Système d'analyse et de visualisation d'images hyperspectrales appliqué aux sciences planétaires

Ludovic Mercier

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MÉMOIRE

présenté par Ludovic Mercier

en vue d'obtenir

LE DIPLÔME D'INGÉNIEUR C.N.A.M.

en INFORMATIQUE

Système d'analyse et de visualisation d'images hyperspectrales appliqué aux sciences planétaires.

Soutenu le 8 avril 2011, à Grenoble

JURY

Président :	M. Eric Gressier-Soudan
Examinateurs :	M. Jean-Pierre Giraudin
	M. André Plisson
	M. Mathias Voisin-Fradin
Tuteurs :	M. Douté Sylvain
	M. Volcke Pierre

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MÉMOIRE

présenté par Ludovic Mercier

en vue d'obtenir

LE DIPLÔME D'INGÉNIEUR C.N.A.M.

en INFORMATIQUE

Système d'analyse et de visualisation d'images hyperspectrales appliqué aux sciences planétaires.

Soutenu le 8 avril 2011, à Grenoble

Les travaux relatifs à ce mémoire ont été effectués au Laboratoire de Planétologie de Grenoble, composante de l'Observatoire des Sciences de l'Univers de Grenoble, sous la direction du Dr. Sylvain Douté et de M. Pierre Volcke.

Avant propos

Conventions typographiques

Afin de faciliter la lecture, le symbole « † » accolé à un mot spécifie une entrée dans le glossaire situé à la fin de ce document.

Remerciements

Merci à Pierre Volcke pour avoir voulu m'encadrer et me conseiller tout au long de ce mémoire.

Je souhaite remercier également toute l'équipe du Laboratoire de Planétologie devenue maintenant l'Institut de Planétologie et d'Astrophysique de Grenoble, avec laquelle je travaille au sein du projet Vahiné et tout particulièrement le Dr. Sylvain Douté.

Je remercie toutes les personnes du CNAM qui me font l'honneur de participer au jury de ce mémoire. Je remercie Monsieur André Plisson, directeur du centre d'enseignement de Grenoble et Monsieur Jean-Pierre Giraudin, professeur à l'Université Pierre-Mendès France de Grenoble et responsable du cycle ingénieur CNAM en informatique à Grenoble. Je remercie aussi Monsieur Mathias Voisin-Fradin directeur-adjoint du centre d'enseignement de Grenoble et je remercie particulièrement Monsieur Eric Gressier, professeur au CNAM de Paris, qui me fait l'honneur de présider le jury.

Par ailleurs, un merci tout particulier au membre de l'équipe informatique du LPG/LAOG/IPAG pour leurs amitiés, leurs soutiens et le temps qu'ils m'ont accordé pour les discussions et relectures de ce document. Merci donc à Sylvain, Guillaume, Laurent, Richard et tous les autres qui ne sont pas cités présentement.

Merci à Philippe pour nos discussions « Debiannesque » et nos tentatives de conversion de l'humanité à un meilleur avenir...

Merci aussi à Olivier et Roland pour les discussions orientées chantier autour du café matinal.

Je ne saurais oublier notre râleur national Ghislain et son colistier Hafid avec qui j'ai passé d'excellents moments.

Enfin mon plus grand remerciement va à ma femme Karen qui m'a soutenu jusqu'au bout et m'a remotivé dans les baisses de moral qui ont accompagnés ces 8 dernières années de cours du soir. A ton tour maintenant de t'y coller ;)

Table des matières

Table des figures

Listings

Liste des symboles

Sigles

ASCII	American Standard Code for Information Interchange.
BIL	Band Interleaved by Line.
BIP	Band Interleaved by Pixel.
BSQ	Band Sequential.
CCD	Charge-Coupled Device ; Capteur photographique de type matriciel.
CRISM	Compact Reconnaissance Imaging Spectrometer for Mars ; Spectromètre travaillant dans l'infrarouge et la lumière visible embarquée à bord de la sonde Mars Reconnaissance Orbiter.
CSV	Comma Separated Value ; format informatique représentant des données tabulaires sous forme de valeurs séparées par des virgules.
ELM	Eigenvalue Likelihood Maximization ; Maximum de vraisemblance des valeurs propres.
FITS	Flexible Image Transport System ; Le format FITS, créé par la NASA et défini par l'IAU (International Astronomical Union), est un format dédié au stockage et à la manipulation de données scientifiques.
GDAL	Geospatial Data Abstraction Library ; Bibliothèque libre de traitement d'images.
GDB	GNU Project Debugger ; Débuggeur du projet GNU.
HMA	Hyperspectral Multi-Angular images ; Images hyperspectrales multiangulaires.
IDL	Interactive Data Language ; Langage de programmation appartenant à la société ITT.
IHM	Interface Homme Machine.
ISIS	Integrated Software for Imagers and Spectrometers ; Ensemble de logiciels permettant le traitement d'images au format PDS.
ITK	Insight Segmentation and Registration Toolkit ; Bibliothèque de traitement d'images.
MIT	Massachusetts Institute of Technology ; Institut de technologie du Massachusetts.
MVC	Modèle Vue Contrôleur.

OMEGA	Observatoire pour la Minéralogie, l'Eau, les Glaces et l'Activité.
OSGeo	Fondation Geospatiale Open Source. Elle a été créée pour soutenir et construire une offre de logiciels libres en géomatique. http://www.osgeo.org/
OTB	Orfeo ToolBox ; bibliothèque développée dans le cadre du programme Orfeo du CNES.
PDS	Planetary Data System ; Format d'archives pour des données scientifiques.
SIG	Système d'Information Géographique.
SSA	Séparation de Sources en Aveugle ; opération consistant à estimer N sources inconnues à partir d'un jeu de P observations.
UML	Unified Modeling Language ; Langage de modélisation unifié permettant de décrire la conception et le fonctionnement d'un logiciel.
Vahiné	Visualization and analysis of multi-dimensional hyperspectral images in Astrophysics ; Visualisation et analyse d'images hyperspectrales multidimensionnelles en Astrophysique.
VCA	Vertex Component Analysis ; Analyse de composants par vertex.
XML	Extensible Markup Language ; Langage extensible de balisage.
XP	eXtrem Programming ; Méthode de développement logiciel faisant partie des méthodes dites « agiles ».
XSLT	eXtensible Stylesheet Language Transformations ; Langage de transformation XML.

Centres de recherche et laboratoires

ANR	Agence Nationale de la Recherche.
CNES	Centre National d'Études Spatiales.
CNRS	Centre National de la Recherche Scientifique.
ESA	European Space Agency ; L'agence spatiale européenne.
GIPSA Lab	Grenoble Images Parole Signal Automatique
INRIA	Institut National de Recherche en Informatique et Automatique
IPAG	Institut de Planétologie et d'Astrophysique de Grenoble.
JAXA	Japan Aerospace eXploration Agency ; L'agence spatiale nationale du Japon.
LPG	Laboratoire de Planétologie de Grenoble.
NASA	National Aeronautics and Space Administration ; L'agence spatiale étasunienne.
OSUG	Observatoire des Sciences de l'Univers de Grenoble.
UJF	Université Joseph Fourier.

Introduction

Ce document constitue le rapport de fin de mémoire menant à l'obtention du titre d'ingénieur du Cnam. Il présente le travail réalisé depuis avril 2010 au sein du laboratoire de planétologie de Grenoble. Ce travail s'inscrit dans le projet Vahiné que nous détaillerons dans cette introduction.

Contexte

Laboratoire de Planétologie de Grenoble

Le Laboratoire de Planétologie de Grenoble (LPG), fondé en 1999 est situé sur le domaine universitaire de Saint-Martin d'Hères. C'est une unité mixte de recherche du CNRS et de l'Université Joseph Fourier (UJF) ainsi qu'une composante de recherche de l'Observatoire des Sciences de l'Univers de Grenoble (OSUG). Il regroupe une quarantaine de personnes (chercheurs, étudiants, ingénieurs et personnel administratif). Le LPG est en pleine fusion avec le laboratoire d'astrophysique de Grenoble (LAOG) pour donner naissance depuis le 1er janvier 2011 à l'Institut de Planétologie et d'Astrophysique de Grenoble (IPAG).

La recherche effectuée au Laboratoire de Planétologie de Grenoble couvre un vaste domaine englobant l'étude des planètes, des comètes et des météorites du système solaire. Elle se fait dans le cadre de l'exploration du système solaire par des équipements embarqués sur les sondes spatiales lancées par les agences telles que le CNES [1], l'ESA [2], la JAXA [3] et la NASA [4]. Les chercheurs du laboratoire sont impliqués dans la préparation et l'interprétation des mesures réalisées par les missions telles que Cassini-Huygens vers Saturne, Mars Express ou encore Venus Express.

Les recherches s'organisent autour de trois thèmes :

- les hautes atmosphères planétaires,
- l'étude des surfaces, subsurfaces et noyaux cométaires,
- la matière moléculaire solide du système solaire.

C'est à ce dernier thème de recherche que le projet de ce mémoire est rattaché. Cet axe de recherche a pour but d'étudier la surface des planètes et des satellites ainsi que la matière constituant les comètes et les météorites. Cette matière nous renseigne sur l'origine et l'évolution ultérieure de ces objets. La planète Mars est l'un des sujets de prédilection du laboratoire. Ses glaces en particulier participent aux grands cycles qui conditionnent le climat de Mars à différentes échelles temporelles. Nous le verrons tout au long de ce mémoire les images

1. Centre national d'études spatiales
2. Agence spatiale européenne
3. Agence spatiale japonaise
4. Agence spatiale étasuniennes

traitées informatiquement sont toutes issues de missions martiennes. Un enjeu important de l'équipe « matière moléculaire » est donc de pouvoir traiter les images issues des spectro-imageurs† des missions martiennes. Ces images sont particulières et sont appelées images hyperspectrales (Voir la section 1.1).

Imagerie hyperspectrale

L'imagerie hyperspectrale visible et infrarouge est une technique importante pour l'étude et le suivi des planètes (Terre comprise) et de l'Univers. Elle est utilisée pour conduire une grande variété d'investigation comme la caractérisation de surfaces planétaires ou de matière en laboratoire.

Cette technique fournie alors un grand nombre d'images à trois dimensions (deux spatiales et une spectrale). La dimension spectrale permet d'accéder aux propriétés chimiques, physiques des matériaux observés. Les derniers progrès technologiques apportent même une dimension supplémentaire aux images, une mesure angulaire. Ces dernières images sont appelées images hyperspectrales multiangulaires (HMA, voir l'annexe A.2). Celles-ci permettent physiquement de pouvoir mieux caractériser les propriétés structurelles des matériaux planétaires et de séparer les contributions du sol et de l'atmosphère dans le signal mesuré par le capteur.

La contrepartie de ces nouveaux instruments est l'accentuation de la taille et de la complexité des données de télédétection. Ces données requièrent des algorithmes d'analyse physique et statistique de plus en plus performants. C'est dans ce cadre que le projet Vahiné a été lancé.

Projet VAHINE

Ce projet dont l'acronyme signifie (en anglais) « Visualization and analysis of multi-dimensional hyperspectral images in Astrophysics » qui peut se traduire par « Visualisation et analyse d'image hyperspectrales multidimensionnelles en Astrophysique » a pour ambition :

- de développer des modèles physiques et paramétriques, des algorithmes statistiques et de traitement du signal, ainsi que des logiciels pour traiter efficacement les données des spectro-imageurs embarqués ;
- de contribuer à coordonner les efforts d'un communauté actuellement dispersée entre différents domaines de compétence (science de la terre, mathématique, etc.).

La principale valorisation du projet, outre la publication d'articles scientifiques, est la réalisation d'un logiciel dédié à la visualisation et à l'analyse d'images multidimensionnelles (principalement hyperspectrales mais aussi HMA) en planétologie.

Problématique

Le but étant la mise à disposition d'un outil logiciel, nous examinons dans la suite les différents problèmes qui apparaissent.

Nous avons abordé précédemment une première problématique qui est la complexité des données. Nous avons à traiter des images multispectrales ou hyperspectrales voir multiangulaires. Les images à traiter ont des tailles allant ainsi suivant leurs complexités ou leurs grandeurs spatiales de quelques dizaine de mega-octets à plusieurs giga-octets.

Leur format est aussi particulier. De nombreux formats en imagerie satellitaire ont été créés : GeoTIFF, Erdas Imagine, SDTS, ... En planétologie un format domine c'est le format

« Planetary Data System » , PDS. Pour les images de type astrophysique (objet lointain notamment) on retrouve couramment le format « Flexible Image Transport System » , FITS.

Concernant les images planétaires une contrainte s'ajoute, due au fait qu'elles peuvent être géoréférencées†. C'est un élément à garder à l'esprit pendant notre phase d'analyse car il aura un impact sur les choix techniques.

Maintenant nous changeons de niveau pour examiner les traitements. Nous voulons un logiciel assez souple pour agréger les algorithmes issus de la recherche en traitement de l'image et signal hyperspectral. Ces traitements et algorithmes produisent des produits que les scientifiques planétologues veulent interpréter et comparer. Cela nous amène à notre dernière problématique : la visualisation. Idéalement le scientifique planétologue aimerait disposer d'un seul outil lui permettant d'effectuer les traitements, de pouvoir lire et écrire de façon optimisée la plupart des formats d'images, d'accéder aux méta-données et de visualiser le résultat de ses analyses.

Nous avons défini dans le cadre du projet Vahiné deux groupes de travaux : le premier a pour tâche le développement et l'implémentation de modules de traitement d'images hyperspectrales pour remonter à la distribution spatiale des matériaux planétaires et à leur propriétés physico-chimiques.

Le deuxième vise à développer un système de visualisation qui devra offrir à l'utilisateur la possibilité d'avoir une vision synthétique de l'information astrophysique contenue dans une image ou un groupe d'images et d'interagir facilement avec elle.

C'est dans ce groupe que s'inscrit le travail de ce mémoire.

Solutions envisagées

Par rapport aux problématiques et à l'environnement décrit précédemment nous pouvons définir trois grands ensembles :

- permettre l'accès aux données de type PDS en lecture et écriture ;
- opérer le portage et le développement du plus grand nombre possible d'algorithmes de traitement d'images hyperspectrales ;
- créer une interface pour manipuler et visualiser à la fois les algorithmes et les images.

Les deux premiers points sont développés comme un ensemble de composants logiciels nommé « Vahiné Framework ». Quand au dernier point, il vise à obtenir un logiciel de visualisation nommé « Vahiné View ».

Pendant ce travail de mémoire j'ai réalisé partiellement ou entièrement ces trois ensembles de tâches.

Un enjeu communautaire

Devant les grandes quantités de données issues des nouveaux instruments, un des enjeux est de pouvoir les traiter rapidement et d'en faciliter l'analyse. Dans le cadre du déploiement du système d'observation de la Terre « Pleiades », le CNES a par exemple mis en place un projet ambitieux l'OrfeoToolBox (OTB), une bibliothèque libre (« open source† ») de traitement d'images. La bibliothèque OTB est constituée d'un ensemble de briques algorithmiques qui capitalise les résultats obtenus dans les différentes actions de « Recherche et Technologie » internes ou commanditées par l'organisme. Nous avons choisi de nous appuyer sur les fonctionnalités de base de l'OTB pour bâtir notre logiciel Vahiné spécialisé en imagerie hyperspectrale. Nous sommes clairement dans l'esprit d'agréger les algorithmes développés en interne ou avec nos partenaires de recherche proches (GIPSA Lab, INRIA) mais aussi de mettre à disposition de la communauté planétologique un premier noyau logiciel sur lequel

pourraient se greffer d'autres processus de traitement. Un des objets de Vahiné est ainsi de contribuer à coordonner les efforts d'une communauté.

Ce point est essentiellement dépendant de la volonté des différents acteurs pour collaborer et développer des échanges de connaissances et de techniques. Mais il est aussi lié au type de technologie et au choix juridique que nous effectuons. Cela nous assujettit le type de licence pour diffuser nos travaux. Le choix d'une licence libre† dite « open source† » permet en effet une implication et contribution aisée de la communauté.

Plan du mémoire

Après ce chapitre introductif, ce mémoire est structuré en quatre chapitres et une conclusion.

Les trois premiers chapitres présentent mes réalisations. Le premier présente les types de données que nous devons traiter et leurs modes d'accès. Le second est lui consacré au traitement numérique réalisé sur les données. L'aspect mathématique y est brièvement abordé. Le troisième chapitre ayant trait au développement concerne l'aspect interface graphique. Celui-ci est introduit par un état des technologies disponibles pour manipuler nos images hyperspectrales.

Le quatrième chapitre abordera les méthodes agiles et les outils associés mis en œuvre dans ce travail de mémoire.

Nous conclurons ce mémoire, en synthétisant les travaux effectués sur chaque partie du projet Vahiné, ainsi que sur nos apports en termes méthodologiques. Je termine par un bilan personnel sur le cursus du CNAM et mes futures attentes professionnelles.

Chapitre 1

Imagerie hyperspectrale et accès aux données

1.1 Imagerie hyperspectrale

L'imagerie hyperspectrale est une technique permettant la représentation d'une même scène suivant de nombreuses bandes spectrales étroites dans des gammes de longueurs d'ondes variées (visible, infrarouge, etc.). C'est une technologie en plein développement qui permet d'accéder à de nombreuses informations sur les propriétés physiques des objets observés comparativement à l'imagerie couleur classique. L'imagerie hyperspectrale est utilisée dans de multiples domaines comme la géologie, l'écologie, l'urbanisme, la foresterie, l'agriculture, dans le domaine militaire ou encore en science des planètes et astrophysique.

Un capteur produisant des images multispectrales ou hyperspectrales est qualifié de spectro-imageur. Il comporte un système optique, un dispositif de sélection des longueurs d'onde et un dispositif d'enregistrement numérique du flux lumineux en général de type CCD.

Les données issues de la télédétection hyperspectrale sont agencées en cube. Un cube est constitué de deux dimensions spatiales notées x et y et d'une dimension spectrale λ. Le nombre de bandes spectrales ou spectels[†] dans la direction λ est noté N_λ. Chaque bande correspond à l'acquisition de la même surface mais à une longueur d'onde différente. Une bande spectrale correspond à un canal d'acquisition du spectro-imageur. Un exemple d'image hyperspectrale et le principe de sa formation sont présentés à la figure 1.1. On notera qu'il existe une imagerie dite « multispectrale » qui se contente d'une dizaine de canaux alors que l'imagerie « hyperspectrale » dépasse la centaine de canaux ($N_\lambda > 100$).

Les images hyperspectrales sont acquises par des spectro-imageurs associés à des microscopes (étude d'échantillon), à des satellites (études de la surface terrestre) ou encore à des sondes spatiales (études de la planète Mars par exemple).

Dans le domaine multispectral on trouvera par exemple, les satellites PLEIADES [1] du CNES qui permettront l'acquisition d'images multispectrales de 70 cm à 250 cm de résolution spatiale [CNES, 2009]. Chaque image aura un volume compris entre 1,8 et 3,6 Go pour seulement 5 canaux spectraux. Ce type d'instrument est développé pour avoir un bon rendu spatial.

Par comparaison dans le domaine hyperspectral nous avons des images fournies par le spectro-imageur OMEGA [ESA, 2009] [2] de l'ESA. Chaque image a une résolution spatiale comprise entre 350 m à 4000 m. Leur taille varie de 50 à 100 Mo pour environ 100000 spectres composés de 256 spectels.

1. Système dual d'observation optique de résolution métrique.
2. Observatoire pour la Minéralogie, l'Eau, les Glaces et l'Activité.

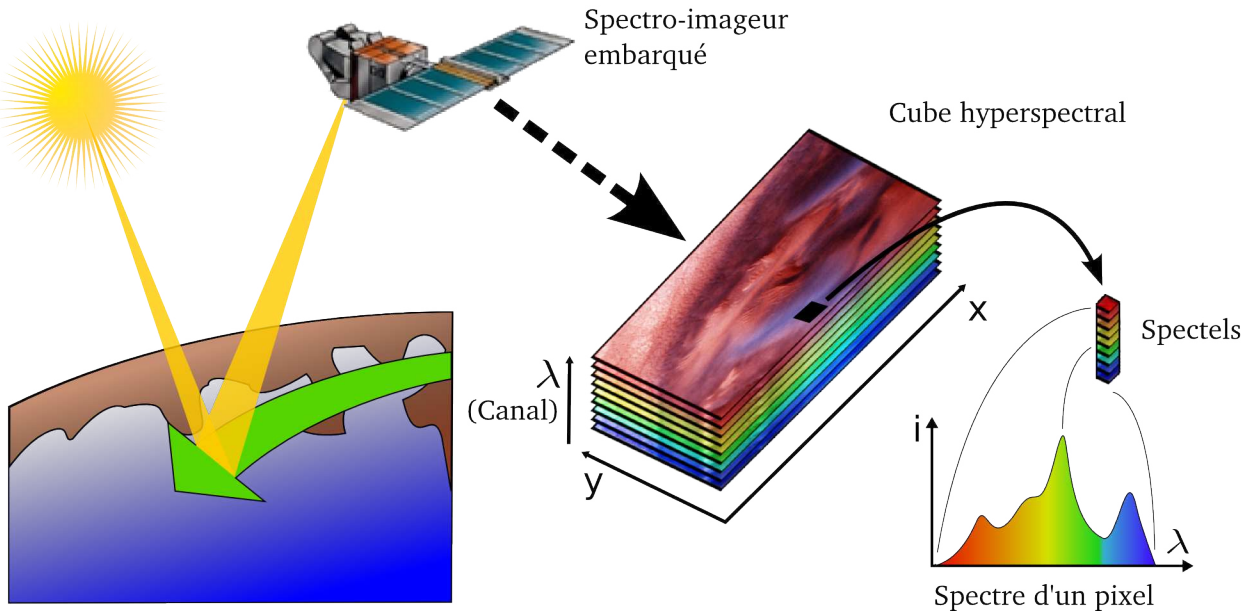

FIGURE 1.1 – Acquisition et décomposition d'une image hyperspectrale.

