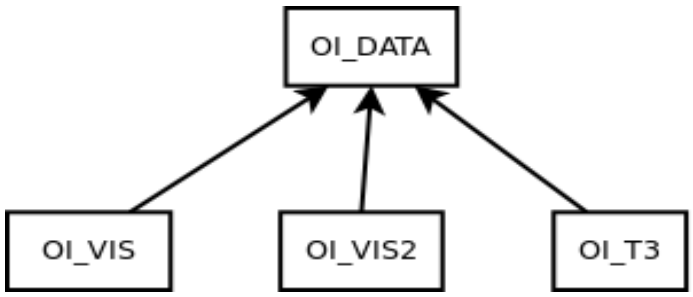
Un autre test a été d'implémenter une classe JythonCalc qui a permis de tester les obstacles et possibilités de Jython. Le principal obstacle a été la création de tableaux 2D. En effet la création de tableaux 2D a posé un problème de dimensionnalité dû au langage Python. Cela a nécessité de trouver le bon moyen de créer ces tableaux. Ce qui a été rendu possible grâce à PyArray, une classe permettant de passer des tableaux Java de toutes dimensions à l'interpréteur Jython.

OIFitsExplorer charge chaque fichier OIFITS en mémoire dans un objet OIFitsFile qui contient une liste de tables (OITable). Ces tables contiennent des mots clés et des colonnes de données.
Voir image ci-dessous.

15 Jython Native Interface



Les tables de données (OIVIS, OI_VIS2, T3) sont aussi des objets OIData qui possèdent des colonnes communes et des colonnes calculées en Java comme le montre le diagramme ci-dessous.



Chaque colonne possède des métadonnées qui donnent des informations sur la colonne. Les métadonnées se déclarent de cette façon : ColumnMeta (nom, type, dimensions, description...)

Il existait déjà des colonnes calculées par du code Java.
Par exemple :
"UCOORD / EFF_WAVE" = UCOORD_SPATIAL

Le but de cet étude a donc été d'étendre cette possibilité de calcul en python pour l'utilisateur et en rendant le code plus générique.

3.5.3 Travail effectué

La conception s'est déroulée pas à pas, au tout début l'expression à calculer était écrite en dur dans une méthode appelant directement la classe Jython et ensuite l'expression a été remontée jusqu'à être entrée directement par l'utilisateur en mode graphique, le fait de remonter cette expression a demandé la modification de l'architecture déjà existante.

Par exemple :
"VIS2DATA / VIS2ERR" = SNR

L'expression à calculer peut s'apparenter à une boite noire qui est gérée par Jython. Il fallait trouver un moyen de déterminer sur quelles tables effectuer les calculs, la solution a été de faire ces calculs sur toutes les tables présentes.

3.5.3.1 Classe JythonEval

C'est cette classe qui se charge de calculer l'expression, elle utilise la méthode eval() qui prend en paramètres une expression et une table pour retourner un tableau 2D. La table en question est la table de donnée sur lequel l'expression s'applique, le tableau retourné est le résultat obtenu.

Cette méthode eval() méthode a permis à l'utilisateur d'entrer des formules simples comme 'VCOORD / EFF_WAVE'
Sans ce travail d'interprétation, l'utilisateur aurait dû entrer tout le script Python qui gère ce calcul.

Par exemple, pour VCOORD / EFF_WAVE, il aurait dû entrer le script ci-dessous :

```
from array import array
from math import *

def user_func():
      return VCOORD / EFF_WAVE

for _i in range(_rows):
      for _j in range(_waves):

            EFF_WAVE = _input_EFF_WAVE[_i][_j]
            TIME = _input_TIME[_i]
            MJD = _input_MJD[_i]
            INT_TIME = _input_INT_TIME[_i]
            VIS2DATA = _input_VIS2DATA[_i][_j]
            VIS2ERR = _input_VIS2ERR[_i][_j]
            UCOORD = _input_UCOORD[_i]
            VCOORD = _input_VCOORD[_i]

            _output[_i][_j] = user_func()
```

On peut voir comment le script Jython fonctionne :
En gras, on déclare la fonction à partir de la formule de l'utilisateur.
Ensuite, le tableau résultat (_output) est rempli grâce à 2 boucles qui traversent les deux dimensions des tableaux de données (rows et wavelengths).
Dans ces boucles, on extrait les scalaires qui permettent d'effectuer les opérations de la fonction

user_func().
Dans notre exemple ci-dessus, il aurait été possible de récupérer uniquement VCOORD et EFF_WAVE mais comme nous ne voulons pas décrypter l'expression, utilisée dans user_func(), il faut déclarer tous les scalaires présents dans la table OIData donnée.

La méthode eval() utilise le moteur Jython qui se lance de cette façon :

```
System.setProperty("python.import.site", "false");
// Create an instance of the PythonInterpreter
PythonInterpreter interp = new PythonInterpreter();
```

Une fois l'interpréteur de commande créé, il est à présent possible de lui demander d'initialiser des variables, des fonctions ou d'exécuter des commandes.
Le but étant de calculer une expression pour un tableau, il est nécessaire de fournir des variables à l'interpréteur comme dans l'exemple ci-dessous :

```
varName = "_input_" + "EFF_WAVE";
interp.set(varName, new PyArray(double[][].class,
oiData.getEffWaveAsDoubles()));
```

Le tableau PyArray est créé à partir d'un tableau Java existant qui est dans ce cas retourné par oiData.getEffWaveAsDoubles(), il faut également préciser un nom au tableau (varName) ainsi que ses dimension et son type (double[][].class).

Pour ce qui est du fonctionnement de cette méthode eval(), elle déclare autant de variables Python qu'il y a de colonnes.
Étant donné qu'il est impossible d'effectuer les calculs sur les tableaux directement, il faut créer des scalaires pour effectuer ces calculs un à un. Une fois ces scalaires obtenus, on peut remplir le tableau de retour en utilisant user_func() qui contient l'expression mathématique.

Cette méthode eval() génère un script jython qui gère aussi bien des colonnes sous forme de tableau 1D que de tableaux 2D. Cette méthode est initialement prévue pour calculer des colonnes 2D. Dans le cas de colonnes 1D, il est facile de convertir le tableau en un tableau 2D.

Par exemple, dans ce morceau de code récupéré dans eval() :

```
if (column.isArray()) {
  // 2D:
  script.append("\t\t" + colName + " = " + varName + "[_i][_j]\n");
  break ;
  }

  // 1D on rows:
  script.append("\t\t" + colName + " = " + varName + "[_i]\n");
```

Cela montre un exemple de la gestion des tableaux 2D et 1D.

3.5.3.2 Modifications dans OIData

Création d'une méthode getExprColumnDoubles() qui permet d'obtenir le tableau de double 2D créé par le moteur Jython, cette méthode ne calcule le tableau qu'au premier appel (mode lazy).
Création de méthodes updateExprColumn() et removeExprColumn() qui permettent de supprimer

ou modifier une expression donnée. À noter que dans le cas de updateExprColumn(), il est nécessaire d'avoir le nom de la colonne à créer et que cette méthode vérifie si une colonne du même nom existe déjà ou non.
Des modifications ont également été apportées dans certaines méthodes afin de pouvoir prendre en charge les expressions de manière générique. En effet, le programme ne s'occupait que d'expressions bien particulières avant la modification.

Les méta-données des colonnes calculées sont gérées par la classe WaveColumnMeta. Pour pouvoir créer des colonnes en fonction d'une expression, il a été nécessaire d'ajouter un attribut expression dans la classe WaveColumnMeta, ainsi que son constructeur associé.

3.5.3.3 Gestion des erreurs et gestion des données

La gestion des erreurs a plusieurs intérêts, elle permet à la fois d'indiquer à l'utilisateur où se situe son erreur de saisie et aussi d'éviter les calculs pour les tables où l'expression n'est pas valide.

Un premier test a été de vérifier que les opérateurs de calcul étaient corrects, pour cela, il a suffi de vérifier que le calcul marchait pour une table donnée.
Il est également important de vérifier que si l'utilisateur rentre dans l'expression un nom de colonne erroné ou qu'elle n'existe pas dans cette table, le logiciel renvoie une erreur.

Cette vérification est plus compliquée car il n'y a pas toujours toutes les tables possibles (VIS, VIS2, T3), cela dépend des fichiers OIFits (et de l'instrument utilisé).

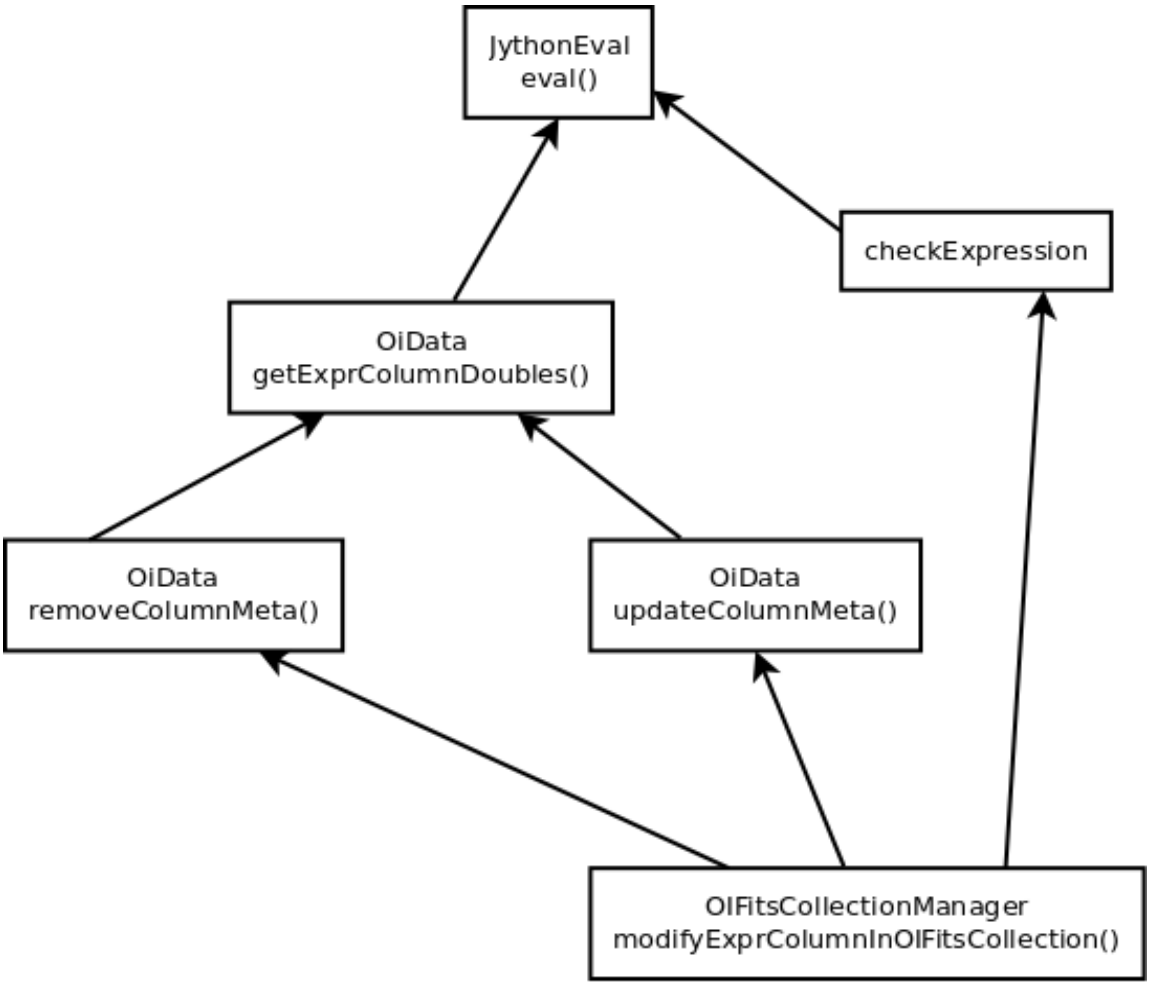
J'ai implémenté l'algorithme ci-dessous pour faire les vérifications :

```
 paramètres : String expression
                   Boolean remove (indique si on souhaite remove ou update
l'expression)
 int n = 0;
 int nBad = 0;
 int nOk = 0;
 OIVis vis = null;

 Pour tous les oifitsfile
      Si Vis non null
            Si oiFitsFile.possedeOiVis()
                  Vis ≤= oiFitsFile.getOiVis()[0]
                  n++
                  try{
                        vis.checkExpression(expression);
                                    ca_marche[0] = true;
                                    nOk++;
                  }
                  catch{
                         nBad++;
                  }

 // idem pour OIVis2 et OIT3
      si n = 3
            break
 Fin pour tout
 //une fois sorti de la boucle, on sait si l'expression est bonne ou non.
 // Si nOk égal zéro c'est qu'aucune table n'est valable le calcul ne sera pas
effectué.
 // Sinon on peut effectuer le calcul

 Pour tous les oifitsfile
      Pour tous les oiData
            Si remove
                  remove(expression)
            Sinon
                  Si ca_marche contient oiVis ou oiVis2 ou oiT3
                        update(expression)
 fin pour tout
```

3.5.3.4 Modifications d'affichage et gestion d'événements.

Création d'une classe UserExprEditor qui permet de créer un éditeur simple avec deux champs :
-Un champ pour l'expression.
-Un champ pour le nom.
-Un bouton pour modifier ou ajouter une expression
-Un bouton pour supprimer une expression

Ci-dessous un diagramme expliquant le fonctionnement de cette partie script en Jython :



Bilan :
Une des alternatives au Jython est JEL[16], c'est une librairie permettant d'interpréter des expressions en Java.
Quelques problèmes surviennent pour certaines expressions. Ces problèmes sont de l'ordre de l'interpréteur Jython. Par exemple, lorsque l'on entre l'expression sqrt(-1) en Jython, l'interpréteur retourne une exception. Alors que si le package Numpy est installé, cette même expression renvoie NaN[17].
Un autre problème de cet interpréteur est la gestion d'erreur, en effet dans certains cas l'interpréteur ne va pas être assez précis dans la description de la source de l'erreur ou même se tromper dans certains cas.
Par exemple, si on entre l'expression 'VIS2DATA / sqrt(-1) ' dans une collection contenant bien la table OIVIS2, l'interpréteur va renvoyer
« Bad expression 'VIS2DATA / sqrt(-1)' [global name 'VIS2DATA' is not defined] ».
Alors que le problème provient de la racine négative (sqrt(-1)). En fait quand l'interpréteur trouve une erreur, il renvoie la première variable entrée dans l'expression. Cette erreur peut se révéler être chronophage pour l'utilisateur dans la compréhension du fonctionnement du logiciel. En effet, si la mauvaise variable est ciblée il sera beaucoup plus difficile de trouver d'où provient l'erreur.
Un autre problème de cette librairie Jython est de la télécharger, celle-ci nécessite plus de 35Mb d'espace sur le disque dur. Il est vrai qu'avec les ordinateurs actuels ce n'est pas vraiment un

16 Java Expressions Library
17 Not a Number

problème, mais quand on sait que le logiciel actuel prend une place d'environ 10Mb. De plus Jython est utilisé pour une seule fonction du logiciel.
Et pour finir sur le bilan de Jython je vais aborder le point des performances.
Par exemple avec l'expression « VIS2DATA / VIS2ERR » (VIS2ERR étant la marge d'erreur de la colonne VIS2DATA). On obtient : « updateExprColumnInOIFitsCollection[VIS2DATA / VIS2ERR] computation time = 617.333235 ms. »
Le résultat est renvoyé en moins d'une seconde ce qui ne provoque pas de gène particulière chez l'utilisateur.

4 Bilan de cette expérience

4.1 Connaissances acquises à l'occasion de ce stage

Durant ce stage j'ai appris dans plusieurs domaines, tout d'abord j'ai été formé sur l'utilisation d'un IDE (NetBeans). Je suis maintenant capable de naviguer facilement grâce aux raccourcis et autres astuces. J'ai appris comment utiliser le debugueur. Et je sais me servir de plusieurs outils proposés par NetBeans tel que le refactor, pratique pour modifier du code automatiquement, je peux également me servir de l'outil d'analyse de différences entre plusieurs versions de code. C'est une partie très importante de la programmation car bien utiliser son IDE permet d'être plus efficace, on peut à la fois coder plus rapidement mais on peut également trouver des bugs plus facilement.

J'ai tout au long du stage, amélioré mes connaissances en programmation notamment en programmation objet à la fois dans les capacités à programmer que dans les stratégies sur la façon d'implémenter le code. Ce qui me permet de faire moins d'erreurs qu'avant et de générer un code plus lisible et plus robuste.
J'ai également apprit à réutiliser des librairies déjà créées au lieu de devoir repartir de zéro à chaque fois, c'est déconcertant aux premiers abords car il faut comprendre le code d'autres personnes mais c'est une méthode qui fait gagner énormément de temps.
Durant ce stage, j'ai souvent été amené à expliquer les modifications apportées lors de commits ou sur la page wiki pour des publics différents, il faut expliquer différemment selon si on s'adresse aux utilisateurs du logiciel ou si on s'adresse à l'équipe de développement. Cela m'a appris a être plus clair dans mes descriptions et dans les rédactions de commentaires dans le code.

4.2 Problèmes rencontrés

La première difficulté rencontrée a été de comprendre le fonctionnement du logiciel, en effet OIFitsExplorer est un logiciel qui est déjà très évolué et il comporte beaucoup de code. De plus, ce logiciel utilise des librairies qui sont implémentées en dehors du logiciel et il a fallu que je m'habitue à me servir de la documentation présente dans le code pour m'aider. L'utilisation de NetBeans (un IDE), m'a également beaucoup servi pour parcourir le code, il existe de nombreux raccourcis qui permettent de naviguer rapidement dans les classes.

Une autre difficulté a été de m'adapter aux méthodes de conception du code, étant donné que le programme est gros, il repose sur une centaine de classes, une trentaine de librairies et cinq modules, il faut penser à dupliquer le code le moins possible (voir pas du tout si possible), ce qui est déconcertant au début mais cela fait gagner beaucoup de temps lors du débugage et maintenance notamment.

Le travail demandé était lui-même également exigeant et cela m'a demandé beaucoup d'efforts pour

que je puisse comprendre les besoins par rapport à l'existant. Cette difficulté m'a appris à travailler avec plus de rigueur et à réfléchir avant de me précipiter dans le codage. Maintenant je suis capable de réfléchir à plusieurs solutions possibles et choisir laquelle est la plus optimale en fonction du besoin du client.