Les missions spatialee, ayant une durée de plusieurs années, le volume total de données généré est très important. Ainsi pour la mission OMEGA, le LPG dispose en interne de 3,3 tera-octets d'images brutes.

Ces exemples montrent que la taille et la complexité des données générées font apparaître de nouveaux défis à la fois mathématique et informatique pour l'analyse. Un des problèmes à résoudre est d'automatiser l'extraction d'informations pertinentes de ces images : détection de zones d'intérêts, distribution et caractérisation des matériaux, etc. Un élément d'automatisation est notamment l'algorithme ELM-VCA que j'ai implémenté et que je présente à la section 2.4.

1.2 Cubes hyperspectraux planétaires

Dans le domaine qui nous intéresse, la planétologie, les spectro-imageurs sont embarqués à bord des missions d'exploration du Système Solaire.

Voici une liste non exhaustive des spectro-imageurs passés et présents dont les données nous intéressent :

- OMEGA à bord de Mars Express (ESA) ;
- VIRTIS à bord de Vénus Express (ESA) ;
- CRISM à bord de Mars Reconnaissance Orbiter (NASA) ;
- NIMS à bord de Galileo ;
- VIMS à bord de Cassini.

Une image hyperspectrale planétaire comporte typiquement des centaines de milliers, voire des millions de pixels, chacun formant un spectre de plusieurs centaines de spectels. Le domaine spectral typique couvert est $0.4 - 5.0\,\mu$m, la résolution spectrale est de l'ordre de 10 nm/spectel. En fonction des conditions d'observation qui sont très variables d'une mission à l'autre, la résolution spatiale s'étend entre 10 m et 100 km/pixel. L'analyse physique est menée sur des données calibrées, organisées sous forme de cubes formatés selon la norme PDS. C'est cette norme que nous abordons dans la section suivante.

```
PDS_VERSION_ID=PDS3
/* File identification and structure */
RECORD_TYPE=FIXED_LENGTH
RECORD_BYTES=512
FILE_RECORDS=18550
LABEL_RECORDS=2
FILE_NAME=result_im41_GMM.cub
/* Pointers to Data Objects */
^QUBE=("result_im41_GMM.cub",1)
/* Qube structure */
OBJECT=QUBE
  AXES=3
  AXIS_NAME=(SAMPLE,LINE,BAND)
  /* Core description */
  CORE_ITEMS=(128,265,70)
  CORE_NAME="RAW DATA NUMBER"
  CORE_UNIT="N/A"
  CORE_ITEM_TYPE=PC_REAL
  CORE_ITEM_BYTES=4
  CORE_BASE=0.0
  CORE_MULTIPLIER=1.0
  /* Suffix description */
  SUFFIX_BYTES=4
  SUFFIX_ITEMS=( 0, 0, 0)
END_OBJECT=QUBE
END
```

Listing 1.1 – Extrait d'un fichier label PDS.

1.3 Norme « Planetary Data System »

Le format PDS est utilisé pour archiver des produits scientifiques (images, tableaux, données numériques ou textuelles) issus des missions planétaires. Ce format a été créé à l'époque de l'exploration lunaire et est toujours maintenu par la NASA. Nous allons dans les paragraphes qui suivent donner un bref aperçu de cette norme. Le lecteur pourra se référer pour plus de détails aux documents officiels [NASA, 2010].

Chaque produit est décrit par une liste d'étiquettes (mots clef PDS) qui fournissent des informations sur les types de données, les valeurs stockées et leur structuration. L'ensemble des étiquettes est listé dans un fichier de format texte (ASCII), dans une structure appelée « Label PDS ». Un exemple de « label » PDS est présenté sur le listing 1.1.

Les données archivées peuvent être de tout type, les plus usuelles étant des images et des tableaux de nombres. On notera que ce format permet d'ajouter par exemple, des données de calibration ou des informations de type cartographique comme un système de coordonnées de référence. Cela est utile pour positionner les images dans un système d'information géographique (SIG) .

Dans sa plus simple expression, un objet PDS consiste en une étiquette et les données de l'objet qu'il décrit. Nous pouvons aussi définir des produits complexes qui peuvent contenir plusieurs objets interdépendants. Une archive PDS peut donc contenir un objet primaire et un ou plusieurs objets secondaires, ou les deux. Un exemple schématique d'une archive est représenté sur la figure 1.2.

Un objet de données primaire est un ensemble de résultats d'une observation scientifique. Les objets primaires sont habituellement décrits en utilisant l'une des structures PDS suivantes : `TABLE, SPREADSHEET, IMAGE, SERIES, SPECTRUM, QUBE`. On remarquera que sur l'exemple du listing1.1 l'objet est de type `QUBE`. C'est le type le plus couramment utilisé en planétologie pour stocker les images hyperspectrales. D'ailleurs on parlera usuellement de « cube hyperspectral » .

Comme nous l'avons vu, afin d'identifier et de décrire l'organisation, le contenu et le format de chaque produit de données, le format PDS nécessite un label distinct du produit

FIGURE 1.2 – Label PDS attaché et détaché.

de données. Ces produits et labels peuvent être construits de deux façons différentes :

Attaché : le label PDS correspondant à un produit de données est disponible au début du fichier de données ;

Détaché : le label PDS est physiquement détaché du produit de données qu'il décrit et réside dans un fichier séparé qui contient un pointeur (le nom d'un fichier) vers le fichier de données. Le fichier label a en général le même nom que son fichier de données associées, mais avec l'extension `.LBL`.

Ces deux méthodes d'organisation d'un label et de ses données associées sont représentées sur la figure 1.2.

1.4 Objet cube hyperspectral

Nous détaillons maintenant un objet particulier, très utilisé dans le domaine de l'imagerie hyperspectrale. Un objet cube défini par le mot clef PDS « QUBE [3] » est un tableau multidimensionnel de valeurs réparties sur plusieurs dimensions. La partie principale notée « core » dans la norme PDS est homogène, et se compose d'octets représentant des nombres entiers non signés, signés ou des réels. Les objets de type « QUBE » peuvent avoir une zone de suffixe optionnel pour chaque axe. Les zones suffixes peuvent être hétérogènes, avec des éléments de types différents (flottant et entier par exemple). De plus, des valeurs particulières peuvent être définies pour la partie « core » et les zones de suffixe permettant de désigner les valeurs manquantes et plusieurs types de valeurs non valides comme par exemple, une saturation instrumentale sur un pixel donné.

Une spécialisation fréquente de l'objet « QUBE » est la norme « ISIS QUBE » , qui est un cube à trois dimensions composé de deux dimensions spatiales et une dimension spectrale. Ses trois axes peuvent être interprétés comme le triplet : (échantillon, ligne, bande). Les deux premières valeurs du triplet représentent les dimensions horizontales et verticales de l'image.

Enfin, la norme ISIS autorise trois types d'organisation de stockage physique :

BIL : signifiant « Band Interleaved by Line » traduisible par bande entrelacée par ligne. On stocke la $1^{\text{ère}}$ ligne pour la $1^{\text{ère}}$ bande puis la $1^{\text{ère}}$ligne pour la $2^{\text{ème.}}$bande, etc.

BIP : signifiant « Band Interleaved by Pixel » que l'on traduit par bande entrelacée par pixel. L'ensemble des spectres est stocké de façon séquentielle ligne par ligne et colonne par colonne.

BSQ : qui signifie « Band Sequential » que l'on traduit par bande séquentielle. On stocke le plan de toute la $1^{\text{ère.}}$ bande puis celui de la $2^{\text{ème.}}$, etc.

Pour illustrer les définitions précédentes le tableau 1.1 montre la différence entre les formats BIL, BIP et BSQ, pour une image classique à trois bandes (†RVB) (rouge, vert, bleu) de cinq pixels de large et trois pixels de haut.

3. dans la suite nous utiliserons indifféremment le mot « cube » et le mot clef « QUBE » pour désigner le même objet.

BIL	BIP	BSQ
R R R R R		R R R R R
V V V V V		R R R R R
B B B B B		R R R R R
R R R R R	R V B R V B R V B R V B R V B	V V V V V
V V V V V	R V B R V B R V B R V B R V B	V V V V V
B B B B B	R V B R V B R V B R V B R V B	V V V V V
R R R R R		B B B B B
V V V V V		B B B B B
B B B B B		B B B B B

TABLE 1.1 – Différences d'organisation des données pour les représentations BIL, BIP et BSQ.

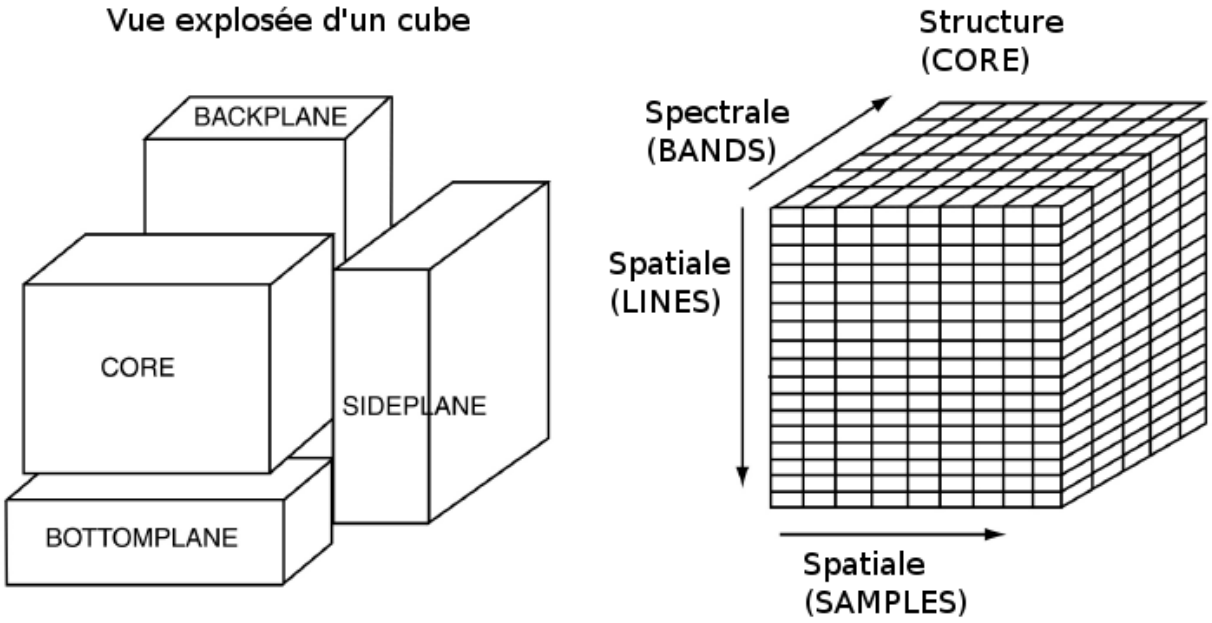

FIGURE 1.3 – Structure générale d'un objet cube : `QUBE`.

Un « QUBE » typique ISIS est un cube hyperspectral contenant des données issues d'un spectro-imageur. Un tel cube est à la fois un ensemble d'images (à différentes longueurs d'onde) de la même zone cible, et un ensemble de spectres pour chaque point de la zone cible. Généralement, le suffixe ou « backplane » d'un tel cube contient des informations géométriques ou sur la qualité de chaque spectre individuel. Le schéma 1.3 illustre la structure générale d'un tel « QUBE » standard ISIS.

1.5 Bases de spectres synthétiques

En planétologie de nombreux algorithmes utilisent une ou plusieurs bases dites d'apprentissage ou bases de spectres synthétiques. Celles-ci sont constituées de spectres calculés. Suivant l'étude à réaliser nous choisissons des paramètres physiques libres à faire varier

(abondances, taille de grains[4], etc.). Nous choisissons aussi une représentation de surface pour chaque unité de terrain : corps purs présents, modes de mélange (géographique, granulaire, stratification, etc.), texture (granulaire ou compacte). La localisation dans l'image des diverses unités de terrains physiques est gérée par un ensemble de masques† binaires. L'ensemble constitue un modèle physique.

On notera qu'en général un modèle est pertinent seulement pour une partie de l'image, un planétologue manipule donc souvent des masques spatiaux. Tout algorithme et a fortiori logiciel de traitement devra être capable de gérer ces masques. Une fois le modèle physique défini, on utilise des codes numériques (de transfert radiatif typiquement) pour générer un ensemble de spectres. Un spectre correspond à une combinaison de paramètres libres et à une géométrie d'acquisition.

Les spectres résultats sont ceux que mesureraient théoriquement l'instrument pour la zone composée de pixels possédant ces propriétés physiques et étant observé dans cette géométrie. L'ensemble des spectres synthétiques X est organisé dans un cube au format PDS de dimension $(n \times m \times d)$, n nombre de combinaisons de valeurs pour les paramètres physiques, m nombre total de géométries et d nombre de spectels.

Le cube de spectres synthétiques est accompagné d'un cube au format PDS de données auxiliaires de dimension $(n, m, p + q)$ qui donne pour chaque spectre de coordonnées (n, m) la valeur des q paramètres géométriques et la valeur des p paramètres physiques Y, qui ont été utilisées pour son calcul.

Un ensemble de spectres synthétiques (un cube et son cube de paramètres) n'est relatif qu'à une unité de terrain donnée. Donc, l'analyse physique procède terrain par terrain soit généralement uniquement pour une portion de l'image (on utilise souvent un masquage spatial. *Voir l'annexe A.3*) .

1.6 Accès aux données

Après avoir détaillé les données sur lesquelles nos algorithmes et logiciels travaillent, nous voyons comment nous pouvons les manipuler. La bibliothèque la plus usitée pour la manipulation d'images de type satellitaires est GDAL, « Geospatial Data Abstraction Library ».

1.6.1 Introduction à GDAL

GDAL est une bibliothèque permettant l'utilisation des formats matriciels† (raster†) de données géospatiales. Elle est publiée sous la licence libre MIT par l'Open Source Geospatial Foundation (OSGeo). Cette bibliothèque propose une abstraction aux différents formats d'images. Tous les formats pris en charge sont représentés par un seul modèle de données (ou « Dataset » en anglais) . Cette bibliothèque est également livrée avec plusieurs utilitaires en ligne de commande utile pour la traduction et le traitement des données.

La bibliothèque associée OGR (disponible avec les source de GDAL) propose des fonctions similaires pour des données vectorielles.

Le fonctionnement de GDAL et notamment de ses pilotes est décrit en détails dans les documents « GDAL Data Model », [GDAL, 2010a] et « GDAL Driver Implementation Tutorial », [GDAL, 2010b].

4. Actuellement c'est la surface de Mars qui est principalement modélisée par un mélange de type granulaire. Voir le paragraphe 2.4.1 et la figure 2.5

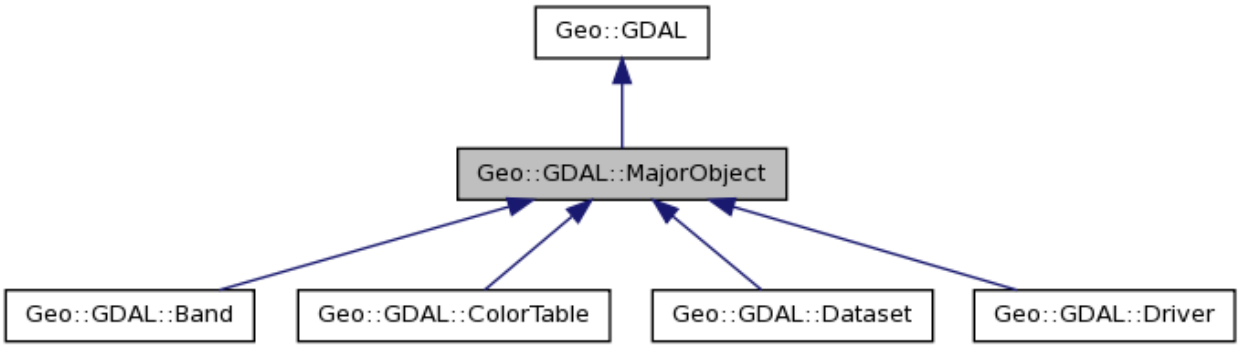

FIGURE 1.4 – Représentation de l'héritage des différentes classes de base de GDAL

1.6.2 Modèle de données ou « dataset »

Un élément central dans la bibliothèque GDAL est la classe `GDALDataset`. Elle représente un ensemble de données issu d'un fichier image sous la forme d'une liste de bandes et quelques informations communes à l'ensemble des bandes. En particulier, à cet ensemble de données est associé une notion de taille (un nombre de pixels et de lignes) qui s'applique à toutes les bandes. La classe `GDALDataset` est également responsable du géoréférencement et notamment d'effectuer les changements de coordonnées pour chaque pixel de chaque bande. Cet ensemble de données peut aussi avoir des méta-données associées. Ces dernières se présentent comme une liste de paires (nom, valeur). Chaque paire est représentée sous forme de chaînes :

$$\{\text{"clef1=valeur1"},\ \text{"clef2=valeur2"},\ ...\}$$

En général les nouveaux formats sont ajoutés à GDAL en mettant en œuvre des sous-classes de `GDALDataset` et de la classe `GDALRasterBand` dans le cas du traitement d'un format matriciel ou raster†.

1.6.3 RasterBand

Une bande est représentée dans GDAL avec la classe `GDALRasterBand`. Elle représente une unique bande, canal ou couche. Elle ne représente donc pas nécessairement une image entière. Par exemple, une image 24 bits RVB devrait normalement être représentée comme un ensemble de trois bandes : une pour le rouge, une pour le vert et une pour le bleu. La classe `GDALRasterBand` a les propriétés suivantes.

- Une largeur en pixel et une hauteur en nombre de lignes. Ce sont les mêmes que celles définies pour l'ensemble des données, si chaque bande a la même résolution.
- Un type de données défini comme un énuméré `GDALDataType`. On trouve par exemple, les valeurs : `Byte, UInt16, Float32, Float64` et des types complexes.
- Une taille de bloc. Il s'agit de choisir une taille de bloc pour l'accès aux données. Pour des images en mosaïque, le bloc sera l'équivalent d'une tuile†. Pour des images classiques ce sera normalement une ligne.
- Une liste de méta-données par paire (dans le même format que décrit précédemment), contenant de l'information spécifique à cette bande.

1.6.4 Implémentation du pilote

Concernant l'implémentation nous ne partons pas de rien. En effet GDAL contient déjà plusieurs pilotes PDS. Pour être plus précis un pilote PDS, un pilote à la norme ISIS2 et un

dernier pilote suivant la norme ISIS3. Le projet Vahiné utilise dans un premier temps seulement des images OMEGA et CRISM dont le stockage est au format PDS norme ISIS2. J'ai modifié en conséquence uniquement le pilote ISIS2 pour améliorer la lecture des fichiers PDS et supporter l'écriture dans ce format. Plusieurs problèmes ont été rencontrés notamment du fait que de nombreuses images ne respectent pas strictement la norme PDS.

Ma première implémentation était limitée, notamment par les points suivants :

- elle ne gérait que les flottants 32 bits (le type usuel en usage au laboratoire) ;
- seuls les fichiers PDS de type détaché (Fig. 1.2) étaient pris en charge ;
- seule la représentation « little endian » était gérée ;
- le code n'était pas assez modulaire en vue d'une fusion des pilotes PDS standards et PDS norme ISIS2.

Cette version du code implémenté n'étant pas optimale, j'ai effectué une deuxième itération avec une refonte complète de sa structure.

Suite aux dernières modifications une livraison officielle a été faite début décembre 2010 à l'équipe des développeurs GDAL de l'OSGeo. Ces développements indispensables au projet Vahiné que nous redistribuons à la communauté ont nécessité environ un mois et demi de travail dans le cadre de ce mémoire.

1.6.4.1 Implémentation du dataset

Le pilote ISIS2 dérive de la classe `RawDataset` déclarée dans `gdal/frmts/raw`. Il est implémenté dans `gdal/frmts/pds/isis2dataset.cpp`. La gestion du format PDS en version standard (non ISIS) est contenue dans le fichier `gdal/frmts/pds/pdsdataset.cpp`, celle pour ISIS3 est dans `gdal/frmts/pds/isis3dataset.cpp`.

Le listing 1.2 présente un extrait du code source du dataset. Nous détaillons quelques points :

① A partir du dataset existant j'ai d'abord ajouté une classe pour la gestion d'erreurs qui dérive de la bibliothèque standard C++.

② La classe `ISIS2Dataset` hérite de la classe GDAL de base `RawDataset`.

③ et ④ Nous définissons deux structures qui permettent de manipuler plus facilement les dimensions d'un cube hyperspectral. Ces deux structures reprennent les attributs d'un cube tel que défini sur la figure 1.3.

Globalement notre implémentation du dataset redéfinit de nombreuses méthodes déclarées virtuelles dans la classe de base `GDALDataset`, mais nous avons une méthode particulière la méthode `Open()` dont la déclaration figure au point ⑤. Celle-ci n'est pas une méthode virtuelle dans la classe de base, elle est déclarée statique. Nous verrons l'intérêt au paragraphe 1.6.4.3. La méthode `Open()` va faire appel à plusieurs sous-méthodes spécialisées (`OpenQUB()`, `OpenImage()`) pour gérer l'ouverture des différents objets PDS. La première étape de la méthode `Open` est de vérifier que le fichier transmis est bien du type géré par ce pilote. Il est important de réaliser que chaque méthode `Open` de chaque pilote est appelée à tour de rôle jusqu'à ce que l'une d'elle confirme qu'elle sait gérer le fichier. Dans le cas où le pilote ne sait pas gérer le fichier il doit retourner uniquement la valeur `NULL`.

De manière générale les pilotes ne doivent produire une erreur que si le fichier correspond au format pris en charge et que le fichier est par exemple corrompu.

La méthode `Open` reçoit en paramètre un pointeur sur un objet `GDALOpenInfo`, ce dernier contient, le premier kilo octet lu du fichier à ouvrir. On peut voir sur l'extrait du code 1.3 que nous sommes donc obligés de tester non seulement la présence d'objet de type image mais aussi des objets n'ayant rien à voir comme des histogrammes. Cela est lié à la définition même du format PDS qui permet un stockage d'objet hétérogène dans un seul fichier comme expliqué à la section 1.3.

```
class ISIS2DatasetError : public std::exception                                    ①
{
public:
  ISIS2DatasetError(CPLString description="ISIS2Dataset internal error") throw() :
    sDescription(description){}

  private:
  CPLString sDescription;

  virtual const char* what() const throw(){
    return sDescription;
  }
};

class ISIS2Dataset : public RawDataset                                             ②
{
    FILE *fpImage; // image data file.

    NASAKeywordHandler oKeywords;

    int         bGotTransform;
    double      adfGeoTransform[6];

    CPLString  osProjection;

    int  parse_label(const char *file, char *keyword, char *value);
    int  strstrip (char instr [], char outstr [], int position);

    struct DataSize {                                                              ③
      unsigned int nSample;
      unsigned int nLine;
      unsigned int nBand;
    };

    struct SuffixSize {                                                            ④
      unsigned int nX; // side plane
      unsigned int nY; // bottom plane
      unsigned int nZ; // back plane
    };

    static GDALDataset *Open( GDALOpenInfo * );                                    ⑤
    static GDALDataset *Create( const char * pszFilename,
                                int nXSize, int nYSize, int nBands,
                                GDALDataType eType, char ** papszParmList );
};
```

Listing 1.2 – Extrait du pilote ISIS2.

```
...
if( strstr ((const char *)poOpenInfo->pabyHeader,"^QUBE") == NULL &&
      strstr ((const char *)poOpenInfo->pabyHeader,"^IMAGE") == NULL &&
      strstr ((const char *)poOpenInfo->pabyHeader,"^SPECTRAL_QUBE") == NULL &&
      // not an image but an PDS data object
      strstr ((const char *)poOpenInfo->pabyHeader,"^HISTORY") == NULL &&
      strstr ((const char *)poOpenInfo->pabyHeader,"^HISTOGRAM") == NULL &&
      strstr ((const char *)poOpenInfo->pabyHeader,"^PALETTE") == NULL &&
      strstr ((const char *)poOpenInfo->pabyHeader,"^HEADER") == NULL )
  {
    CPLDebug("header ",CPLString().Printf("nHeaderBytes = %d",poOpenInfo->nHeaderBytes), (const char *)
        poOpenInfo->pabyHeader);
      return NULL;
  }
...
```

Listing 1.3 – Extrait de la méthode `Open()` du pilote ISIS2.

```
...
CPLString sLayout("BSQ"); //default to band seq.
CPLString sValue = poDS->GetKeyword( "QUBE.AXIS_NAME", "(SAMPLE,LINE,BAND)" );                    ⑥
if (EQUAL(sValue,"(SAMPLE,LINE,BAND)")
  sLayout = "BSQ"
  dataSize.nSample = atoi(poDS->GetKeywordSub("QUBE.CORE_ITEMS",1));                               ⑦
  dataSize.nLine = atoi(poDS->GetKeywordSub("QUBE.CORE_ITEMS",2));
  dataSize.nBand = atoi(poDS->GetKeywordSub("QUBE.CORE_ITEMS",3));
...
```

Listing 1.4 – Sélection des dimensions du cube. Extrait du pilote ISIS2.

```
...
static CPLString GetSampleType(CPLString);
static void GetSampleFormat(int itype, CPLString sDataType, GDALDataType & eDataType, double & dfNoData, bool
      & bNoDataSet, bool bByteUnit);
static char GetWordOrder(CPLString sSampleType);
...
```

Listing 1.5 – Déclaration des méthodes permettant le traitement du type de donnée contenues dans le cube. Extrait du pilote ISIS2.

Il serait trop long et fastidieux de décrire l'ensemble des opérations réalisées par le pilote. Nous allons maintenant nous focaliser sur quelques points que j'ai développé complètement ou remanié en profondeur par rapport au pilote originel de l'équipe GDAL. Ainsi nous n'expliquerons pas le système de géoréférencement, tout simplement car nous ne l'utilisons pas encore dans les traitements Vahiné. Nos traitements numériques sont indépendants de la position géographique des images.

La gestion des différents types d'organisation du « core » d'un cube n'a été que partiellement modifiée. Elle se fait via l'analyse des méta-données et plus précisément de la chaîne `AXIS_NAME` comme le montre le point ⑥ de l'extrait de code 1.4.

Comme expliqué dans la section 1.3, il y a trois possibilités d'organisation qui sont définies par trois valeurs possibles de la chaîne `AXIS_NAME`. Dans le même temps, j'en profite pour remplir la structure de données `dataSize` avec les dimensions de notre hypercube. Par exemple, pour une organisation de type BSQ nous avons les trois lignes situées au point ⑦.

Un autre élément important est la gestion des valeurs nulles. Celle-ci a été améliorée pour éviter d'avoir des effets de bords numériques catastrophiques. Par exemple, en traitant une valeur nulle comme une mesure classique nous pouvons nous retrouver avec des valeurs ayant plusieurs ordres de grandeurs différents (une valeur nulle égale à -1.10^6 et les valeurs des données dans la plage $[-1, 1]$).

J'ai aussi ajouté la gestion complète des différents types de donnés supportés par la norme PDS grâce à l'implémentation des méthodes décritent dans le listing 1.5.

Ces méthodes permettent la détection et gestion de la représentation interne des types : l'endianness†, le format des données (`INTEGER, UNSIGNED_INTEGER, REAL`, etc.) et leur taille (`GDT_Byte, GDT_Int32`, etc.). Les trois méthodes sont basées sur la reconnaissance de chaînes contenues dans les méta-données.

1.6.4.2 Création de fichiers

Il existe deux approches pour la création d'un fichier dans GDAL. La première méthode est appelée `CreateCopy` et implique la mise en œuvre d'une fonction qui puisse écrire un

```
...
/**
 * Creation Options:
 * INTERLEAVE=BSQ/BIP/BIL: Force the generation specified type of interleaving.
 *   BSQ --- band sequental (default),
 *   BIP --- band interleaved by pixel,
 *   BIL --- band interleaved by line.
 * LABELING_METHOD=attached/detached
 * LABEL_EXTENSION=???, if null default is "pds"
 * OBJECT=QUBE/IMAGE/SPECTRAL_QUBE, if null default is QUBE
 */

GDALDataset *ISIS2Dataset::Create(const char* pszFilename,
    int nXSize, int nYSize, int nBands,
    GDALDataType eType, char** papszParmList) {
...
 try{
    GUIntBig iRecords = ISIS2Dataset::RecordSizeCalculation(nXSize, nYSize, nBands, eType);
  CPLDebug("ISIS2","irecord = %i",static_cast<int>(iRecords));
    GUIntBig iLabelRecords(2);
    if( bAttachedLabelingMethod ) {                                                    ⑧
      ISIS2Dataset::WriteLabel(osRasterFile, "", sObject, nXSize, nYSize, nBands, eType, iRecords, pszInterleaving,
          iLabelRecords, true);
    }else{
      ISIS2Dataset::WriteLabel(osLabelFile, osRasterFile, sObject, nXSize, nYSize, nBands, eType, iRecords,
          pszInterleaving, iLabelRecords);
    }
    ISIS2Dataset::WriteRaster(osRasterFile, bAttachedLabelingMethod, iRecords, iLabelRecords, eType, pszInterleaving);
  }catch (ISIS2DatasetError){
    CPLError(CE_Failure, CPLE_AppDefined, "Error on WriteRaster or Label");
    return NULL;
  }

  return (GDALDataset *) GDALOpen( osOutFile, GA_Update );
...
```

Listing 1.6 – Fonction de création d'un cube. Extrait du pilote ISIS2.

fichier dans le format voulu, en récupérant l'ensemble des informations nécessaires à partir d'un objet `GDALDataset`. La seconde, est une méthode de création dynamique qui consiste à mettre en œuvre la méthode `Create` pour créer la structure du fichier (une sorte de coquille vide mais valide). Puis, l'application complète le fichier de sortie par des appels successifs à la méthode `WriteBlock` de la classe `GDALRasterBand`. Cette méthode accède à un bloc mémoire d'une bande « raster » sans rééchantillonnage, ou conversion et est chargée de l'écriture d'une façon qui se veut « efficace ».

Les avantages de la première méthode est que toutes les informations sont disponibles au moment ou le fichier de sortie est créé. Elle peut accéder ainsi à des informations comme les valeurs extrêmes de l'image, les indications de géoréférencement, etc. L'inconvénient est que l'empreinte mémoire est équivalente à la taille du fichier de sortie.

La seconde méthode permet la création dynamique. Elle a l'avantage d'avoir besoin seulement de connaître : la taille, le nombre de bandes et le type de données affecté à chaque pixel. Ainsi une application peut créer un fichier vide et écrire les résultats au fur et à mesure qu'ils deviennent disponibles. La mémoire nécessaire est moindre, c'est un avantage non négligeable quand la taille des fichiers atteint plusieurs centaines de méga-octets.

L'équipe de développement de GDAL conseille pour les formats très importants que les deux méthodes soient mises en œuvre. Dans notre cas, j'ai seulement implémenté la seconde solution.

Le listing 1.6 présente un extrait du code de création. La création du fichier passe par le choix (point ⑧) du type d'organisation des données avec un fichier « label » attaché ou détaché (cf section 1.3). Les méthodes `WriteRaster` et `WriteLabel` sont reproduites en annexe (B.1) à titre informatif. Notre pilote est maintenant prêt à pouvoir lire et écrire des

fichiers de type PDS. Nous allons voir comment le finaliser.

1.6.4.3 Intégration du pilote

Pour chaque format, une instance† de la classe `GDALDriver` est créée et enregistrée avec le gestionnaire `GDALDriverManager` pour s'assurer que la bibliothèque connaisse et traite le format décrit.

Nous nous intéressons à la fonction `GDALRegister_ISIS2` reproduite sur le listing 1.7. Cette fonction permet d'enregistrer notre pilote au près de la classe `GDALDriverManager` . Lors d'un premier appel, une instance de la classe `GDALDriver` est créée (cf point ①). La bibliothèque GDAL permet de fixer plusieurs attributs du pilote avant de l'enregistrer auprès de la classe `GDALDriverManager()`. Ainsi à partir du point ② du listing on peut définir notamment les attributs suivants :

SetDescription() : La description est le nom court désignant le format géré par ce pilote. Il s'agit d'un nom unique pour identifier le pilote. Il correspond habituellement au préfixe de la classe définissant le format.

GDAL_DMD_LONGNAME : C'est une chaîne de caractères permettant une description plus longue concernant le format du pilote.

GDAL_DMD_HELPTOPIC : C'est le nom de la rubrique d'aide au format HTML[5] à afficher pour ce pilote.

GDAL_DMD_CREATIONOPTIONLIST : C'est une chaîne de caractères définissant l'ensemble des options de création acceptées par ce pilote.

GDAL_DMD_CREATIONDATATYPES : Permet de définir la liste des types de données pris en charge lors de la création de nouveaux ensembles de données par les fonctions de création. Par exemple, un entier sur deux octets noté « Int16 ».

Une fois les options décrites les instructions situées au point ③ et suivant affectent les fonctions déclarées « static » aux pointeurs des fonctions du pilote. Celui-ci est donc maintenant en mesure d'appeler la fonction `Open` ou `Create` selon les besoins.

pfnOpen : Définit un pointeur sur la fonction à appeler pour tenter l'ouverture d'un fichier de ce format.

pfnCreate : Défini un pointeur sur la fonction à appeler pour créer un nouveau fichier au format ISIS2 dans notre cas.

pfnCreateCopy : Permet de définir la fonction à appeler pour créer un nouveau fichier à partir d'un ensemble de données issues d'une autre source. Dans notre cas cette méthode n'étant pas implémentée ce pointeur est positionné à la valeur `NULL`.

pfnDelete : Enfin ce dernier pointeur de fonction permet de définir une fonction à appeler pour supprimer un ensemble de données. Dans notre cas nous n'avons pas d'opération particulière a effectuer sur une destruction du « Dataset » donc il reste à sa valeur par défaut `NULL`.

1.6.5 Analyseur syntaxique de méta-données

Nous avons vu dans les paragraphes précédents de nombreuses références aux mots clefs contenus dans les méta-données. Ce sont des éléments importants dans le traitement d'un fichier à la norme PDS.

5. Hypertext Markup Language

```
void GDALRegister_ISIS2()
{
    GDALDriver *poDriver;

    if( GDALGetDriverByName( "ISIS2" ) == NULL )
    {
        poDriver = new GDALDriver();                                          ①

        poDriver->SetDescription( "ISIS2" );                                  ②
        poDriver->SetMetadataItem( GDAL_DMD_LONGNAME,
                                   "USGS Astrogeology ISIS cube (Version 2)" );
        poDriver->SetMetadataItem( GDAL_DMD_HELPTOPIC,
                                   "frmt_various.html#ISIS2" );
    poDriver->SetMetadataItem( GDAL_DMD_CREATIONDATATYPES, "Byte Int16 Int32 Float32 Float64");

        poDriver->pfnOpen = ISIS2Dataset::Open;                               ③
    poDriver->pfnCreate = ISIS2Dataset::Create;
        poDriver->pfnCreateCopy = NULL;

        GetGDALDriverManager()->RegisterDriver( poDriver );
    }
}
```

Listing 1.7 – Méthode d'enregistrement du pilote ISIS2. Extrait du pilote ISIS2.

L'analyse lexicale de l'entête PDS est dévolue à la classe `NASAKeywordHandler`. Celle-ci est implémentée dans deux fichiers `frmts/pds/nasakeywordhandler.cpp` et `frmts/pds/nasakeywordhandler.h`. La classe est utilisée à la fois par le pilote ISIS2 et par le pilote PDS de base. Ce *parseur* qui a été implémenté de façon ad hoc pour GDAL est loin d'être parfait et est très sensible à la qualité des entêtes. Il manque clairement de robustesse et a souvent été la cause de défaillances des jeux de tests. Pour améliorer cela, il serait envisageable d'intégrer la bibliothèque Idaeim (http://pirlwww.lpl.arizona.edu/software/idaeim/PVL/) qui paraît plus robuste. J'ai établi un premier contact avec le développeur de Idaeim qui pourrait fournir du support à l'intégration de son code dans GDAL.

Devant le manque de temps, j'ai seulement effectué quelques modifications sur l'analyseur lexical pour pouvoir récupérer l'ensemble des méta-données et améliorer la lecture de certains fichiers image. Une des modifications porte sur la suppression de l'ensemble des caractères situés entre une fin de commentaire et la fin de la ligne. Cette modification est visible sur le listing 1.8 qui est un fichier de différences issues de notre dépôt Subversion [6].

```
  pszHeaderNext += 2;
+            // consume till end of line.
+            // reduce sensibility to a label error
+            while( *pszHeaderNext != '\0'
+                && *pszHeaderNext != 10
+                && *pszHeaderNext != 13 )
+            {
+             pszHeaderNext++;
+            }
  continue;
}
@@ -419,2 +428,11 @@ }
+/************************************************************************/
+/*                          GetKeywordList()                            */
+/************************************************************************/
+
+char **NASAKeywordHandler::GetKeywordList()
+{
+    return papszKeywordList;
+}
+
```

Listing 1.8 – Extrait de la modification de l'analyseur lexical PDS.

6. Logiciel de gestion de version. Voir le paragraphe 4.3.4

J'ai également ajouté une méthode d'accès à l'ensemble des paires constituant les métadonnées de l'image.

1.7 Remarques sur l'implémentation

Pour compléter les points de développement abordés précédemment nous ne serions pas complets sans parler des tests logiciels. En effet, nos développements visent à intégrer la bibliothèque GDAL et à être mis en production au sein de notre laboratoire, voire à terme auprès de la communauté scientifique. Nous devons donc fournir un effort sur la qualité du code, cela passe par l'écriture de tests non seulement unitaires mais également fonctionnels. Le tout étant idéalement automatisé dans une architecture d'intégration continue. Nous aborderons ce dernier point plus en détail dans le chapitre 4.

Pour le code du pilote ISIS2 de GDAL ou pour les développements des traitements numériques présentés au chapitre 2, j'ai fait le choix d'utiliser la bibliothèque de test « Boost_Test ».

1.7.1 Bibliothèque Boost Test

La bibliothèque Boost [Boost, 2010] désigne en fait un ensemble de sous-bibliothèques écrites en C++. Cette bibliothèque a comme originalité que les nouveaux modules sont soumis à un comité de lecture. Une partie des auteurs des modules et des membres du comité sont aussi des membres du groupe de travail définissant les futurs standards du langage C++. Cette bibliothèque se veut ainsi complètement compatible avec la bibliothèque standard C++ (libStd). La licence choisie pour cette bibliothèque autorise l'usage commercial et non commercial et de ce fait encourage son utilisation dans un grand nombre d'applications.

La sous-bibliothèque `Boost_Test` fournit un ensemble de composants et des macro-définitions† C++ permettant d'écrire et d'organiser des tests simples (test unitaire) ou des jeux de tests complets (module de tests).

Cette bibliothèque fournit entre autres des directives† préprocesseur et des macro-définitions acceptant des prédicats†. Voici une liste non exhaustive des directives et macro-définitions proposées :

BOOST_TEST_MODULE : Directive permettant la déclaration du nom du module de test.

BOOST_CHECK(prédicat) : Macro-définition permettant de tester si un prédicat est vrai. Quelque soit le résultat les tests continueront après.

BOOST_REQUIRE(prédicat) : Macro-définition permettant de tester un prédicat mais contrairement à la macro-définition précédente une exception est levée en cas d'erreur.

BOOST_CHECK_THROW(instruction, exception) : Macro-définition permettant de détecter si une exception est bien levée à l'exécution d'une instruction.

Nous rediscutons de cette bibliothèque dans le dernier chapitre consacré a la méthodologie 4.3.2.

1.7.2 Tests pour le pilote ISIS2

Pour donner un ordre de grandeur, les tests que j'ai implémentés dans GDAL sont répartis dans une douzaine de fichiers et représentent plus de 1560 lignes de code à comparer avec l'unique fichier du pilote comprenant 1450 lignes. J'ai établi une organisation de tests. Ceux-ci se déclinent en trois groupes :

- les tests concernant les méta-données situés dans les fichiers `MetaOperation(.cxx, h)` et `MetaTests.cxx` ;
- les tests concernant les données de l'image : la partie raster située dans les fichiers `RasterOperation(.cxx, .h)` et `RasterTests.cxx` ;
- enfin les tests génériques de plus haut niveau comme la création d'une image située dans les fichiers `PDSTests.cxx` et `ISIS2Tests.cxx`.

Le listing 1.9 montre un extrait d'un fichier de tests. Nous allons commenter l'usage de la bibliothèque Boost_Test dans ce listing.

① **BOOST_TEST_DYN_LINK :** C'est une directive préprocesseur permettant l'usage de la bibliothèque dynamique.

② **BOOST_TEST_MODULE :** Permet la désignation du nom unique du module de test.

③ **:** A ce point du code nous définissons une « fixture† » pour le jeu de test qui va suivre ⑥.

④ **BOOST_CHECK_EQUA :** Macro-définition permettant le test d'un prédicat. Ici nous testons l'existence d'un fichier.

⑤ **BOOST_CHECK_CLOSE :** Macro-définition utilisée pour vérifier que les valeurs de gauche et à droite sont proches numériquement. Le critère de comparaison est la troisième opérande de type pourcentage.

⑥ **BOOST_FIXTURE_TEST_SUITE :** Ici nous définissons un nouveau jeu de test en héritant de la « fixture » définit précédemment. Nous pouvons définir de nombreux jeux de tests pour un module de test donné.

⑦ **BOOST_AUTO_TEST_CASE :** Cette macro-définition permet de définir un test unitaire.