4.3 Manuel utilisateur

4.3.1 Ligne de commande :

Tout d'abord il faut exécuter le programme, pour cela il faut entrer cette ligne :
java -Djava.awt.headless=true -jar ./target/oiexplorer-TRUNK-jar-with-dependencies.jar -v 3
A savoir que l'option -v 3 indique le niveau de verbosité, le programme est fait pour accepter un niveau de verbosité allant de 0 à 5.
Ensuite, étant donné que nous sommes en mode console, il est nécessaire d'indiquer un fichier à ouvrir ainsi que le fichier à exporter.
Pour ce qui est du fichier à ouvrir, il se fait avec l'option -open ainsi que le nom du fichier souhaité, le logiciel OIFitsExplorer chargera alors les données si toutefois elles sont prises en charge.
Pour ce qui est du fichier à exporter, il faut renseigner le format d 'exportation, donc -pdf pour enregistrer en PDF, -png pour exporter en PNG et enfin -jpg pour obtenir du JPG dans touts les cas le format doit être précédé par un nom de fichier dans le répertoire souhaité. A savoir qu'ile st possible d'exporter plusieurs formats a la fois et que le logiciel corrige le nom du fichier si il contient une erreur dans l'extension.
Il est possible d'afficher l'aide en tapant -h ou -help, l'aide s'affiche automatiquement quand l'utilisateur entre de mauvais arguments.
Pour indiquer le mode d'exportation souhaité (single, multi ou default) il suffit d'entrer -mode suivi du mode souhaité.
Dans le cas de l'exportation d'une image on peut indiquer ses dimensions voulues (en pixels) en indiquant -dims ensuite il faut renseigner la largeur puis la hauteur séparées par une virgule (largeur, hauteur).

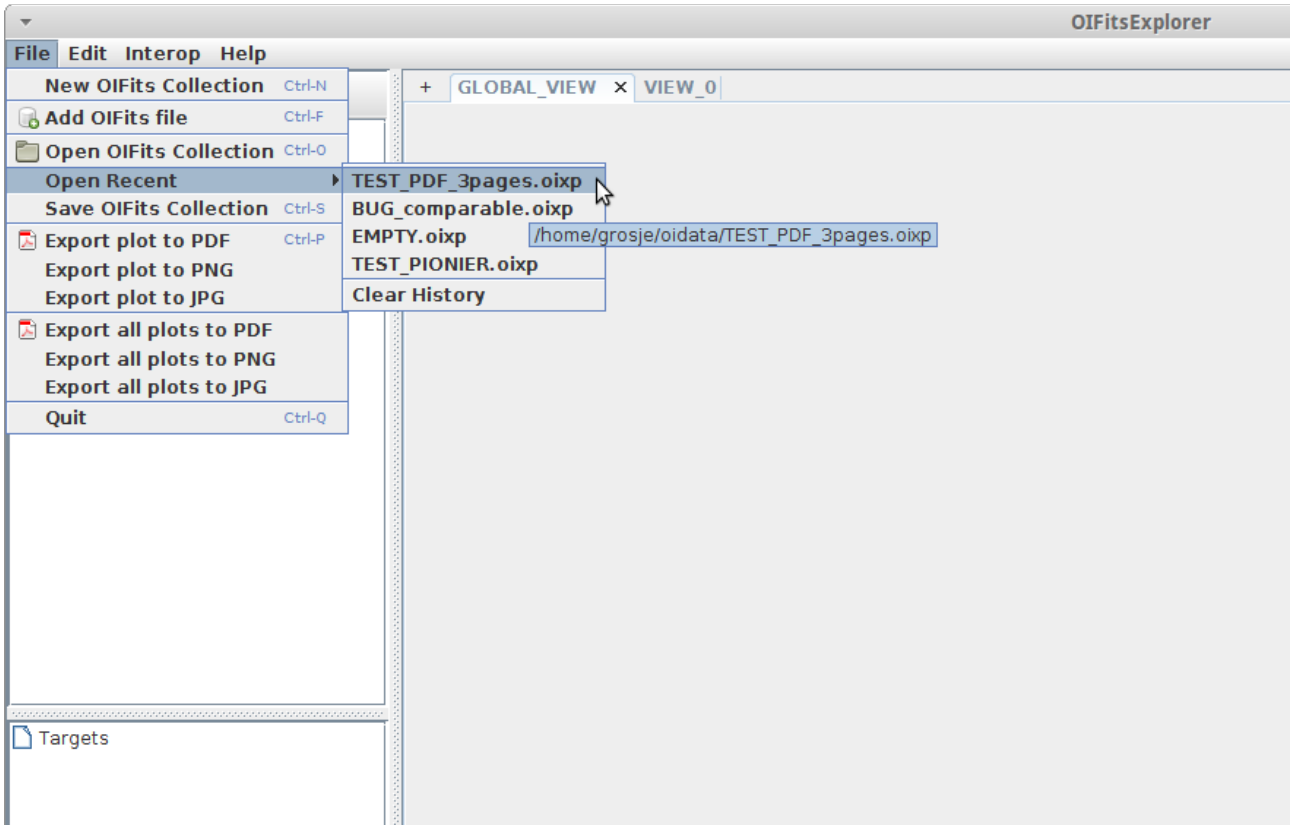
L'exécution se déroule ainsi :

./OIFitsExplorer -open fichier.oixp [-pdf fichier[.pdf]] [-help] [-mode [single/multi/default]] [-png fichier[.png]] [-jpg fichier[.jpg]] [-dims int,int]

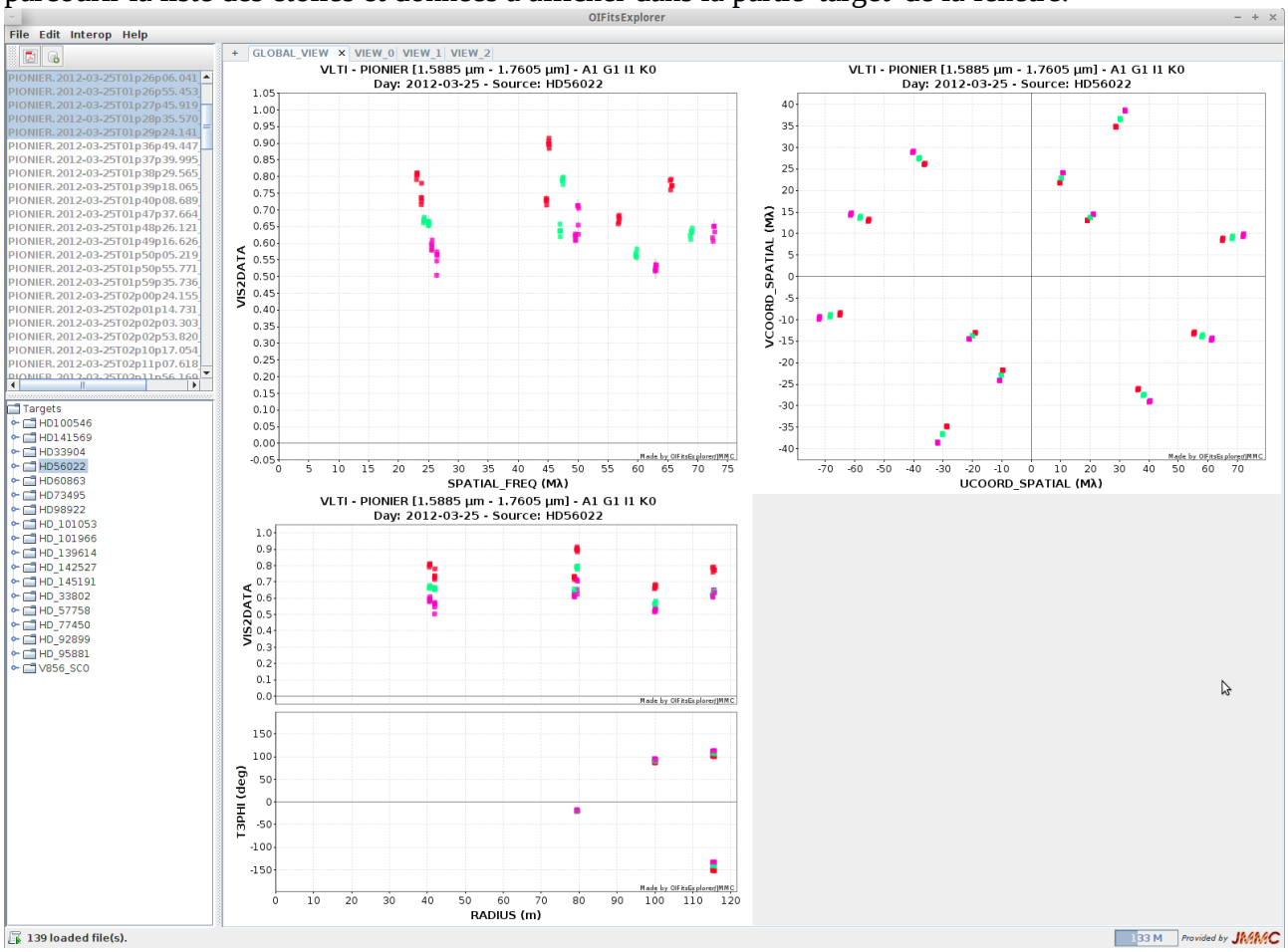
4.3.2 Utilisation d'OiFitsExplorer

4.3.2.1 Approche exportation

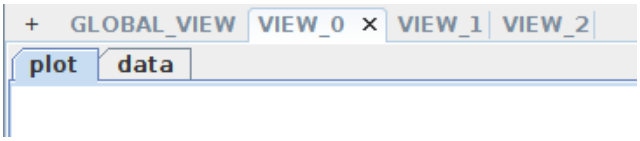
Lorsque OIFitsExplorer est lancé, il faut lui indiquer quelle collection charger. Comme expliqué précédemment il doit donner un fichier oixp. Il peut alors soit chercher lui-même le fichier sur son ordinateur ou renseigner un fichier qu'il a déjà utilisé comme sur l'image ci-dessous.



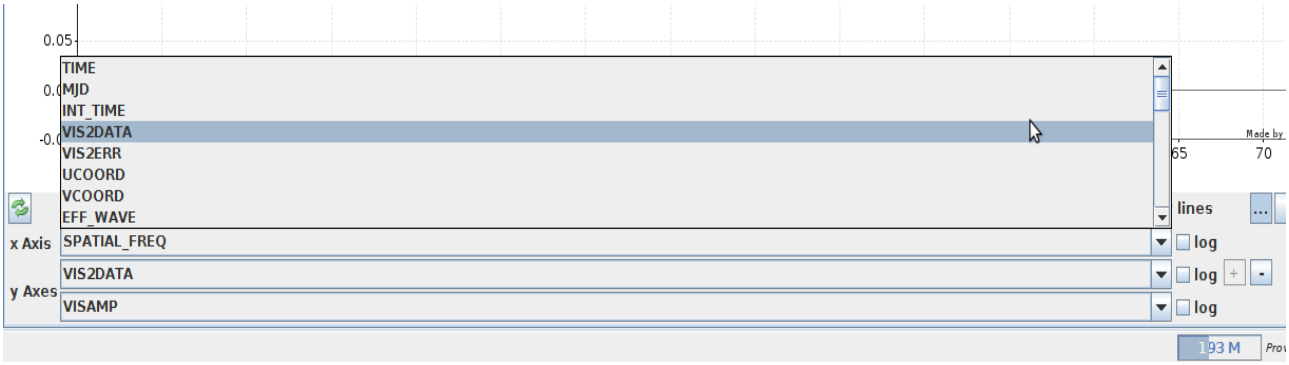
Une fois les collections chargées, OIFitsExplorer se charge d'afficher les vues qui ont été prédéfinies dans le fichier oixp. Dans notre exemple il y a 3 vues et la vue globale.
Ensuite libre à l'utilisateur d'ajouter des vues ou supprimer des vues existantes, il peut également parcourir la liste des étoiles et données à afficher dans la partie 'target' de la fenêtre.



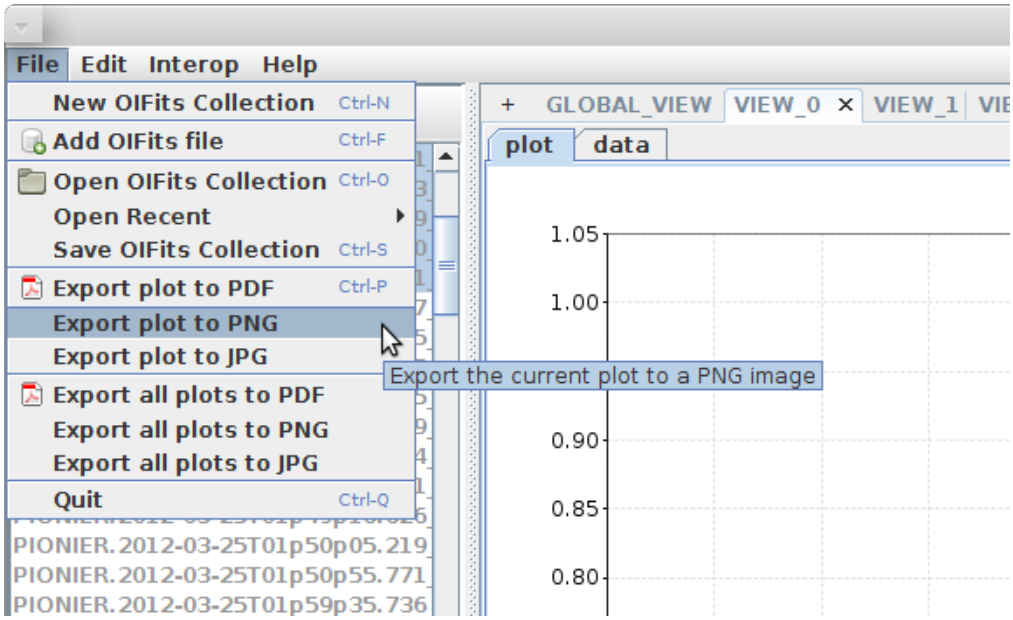
Zoom sur la partie de gestion des vues :



Il est donc possible de naviguer entre les différentes vues, de supprimer chaque vue séparément en cliquant sur sa croix associée.
Pour ajouter une vue, il suffit de cliquer sur le bouton '+' il faudra ensuite renseigner les colonnes à afficher. Pour se faire il faut sélectionner la nouvelle vue crée et entrer dans le menu en cliquant sur le bouton '…' comme ceci :



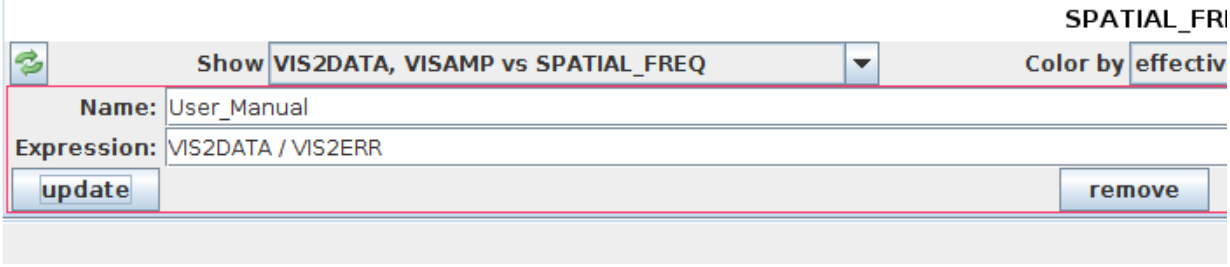
Une fois que nous avons sélectionné ce que nous voulons afficher, il est possible d'exporter ces résultats. Il est possible d'exporter tous les graphiques sur une page, dans ce cas toutes les vues y compris la vue globale seront exportées dans le format indiqué (Export all plots to ...). Il est également possible d'exporter uniquement la vue sélectionnée dans le format que l'on souhaite (Export plot to ...).
Pour exporter, il suffit de cliquer dans le menu File comme ceci :



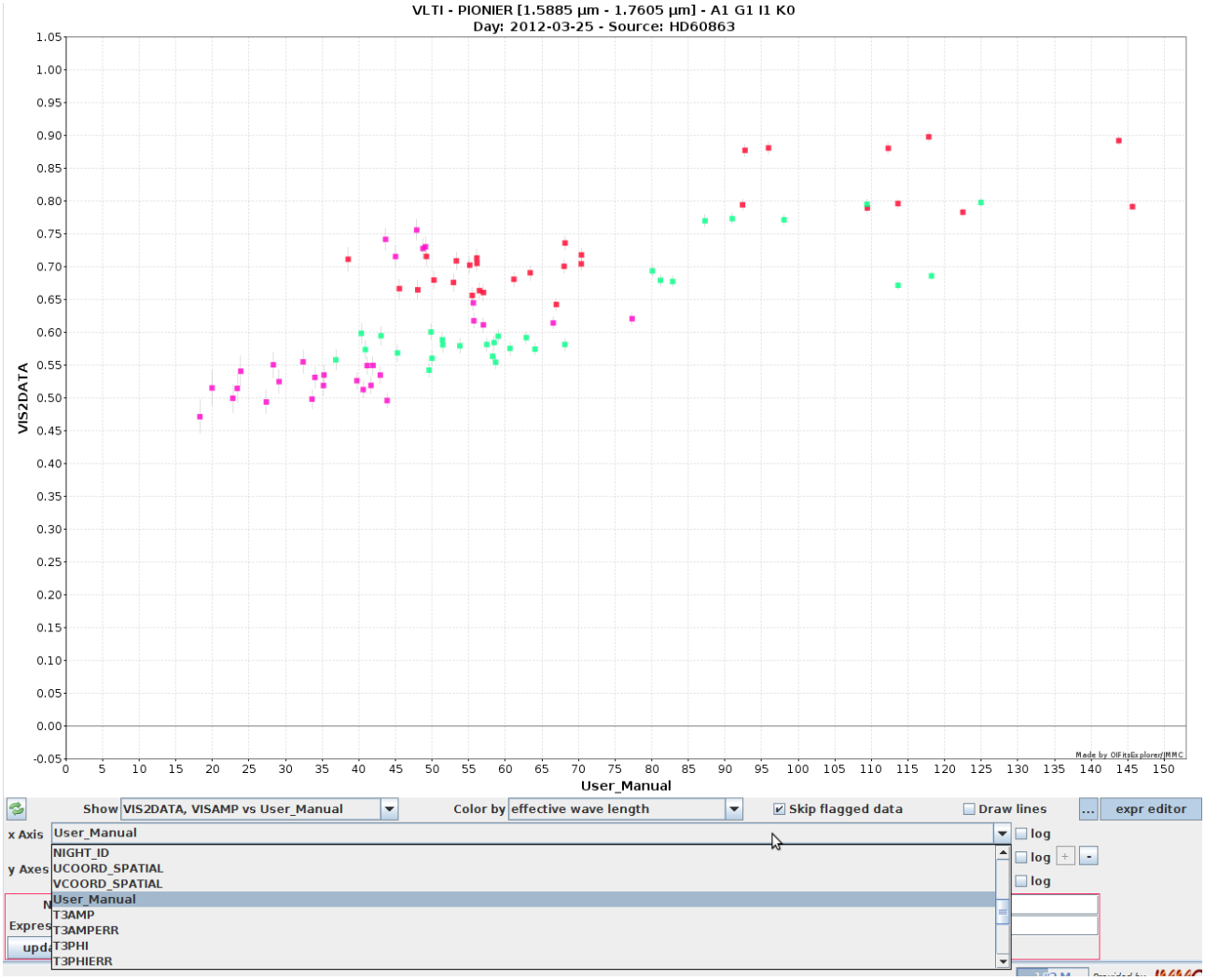
4.3.2.2 Approche script Jython

Pour ce qui est de l'approche script Jython, elle commence comme l'approche exportation, il faut

charger sa collection avec un fichier .oixp puis il faut sélectionner une vue pour pouvoir afficher le résultat des calculs. Ensuite il suffit de cliquer sur le bouton 'expr editor'. Deux nouveaux champs texte apparaissent et il est maintenant possible de renseigner une nouvelle colonne à afficher ou d'en supprimer une existante.