J'ai défini quatre jeux de tests (`testSuitePds, testSuiteISIS2, testSuiteRaster, testSuiteMeta`) qui sont tous inclus dans mon module `GdalISIS2Tests`. Cet ensemble de tests se base sur des données simulées ou précalculées à partir de code Matlab ou Octave[7] mais aussi de données réelles. Ces dernières proviennent des instruments : Omega, Nims, Vims et Virtis. Cela nous permet d'avoir un panel assez vaste de données issues de la communauté planétologique et rajoute un argument pour la diffusion de nos développements de logiciels.

1.8 Conclusion

Je peux résumer mes travaux sur la couche logiciel d'accès aux données par une amélioration et une refonte du pilote GDAL PDS à la norme ISIS2.

1.8.1 Extension du pilote PDS-ISIS2 de GDAL

J'ai ainsi intégré différents aspects supplémentaires de la norme PDS comme par exemple, la gestion de l'ensemble des types disponibles (entier, flottant, etc). L'extension la plus importante étant l'ajout du support en écriture. J'ai aussi créé des jeux de tests de différents niveaux sur un échantillon typique d'images que nous sommes ammenés à traiter en sciences planétaires.

Enfin, soulignons que j'ai effectué une livraison officielle de notre code sur la liste de diffusion de GDAL (https://groups.google.com/group/gdal) début décembre 2010. A l'heure actuelle, après plusieurs courriers électroniques d'échange et une proposition de correctif

7. Octave est un logiciel libre de calcul numérique ainsi que Matlab son pendant propriétaire.

```
#define BOOST_TEST_DYN_LINK                                                    ①
#define BOOST_TEST_MODULE GdalISIS2Tests                                       ②
#include <boost/filesystem.hpp>
#include <boost/test/unit_test.hpp>
#include <gdal/gdal_priv.h>
#include "MetaOperation.h"
#include "RasterOperation.h"
...
struct Isis2Fixture {                                                          ③
  Isis2Fixture() {
    BOOST_TEST_MESSAGE("prepare testing : cleanup Result directory");
    createDir();
  }
    ...
  // create a random cube with detached label
  void TestCreateRandomCube(string labelname, unsigned int band, unsigned int line, unsigned int sample){
    RasterOperation *raster = new RasterOperation();
    raster->m_attachedLabel = false;
    vectDouble mean1;
    mean1 = raster->createCube(labelname, band, line, sample);

    BOOST_CHECK_EQUAL(boost::filesystem::exists(labelname), true);             ④

    unsigned int b, l, s;
    raster->getBandLineSample(labelname, b, l, s);
    BOOST_CHECK_EQUAL(band, b);
        ...

    //check by try to read cube with compare mean by band
    double pourcent(1E-6);
    vectDouble mean2;
    mean2 = raster->getMeanByBand(labelname);
    BOOST_CHECK_EQUAL(mean1.size(), mean2.size());
    //cout<<"size == "<<mean1.size()<<endl;
    vectDouble::iterator it1 = mean1.begin();
    vectDouble::iterator it2 = mean2.begin();
    while(it1 != mean1.end() && it2 != mean2.end()){

      BOOST_CHECK_CLOSE(*it1, *it2, pourcent );                                ⑤
      ++it1;
        ...
};  // end struct  Isis2Fixture

BOOST_FIXTURE_TEST_SUITE( testSuiteIsis2, Isis2Fixture)                        ⑥

...

BOOST_AUTO_TEST_CASE(testDriverSupport){                                       ⑦
  MetaOperation *meta = new MetaOperation();
  bool create, createCopy;
  meta->checkSupport(create, createCopy);
  BOOST_CHECK_EQUAL(create, true );
  // no createCopy driver implemented
  BOOST_CHECK_EQUAL(createCopy, false );
  delete meta;
}
```

Listing 1.9 – Extrait du fichier de test `ISIS2Tests.cxx` du pilote ISIS2.

complémentaire, je n'ai pas de calendrier d'intégration dans la branche officielle, mais notre code figure dans le dépôt à l'adresse suivante : http://trac.osgeo.org/gdal/attachment/ticket/3869/gdal-1.7.3-lpg-613.patch. Il est disponible sous le ticket #3869.

1.8.2 Perspectives d'amélioration

La norme PDS étant très vaste, il serait souhaitable de continuer à implémenter quelques fonctionnalités dans le pilote GDAL. Je pense notamment en priorité à la gestion des différentes valeurs nulles, ou à une gestion complète des suffixes (« Backplane », etc. tel que schématisé sur la figure 1.3).

Idéalement il faudrait pour suivre la philosophie GDAL, implémenter une bibliothèque dédiée au format PDS/ISIS2 pour extraire le code de gestion de GDAL. On se retrouverait dans le cas du pilote FITS par exemple qui utilise la bibliothèque « libfits » liée dynamiquement à GDAL au moment de la compilation. Cet usage converge vers l'amélioration du parseur comme précisé au paragraphe 1.6.5.

Nous reviendrons sur les tests plus loin dans ce document, mais il faudrait encore améliorer les jeux de test en ajoutant des contrôles sur l'extraction des données de géoréférencement.

Les développements effectués dans ce chapitre nous permettent d'accéder à nos images. Nous pouvons maintenant leur faire subir des traitements numériques particuliers pour pouvoir en extraire de l'information. Ce sont ces traitements numériques et notamment l'algorithme ELM-VCA que nous présentons dans le chapitre suivant.

Chapitre 2

Traitement numérique et algorithmes

Les algorithmes que nous déployons dans le cadre du projet Vahiné sont tous issus de travaux récents de recherche. Ils sont donc soumis à des contraintes de droit d'auteur. Avant d'en préciser les aspects plus techniques nous abordons le volet juridique.

2.1 Aspects juridiques

Mon travail étant financé par le CNES et l'ANR, un contrat a été établi entre eux et le LPG. Celui-ci a été rédigé dans le cadre de l'ANR Vahiné. Ce contrat impose des livraisons de logiciels sous licence Cecill. C'est pourquoi comme nous l'avons vu nous travaillons exclusivement avec des logiciels libres dits « open-source ». La licence Cecill a été créée par le CEA, l'INRIA et le CNRS dans le but d'adapter au droit français les développements qui étaient effectués sous des licences classiques libres tel la licence GPL. Cela permet d'accroître la sécurité juridique en utilisant le droit français. La licence Cecill est détaillée sur le site internet http://www.cecill.info/.

Les algorithmes de calcul que nous livrons au CNES, quant à eux, visent à être intégrés dans la bibliothèque OTB. Pour faciliter leurs usages et donc leurs plus grandes diffusions les licences associées à ces algorithmes sont de type Apache. L'élément principal de la licence Apache est d'autoriser la modification et la distribution d'un logiciel sous toutes les formes (libre ou propriétaire, gratuit ou commercial) et d'obliger le maintien du copyright lors de toute modification.

J'ai dû obtenir l'accord de nos partenaires mais aussi des auteurs de certaines parties d'algorithmes avec qui le laboratoire ne collabore pas directement. Ainsi José Bioucas m'a donné son accord pour l'utilisation au sein de l'OrfeoToolBox de l'algorithme « Vertex Components Analysis ».

2.2 Choix du langage et des bibliothèques

2.2.1 Introduction

Comme nous l'avons vu au chapitre précédent ces algorithmes sont amenés à manipuler des données de taille importante (plusieurs centaine de méga-octets au minimum). Ils requièrent donc des choix technologiques particuliers en terme de fonctionnalités mathématiques, numériques et vitesse de calcul. A la suite d'une veille technique que j'ai réalisée concernant l'analyse et la visualisation d'images astrophysiques, (voir le document vfw-state art-10 [Mercier, 2008]) j'ai fait les choix techniques décrits ci-dessous.

2.2.2 Choix réalisés

Le code implémentant les algorithmes est regroupé au sein de la bibliothèque que nous désignons par l'acronyme VFW pour VahineFrameWork. Cette bibliothèque est écrite en C++, langage qui offre de bonnes performances numériques, une intégration aisée de bibliothèques externes, avec en plus la facilité de maintenance d'un langage objet. Ce choix est aussi lié à l'usage de la bibliothèque OTB (OrfeoToolBox) développée par le CNES. Dans une première phase du projet, avant le développement de l'interface de visualisation, j'ai développé un exécutable de pilotage des traitements numériques. Pour ce dernier j'ai retenu le langage de script Python. Ce choix est motivé par deux points :

- le langage Python dispose de nombreuses bibliothèques scientifiques (NumPy http://numpy.scipy.org/) ;
- il y a une très bonne intégration entre Python et C++ grâce à la bibliothèque Cableswig http://www.itk.org/ITK/resources/CableSwig.html. Cette dernière permet de générer des interfaces Python pour encapsuler des appels à des méthodes et l'utilisation d'objets C++.

Le choix de la bibliothèque de traitement a été rapide. La bibliothèque OTB s'est naturellement imposée car elle intègre la célèbre bibliothèque de traitement d'images médicales ITK (Insight Segmentation and Registration Toolkit) http://www.itk.org/. ITK est une bibliothèque dont le développement a été commandité par l'institut de la santé états-unienne (National Institute of Health). Cette bibliothèque est très utilisée dans le milieu médical pour effectuer de la segmentation† d'images. La bibliothèque ITK est devenue une référence dans le domaine du traitement d'images médicales par le nombre et la qualité des algorithmes qu'elle intègre et sa communauté notamment d'universitaire l'utilisant.

De plus, OTB est en train de devenir de facto le standard en bibliothèque « open source » dédiée au traitement d'images satellitaires.

2.2.3 Bibliothèque OrfeoToolBox ou OTB

Dans le cadre du programme d'accompagnement Orfeo lié au lancement des futurs satellites d'observation PLEIADE [1], le CNES a développé des outils logiciels libres pour la manipulation et le traitement d'images multispectrales.

OTB met à disposition des développeurs une riche palette de fonctionnalités s'étalant sur plusieurs niveaux d'abstraction pour le traitement de l'image de télédétection en général. Cette bibliothèque nous impose des contraintes de programmation avancée liées à une programmation spécifique en C++ (programmation générique par « template† » et « smart pointer » [2] notamment). Elle intègre de nombreuses bibliothèques performantes :

ITK : bibliothèque de traitement d'images (segmentation, etc.) ;

VXL-VNL : bibliothèque de calcul linéaire (the Vision-Numerical-Libraries) ;

GDAL : bibliothèque permettant de gérer les opérations d'entrée-sortie sur de nombreux formats d'images.

Au final la bibliothèque OTB est un outil puissant pour les travaux en imagerie satellitaire. Techniquement elle est implémenté en C++ et est essentiellement basé sur ITK. La bibliothèque ITK est ainsi la librairie centrale d'OTB et les classes d'OTB héritent des classes d'ITK.

La bibliothèque OTB s'inspire aussi de la bibliothèque ITK pour les méthodes de développement (méthode agile, tests logiciels, programmation générique). La documentation

1. Satellites d'observation de la terre.
2. Voir le paragraphe 2.2.4

et les exemples sont en grande partie extraits de la bibliothèque ITK. La programmation par « template » en C++ est très efficace et permet de soulever de nombreux problèmes à la compilation au lieu de les obtenir au cours de l'exécution du programme. ITK et OTB sont multiplateformes. Elles utilisent l'environnement de construction CMake pour gérer le processus de configuration et de compilation.

Le choix d'OTB comme brique de base pour la programmation des algorithmes se justifie aussi par un gage de pérennité (le CNES est maître d'ouvrage d'OTB) et de visibilité dans la communauté travaillant dans le domaine de l'imagerie satellitaire.

Nous présentons maintenant quelques points clef de la bibliothèque OTB.

2.2.4 Programmation générique

La programmation générique est une technique qui vise à organiser les bibliothèques en composants réutilisables. L'idée est de construire des composants logiciels sous forme de briques de base. Il nous faut donc des objets dit conteneurs, des itérateurs pour les parcourir et enfin des algorithmes génériques pour les manipuler. Pour augmenter cette généricité d'un programme, certains langages comme C++ offrent la possibilité de faire de la méta-programmation. Celle-ci est utilisée pour générer du code interprété par le compilateur. Cela permet de générer des structures de données adéquates aux types de données traitées. Le code est alors très générique et facilite la maintenance.

En C++ la méta-programmation utilise les fonctions, méthodes ou classes « template ». Cela permet d'éviter de définir une implémentation d'une fonction par type de données.

```
template <class T>
T minimum (T a, T b){
  return ((a < b) ? a : b);
}
```

Listing 2.1 – Définition de la fonction générique minimum.

Un exemple trivial de cette possibilité est exposé sur le listing 2.1. Nous définissons ici un patron de la fonction `minimum` qui peut être instancié pour tout type `T`. La seule contrainte est que l'opérateur « inférieur à » noté `<` soit défini pour l'ensemble des type `T` envisagés.

Dans notre cas nous travaillons avec des classes et méthodes « template » bien plus complexes (voir 2.2). Par exemple nous manipulons différents types de pixels ou même d'images.

```
template <class TImage>
  class ITK_EXPORT VahineElmVcaFilter
    : public itk::ImageToImageFilter<TImage, TImage> {
    public:
      typedef TImage VectorImageType;
    typedef VahineElmVcaFilter Self;
    typedef itk::ImageToImageFilter<VectorImageType, VectorImageType> Superclass;
    typedef itk::SmartPointer<Self> Pointer;
    typedef itk::SmartPointer<const Self> ConstPointer;
 ...
```

Listing 2.2 – Extrait de la classe template `VahineElmVcaFilter`.

2.2.5 Smart pointer

Autre notion clef utilisée dans la bibliothèque OTB c'est l'usage de pointeurs particuliers appelés « smart pointer ». Le langage C++ donne la possibilité au développeur de gérer lui même la mémoire qu'il aura allouée dynamiquement. Cela peut être un avantage par moment mais source également de nombreux disfonctionnement (« bugs »). Pour pallier à cela il est possible d'utiliser des « smart pointer » au lieu des pointeurs normaux. Les « smart pointer

» ont la faculté de libérer automatiquement la mémoire allouée pour les objets lorsqu'ils sont détruits. OTB utilise ce mécanisme.

Pour être plus précis grâce à la classe `itk::SmartPointer`, les classes d'OTB utilisent des « smart pointer » avec comptage de références. A chaque objet créé est associé un compteur de références. Celui-ci compte le nombre de pointeurs désignant l'objet. L'objet sera détruit seulement quand le compteur de références sera à zéro. A cet instant plus aucun pointeur ne désigne notre objet. Ce mécanisme est l'équivalent du ramasse-miettes (ou « garbage collector ») disponible avec la technologie Java.

Ce processus est complètement transparent dans OTB. Il est déclenché automatiquement grâce à la méthode `New()` d'instanciation des objets ITK. Les objets doivent hériter de la classe `itk::LightObject` pour bénificier de ce mécanisme.

2.2.6 Représentation des données

La bibliothèque OTB supporte les images quelque soient leurs types de pixels et quelque soient leurs dimensions. Dans le cadre d'une image standard c'est la classe `OTB::Image` qui sert à représenter et à manipuler les données.

```
typedef otb::Image< unsigned short, 2 > ImageType;
ImageType::Pointer image = ImageType::New();
 ...
typedef float InternalPixelType;                                                    ①
const unsigned int Dimension = 2;
 ...
typedef otb::VectorImage< InternalPixelType, Dimension > VectorImageType;
VectorImagePointer output = VectorImageType::New();
 ...
RegionType imageRegion = this->GetInput()->GetLargestPossibleRegion();             ②
 ...
output->SetRegions(imageRegion);
output->SetNumberOfComponentsPerPixel(m_NbComponents);                              ③
output->Allocate();
```

Listing 2.3 – Exemple de déclarations d'images.

Ces classes utilisent la notion de programmation générique comme présentée précédemment à la section 2.2.4. Ce sont des classes dites template dans le jargon du langage C++.

Les images hyperspectrales sont représentées grâce à la classe `OTB::VectorImage`. Pour définir une nouvelle instance de celle-ci nous devons définir 3 éléments.

- Le type de données manipulées ou intrinsèques à cette classe comme défini au point ①.
- La taille géométrique d'un plan de notre image ou dans notre cas d'une bande spectrale. Cette taille est définie par l'intermédiaire de l'objet `imageRegion`. C'est la déclaration située au ②.
- Et enfin la taille dans la troisième dimension. C'est-à-dire le nombre de plans spectraux. Celui-ci est défini par la variable `m_NbComponents` et est initialisé au niveau de l'image au point ③ du listing 2.3.

2.2.7 Itérateurs

Un élément important en programmation générique est le concept d'itérateur†. Disponibles dans la bibliothèque standard associée au langage C++, les itérateurs permettent de travailler facilement sur les conteneurs classiques comme les vecteurs ou les listes.

La bibliothèque ITK et la bibliothèque OTB généralisent ce concept aux objets manipulés dans leur cadre : les images. ITK a ainsi une grande variété d'itérateurs sur les images pour simplifier les tâches courantes de traitement. Sur le listing 2.4 nous avons un exemple

```
...
typedef itk::ImageRegionConstIterator<VectorImageType> ConstIteratorType;                    ④
...
ConstIteratorType itInput( this->GetInput(), imageRegion );                                  ⑤
...
for(itInput.GoToBegin(); !itInput.IsAtEnd(); ++itInput){
  PixelType pixel = itInput.Get();                                                            ⑥
  VnlVector column(m_NumberOfBand);
  for(unsigned int j(0); j<m_NumberOfBand; ++j){
    column[j] = static_cast<double>(pixel[j]);
  }
  data.set_column(idx, column);
  ++idx;
}
...
```

Listing 2.4 – Utilisation d'un itérateur constant. Extrait du filtre Vahiné `VcaFilter.h (.txx)`

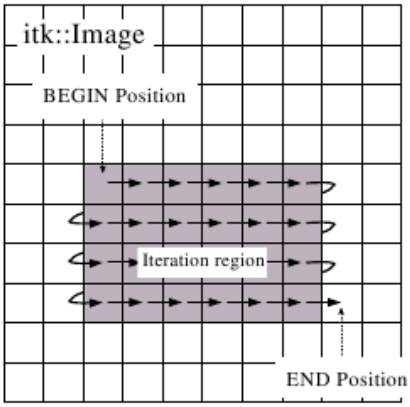

FIGURE 2.1 – Lecture des pixels d'une image via la classe `Itk::iterator`.

d'utilisation dans le cadre du projet Vahiné. Au point ⑤ nous déclarons une spécialisation ITK d'un itérateur sur image. Ce dernier est un itérateur constant (qui n'accédera aux données qu'en lecture). Il est de type `itk::ImageRegionConstIterator`.
De façon générale tous les itérateurs d'images ont au moins un paramètre « template » qui définit le type d'images sur lequel ils vont devoir travailler. Dans notre cas, au point ④ du listing 2.4, notre itérateur travaille sur des images de type `VectorImageType`. Au moment de l'instanciation ⑤ deux paramètres sont nécessaires : un pointeur sur l'image à balayer et une région de l'image. La région pouvant englober tout ou partie de l'image.

L'itérateur est manipulable via les structures classiques du langage C++. La figure 2.1 représente l'itération d'une région d'une image. Il nous permet d'accéder à l'objet représentant un pixel grâce à la méthode `Get()` comme le montre le point ⑥.

Maintenant que nous avons défini une image et que nous savons y accéder nous pouvons décrire la création de filtres OTB.

2.2.8 Filtre et pipeline

La bibliothèque OTB est conséquente, nous terminerons notre aperçu avec la notion de filtre et leur chaînage ou « pipeline ».

On peut voir un filtre comme un processus qui prend un ou plusieurs éléments en entrée et produit un ou plusieurs éléments en sortie. Par élément, nous entendons une image matricielle, vectorielle, une valeur numérique ou une chaîne de caractères par exemple. Les

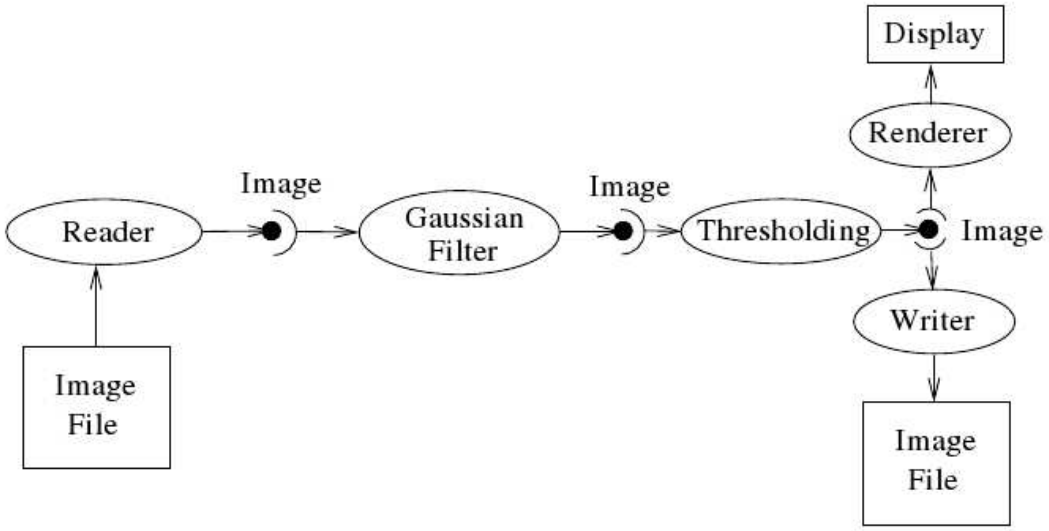

FIGURE 2.2 – Représentation d'un flux de données typique OTB

filtres sont définis par rapport au type de données qu'ils doivent traiter en entrée et au type de données qu'ils doivent renvoyer. L'élément clef de la création d'un filtre OTB est donc le nombre et la nature des entrées-sorties pour le filtre considéré.

En général l'écriture d'un filtre passe par l'héritage d'une classe mère. Cela permet de simplifier la conception.

Dans notre cas nos filtres principaux héritent le plus souvent de la classe `itk::ImageToImageFilter`. Ce filtre de base d'OTB accepte une seule image en entrée et renvoie une image en sortie. Cette classe étant une classe « template », son usage est très souple et nous permet d'utiliser deux types complètement différents d'image pour l'entrée et la sortie.

OTB fournit déjà un grand nombre de filtres. Par exemple, deux filtres importants sont : `Reader` et `Writer`. Ce sont les filtres d'entrée-sortie permettant la lecture et l'écriture sur disque.

La figure 2.2 présente l'imbrication typique d'une série de filtres dans le but ici de :

1. lire un fichier `Reader` ;
2. appliquer un traitement numérique `GaussianFilter` ;
3. appliquer un autre traitement `Thresholding` (ici un seuillage) ;
4. écrire sur le disque `Writer`.

Dans le cadre de ce mémoire et dans celui plus large de l'ensemble du projet Vahiné j'ai ainsi défini une douzaine de filtres OTB. L'annexe B.2 présente le code source complet du filtre `ElmVca`.

Le mécanisme de filtre défini dans la bibliothèque ITK et repris par OTB permet de chaîner autant de filtres que l'on souhaite. Nous avons ainsi un parfait découpage de la bibliothèque en composant de base (les filtres) et la possibilité de les réutiliser (les chaîner) dans le plus pur esprit de programmation générique.

Ce système de chaînage (ou « pipeline ») permet aussi de résoudre le problème du traitement d'images ayant des tailles importantes. La bibliothèque ITK offre la possibilité de travailler sur une portion de l'image seulement. On peut donc découper l'image en région et procéder à un « streaming » de l'image par petits morceaux. C'est là que la notion de « pipeline » prend toute sa puissance. les bibliothèques ITK et OTB fournissent l'ensemble

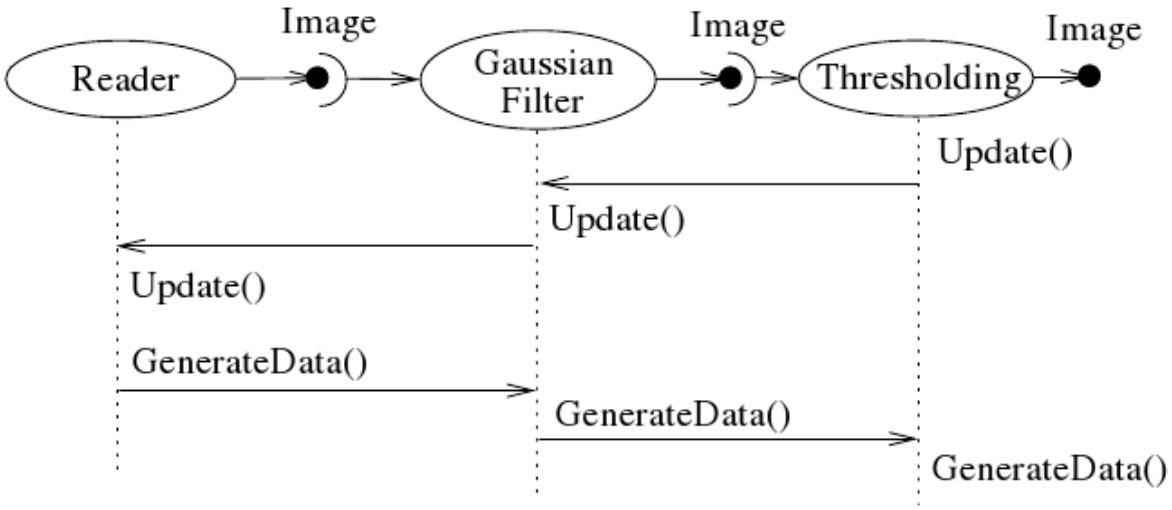

FIGURE 2.3 – Représentation d'un flux de données typique OTB

des services pour gérer un découpage de l'image et une exécution en parallèle des différents processus sur ce découpage.

Ce système de chaînage remplit aussi plusieurs fonctions comme :

- la détermination du filtre ayant besoin d'être exécuté, cela permet de minimiser le temps d'exécution globale ;
- l'initialisation des objets en sortie de toute la chaîne,
- la gestion automatique de la subdivision des données d'entrée en fonction de la demande du dernier filtre en sortie ;
- la gestion de la mémoire : allocation et désallocation dynamique.

Le processus global est contrôlé par la demande d'une région de taille particulière (qui peut être l'ensemble de l'image). L'élément important à retenir est que cela permet à l'utilisateur de demander à traiter seulement une région d'intérêt dans une image de grande taille ou de pouvoir gérer de grandes données tout en ayant une limitation de mémoire.

La figure 2.3 présente une exécution typique d'une séquence de filtres OTB. L'exécution du « pipeline » est déclenchée lorsque l'on invoque la méthode `Update()` sur un objet de la chaîne. Cet appel est alors propagé récursivement de la fin vers le début du traitement ou alors jusqu'à ce qu'un filtre dont les données sont à jour soit rencontré. A partir de cet instant les traitements sont activés les uns après les autres jusqu'à l'objet qui a provoqué l'appel à la méthode `Update()` initiale.

Pour accélérer le traitement, la bibliothèque OTB fournit de plus un moyen très simple. Il suffit d'implémenter dans le filtre voulu la méthode `ThreadedGenerateData()` et la bibliothèque s'occupera automatiquement de découper l'image en régions. Nous avons juste à implémenter le traitement voulu sur une région au sein de `ThreadedGenerateData()`. Cette technique a été expérimentée au cours de ce mémoire notamment sur le filtre `CovFilter()` pour accélérer un calcul de covariance sur l'ensemble d'une image hyperspectrale.

2.3 Pilotage des traitements

Dans les méthodes de développement dites agiles, un des objectifs est d'avoir à tout moment des logiciels utilisables même si les fonctionnalités implémentées sont minimales. Étant donnés les objectifs que nous devons atteindre, j'ai décidé en attendant d'avoir une

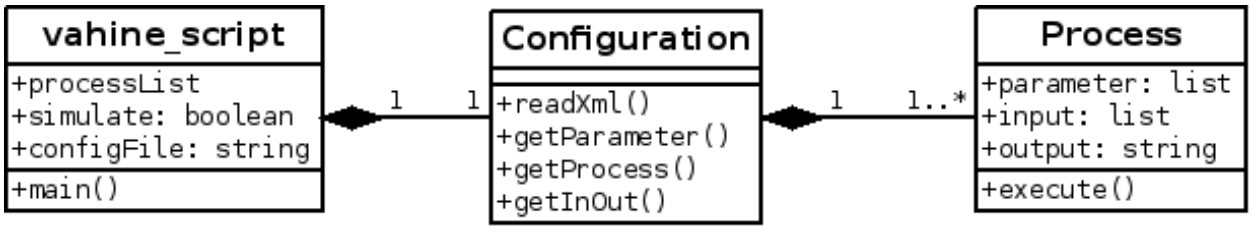

FIGURE 2.4 – Diagramme de classes définissant le script `vahine.py`.

interface de pilotage, que les algorithmes de traitement seraient exécutables en ligne de commande. Pour faciliter cela, j'ai donc créé un script python `vahine.py` qui pilote l'ensemble des filtres développés dans le cadre du projet Vahiné. Nous reproduisons l'ensemble du script `vahine.py` en annexe B.3.

Ce script se décompose en deux classes `Configuration` et `Process`. La classe configuration est chargée d'analyser la syntaxe du fichier d'entrée au format XML. Elle récupère la configuration pour chaque filtre de base que j'ai implémenté pour Vahiné. La classe `Process` va elle être instanciée autant de fois que nous avons de processus décrits dans le fichier de configuration. La fonction `main` définie au point ① du listing B.7 va créer une liste `processList` contenant l'ensemble des processus de traitement (ou filtre Vahiné-OTB) à activer.

Puis cette fonction `main` va appeler la méthode `execute` de la classe `Process` pour chaque objet de la liste `processList`. Nous chaînons ainsi l'exécution des filtres Vahiné.

Pour configurer l'ensemble d'un traitement Vahiné le script `vahine.py` admet un seul paramètre obligatoire à l'appel : un fichier de pilotage au format XML. Un diagramme de classes est présenté sur la figure 2.4.

2.3.1 Utilisation de XML et paramétrage

Un document XML est un fichier dont les données sont structurées à l'aide de balises. Les facilités d'écriture et de traitement des données offertes par ce format sont particulièrement bien adaptées au stockage pérenne des données, grâce à son auto-descriptivité. C'est donc tout naturellement que j'ai fait le choix de ce format pour la gestion des paramètres des algorithmes. Dans notre cas nous disposons d'un jeux réduit de balises que nous détaillons ci-dessous.

vahine : Balise racine pour les paramètres concernant les algorithmes du Framework vahiné. Cette balise est obligatoire.

process : Balise permettant la désignation du processus de calcul à exécuter. Elle prend deux paramètres obligatoires. Le premier `name` désigne le nom du processus. Le second paramètre `run` qui est un drapeau booléen pour désactiver ou non le processus. Ce dernier est utile en cas de chaînage de plusieurs processus.

input : Balise contenant toujours le paramètre `name` qui prend la valeur « default ». Elle permet de désigner le fichier d'entrée dont le nom est passé par le paramètre value.

parameter : Balise permettant d'associer un paramètre spécifique au processus. Par exemple dans le cas du processus `VahineElmVcaFilter`, celui-ci n'a qu'un et unique paramètre : `endmembers`.

ouput : Balise contenant toujours le paramètre `name` qui prend la valeur « default ». Elle permet de désigner le fichier de sortie dont le nom est passé par le paramètre `value`.

Le module ELM-VCA que nous présentons dans la section 2.4 peut être lancé grâce à l'usage du script python `vahine.py` et du fichier de configuration présenté au listing 2.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<vahine>
  <process name="VahineElmVcaFilter" run="True">
    <input name="default" value="Data/img_41_v04/ORB0041_1_L_V04.REF.lbl"/>
    <parameter endmembers="0" />
    <output name="default" value="Result/img_41_elmvca.lbl"/>
  </process>
</vahine>
```

Listing 2.5 – Exemple de fichier de paramètres pour l'utilisation de l'algorithme ELM-VCA.

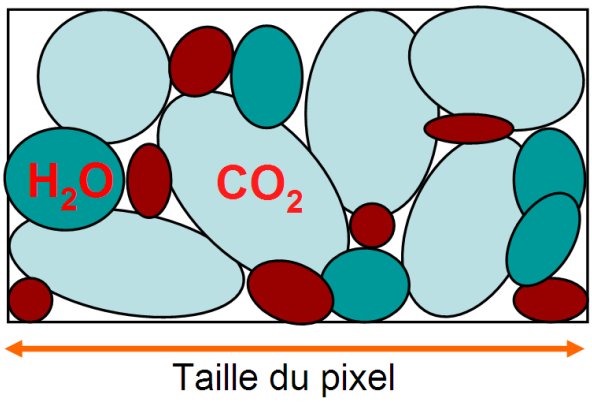

FIGURE 2.5 – Schéma d'un mélange granulaire à trois corps auquel sera associé un mélange spectral.

L'exécution se fait par la ligne de commande suivante :

```
./vahine.py -c elmvca.xml
```

Dans cet exemple, le fichier image traité en entrée est l'image au format PDS : `ORB0041_1_L_V04.REF.lbl`. Une balise particulière `VahineElmVcaFilter` permet de lancer le traitement ELM-VCA. Cette balise particulière a un seul paramètre `endmembers`. Ce dernier est de type entier, s'il est nul alors le processus fera une estimation du nombre de pôles sinon il lancera directement la procédure qui exécutera l'algorithme VCA d'extraction des sources.

2.4 Algorithme ELM-VCA

De la recherche Vahiné sont issus de nouveaux algorithmes de traitement d'images hyperspectrales. Dans le cadre de ce mémoire, je décris l'implémentation de l'algorithme ELM-VCA que nous avons réalisée dans le « Vahiné Framework ».

2.4.1 Introduction

En collaboration avec le GIPSA-Lab, le LPG a élaboré plusieurs algorithmes de séparation de sources† spectrales en aveugle (SSA). L'objectif est d'extraire les pôles† d'une image hyperspectrale et de déterminer leur proportion respective au sein de chaque pixel. Ces deux opérations font l'hypothèse que le signal spectral mesuré résulte d'un mélange linéaire des pôles. En effet la résolution spatiale souvent kilométrique des spectro-imageurs planétaires ne permet pas toujours de résoudre spatialement les zones de terrains chimiquement homogènes (Voir la représentation du cas sur la figure 2.5).

La nouvelle méthode (ELM-VCA) est parfaitement adaptée à cet objectif car elle travaille de façon géométrique dans l'espace n-dimensionnel de la population de spectres. Elle estime d'abord le nombre de pôles spectraux purs présents dans une image hyperspectrale. Cette étape est menée à partir du calcul des matrices de covariance et de corrélation de l'image. Ensuite l'algorithme VCA (Vertex Component Analysis) utilise ce nombre de pôles comme paramètre d'entrée pour extraire de façon non supervisée† les spectres sources† et leurs cartes d'abondances respectives. L'hypothèse clef sous-jacente de cette méthode est qu'il existe des pixels purs† dans l'image. Cette méthode a été validée par des expériences effectuées sur des bases de spectres synthétiques† et sur des images hyperspectrales réelles OMEGA et CRISM [Luo *et al.*, 2009, Douté *et al.*, 2011].

Dans la suite, nous explicitons la partie implémentation de cet algorithme sans approfondir l'aspect mathématique et méthodologique. Le lecteur est invité à consulter l'article de référence concernant cet algorithme [Luo *et al.*, 2009].

2.4.2 Analyse de la maquette

Le module ELM-VCA a été développé à partir d'une maquette matlab et d'un jeu d'images tests planétaires acquises par le spectro-imageur OMEGA [Bibring *et al.*, 2004].

La première implémentation informatique de ELM-VCA a été effectuée en Matlab par les soins du GipsaLab. C'est cette maquette Matlab et les résultats qu'elle produit qui servent de référence au code ELM-VCA du cadriciel (« framework » en français) Vahiné.

L'algorithme initial commence par charger une image hyperspectrale et par lui appliquer un filtrage de bande pour ne sélectionner que les bandes pertinentes. Ensuite, le processus de calcul passe par l'exécution d'une fonction Matlab appelée `demelange()`. Celle-ci effectue l'estimation du nombre de sources grâce à une sous-fonction `ELM()`.

La fonction `demelange()` effectue ensuite un calcul de covariance sur les données initiales, opération coûteuse en temps de calcul. Le nombre de sources obtenues est passé comme paramètre à la méthode `VCA()`. Pour cette dernière méthode, le lecteur se référera à l'article [Nascimento et Dias, 2004]. L'algorithme VCA permet d'obtenir la matrice de mélange qui, multipliée par les données initiales, nous donnes alors les cartes d'abondances.

La maquette Matlab génère un fichier résultat constitué d'un cube image dont les deux premières dimensions reprennent les dimensions de l'image initiale et dont la troisième dimension est constituée d'autant de plans que de pôles estimés. Chaque plan est la projection de l'image d'origine sur chaque spectre identifié comme pôle (Endmember). La visualisation de ces plans, nous donne alors des cartes du degré de couverture de chaque pôle au sein de chaque pixel.

2.4.3 Implémentation avec OTB

Les données initiales sont de type hyperspectral au format PDS. Pour ce module, je réutilise donc une partie des développements que j'ai effectués précédemment pour le module GRSIR (Voir l'annexe C). Dans la suite, les données hyperspectrales sont notées sous forme matricielle avec $\mathbf{X}$ tel que :

$$\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_N \tag{2.1}$$

Le spectre du pixel k est noté $\mathbf{x}_k$ et l'hypothèse sous-jacente est qu'il correspond a un mélange linéaire des N_c spectres sources décomposés sur N_s bandes spectrales. Pour résumer

```
namespace otb {
  template <class TImage>
  class ITK_EXPORT VahineElmVcaFilter
    : public itk::ImageToImageFilter<TImage, TImage> {
  public:
    typedef TImage VectorImageType;
  ...
};
```

Listing 2.6 – Classe Vahine implémentant ELM-VCA. Extrait de `ElmVcaFilter.txx`

```
template<class TImage>
unsigned int VahineElmVcaFilter<TImage>::Demelange(unsigned int nbObserv){
  ...
}
```

Listing 2.7 – Méthode Demelange implémentant l'algorithme ELM.

nous avons donc le modèle suivant :

$$\mathbf{X} = \mathbf{MS} + \mathbf{n} \tag{2.2}$$

où $\mathbf{M} = \{\mathbf{m}_1, ..., \mathbf{m}_{N_c}\}$ est la matrice de mélange, et $\mathbf{m}_n$ est le n-ième spectre source. $\mathbf{S}$ est la matrice des abondances. $\mathbf{n}$ représente du bruit additif centré sur zéro.

Du point de vue de la programmation C++ les filtres `ElmVcaFilter` et `VcaFilter` sont codés suivant la philosophie de la bibliothèque OTB et utilisent la méthode des « templates » que nous avons abordée à la section 2.2.4. Les deux filtres sont paramétrés par la variable « template » `TImage` qui doit être de type `VectorImage` mais avec un type interne libre. J'essaye de travailler en interne uniquement avec des nombres codés en double précision pour diminuer les erreurs dûes aux arrondis imposés par les calculs avec des nombres flottants. On notera que généralement les données initiales sont des réels simple précision. Il faut être conscient de la limite des calculs numériques avec cette méthode basique. Seul l'usage d'une méthode de calcul par intervalle peut nous donner des résultats arrondis mais fiables avec notamment l'intervalle d'erreur associée.

L'implémentation du module ELM-VCA est effectuée à travers la classe `VahineElmVcaFilter` qui est donc un nouveau filtre OTB à part entière. Nous voyons sur le listing 2.6 que cette classe dérive de `itk::ImageToImageFilter`.

Le filtre Vahiné ELM-VCA hérite du filtre OTB/ITK `ImageToImageFilter` puisqu'il prend comme paramètre d'entrée l'image à traiter et renvoie les cartes de paramètres pour chaque spectre source estimé. Le seul paramètre supplémentaire et celui qui offre la possibilité de fixer par avance le nombre de spectres sources via la méthode `SetNbComponents`. Cela permet de court-circuiter la partie ELM et d'utiliser uniquement le module VCA.

2.4.3.1 ELM (Eigenvalue Likelihood Maximization)

La méthode ELM permet l'estimation du nombre de spectres initiaux N_c. Cette méthode est implémentée dans la méthode interne de la classe `VahineElmVcaFilter` : `Demelange`. Son entête est visible sur le listing 2.7.

Le calcul ELM commence par l'obtention des matrices de covariance et de corrélation de $\mathbf{X}$ respectivement $\mathbf{K}$ et $\mathbf{R}$. Ce calcul est effectué par la classe `itk::Statistics::CovarianceCalculator` d'OTB (voir l'extrait du code sur le listing 2.8).

Le calcul statistique de la covariance se base sur un objet `ListSample`. Nous procédons donc à une transformation de l'image d'origine en une liste d'échantillons (voir 2.9). J'ai

```
typedef typename itk::Statistics::ListSample<itk::VariableLengthVector<float>> SampleType;
typedef itk:: Statistics :: CovarianceCalculator< SampleType > CovarianceAlgorithmType;
typedef typename CovarianceAlgorithmType::Pointer CovarianceAlgorithmPointer;
...
CovarianceAlgorithmPointer covarianceAlgorithm = CovarianceAlgorithmType::New();
covarianceAlgorithm->SetInputSample(sample);
covarianceAlgorithm->SetMean(0);
covarianceAlgorithm->Update();
const itk:: VariableSizeMatrix<double> *covarianceVSMat=covarianceAlgorithm->GetOutput();
VnlMatrix covarianceMatrix = covarianceVSMat->GetVnlMatrix();
```

Listing 2.8 – Calcul de covariance

```
SampleType::Pointer sample = SampleType::New();
RegionType imageRegion = this->GetInput()->GetLargestPossibleRegion();
ConstIteratorType it( this->GetInput(), imageRegion );
for(it.GoToBegin(); !it.IsAtEnd(); ++it){
  sample->PushBack( it.Get() );
}
```

Listing 2.9 – Transformation de l'image en liste

implémenté différentes versions du calcul de covariance pour effectuer des comparatifs de temps de calcul.

Après l'obtention des matrices $\mathbf{K}$ et $\mathbf{R}$, nous récupèrons les valeurs propres grâce aux méthodes de la bibliothèque VNL. Nous obtenons λ_i et $\hat{\lambda}_i$ respectivement la i-ième valeur propre de $\mathbf{K}$ et $\mathbf{R}$ grâce à la méthode `vnl_symmetric_eigensystem()`. S'il y a N_c spectres sources, les valeurs propres λ_i et $\hat{\lambda}_i$ pour $i > N_c$ correspondent aux variances du bruit. L'approche utilisée dans l'algorithme ELM consiste à définir une variable $z_i = \hat{\lambda}_i - \lambda_i$ qui permet d'estimer le nombre de spectres sources. La distribution de z_i se modélise ([Luo *et al.*, 2009]) par

$$\begin{aligned} z_i &\sim \aleph(\mu_i, \sigma_i^2),\ i \leq N_c \\ z_i &\sim \aleph(0, \sigma_i^2),\ i > N_c \end{aligned} \tag{2.3}$$

où μ_i est inconnue et $\sigma_i^2 \approx \frac{2}{N}(\hat{\lambda_i^2} - \lambda_i^2)$ si le nombre d'échantillons est suffisamment élevé.

Pour obtenir N_c, nous établissons à partir de l'équation 2.3 une fonction de vraisemblance† $H(i)$ tel que :

$$H(i) = \Pi_{l=i}^{N_s} \frac{1}{\sigma_l} exp(-\frac{z_l^2}{2\sigma_l^2}) \tag{2.4}$$

Ce qui nous intéresse, c'est le logarithme de la fonction de vraisemblance qui peut être décomposé en une somme de deux termes :

$$\tilde{H(i)} = log\, H(i) = A(i) + B(i)$$

Nous ne développerons pas ici l'analyse mathématique complète. Au final, le nombre de spectres sources N_c peut être déterminé par le premier maximum local, qui est aussi souvent le maximum global, de $\tilde{H(i)}$. Les deux maxima sont différents si l'image présente des artefacts résiduels comme une bande corrompue par exemple.

Le logarithme de la fonction de vraisemblance† est obtenu par la méthode `LikelihoodLog` et les maxima locaux par `LocalMaximum()`.

On notera que l'appel de la méthode `Demelange`(Voir la figure 2.10) qui permet d'obtenir le nombre de spectres sources est effectué dans la méthode `GenerateOutputInformation()`.