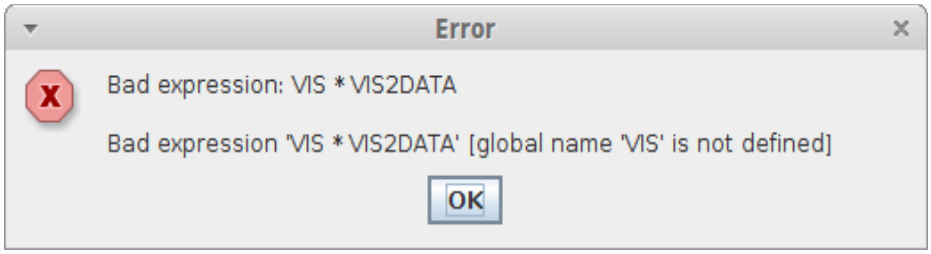
Pour visionner le résultat, il faut sélectionner la colonne créée dans la liste des colonnes.



Cette vue est donc maintenant exportable tout comme les autres vues.

Dans le cas d'une mauvaise expression, un message sera affiché.

Par exemple si on entre une opération sur une colonne qui n'existe pas, on obtient ce message.

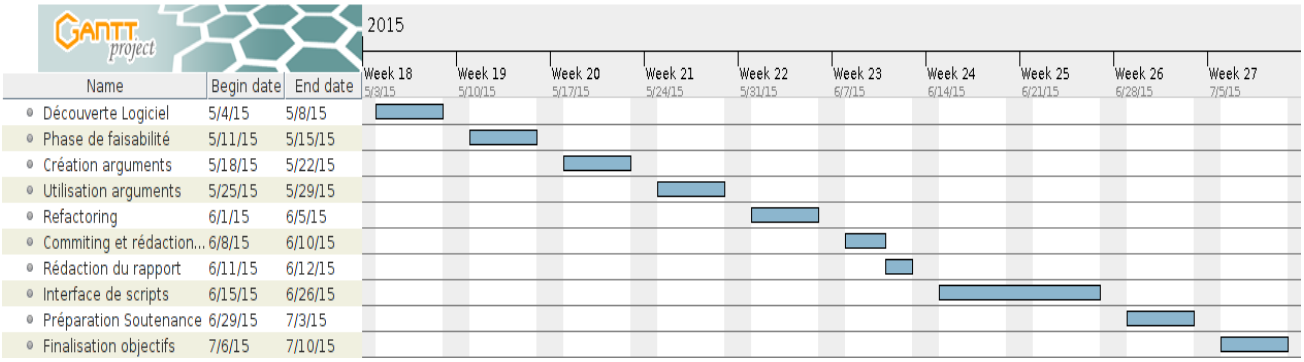


Comme le logiciel gère bien les exceptions, l'utilisateur n'a pas besoin de relancer le logiciel, une fois renseigné du message, il peut fermer la fenêtre d'erreur et rentrer à nouveau une expression.

4.4 Diagramme de Gantt

Comme l'équipe fonctionne en programmation agile, il n'y a pas eu de diagramme de Gantt prévisionnel.

Le diagramme ci-contre est le diagramme de Gantt final. On peut facilement remarquer que c'est la première partie du sujet qui m'a demandé le plus de temps. En effet, nous avons décidé de finaliser ce travail avec la phase de refactoring pour valider ces changements.



5 Annexes

5.1 Webographie

1. Site de l'institut de planétologie et d'Astrophysique :
http://ipag.osug.fr

2. Site du Centre Jean-Marie Mariotti :
http://www.jmmc.fr

3. Wiki sur l'avancement du projet :
http://www.jmmc.fr/twiki/bin/view/Jmmc/Software/JmmcOIFitsExplorerStageJPGros

4. Lien pour télécharger OIFitsExplorer :
http://www.jmmc.fr/oifitsexplorer_page.htm

5. Lien vers site Jython :
http://www.jython.org/

6. Lien description collection oifits :
http://www.mrao.cam.ac.uk/research/optical-interferometry/oifits/

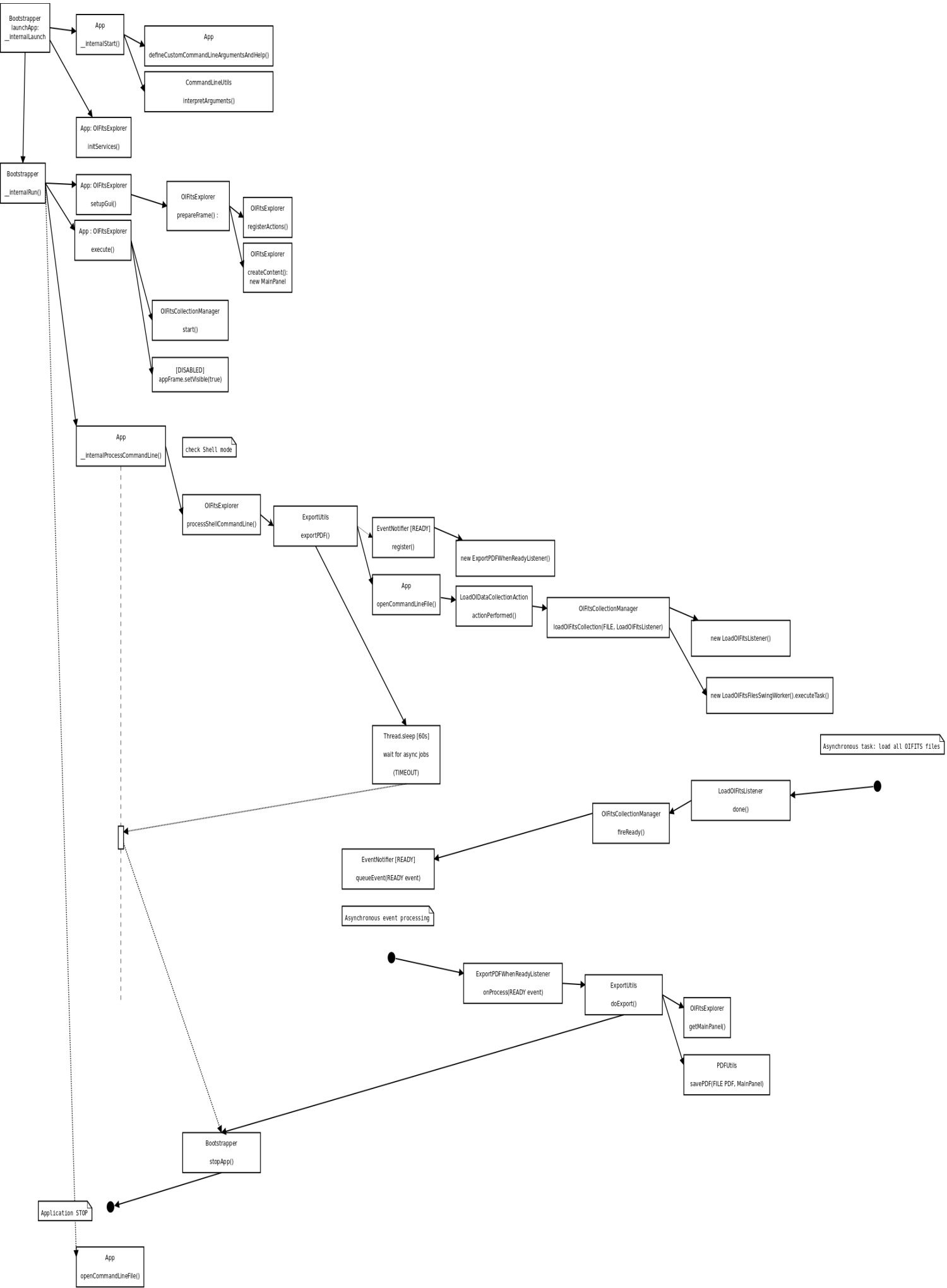
7. Lien explication svn :
https://subversion.apache.org/