```
template<class TImage>
void VahineElmVcaFilter<TImage>::UpdateNumberOfComponents(){
  RegionType imageRegion = this->GetInput()->GetLargestPossibleRegion();
  SizeType size = imageRegion.GetSize();
  unsigned int NObserv = size[0]*size[1];
  m_NumberOfBand = this->GetInput()->GetNumberOfComponentsPerPixel();
  if(m_NbComponents == 0){
    m_NbComponents = Demelange(NObserv);
  }
}
```

Listing 2.10 – Appel de la méthode `Demelange()` depuis `UpdateNumberOfComponents`

Cela permet de fixer la taille de l'image de sortie. Cela est indispensable dans le cas où le filtre est chaîné avec d'autre. La contre partie est le temps de calcul prohibitif sur des images de taille importante lors de l'appel au filtre `ELM-VCA`, avant même l'appel de la méthode `update`. Enfin la méthode `GenerateData` contient l'allocation mémoire pour l'image de sortie et le calcul de la projection des données initiales sur la matrice de « mixing » issue du calcul du filtre VCA.

2.4.3.2 VCA (Vertex Component Analysis)

Le filtre VCA ne dépend pas d'autres filtres OTB. Pour le détail du calcul, on se reportera à l'article [Nascimento et Dias, 2004]. Le coeur du calcul s'effectue dans la méthode `VCA`. La méthode mathématique VCA cherche les sommets d'un simplex† qui contient les données initiales. L'algorithme est itératif et projette les données dans la direction orthogonale au sous-espace donné par les spectres sources déjà déterminés. Le spectre source suivant correspond alors à la projection la plus extrême dans cette direction. L'algorithme itère jusqu'à ce que tous les sommets correspondant au nombre de spectres sources soient déterminés.

La première partie du calcul consiste à déterminer le rapport signal sur bruit SNR (« Signal-to-noise ratio ») lié au nombre de sources et le niveau de bruit SNR théorique. En fonction des résultats on appliquera l'une des deux projections suivante.

- void ProjectivProjection(VnlMatrix const& M, VnlMatrix & Rp) ;
- void SubspaceProjection(VnlMatrix const& M, VnlMatrix & Rp) ;

Si le ratio théorique est supérieur alors la première projection est choisie.

```
if(snr < snr_th){
  vahineDebug("Select␣the␣projectiv␣projection");
  ProjectivProjection(data, m_ProjectedData);
}else{
  vahineDebug("Select␣␣projection␣to␣subspace");
  SubspaceProjection(data, m_ProjectedData);
}
...
```

Listing 2.11 – Choix de la projection dans les sous-espaces

Ensuite le calcul se poursuit par la recherche des pixels les plus purs correspondant donc aux sources initiales et par la génération de la matrice de « mixing ». Nous reproduisons dans l'encart 2.12 le calcul central de la méthode. Les seuls paramètres d'entrée sont le nombre de sources « EndMember » et les données issues de l'image sous forme matricielle.

```
vahineDebug("start␣VCA␣algorithm");
VnlMatrix A(m_NumberEndMember, m_NumberEndMember, 0);
A(m_NumberEndMember-1, 0) = 1;
VnlVector w(m_NumberEndMember);
VnlVector f, fsquare, v, col;
for(unsigned int i=0; i<m_NumberEndMember; i++){
  FillRandom(w);
  vnl_svd<double> Asvd(A);
```

```
  VnlMatrix Ainv = Asvd.pinverse();
  f = w - A*Ainv*w;
  fsquare = f.apply(&mysquare);
  f = f / sqrt(fsquare.sum());
  v = f*m_Y;
  v=v.apply(&myabs);
  // search pure pixel
  unsigned int j = arg_max(v);
  col = m_Y.get_column(j); // save Y columns values
  m_PurePixelsIndices.push_back(j); // save indice number
  A.set_column(i, col);
}
```

Listing 2.12 – Calcul de VCA et itération sur les endmembers.

2.4.4 Futur de ELM-VCA

Le développement du module ELM-VCA va continuer avec l'ajout du module ELM-MVSA qui permet d'obtenir des cartes d'abondances à partir d'une image où n'existe aucun pixel « pur ». Une modification souhaitable est d'améliorer le temps de calcul de la covariance par l'application préalable d'un module de réduction de dimensionnalité, par exemple pour accélérer le calcul. Une autre solution qui conserverait la précision des matrices de covariance et corrélation K et R, est de séparer le calcul de corrélation sur plusieurs processeurs. Néanmoins l'usage du filtre OTB `otb::StreamingStatisticsVectorImageFilter` n'a pas été concluant pour le moment car le temps de calcul n'est pas diminué. La suite du développement de ce module continuera avec l'intégration de l'algorithme ELM-MVSA décrit lui aussi dans l'article [Luo *et al.*, 2009].

2.5 Autres algorithmes

2.5.1 Algorithme GRSIR

L'algorithme GRSIR est une spécialisation de la méthode SIR (Sliced Inver Regression). Celle-ci est une méthode de régression semi-paramétrique ayant un temps de calcul court par rapport aux autres méthodes de régression† semi-paramétriques.

Le lecteur trouvera une description complète des fondements théoriques des méthodes SIR et GRSIR dans les références suivantes [Bernard-Michel *et al.*, 2007, Bernard-Michel *et al.*, 2009a, Bernard-Michel *et al.*, 2009b]. Quelques corrections de bugs et modifications mineures ont été effectués pendant ce stage sur cette méthode ; c'est pourquoi nous ne détaillons pas plus la méthode GRSIR et son implémentation. Un aperçu du travail effectué en amont de ce stage est présenté en annexe C.

2.5.2 Futurs algorithmes

Le projet Vahiné va se poursuivre au moins jusqu'en juin 2011, dans ce cadre là nous présentons de façon succincte les prochains algorithmes d'intérêt. Ceux-ci sont disponibles sous forme de maquette et pourraient être implémentés grâce à la bibliothèque OTB.

2.5.2.1 BPSS

L'algorithme BPSS « Bayesian positive source separation » est une approche non supervisée† pour le démixage de données hyperspectrales. Il vise le même but que ELM-VCA mais effectue des calculs de type Monte-Carlo† par chaîne de Markov. L'algorithme actuel

est disponible comme maquette scientifique programmée avec Matlab. C'est un algorithme très gourmand en mémoire et temps de calcul mais qui donne de très bons résultats.

Cet algorithme sera le prochain développement dans le cadre du projet Vahiné. Son implémentation avec OTB pourra sûrement lui faire bénéficier d'une amélioration en terme de temps de calcul grâce aux facultés de parallélisation d'OTB.

Je commence actuellement l'étude mathématique de cet algorithme sur la base de l'article de référence [Dobigeaon *et al.*, 2009].

2.5.2.2 Waveanglet-MSF

A plus long terme, il est envisagé de développer un nouvel algorithme, fusion de deux maquettes existantes. Nous associerions l'algorithme Wavanglet à un algorithme de type Minimum Spanning Forest (MSF). Ce projet pourrait faire en plus l'objet d'une publication pour une étude comparative.

Waveanglet est une méthode de détection automatique supervisée. Elle nécessite une base spectrale de référence des corps chimiques supposés présents dans l'image. Elle utilise une transformée en ondelette pour effectuer une réduction de dimensionnalité spectrale. Enfin grâce à des critères de distance† mathématiques elle va classifier les spectres présents dans l'image. Cet algorithme a été développé initialement pendant un travail de thèse [Schmidt, 2007].

Minimum Spanning Forest est un algorithme de segmentation basé sur la théorie des graphe.

Étant donné un graphe non orienté et connexe, un arbre couvrant de ce graphe est un sous-ensemble qui est un arbre qui connecte tous les sommets [Wikipedia, 2011].

Un graphe peut comporter plusieurs arbres couvrants différents. On peut associer un poids à chaque arête (le coût de cette arête) et prendre la somme des poids des arêtes de l'arbre couvrant. Un arbre couvrant de poids minimal est un arbre couvrant dont le poids est plus petit ou égal à celui de tous les autres arbres couvrants du graphe.

En considérant que chaque pixel d'une image est sommet d'un arbre et si on définit une fonction coût entre chaque pixel adjacent alors la détermination d'un arbre couvrant minimal par rapport à un seuil permet de définir des zones homogènes. Dans le cas d'un seul arbre, on dit que l'on utilise l'algorithme « minimum spanning tree », mais si on généralise cela à un ensemble de graphes déconnectés on est dans le cas du « Minimum Spanning Forest ».

Cet algorithme permet à partir d'un pixel initial appartenant à une classe donnée de faire croître de proche en proche l'ensemble des pixels de cette classe. Nous avons une sorte de croissance régionale dont la caractéristique principale est d'être un arbre connexe. En définitive, une image peut être segmentée en régions homogènes en prenant en compte la cohérence spatiale des structures de l'image et pas seulement les signatures spectrales de chaque pixel pris indépendamment.

2.6 Conclusion

Pour résumer le travail effectué, j'ai fait le choix d'utiliser la bibliothèque OTB de par ses possibilités algorithmiques et pour satisfaire les contraintes juridiques du projet Vahiné. D'un point de vue technique de programmation, j'ai dû prendre en main des aspects avancés du langage C++ comme par exemple, la programmation générique.

Dans ce chapitre, nous avons vu l'implémentation d'un des algorithmes du projet Vahiné. Cet algorithme est représentatif du travail à effectuer pour tout autre module. Pour chaque algorithme, j'ai dû assimiler les notions mathématiques qui le sous tendent afin d'essayer de trouver la meilleure implémentation possible avec la bibliothèque OTB. Pour cela, une connaissance approfondie de cette bibliothèque s'est avérée indispensable. L'apprentissage a nécessité un important investissement en terme de temps mais chaque implémentation s'est révélée être un challenge intellectuel passionnant.

Pour le futur, l'expérience accumulée sur la bibliothèque me permettra d'améliorer l'implémentation des algorithmes. Ainsi, nous pourrions envisager sur ELM-VCA d'abaisser le temps de calcul en réalisant une seconde version utilisant au mieux l'aspect multiprocessus de la bibliothèque OTB.

Ce travail de calcul numérique nous permet de générer des fichiers d'images hyperspectrales contenant le résultat d'un ou de plusieurs traitements. L'étape suivante du point de vue du planétologue consiste maintenant à pouvoir les interpréter. Cela n'est possible que grâce à des outils de visualisation. Ceux-ci font l'objet du chapitre suivant.

Chapitre 3

Visualisation d'images hyperspectrales

Le projet Vahiné, qui a pour but de mettre en œuvre de nouveaux algorithmes, a aussi l'objectif de développer un logiciel permettant de piloter ces algorithmes mais aussi de visualiser et de manipuler les images hyperspectrales et leurs produits d'analyse. Je vais dans ce chapitre présenter les développements faits sur ce module de visualisation nommé VahinéView.

3.1 Problématique et objectifs

Le but est de réaliser un logiciel de visualisation et d'analyse pour les cubes hyperspectraux. Le cahier des charges est simple, nous voulons à minima les fonctionnalités suivantes :

- visualiser des images hyperspectrales ; c'est-à-dire naviguer à travers les cubes pour sélectionner des plans image ;
- avoir un outil graphique pour localiser des régions d'intérêt (ROI) ;
- extraire leurs spectres ou des statistiques ;
- exporter des spectres pour les intégrer sur d'autres outils de calcul ou des chaînes de simulation ;
- sauvegarder des tracés de spectres pour les intégrer dans des publications ou des rapports. Ce dernier point peut paraître trivial mais les formats classiques bitmap ne sont pas forcément les meilleurs pour être intégré dans des documents de type publications scientifiques.

Pour un usage courant, dès que nous voulons visualiser des cubes hyperspectraux, l'application a besoin de fonctionnalités basiques telles que la manipulation (zoom, superposition, déplacement, etc.) de couches (ou « layer ») raster. J'ai donc décidé de ne pas partir de zéro et surtout de ne pas « réinventer la roue ».

Nos développements devant être contractuellement sous licence libre « CeCill », j'ai décidé de réutiliser un logiciel et de lui greffer nos besoins. Voici mes critères principaux pour ce choix :

- il doit être sous licence open-source ;
- si possible multiplateforme, avec dans l'ordre de préférence Linux/Unix, MacOSX et Windows ;
- contenant des fonctionnalités basiques de traitement d'images ;
- facilement extensible grâce à l'usage de greffons (plugins) par exemple.

Après avoir effectué une étude de l'existant [Mercier, 2008], j'ai sélectionné quelques logiciels intéressants que je présente dans la section suivante.

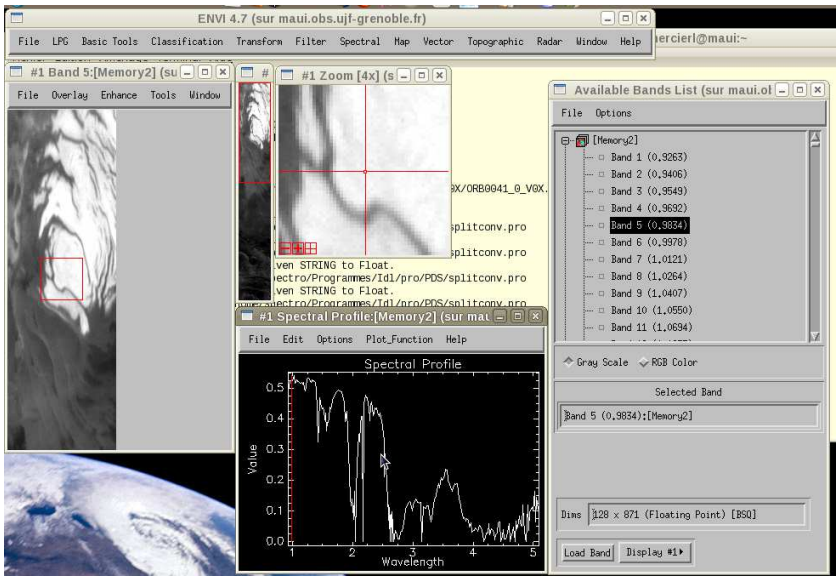

FIGURE 3.1 – Session ENVI. Invite de commande ENVI/IDL et visualisation d'un spectre.

3.2 Présentation des solutions disponibles

3.2.1 ENVI

Nous commencons par un logiciel n'étant pas une solution envisagée, mais qui a une part importante dans le milieu planétologique et plus généralement dans le domaine de l'imagerie spatiale. Ce logiciel ENVI me sert de référence dans notre projet.

De nombreux scientifiques utilisent et développent leurs algorithmes sous ENVI. Cela implique pour le laboratoire un coût de licence élevé et une dépendance vis-à-vis de l'éditeur et de sa politique de mise à jour. Suite à la naissance du projet Vahiné nous avons eu l'idée que si nous disposons d'un logiciel capable d'effectuer une certain nombre de traitements réduits mais spécifiques nous pouvons alors réserver ENVI à des usages plus restreints comme la mise au point de nouveaux algorithmes. Au final, nous réduirons notre dépendance à l'éditeur et le coût financier associé.

ENVI est un logiciel de traitement d'images qui intègre un moteur de script IDL (Interactive Data Language[1]). De fait ENVI est un environnement de calcul et de visualisation. Le lecteur trouvera de plus amples informations sur le site internet de l'éditeur http://www.ittvis.com/ProductServices/ENVI.aspx. La figure 3.1 présente un exemple de session ENVI sur notre serveur de calcul « Maui ».

3.2.2 QFitsView

Le premier candidat sélectionné provient du domaine astrophysique. Le logiciel QFitsView permet la visualisation de fichiers image FITS. Il a été développé grâce à la bibliothèque Qt et il est multisystème. Il intègre un interpréteur de script DPUSER. Ce dernier est un langage interprété qui est capable de manipuler des nombres (réels et complexes), des chaînes de caractères et des matrices. Son objectif principal est de pouvoir faire de l'analyse d'images astronomiques. Pour cela il fournit un ensemble de fonctions mathématiques et de traitement d'images.

1. Langage de programmation propriétaire dédié à la visualisation et au traitement de données. Ce langage est très répandu dans la recherche.

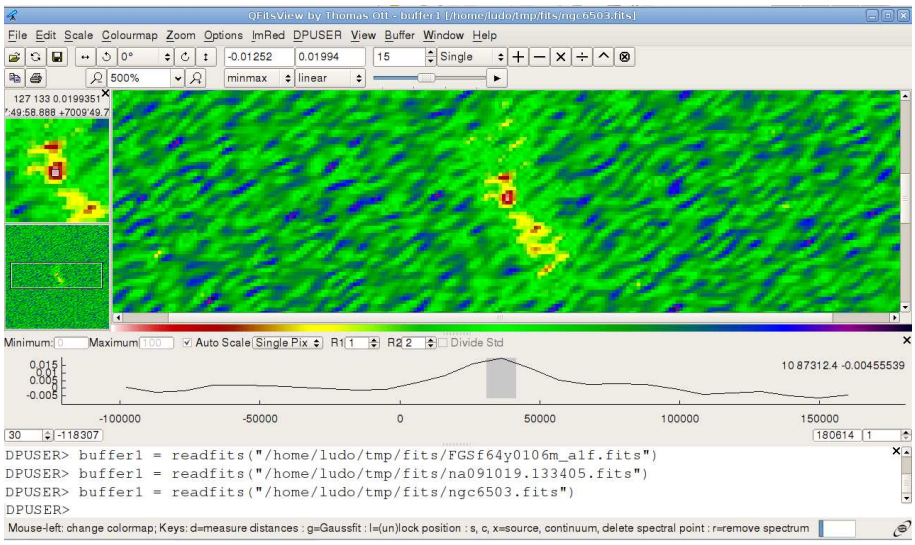

FIGURE 3.2 – Session de travail QFitsView. Visualisation d'un spectre et invite de commande DPUSER.

Le point faible de QFitsView réside dans le peu d'activités de développement et une communauté d'utilisateurs et de développeurs inexistante sur internet. Du moins elle n'est pas visible. Son point fort est qu'il dispose d'une bonne interface avec des fonctionnalités intéressantes comme un système de calculatrice de bandes spectrales. C'est une des fonctionnalités qui nous intéresse pour Vahiné. Chaque bande spectrale d'une image raster peut être vue comme une matrice. On peut donc lui appliquer des opérations mathématiques matricielles et construire des expressions avec comme opérandes des bandes spectrales.

Un exemple d'utilisation de QFits est présenté sur la figure 3.2.

Pour plus de renseignements sur ce logiciel le lecteur pourra se référer à la page internet http://www.mpe.mpg.de/~ott/QFitsView/.

3.2.3 QGis

QGis est un logiciel libre sous licence GPL dont le développement et l'hébergement est géré par la fondation OSGEO http://www.osgeo.org/. Les points forts de ce logiciel sont les suivants :

- Il peut être enrichi par l'ajout de greffons (« plugins ») . Ceux-ci peuvent être développés en Python ou C++.
- La communauté de développeurs et d'utilisateurs est très importante, nous bénéficions donc de nombreux exemples de développements et d'usages.
- Des tests d'intégration de la bibliothèque OTB ont déjà été effectués par l'équipe de développeurs OTB.

Le point faible dans notre cas est l'absence totale de traitement hyperspectral. Une copie d'écran d'une session de travail est présentée sur la figure 3.3.

3.2.4 Monteverdi

C'est un logiciel libre développé par le CNES dans le cadre du programme Orfeo. Ce logiciel est développé en C++ et permet l'utilisation de la boîte à outils OrpheoToolBox. Il est donc parfaitement adapté pour intégrer et piloter des algorithmes de calcul basés sur l'OTB. Il intègre optimalement la notion de chaîne de traitements (ou « pipeline ») avec en

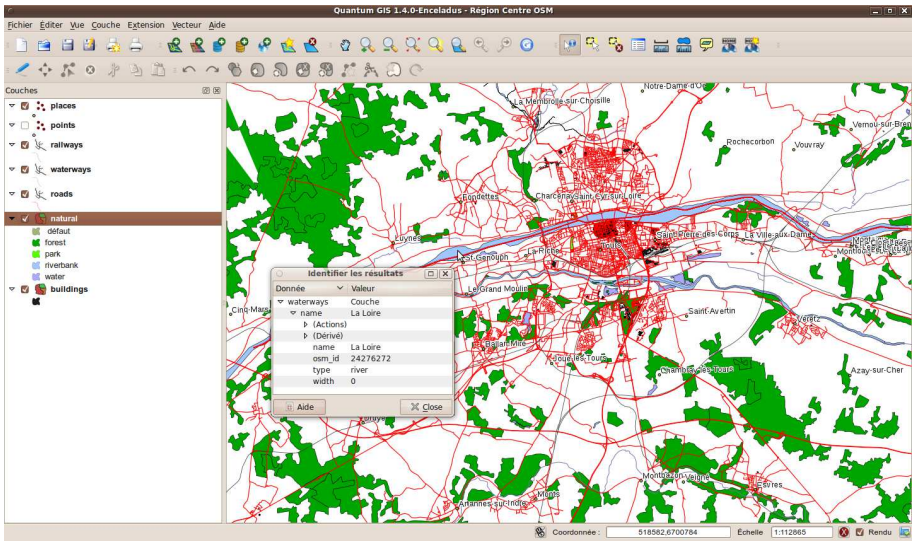

FIGURE 3.3 – Session de travail QGis sur une carte terrestre.

parallèle la possibilité d'effectuer une lecture en temps réel de l'image (« streaming »). Le point faible relevé est une IHM entièrement en C++ donc longue et complexe à développer et modifier. Ce logiciel ne bénéficie pas non plus actuellement d'une grande communauté.

3.2.5 Choix et conclusion

Suite à l'analyse des points forts et des points faibles j'ai retenu comme base de développement le logiciel QGis. Son développement et la communauté associée est très active. L'aspect géotraitement et les fonctionnalités disponibles qui nous permettraient de nous connecter au serveur GIS du laboratoire sont un atout considérable pour le futur. L'idée initiale était aussi de disposer de l'interfaçage python mis en place par l'équipe OTB. Cela nous permettrait de connecter nos algorithmes de calcul à QGis par l'intermédiaire d'un greffon Vahiné développé en Python ou en C++.

3.3 Multivue et visualisation spectrale

La première fonctionnalité à implémenter dans QGis est la visualisation multivue. En effet si dans QGis nous pouvons charger en mémoire plusieurs images, nous ne pouvons en visualiser plusieurs en même temps. Le problème de cette fonctionnalité et de son implémentation dans QGis est qu'elle touche au cœur du logiciel. La documentation étant peu développée sur le fonctionnement interne du logiciel j'ai passé du temps à comprendre le fonctionnement interne de QGis en effectuant un travail d'ingénierie inverse†.

Celui-ci m'a permis d'établir un certain nombre de diagrammes UML (« Unified Modeling Language ») pour modéliser le fonctionnement interne de QGis. Une partie de ce travail est exposée dans la section suivante.

3.3.1 Ingénierie inverse et modélisation de QGis

La classe `QgsMapCanvas` et l'élément graphique qui en découle est probablement la plus importante au sein de QGis. Elle permet l'affichage des cartes et des différentes couches (ou « layer ») ainsi que l'interaction avec la palette d'outils disponibles dans l'interface. La figure 3.5 montre les liens entre la classe `QgsMapCanvas` et ses collaboratrices. L'implémentation

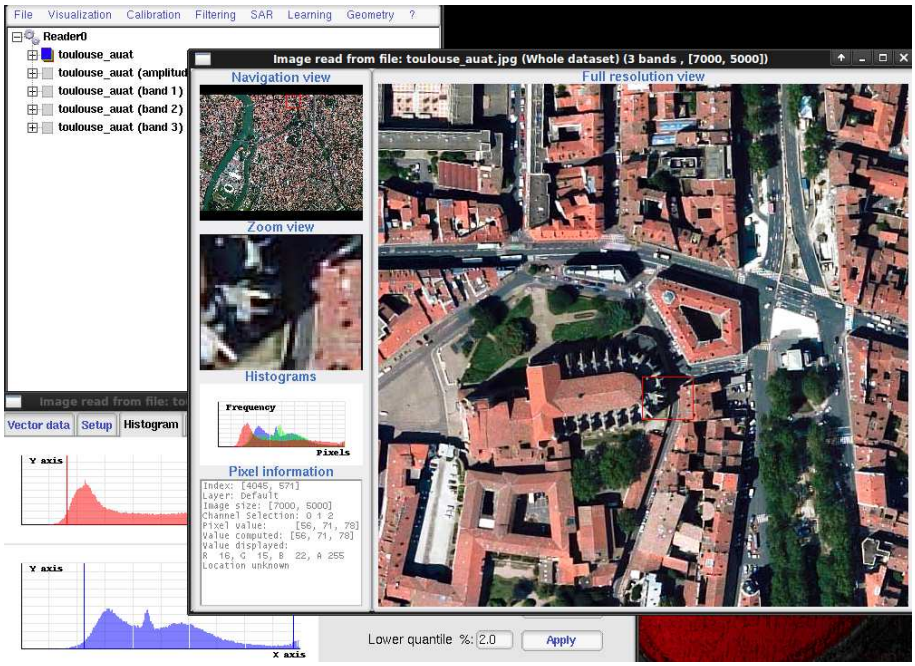

FIGURE 3.4 – Visualisation d'une image hyperspectrale sous Monteverdi.

de `QgsMapCanvas` est basée sur l'usage et les propriétés de la bibliothèque Qt. Dès que la carte affichée subie des transformations de déplacement (l'outil « panned ») ou de zoom, le rendu de la carte est recalculé en tenant compte des nouvelles frontières d'affichage. Les différentes couches sont affichées comme une image grâce à la classe `QgsMapRenderer`, qui fait elle même appel à la classe Qt : `QImage`. Cette image représentant les différentes couches visibles est alors affichée dans un « canevas† » Qt. C'est la classe `QgsMapCanvasMap` qui est responsable de l'affichage de la carte et qui contrôle son rendu. Si nous avons besoin d'autres éléments graphiques à afficher sur le « canevas » alors ces éléments doivent hériter de la classe `QgsMapCanvasItem`. C'est le cas si l'on désire afficher en surimpression des polygones ou zones d'intérêts.

L'étude du code source de QGis a permis de localiser une autre classe importante : la classe `QgsMapLayerRegistry`. Celle-ci est définie comme un patron de développement (ou « design pattern† » en anglais) de type « singleton ». Le pattern singleton garantit qu'une classe a seulement une unique instance et fournit un point d'accès global à cette instance [Freeman et Freeman, 2005].

Cette classe permet d'assurer un suivi des différentes couches qui sont à un instant donné chargées en mémoire. Elle fournit un moyen de récupérer un pointeur sur l'une des couches pour la manipuler. La figure 3.6 montre le lien entre `QgsMapLayerRegistry` et les classes de gestion des différents types de couches. Le listing 3.1 montre l'utilisation de la classe singleton `QgsLayerRegistry` avec l'appel de son instance unique.

Pour terminer cet aperçu des éléments centraux de QGis, il nous faut rajouter la gestion des légendes. Ce sont les éléments graphiques qui permettent d'identifier chaque couche. Cette gestion est effectuée par la classe `QgsLegend`. Celle-ci permet de gérer les différentes couches par l'intermédiaire d'un objet de l'IHM dédié. La classe `QgsLegend` est une spécialisation de l'objet Qt : `QListView`. Cet objet est conçu pour afficher : des groupes de couches, les couches de la carte, les propriétés et les symboles associés à chaque couche.

Cet objet graphique permet aussi la gestion des opérations simples telles que l'affichage d'une liste ordonnée de couches disponibles sur la carte, et des opérations complexes telles que le glisser / déposer, la symbologie des couches comme la carte de couleur associée par

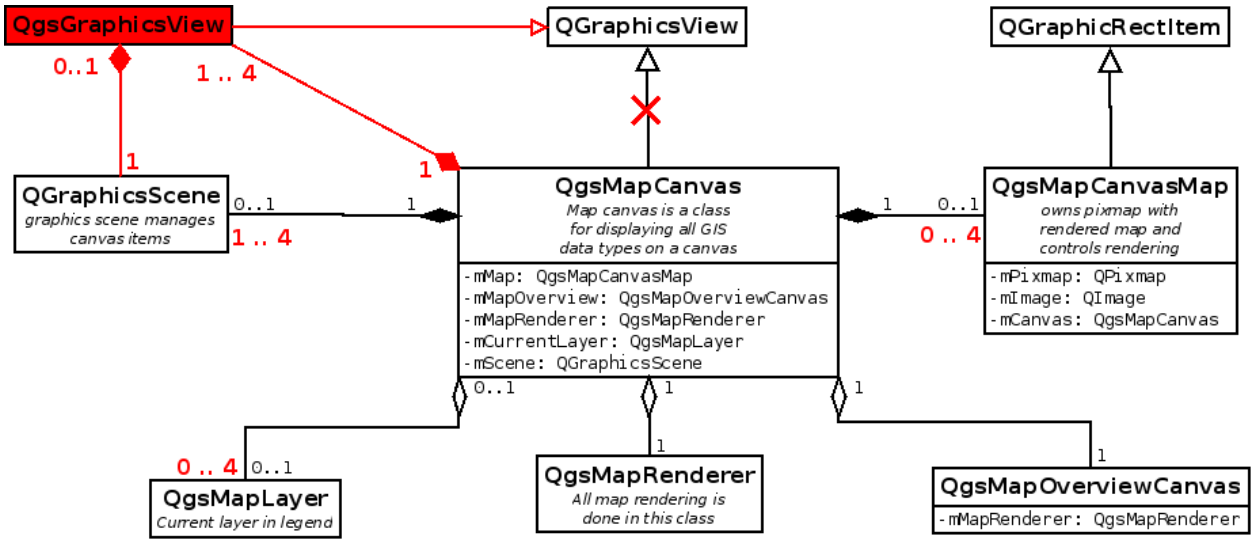

FIGURE 3.5 – Diagramme de classes pour `QgsMapCanvas`.

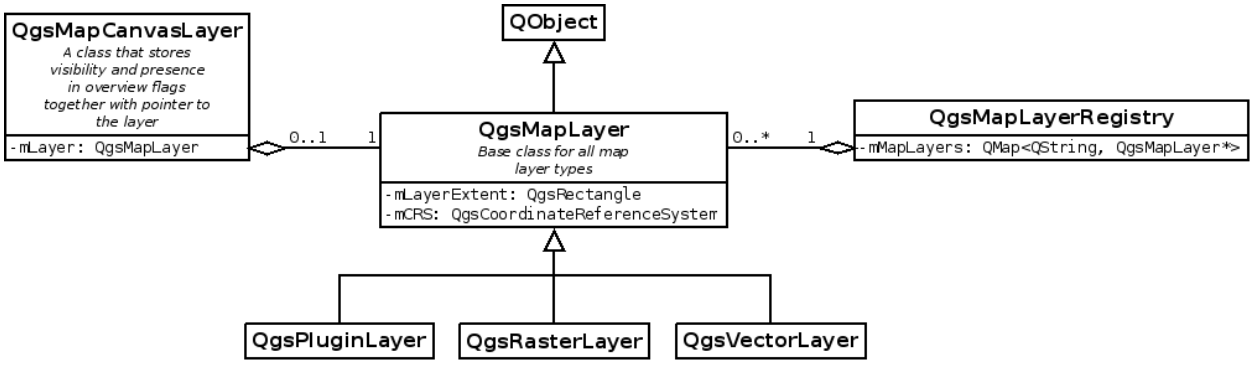

FIGURE 3.6 – Diagramme de classes pour `QgsMapLayer`. Les modifications apportées sont figurées en rouge.

```
...
QgsMapLayerRegistry::instance()->removeMapLayer( ml->getLayerID() );
...
```

Listing 3.1 – Exemple d'utilisation de l'instance de la classe `QgsMapLayerRegistry`.

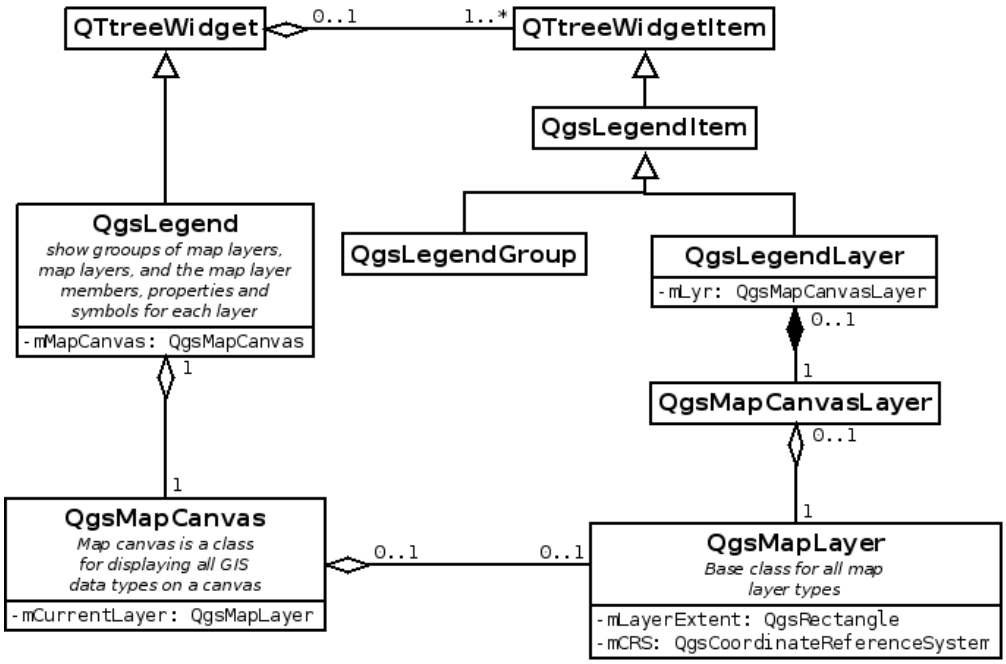

FIGURE 3.7 – Diagramme de classes pour `QgsLegend`.

exemple. Il existe plusieurs objets qui peuvent apparaître dans un objet `QgsLegend`. Tous les éléments ajoutés à un `QgsLegend` doivent hériter de la classe `QgsLegendItem`. Nous avons donc à disposition deux classes : `QgsLegendGroup` et `QgsLegendLayer` comme nous pouvons le voir sur la figure 3.7.

3.3.2 Interactions entre classes

Les interactions entre instances de classes dans QGis sont gérées par des appels directs aux méthodes de classes publiques tel que le permet le paradigme objet et le langage C++ mais aussi grâce à l'usage intensif de la technologie des signaux de la bibliothèque Qt.

3.3.2.1 Signaux Qt

La bibliothèque Qt propose un moyen de communication entre les objets appelés signaux et slots. Ce système est très utilisé dans la gestion des IHM car il permet de découpler fortement les classes entre elles et d'obtenir des comportements fluides.

Voici les grands principes de l'usage de cette technique :

- toutes les classes peuvent avoir plusieurs slots et autant de signaux que voulus ;
- tout signal peut se connecter à autant de slots que voulus et vice versa ;
- si les signaux et les slots sont des propriétés publiques, toute classe peut se connecter à chaque fois qu'elle le souhaite ;
- chaque classe peut déconnecter un de ses slots ou un de ses signaux à n'importe quel moment si elle n'est plus intéressée par les événements associés ;
- si une classe est détruite, elle se déconnecte automatiquement de l'ensemble de ses signaux et ses slots.

Un exemple d'interaction est présenté sur la figure 3.8. Sur cette figure le « signal 1 » de la classe C est connecté à la fois au « slot X » de la classe A et au « slot Y » de la classe B. Le « slot X » de la classe A reçoit aussi le « signal 1 » de la classe B.

Un des avantages de cette technique est qu'elle minimise les modifications. Une nouvelle

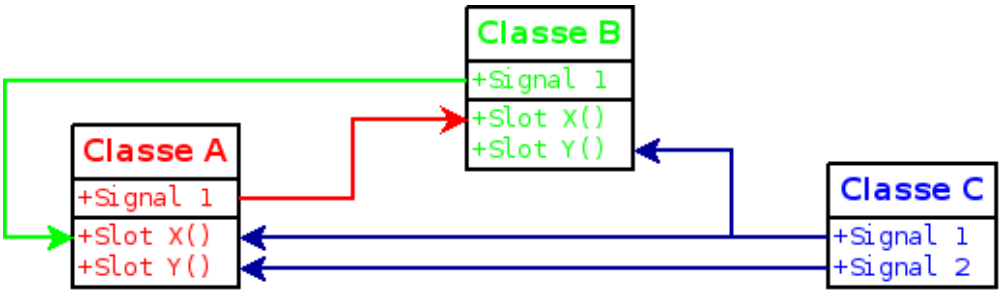

FIGURE 3.8 – Illustration de la technologie signaux/slots de Qt.

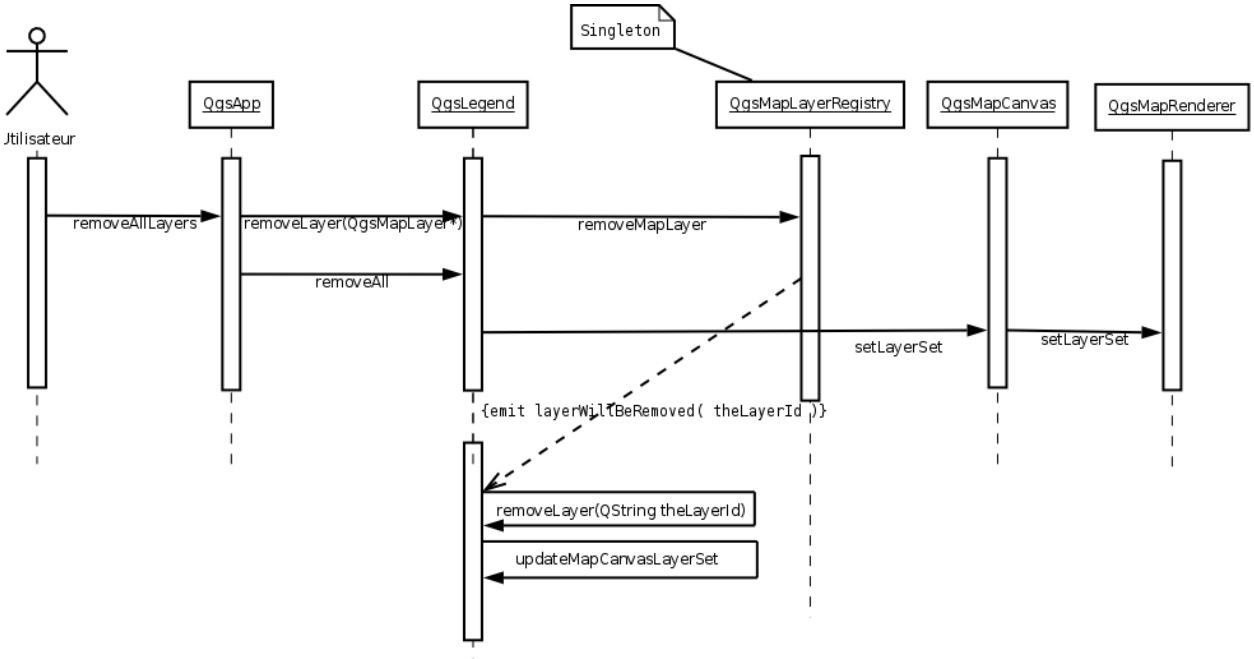

FIGURE 3.9 – Diagramme de séquence montrant l'interaction entre layer et legend.

classe qui veut être informée d'un signal associé à la classe A a juste à déclarer une connection (slot) sur ce signal. Aucune modification de la classe A n'est nécessaire.

3.3.2.2 Interaction dans QGis

La déclaration d'un slot est juste liée à la déclaration d'une fonction associée appelée « call-back » ou fonction de rappel. Cette fonction est exécutée quand un événement (ou signal) arrive sur le slot. Sur le listing 3.2, la macro-définition située en ① permet de simplifier la déclaration des signaux et slots. Ainsi au point ② nous déclarons deux slots. Déclarations qui sont elles-mêmes les entêtes de deux méthodes de notre classe `QgisApp`. De même la déclaration des signaux est très simplifiée. Au point ③ un signal est déclaré, son nom est le nom de la méthode, ici `keyPressed` et il accepte même un paramètre qui pourra être traité au moment de la réception par la méthode de « call-back » appropriée.

L'usage des « signaux et slots » Qt dans QGis est essentiellement géré dans la classe `QGisApp`. Un extrait des déclarations de connexions entre signaux et slots est présenté sur le listing 3.2. La figure 3.9 montre quant à elle un exemple d'interaction entre les classes de gestion des couches et la légende. L'association au niveau du signal Qt lancée par la classe `QgsMapLayerRegistry` est déclarée au point ④ du listing.