8. Lien vers Maven :
https://maven.apache.org/

9. Information et tutoriels divers
Développez : http://www.developpez.com/
OpenClassrooms : http://openclassrooms.com/

10. Lien vers NetBeans :
https://netbeans.org/

5.2 Diagramme du cycle de vie de l'application



5.3 Code source

5.3.1 GlobalView

```
/**
 * ****************************************************************************
 * JMMC project ( http://www.jmmc.fr ) - Copyright (C) CNRS.
 * ****************************************************************************
 */
package fr.jmmc.oiexplorer.core.gui;

import fr.jmmc.oiexplorer.core.export.DocumentExportable;
import fr.jmmc.oiexplorer.core.export.DocumentOptions;
import fr.jmmc.oiexplorer.core.export.DocumentSize;
import fr.jmmc.oiexplorer.core.export.Orientation;
import fr.jmmc.oiexplorer.core.gui.action.ExportDocumentAction;
import fr.jmmc.oiexplorer.core.gui.chart.ChartUtils;
import fr.jmmc.oiexplorer.core.util.Constants;
import java.awt.GridLayout;
import java.util.ArrayList;
import java.util.List;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.ui.Drawable;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
/**
 * Plot view implementation
 *
 * @author Jean-Philippe GROS.
 */
public final class GlobalView extends javax.swing.JPanel implements
DocumentExportable {

    /**
     * default serial UID for Serializable interface
     */
    private static final long serialVersionUID = 1;
    /**
     * Class logger
     */
                    private static final Logger logger =
LoggerFactory.getLogger(GlobalView.class.getName());
    /** List of charts */
    private final List<ChartPanel> chartPanelList;

    /**
     * Creates new form PlotView
     */
    public GlobalView() {
        chartPanelList = new ArrayList<ChartPanel>();

        // Build GUI
        initComponents();
        // Finish init
        postInit();
    }
```

```
    public void addChart(JFreeChart chart) {
        logger.debug("addChart: {}", chart);
        ChartPanel chartPanel = ChartUtils.createChartPanel(chart, false);

        // zoom options :
        chartPanel.setDomainZoomable(Constants.ENABLE_ZOOM);
        chartPanel.setRangeZoomable(Constants.ENABLE_ZOOM);

        // enable mouse wheel:
        chartPanel.setMouseWheelEnabled(true);

        chartPanelList.add(chartPanel);
        refreshJPanel();
    }

    public void removeChart(JFreeChart chart) {
        logger.debug("removeChart: {}", chart);

        for (int i = 0; i < chartPanelList.size(); i++) {
            ChartPanel chartPanel = chartPanelList.get(i)
            if (chart == chartPanel.getChart()) {
                // remove(chartPanel):
                chartPanelList.remove(i);
                refreshJPanel();
                break;
            }
        }
    }

    public void refreshJPanel() {
        final int nCharts = chartPanelList.size();

        this.removeAll();

        int nCols = (nCharts == 1) ? 1 : 2;
        ((GridLayout) this.getLayout()).setColumns(nCols);

        for (int i = 0; i < nCharts; i++) {
            this.add(chartPanelList.get(i));
        }
    }

    /**
       * This method is useful to set the models and specific features of
initialized swing components :
     */
    private void postInit() {
    }

    /**
     * This method is called from within the constructor to initialize the form.
WARNING: Do NOT modify this code. The
     * content of this method is always regenerated by the Form Editor.
   */ @SuppressWarnings("unchecked")    // <editor-fold defaultstate="collapsed"
desc="Generated Code">
    private void initComponents() {

        panel1 = new java.awt.Panel();
```

```
    jColorChooser1 = new javax.swing.JColorChooser();

    setLayout(new java.awt.GridLayout(0, 2));
}// </editor-fold>
// Variables declaration - do not modify
private javax.swing.JColorChooser jColorChooser1;
private java.awt.Panel panel1;
// End of variables declaration

/**
 * Export the chart component as a PDF document.
 * @param action export action to perform the export action
 */
@Override
public void performAction(final ExportDocumentAction action) {
    if (!chartPanelList.isEmpty()) {
        action.process(this);
    }
}

/**
 * Return the PDF default file name
 *
 * @param fileExtension  document's file extension
 * @return PDF default file name
 */
@Override
public String getDefaultFileName(final String fileExtension) {
    return null;
}

/**
 * Prepare the page layout before doing the export:
 * Performs layout and modifies the given options
 * @param options document options used to prepare the document
 */
@Override
public void prepareExport(final DocumentOptions options) {
    options.setNormalDefaults();
}


/**
 * Return the page to export given its page index
 * @param pageIndex page index (1..n)
 * @return Drawable array to export on this page
 */
@Override
public Drawable[] preparePage(final int pageIndex) {
    final int nCharts = chartPanelList.size();
    final JFreeChart[] charts = new JFreeChart[nCharts];


    for (int i = 0; i < nCharts; i++) {
        charts[i] = chartPanelList.get(i).getChart();
      }
    return charts;
}
```

```
    /**
      * Callback indicating the PDF document is done to reset the component's
state
     */
    @Override
    public void postExport() {

    }
}
```

5.3.2 Writer

```
/*******************************************************************************
 * JMMC project ( http://www.jmmc.fr ) - Copyright (C) CNRS.
 ******************************************************************************/
package fr.jmmc.oiexplorer.core.export;

import fr.jmmc.jmcs.data.MimeType;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.io.File;
import java.io.IOException;
import org.jfree.ui.Drawable;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 *
 * @author grosje
 */
public abstract class Writer {

    /** Class logger */
                    protected     static     final     Logger     logger     =
LoggerFactory.getLogger(Writer.class.getName());

    /** flag to draw a red border around the rectangle area */
    private final static boolean DEBUG_RECTANGLE = false;

    /**
     * Give the instance that corresponds to the mime type
     * @param mimeType mime type
     * @return Wanted instance
     */
    public static Writer getInstance(final MimeType mimeType) {
        if (mimeType == MimeType.PDF) {
            return new PDFWriter();
        }
        if (mimeType == MimeType.PNG || mimeType == MimeType.JPG) {
            return new ImageWriter();
        }
                throw  new  IllegalStateException("Unsuported  Mime  type  ["  +
mimeType.getId()
                + "] (only PDF, PNG or JPG expected) !");
    }

    /**
     * Save the given exportable component as a document in the given file
     * @param file file to create
```

```
     * @param exportable exportable component
     * @param options options
     *
     * @throws IOException if the file exists but is a directory
     *                   rather than a regular file, does not exist but cannot
     *                   be created, or cannot be opened for any other reason
     * @throws IllegalStateException if a document exception occurred
     */
    public abstract void write(final File file,
            final DocumentExportable exportable,
                        final  DocumentOptions  options)  throws  IOException,
IllegalStateException;

    /**
     * Draw the given Drawable instances on the given Graphics2D
     * @param drawables array of Drawable
     * @param g2 graphic object
     * @param width of the page
     * @param height of the page
     */
    public static void draw(Drawable[] drawables, Graphics2D g2,
            final float width, final float height) {

        final int nCharts = drawables.length;
        final int nbCol = (nCharts > 1) ? 2 : 1;
        final int nbLine = (nCharts / nbCol) + (nCharts % nbCol);

        for (int i = 0; i < nCharts; i++) {
            final int heightAdjust = i / nbCol;

            final Rectangle2D drawArea = new Rectangle2D.Double(
                    (i % nbCol) * width / nbCol,
                    (heightAdjust * height) / nbLine,
                    width / nbCol,
                    height / nbLine);

            //draw chart:
            drawables[i].draw(g2, drawArea);

            if (DEBUG_RECTANGLE) {
                g2.setColor(Color.RED);
                g2.draw(drawArea);
            }
        }
    }

}
```

5.3.3 DocumentOptions

```
/*******************************************************************************
 * JMMC project ( http://www.jmmc.fr ) - Copyright (C) CNRS.
```

```
 *******************************************************************************/
package fr.jmmc.oiexplorer.core.export;

import fr.jmmc.jmcs.data.MimeType;
import java.awt.geom.Rectangle2D;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public abstract class DocumentOptions {

    /** Class logger */
                    protected     static     final     Logger     logger     =
LoggerFactory.getLogger(DocumentOptions.class.getName());

    // Factory pattern
    /**
     * Create the options of the export document
     *@param mimeType mime type of the export document
     *@return options of the export document depending on the mime type
     *@throws IllegalArgumentException
     */
    public static DocumentOptions createInstance(final MimeType mimeType) throws
IllegalArgumentException {
        if (mimeType == MimeType.PDF) {
            return new PDFOptions(mimeType);
        }
        if (mimeType == MimeType.PNG || mimeType == MimeType.JPG) {
            return new ImageOptions(mimeType);
        }
              throw  new  IllegalArgumentException("Unsupported  mimetype  :  "  +
mimeType);
    }

    /* members */
    /** mime type */
    private final MimeType mimeType;
    /** Mode of exportation (single/multi/default) */
    private DocumentMode mode = null;
    /** Page size */
    private DocumentSize documentSize = null;
    /** Page orientation */
    private Orientation orientation = null;
    /** number of pages */
    private int numberOfPages = 0;

    protected DocumentOptions(final MimeType mimeType) {
        assert (mimeType != null);
        this.mimeType = mimeType;
    }

    /**
     * Adjusts the size of the document depending on the nature of the document
     * @return a rectangle with calculated size
     */
    public abstract Rectangle2D.Float adjustDocumentSize();

    /**
      *Overwrite options of the method parameter if the command options are not
null or default
```

```
     * @param other default options of exporting
     */
    public void merge(DocumentOptions other) {
        logger.debug("merge: other : {} this : {}", other, this);
        if (this.getClass() != other.getClass()) {
                throw new IllegalStateException("Incompatible DocumentOptions
class !");
        }
        if (this.getMimeType() != other.getMimeType()) {
            throw new IllegalStateException("Incompatible mimetype !");
        }
        if (other.getMode() != null) {
            this.setMode(other.getMode());
        }
        if (other.getDocumentSize() != null) {
            this.setDocumentSize(other.getDocumentSize());
        }
        if (other.getOrientation() != null) {
            this.setOrientation(other.getOrientation());
        }
        if (other.getNumberOfPages() != 0) {
            this.setNumberOfPages(other.getNumberOfPages());
        }
        logger.debug("merge(DocumentOptions): this: {}", this);
    }

    /**
     * Set default options: DocumentSize: Normal, Orientation: Landscape, 1 page
    */
    public final void setNormalDefaults() {
        setDocumentSize(DocumentSize.NORMAL)
                .setOrientation(Orientation.Landscape)
                .setNumberOfPages(1);
    }
    /**
     * Set default options: DocumentSize: Normal, Orientation: Landscape, 1 page
    */
    public final void setSmallDefaults() {
        setDocumentSize(DocumentSize.SMALL)
                .setOrientation(Orientation.Landscape)
                .setNumberOfPages(1);
    }

    /** Return the mime type
     @return mime type
     */
    public MimeType getMimeType() {
        return mimeType;
    }

    /** Return the mode
     * @return the mode
     */
    public DocumentMode getMode() {
        return mode;
    }

    /** Set the mode within a Document mode parameter
     * @param mode
```

```
     * @return this (fluent API)
     */
    public DocumentOptions setMode(DocumentMode mode) {
        this.mode = mode;
        return this;
    }

    /** Set the mode of exportation (single/multi/default)
     * @param mode
     * @return the mode
     * @throws IllegalArgumentException
     */
    public DocumentOptions setMode(String mode) throws IllegalArgumentException
{
        //mode verifications
        if (mode == null) {
            return setMode(DocumentMode.DEFAULT);
        }
        return setMode(DocumentMode.parse(mode));
    }

    /**
     * Return the page size
     * @return page size
     */
    public DocumentSize getDocumentSize() {
        return documentSize;
    }

    /**
     * @param documentSize
     * @return this (fluent API)
     */
    public DocumentOptions setDocumentSize(DocumentSize documentSize) {
        this.documentSize = documentSize;
        return this;
    }

    /**
     * Return the page orientation
     * @return page orientation
     */
    public Orientation getOrientation() {
        return orientation;
    }

    /**
     * Set the page orientation
     * @param orientation
     * @return this (fluent API)
     */
    public DocumentOptions setOrientation(Orientation orientation) {
        this.orientation = orientation;
        return this;
    }

    /**
     * Return the number of pages
     * @return number of pages
```

```
     */
    public int getNumberOfPages() {
        return numberOfPages;
    }

    /**
     * Set the number of pages
     * @param numberOfPages
     * @return this (fluent API)
     */
    public DocumentOptions setNumberOfPages(int numberOfPages) {
        this.numberOfPages = numberOfPages;
        return this;
    }

    @Override
    public String toString() {
        return "mimeType: " + getMimeType()
                + " mode: " + getMode() + " documentSize: " + getDocumentSize()
                   + " orientation: " + getOrientation() + " numberOfPages: " +
getNumberOfPages();
    }
}
```

5.3.4 PDFOptions

```
/*******************************************************************************
 * JMMC project ( http://www.jmmc.fr ) - Copyright (C) CNRS.
 ******************************************************************************/
package fr.jmmc.oiexplorer.core.export;

import fr.jmmc.jmcs.data.MimeType;
import java.awt.geom.Rectangle2D;


public final class PDFOptions extends DocumentOptions {

    protected PDFOptions(final MimeType mimeType) {
        super(mimeType);
    }

    @Override
    public Rectangle2D.Float adjustDocumentSize() {
        // adjust document size (SMALL, A3, A2) and orientation according to the
options :
        com.lowagie.text.Rectangle documentPage;
        switch (getDocumentSize()) {
            default:
            case SMALL:
                documentPage = com.lowagie.text.PageSize.A4;
                break;
            case NORMAL:
                documentPage = com.lowagie.text.PageSize.A3;
                break;
            case LARGE:
                documentPage = com.lowagie.text.PageSize.A2;
                break;
        }

        if (Orientation.Landscape == getOrientation()) {
            documentPage = documentPage.rotate();
        }
                        return   new   Rectangle2D.Float(documentPage.getLeft(),
documentPage.getBottom(),
                documentPage.getWidth(), documentPage.getHeight());
    }

    /**
     * Overwrite options of the method parameter if the command options are not
null or default
     * @param otherOptions
     */
    @Override
    public void merge(final DocumentOptions otherOptions) {
        super.merge(otherOptions);
        logger.debug("merge(PDFOptions): this: {}", this);
    }
}
```

5.3.5 ImageOptions

```
/*******************************************************************************
 * JMMC project ( http://www.jmmc.fr ) - Copyright (C) CNRS.
 ******************************************************************************/
package fr.jmmc.oiexplorer.core.export;

import fr.jmmc.jmcs.data.MimeType;
import java.awt.geom.Rectangle2D;

public final class PDFOptions extends DocumentOptions {

    protected PDFOptions(final MimeType mimeType) {
        super(mimeType);
    }

    @Override
    public Rectangle2D.Float adjustDocumentSize() {
        // adjust document size (SMALL, A3, A2) and orientation according to the
options :
        com.lowagie.text.Rectangle documentPage;
        switch (getDocumentSize()) {
            default:
            case SMALL:
                documentPage = com.lowagie.text.PageSize.A4;
                break;
            case NORMAL:
                documentPage = com.lowagie.text.PageSize.A3;
                break;
            case LARGE:
                documentPage = com.lowagie.text.PageSize.A2;
                break;
        }

        if (Orientation.Landscape == getOrientation()) {
            documentPage = documentPage.rotate();
        }
                        return   new   Rectangle2D.Float(documentPage.getLeft(),
documentPage.getBottom(),
                documentPage.getWidth(), documentPage.getHeight());
    }

    /**
     * Overwrite options of the method parameter if the command options are not
null or default
     * @param otherOptions
     */
    @Override
    public void merge(final DocumentOptions otherOptions) {
        super.merge(otherOptions);
        logger.debug("merge(PDFOptions): this: {}", this);
    }
}
```

5.3.6 PDFWriter

```
/*******************************************************************************
*   JMMC   project   (   http://www.jmmc.fr   )   -   Copyright   (C)   CNRS.
*******************************************************************************/
package fr.jmmc.oiexplorer.core.export;
import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
import com.lowagie.text.Rectangle;
import com.lowagie.text.pdf.DefaultFontMapper;
import com.lowagie.text.pdf.FontMapper;
import com.lowagie.text.pdf.PdfContentByte;
import com.lowagie.text.pdf.PdfTemplate;
import com.lowagie.text.pdf.PdfWriter;
import fr.jmmc.jmcs.data.MimeType;
import fr.jmmc.jmcs.data.app.ApplicationDescription;
import fr.jmmc.jmcs.util.FileUtils;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import org.jfree.ui.Drawable;

/**
 * This class is dedicated to export charts as PDF documents
 */
public final class PDFWriter extends Writer {

    /**
     * Force text rendering to use java rendering as Shapes.
     * This is a workaround to get Unicode greek characters rendered properly.
          *  Embedding  fonts  in  the  PDF  may  depend  on  the  Java/OS  font
configuration ...
     */
    public final static boolean RENDER_TEXT_AS_SHAPES = true;

    /**
     * protected Constructor
     */
    PDFWriter() {
        // no-op
    }

    /**
     * Save the given chart as a PDF document in the given file
     * @param pdfFile PDF file to create
     * @param exportable exportable component
     * @param options PDF options
     *
     * @throws IOException if the file exists but is a directory
     *                   rather than a regular file, does not exist but cannot
     *                   be created, or cannot be opened for any other reason
     * @throws IllegalStateException if a PDF document exception occurred
     */
    @Override
     public void write(final File pdfFile, final DocumentExportable exportable,
final DocumentOptions options)
```

```
            throws IOException, IllegalStateException {

        // Fix file extension:
        final File file = MimeType.PDF.checkFileExtension(pdfFile);

        final long start = System.nanoTime();

        BufferedOutputStream bo = null;
        try {
            bo = new BufferedOutputStream(new FileOutputStream(file));

            writeChartAsPDF(bo, exportable, options);

        } finally {
            FileUtils.closeStream(bo);

            if (logger.isInfoEnabled()) {
                    logger.info("write[{}] : duration = {} ms.", file, 1e-6d *
(System.nanoTime() - start));
            }
        }
    }

    /**
       * Create a PDF document with the given chart and save it in the given
stream
     * @param outputStream output stream
     * @param exportable exportable component
     * @param options PDF options
     *
     * @throws IllegalStateException if a PDF document exception occurred
     */
    private static void writeChartAsPDF(final OutputStream outputStream,
             final DocumentExportable exportable, final DocumentOptions options)
throws IllegalStateException {

        Graphics2D g2 = null;

         // adjust document size (A4, A3, A2) and orientation according to the
options :
        final Rectangle2D.Float documentPage = options.adjustDocumentSize();

        final Document document = new Document(new Rectangle(documentPage.x,
                documentPage.y, documentPage.width, documentPage.height));

        final Rectangle pdfRectangle = document.getPageSize();

        final float width = (int) pdfRectangle.getWidth();
        final float height = (int) pdfRectangle.getHeight();

        /*
         Measurements
          When creating a rectangle or choosing a margin, you might wonder what
measurement unit is used:
         centimeters, inches or pixels.
           In fact, the default measurement system roughly corresponds to the
various definitions of the typographic
          unit of measurement known as the point. There are 72 points in 1 inch
(2.54 cm).
```

```
         */
        // margin = 1 cm :
        final float marginCM = 0.5f;
        // in points :
        final float margin = marginCM * 72f / 2.54f;

        final float innerWidth = width - 2 * margin;
        final float innerHeight = height - 2 * margin;

        try {
                        final  PdfWriter  writer  =  PdfWriter.getInstance(document,
outputStream);

            document.open();

            definePDFProperties(document);

            PdfTemplate pdfTemplate;
            Drawable[] drawables;

            final PdfContentByte pdfContentByte = writer.getDirectContent();

             for (int pageIndex = 1, numberOfPages = options.getNumberOfPages();
pageIndex <= numberOfPages; pageIndex++) {
                // new page:
                document.newPage();

                pdfTemplate = pdfContentByte.createTemplate(width, height);

                if (RENDER_TEXT_AS_SHAPES) {
                    // text rendered as shapes so the file is bigger but correct
                              g2  =  pdfTemplate.createGraphicsShapes(innerWidth,
innerHeight);
                } else {
                       // depending on the font mapper, special characters like
greek chars are not rendered:
                        g2 = pdfTemplate.createGraphics(innerWidth, innerHeight,
getFontMapper());
                }

                // Get Drawables:
                drawables = exportable.preparePage(pageIndex);

                draw(drawables, g2, innerWidth, innerHeight);

                pdfContentByte.addTemplate(pdfTemplate, margin, margin);

                // free graphics:
                g2.dispose();
                g2 = null;
            }

        } catch (DocumentException de) {
            throw new IllegalStateException("PDF document exception : ", de);
        } finally {
            if (g2 != null) {
                g2.dispose();
            }
            document.close();
```

```
        }
    }

    /**
     * Define PDF properties (margins, author, creator ...)
     * @param document pdf document
     */
    private static void definePDFProperties(final Document document) {
        document.addCreator(ApplicationDescription.getInstance().getProgramNameW
ithVersion());
    }

    /**
       * Return the font mapper used to translate Java2D Fonts to PDF Fonts
(virtual or embedded)
     * @return font mapper
     */
    private static FontMapper getFontMapper() {

         // ChartFontMapper (test) substitutes SansSerif fonts (plain and bold)
by DejaVu fonts which supports both unicode and greek characters
           // However, fonts must be distributed (GPL) and many problems can
happen...
    /* return new ChartFontMapper(); */
         // default font mapper uses Helvetica (Cp1252) which DOES NOT SUPPORT
UNICODE CHARACTERS:
        return new DefaultFontMapper();
    }
}
```

5.3.7 JythonEval

```
/*******************************************************************************
 * JMMC project ( http://www.jmmc.fr ) - Copyright (C) CNRS.
 ******************************************************************************/
package fr.jmmc.oitools.model;

import fr.jmmc.oitools.meta.ColumnMeta;
import java.util.Arrays;
import java.util.Collection;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.python.core.PyArray;
import org.python.core.PyException;
import org.python.core.PyInteger;
import org.python.util.PythonInterpreter;

/**
 *
 * @author grosje
 */
public final class JythonEval {

    private final static boolean DEBUG = false;
    /** logger */
                    private     final     static     Logger     _logger     =
Logger.getLogger(JythonEval.class.getName());

    private JythonEval() {

    }

    static double[][] eval(final OIData oiData, String expr,
            boolean testOnly) throws RuntimeException {

        // From OIData table:
        int nRows = oiData.getNbRows();
        int nWaves = oiData.getNWave();

        if (testOnly) {
            nRows = Math.min(nRows, 1);
            nWaves = Math.min(nWaves, 1);
        }

        // attention: dimensions en sortie ?
        // deduire la dimensionalité ? ucoord /3 => 1d
        final double[][] result = new double[nRows][nWaves];

        final StringBuilder script = new StringBuilder(1024);

        try {
            System.setProperty("python.import.site", "false");

            // Create an instance of the PythonInterpreter
            PythonInterpreter interp = new PythonInterpreter();

            String colName, varName;

            interp.set("_rows", new PyInteger(nRows));
            interp.set("_waves", new PyInteger(nWaves));
```

```
            // Set Inputs within the PythonInterpreter instance
            // Special case for wavelengths:
            colName = "EFF_WAVE";
            varName = "_input_" + "EFF_WAVE";
                                interp.set(varName,  new  PyArray(double[][].class,
oiData.getEffWaveAsDoubles()));

            if (DEBUG) {
                interp.exec("print '" + colName + ":'," + varName);
            }

                             final  Collection<ColumnMeta>  columnsDescCollection  =
oiData.getColumnDescCollection();

            // For each data column from OIData table:
            // get column name + values as 1d or 2d array
            // attention au type ! double[][] ou double[] ou float[]
            // sets variables in python: '_input_<column_name>'
            for (ColumnMeta column : columnsDescCollection) {
                colName = column.getName();
                varName = "_input_" + colName;

                 // Warning: only columns with double[][] or double[] values in
PythonInterpreter:
                // TODO: handle null values !
                switch (column.getDataType()) {
                    case TYPE_DBL:
                        if (column.isArray()) {
                                                    final  double[][]  dValues  =
oiData.getColumnDoubles(colName);

                                interp.set(varName, new PyArray(double[][].class,
dValues));

                            if (DEBUG) {
                                     interp.exec("print '" + colName + ":'," +
varName);
                            }
                            break;
                        }
                                                     final  double[]  dValues  =
oiData.getColumnDouble(colName);

                                 interp.set(varName,  new  PyArray(double[].class,
dValues));

                        if (DEBUG) {
                            interp.exec("print '" + colName + ":'," + varName);
                        }
                        break;

                    case TYPE_CHAR:
                    case TYPE_INT:
                    case TYPE_REAL:
                    case TYPE_LOGICAL:
                    case TYPE_COMPLEX:
                    default:
//                        System.out.println("Unsupported column: [" + colName
```

```
//                                + "] type=" + column.getDataType());
                    // ignore
                }
            }

            // Output (fixed)
            interp.set("_output", new PyArray(double[][].class, result));

            script.append("from array import array \n");
            script.append("from math import *\n\n");

            script.append("def user_func():\n");
            script.append("\treturn " + expr + "\n\n");

            // Loops on rows x wavelengths:
            script.append("for _i in range(_rows):\n");
            script.append("\tfor _j in range(_waves):\n");

            if (DEBUG) {
                script.append("\t\tprint 'indices: ', _i, _j \n");
            }
            script.append("\n");

            // Get scalar values from input arrays:
            // Special case for wavelengths:
            colName = "EFF_WAVE";
            varName = "_input_" + colName;
            // 1D on wavelengths:
            script.append("\t\t" + colName + " = " + varName + "[_i][_j]\n");
            // attention aux tableaux:
            // 2D [i][j]
            // 1D [i] // rows
            // 1D [j] // wavelengths
            // For each data column from OIData table:
            for (ColumnMeta column : columnsDescCollection) {
                colName = column.getName();
                varName = "_input_" + colName;

                 // Warning: only columns with double[][] or double[] values in
PythonInterpreter:
                switch (column.getDataType()) {
                    case TYPE_DBL:
                        if (column.isArray()) {
                            // 2D:
                             script.append("\t\t" + colName + " = " + varName +
"[_i][_j]\n");
                            break;
                        }

                        // 1D on rows:
                             script.append("\t\t" + colName + " = " + varName +
"[_i]\n");
                        break;

                    case TYPE_CHAR:
                    case TYPE_INT:
                    case TYPE_REAL:
                    case TYPE_LOGICAL:
                    case TYPE_COMPLEX:
```

```
                    default:
                    // ignore
                }
            }
            script.append("\n\t\t_output[_i][_j] = user_func() \n");

            if (DEBUG) {

                // Special case for wavelengths:
                colName = "EFF_WAVE";
                   script.append("\n\t\tprint '" + colName + ":'," + colName +
"\n");

                for (ColumnMeta column : columnsDescCollection) {

                    colName = column.getName();

                     // Warning: only columns with double[][] or double[] values
in PythonInterpreter:
                    switch (column.getDataType()) {
                        case TYPE_DBL:
                                script.append("\t\tprint '" + colName + ":'," +
colName + "\n");
                            break;

                        case TYPE_CHAR:
                        case TYPE_INT:
                        case TYPE_REAL:
                        case TYPE_LOGICAL:
                        case TYPE_COMPLEX:
                        default:
                        // ignore
                    }
                }
                script.append("\t\tprint 'output:',_output[_i][_j] \n");
            }

            if (DEBUG) {
                _logger.info("script:\n" + script);
            }

            interp.exec(script.toString());

        } catch (PyException pe) {
            if (testOnly) {
                // TODO: extract good error message:
                 throw new RuntimeException("Bad expression '" + expr + "' [" +
pe.value + "]", pe);
            }

            _logger.log(Level.INFO, "Python script error:" + pe.value);
            _logger.info("Script executed: \n" + script);

            // reset results to NaN
            for (int i = 0; i < nRows; i++) {
                Arrays.fill(result[i], Double.NaN);
            }
        }
```

```
        // retourner les resultats ...
        if (_logger.isLoggable(Level.FINE)) {
            _logger.fine("result: " + Arrays.deepToString(result));
        }

        return result;
    }

}
```