```
class QgisApp :
{
  Q_OBJECT                                                                    ①
...
public slots:                                                                 ②
  void addRasterLayer();
  void addDatabaseLayer();
...
signals :                                                                     ③
    /** emitted when a key is pressed and we want non widget sublasses to be able
      to pick up on this (e.g. maplayer) */
    void keyPressed( QKeyEvent *e );
...
// connect map layer registry signals to legend
  connect( QgsMapLayerRegistry::instance(), SIGNAL( layerWillBeRemoved( QString ) ),   ④
           mMapLegend, SLOT( removeLayer( QString ) ) );
  connect( QgsMapLayerRegistry::instance(), SIGNAL( removedAll() ),
           mMapLegend, SLOT( removeAll() ) );
  connect( QgsMapLayerRegistry::instance(), SIGNAL( layerWasAdded( QgsMapLayer* ) ),
           mMapLegend, SLOT( addLayer( QgsMapLayer * ) ) );
  connect( mMapLegend, SIGNAL( currentLayerChanged( QgsMapLayer* ) ),
           this, SLOT( activateDeactivateLayerRelatedActions( QgsMapLayer* ) ) );
  connect( mMapLegend, SIGNAL( currentLayerChanged( QgsMapLayer* ) ),
           mUndoWidget, SLOT( layerChanged( QgsMapLayer* ) ) );
  connect( mMapLegend, SIGNAL( currentLayerChanged( QgsMapLayer* ) ),
           mMapTools.mNodeTool, SLOT( currentLayerChanged(
           QgsMapLayer* ) ) );
...
```

Listing 3.2 – Extrait de la classe `QGisApp`. Connexion de différents signaux Qt

3.3.2.3 Modèle de conception : le pattern Observateur

Pour conclure sur le fonctionnement des interactions et du système signaux/slots Qt, il nous faut revenir au design pattern. Ici c'est le pattern observateur qui est utilisé. Il définit une relation entre objet de type un-à-plusieurs, de façon que, lorsqu'un objet change d'état, tous ceux qui en dépendent en soient notifiés et soient mis à jour automatiquement [Freeman et Freeman, 2005]. Ce pattern permet un couplage faible entre objets. En minimisant l'interdépendance, cela permet de construire un système orienté objet souple et évolutif. Dans le cas de Qgis, cela permet une architecture modulaire et donc flexible dans les usages finaux.

3.3.3 Modification apportée : l'aspect multivue

Après réflexion, j'ai fait le choix d'introduire un nouvel objet dans les collaborations entre classes. Ainsi j'ai développé la classe `QgsGraphicView` qui vient s'interposer entre la classe `QgsMapCanvas` et la classe `QgsMapLayer`. Les nouvelles interactions sont représentées en rouge sur la figure 3.6.

J'ai décidé au vu de l'usage fait dans le laboratoire de pouvoir disposer de quatre vues indépendantes. Les interactions initialement liées à la classe `QgsMapCanvas` ont été déplacées sur la classe `QgsGraphicView`. Celles-ci qui s'appuyaient sur des pointeurs uniques sont maintenant gérées par des listes de pointeurs contenant quatre éléments. La valeur maximum « quatre » est définie globalement et pourrait être augmentée. Il faudrait alors revoir le type de disposition que nous pourrions proposer au niveau de l'IHM.

En plus des nombreuses modifications autour de la classe `QgsMapCanvas`, j'ai aussi ajouté une entrée dans la barre de menu principal de QGis. Cette partie fait appel au système de gestion d'évènements Qt (signaux/slots) présenté précédemment.

Notre modification permet d'afficher au choix :

– une image ;

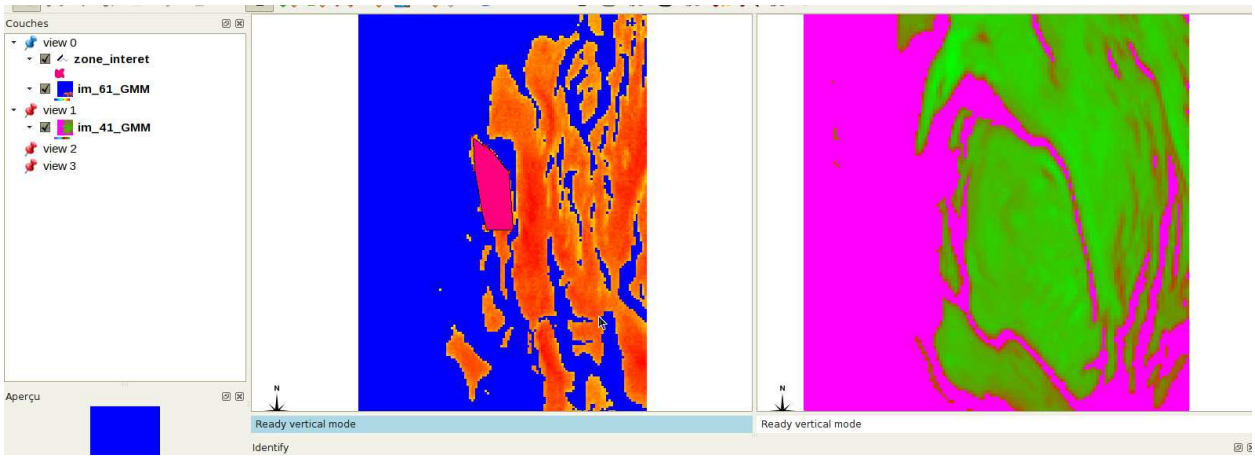

FIGURE 3.10 – Nouvel affichage de QGis avec la fonctionnalité multivue activée pour un mode vertical.

FIGURE 3.11 – Le menu d'activation du mode multivue.

– deux images verticales ;
– deux images horizontales ;
– quatre images dans une grille 2x2.

Une copie d'écran du résultat obtenu est disponible sur la figure 3.10. Celle-ci présente le mode d'affichage multivue activé en mode deux vues verticales. La modification de l'affichage est déclenchée par l'utilisateur via le menu « Vue ». Une copie d'écran de ce menu est présentée sur la figure 3.11.

Des discussions sont en cours sur la liste de diffusion dédiée aux développeurs de QGis pour essayer d'intégrer cette fonctionnalité, dans une prochaine version. Il faudrait dans un premier temps effectuer un travail de restructuration pour pouvoir livrer la modification sous la forme d'une série de « patch » de taille minimale. De nombreux utilisateurs étant demandeurs de cette fonctionnalité la communauté Qgis devra mettre en place un groupe de travail pour réaliser cette tâche.

3.3.4 Nouvelles fonctionnalitées hyperspectrales

Une part importante des développements a été consacrée à la réalisation d'un nouvel outil permettant d'extraire et d'afficher les informations dites hyperspectrales. Classiquement Qgis affiche des images avec seulement trois bandes (typiquement au format RVB[†]). Nos images comportant plusieurs centaines de bandes, j'ai mis au point un outil dédié dans QGis. En fait Qgis initialement dispose d'un outil appelé « identify » qui permet l'identification

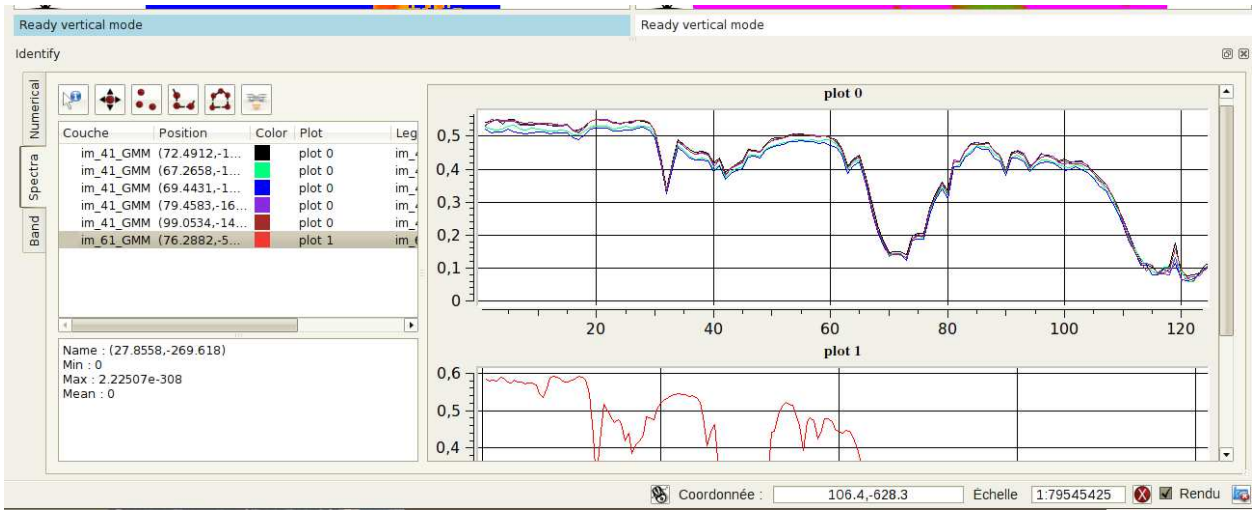

FIGURE 3.12 – Nouvelles fonctionnalités d'affichage des données hyperspectrales.

d'éléments disponibles pour un pixel donné. J'ai greffé nos fonctionnalités sur cet outil. Les développements ont conduit à modifier l'IHM de l'outil d'identification grâce au logiciel de développement QtDesigner. Celui-ci permet de créer ou modifier une interface de façon triviale. Il génère un fichier de configuration XML qui sera transformé à la compilation en classes gérant l'IHM. Ces classes peuvent alors être utilisées dans le code associé.

J'ai découpé l'interface utilisateur en trois zones comme le montre la figure 3.12. En haut à gauche nous avons un objet Qt `QTreeWidget` qui nous permet de lister chaque point sélectionné avec l'outil d'identification. Cette liste permet d'accéder aux attributs d'un point (position, couche d'appartenance, etc.). En bas à gauche, une fenêtre d'information non éditable nous permet d'afficher des informations générales. Nous affichons notamment les valeurs d'un certain nombre de fonctions mathématiques (minimum du spectre, moyenne, etc.). Ces valeurs sont précalculées dès la sélection d'un point. Enfin, la partie droite permet d'afficher autant de tracés de spectres que l'utilisateur désire. J'ai ajouté à cette visualisation des méthodes d'exportation sous forme vectorielle et tabulaire (format CSV) pour l'ensemble des courbes présentes sur un graphique.

Pour réaliser cette interface, j'ai utilisé la bibliothèque d'objets graphiques Qwt (`http://qwt.sourceforge.net/`). Celle-ci a du être intégrée dans le projet via le système de construction CMake (4.4.1) par modification du fichier `CMakeLists.txt` principal.

3.3.5 Greffon Vahiné

Nous avons vu dans la section 3.3 deux aspects de l'interface utilisateur du projet « Vahiné View ». Le dernier point à développer est d'établir un lien entre l'interface et nos algorithmes de calcul notamment l'algorithme ELM-VCA (Voir la section 2.4). L'idée pour réaliser cette fonctionnalité est d'utiliser le système d'extensions de QGis. Celui-ci permet d'étendre les fonctionnalités initiales en proposant un accès à l'interface de programmation de QGis. Seul le squelette d'un greffon a été implémenté par manque de temps. Je n'ai donc pas encore intégré nos algorithmes.

3.3.6 Architecture d'un greffon QGis

La construction d'un greffon QGis se fait en plusieurs étapes.

La première étape consiste à faire reconnaître le greffon par le gestionnaire (plugin manager). Nous devons pour cela créer une nouvelle entrée appelée « vahine » dans l'arborescence des greffons. Le fichier CMakeLists.txt parent est modifié en conséquence.

Ce répertoire vahine va contenir lui aussi un fichier `CMakeLists.txt`. Celui-ci définit les fichiers contenant le code de gestion de l'interface et les fichiers de ressources (image, fichier XML provenant de QtDesigner, etc.).

A ce point, nous avons donc six fichiers :

- `CMakeLists.txt` ;
- `qgsvahine.cpp` ;
- `qgsvahine.h` ;
- `qgsvahinegui.cpp` ;
- `qgsvahinegui.h` ;
- `vahineplugin.png`.

Le fichier bitmap `vahineplugin.png` définit l'icone représentant notre futur greffon dans l'interface de QGis. Les fichiers principaux `qgsvahine.h` et `qgsvahine.cpp` contiennent la classe principale `QgsVahinePlugin`, qui hérite de la classe `QgisPlugin`. Cette dernière est la classe parente de tout greffon Qgis. Les fichiers `qgsvahinegui.h/.cpp` contiennent le code de gestion de l'interface, c'est la partie contrôleur si on se place dans le cadre du modèle MVC. La vue est décrite dans le fichier `vahineguibase.ui` qui n'est pas listé ici. Il est issu de QtDesigner. Le modèle correspond à la classe `QgsVahinePlugin`.

```
class QgsVahinePlugin: public QObject, public QgisPlugin
{
    Q_OBJECT
  public:
    QgsVahinePlugin( QgisInterface * theInterface );
 ...
```

Au démarrage de QGis le processus principal va instancier l'ensemble des greffons censés être actifs (l'activation ou non d'un greffon est déterminée par la configuration). Le processus d'instanciation passe par l'usage d'une fonction `classFactory` qui est chargée de créer une instance de notre classe `QgsVahinePlugin`.

```
QGISEXTERN QgisPlugin * classFactory( QgisInterface * theQgisInterfacePointer )
{
  return new QgsVahinePlugin( theQgisInterfacePointer );
}
```

Nous sommes ici typiquement dans un cas d'usage du pattern fabrication. Celui-ci permet à une classe de déléguer l'instanciation à des sous-classes. L'application principale connait maintenant notre nouveau greffon.

La seconde étape du développement d'un nouveau greffon est donc de générer l'interface graphique. J'utilise pour cela QtDesigner. Cela nous génère le fichier de description d'interface `vahineguibase.ui`. Ce fichier de description qui en fait au format XML sert d'élément d'entrée à un analyseur syntaxique pour obtenir une classe décrivant notre interface.

3.4 Conclusion

Le développement de notre outil graphique a été assez long et n'est pas encore terminé. Le fait de devoir modifier le cœur de l'application QGis n'avait pas été prévu au début du projet. Un patch a néanmoins été proposé à la communauté QGis. Après discussion sur la liste de diffusion, il s'avère qu'une équipe de développeurs a aussi travaillé en parallèle sur l'aspect multivue. L'équipe qui gère le planning de QGis ne désire pas pour le moment

intégrer ces modifications car cela impacte fortement la partie centrale de l'application. Je vais poursuivre ma collaboration avec l'équipe de QGis en essayant de définir une suite de modifications minimales qui pourraient être intégrées incrémentalement. La base de nos discussions va porter sur les diagrammes des collaborations présentées précédemment (figure 3.7).

Concernant les outils hyperspectraux, ceux-ci ont été pour le moment intégrés dans le code même de QGis. A long terme compte tenu du développement très rapide de QGis (de deux à trois version par an) ce n'est pas une solution optimale. La meilleure stratégie semble être de migrer ces développements sous forme de greffons Python qui pourront être maintenus plus facilement au sein du laboratoire. La suite du travail va consister aussi à améliorer l'outil d'analyse hyperspectrale en lui ajoutant notamment des fonctions de moyennage de zones d'intérêt.

Le squelette du greffon C++ présenté précédemment va quant à lui, permettre de lancer l'algorithme ELM-VCA ou un autre algorithme depuis QGis. Le but ultime étant, pour rappel, d'appliquer les différents traitements numériques à une image chargée dans QGis et de visualiser les résultats dans QGis.

Pour effectuer les développements de ce chapitre et des précédants, j'ai essayé d'appliquer une méthode agile. Celle-ci implique une planification et un certain nombre d'outils que nous décrivons dans le chapitre suivant.

Chapitre 4

Planification, méthodologie et outils

4.1 Planification

Nous présentons ici un résumé du déroulement des différentes tâches. La figure 4.1 reprend de façon synthétique les grandes étapes du planning réellement effectué.

Nous avons vu au chapitre 3 le développement de l'aspect multivue de l'interface. Celui-ci n'était pas prévu initialement dans notre planning. Il s'est avéré très coûteux en temps et a conduit à un glissement de l'ensemble des tâches. Néanmoins, une partie du temps de glissement a pu être résorbée. Sur les quatre algorithmes que je devais implémenter, nous avons aussi eu un changement de priorité. L'algorithme ELM-VCA a été développé. L'algorithme SSA-MVSA n'est plus prioritaire au vu des derniers résultats scientifiques obtenus entre temps avec les maquettes. Il est peu performant par rapport à ELM-VCA. Quant à l'algorithme Waveanglet, il est encore intéressant, mais sa priorité a été revue car un nouvel algorithme nommé BPSS plus pertinent sera implémenté prochainement. J'ai pu commencer à travailler sur l'aspect mathématiques de BPSS pour sa réalisation dans les prochains mois.

Comme on peut le voir, le cahier des charges est assez changeant. C'est à la fois une difficulté liée au milieu de la Recherche, mais aussi un challenge en terme de gestion de projet et de réalisation technique.

Concernant les couches basses d'accès aux données elles sont pleinement opérationnelles. Elles sont en cours d'intégration dans l'équipe de GDAL.

4.2 Processus de travail

Nous avons exposé au chapitre 2 l'implémentation des traitements numériques dont l'algorithme ELM-VCA.

Le développement de ces algorithmes au sein de Vahiné est décrit dans des articles scientifiques et s'incarne dans des maquettes plus ou moins bien documentées. Suite à ce constat, j'ai établi un protocole d'implémentation pour ces algorithmes. Celui-ci se base sur la méthode de développement logiciel « Test Driven Development » (TDD) que nous pouvons traduire par « développement piloté par les tests ».

Voici un aperçu de ce processus de conception et développement :

1. Analyse mathématique de l'algorithme et traduction en terme de flux de données.
2. Analyse de la maquette et repérage des fonctions critiques comme par exemple, une fonction de haut niveau en Matlab qui n'a pas d'équivalent en C++.

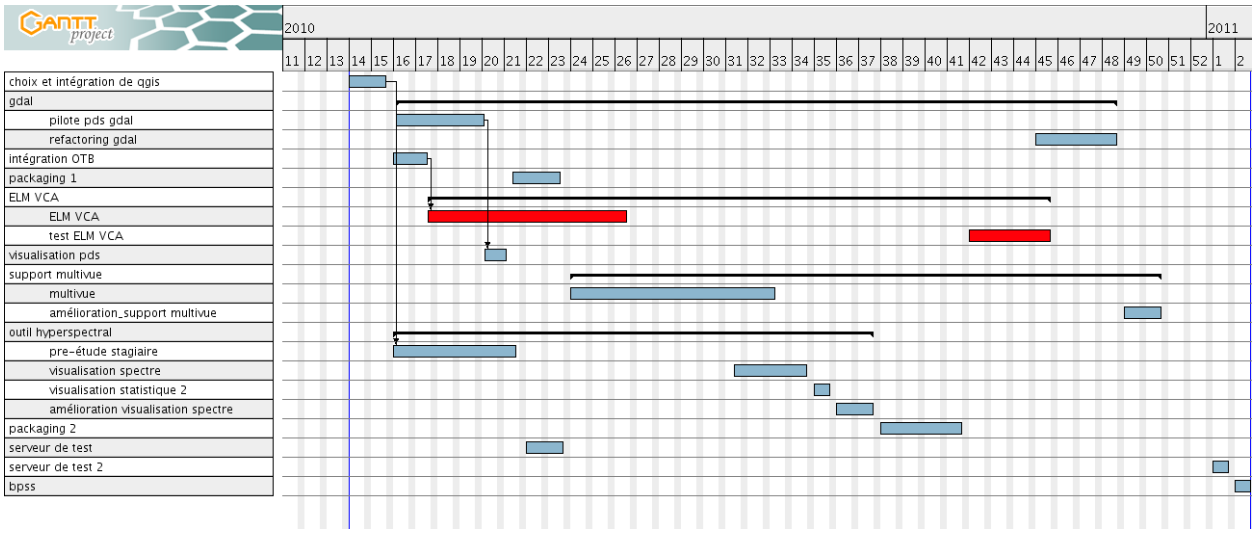

FIGURE 4.1 – Planning synthétique des tâches effectuées.

3. Création d'un jeu de tests dans le langage de la maquette [1]. Celui-ci permet de découper la maquette en blocs plus simples.
4. Modélisation en terme de classes. A cette étape, j'essaye de trouver les classes OTB intéressantes. Je modélise aussi si nécessaire les interactions entre classes. Nous avons à minima un diagramme UML de classes et si nécessaire un diagramme de séquence.
5. Création des jeux de tests C++. C'est la partie TDD officielle. Le lecteur se reportera à la section 4.3 pour plus de détails.
6. Implémentation du code de calcul en C++/OTB.
7. Tests numériques et scientifiques. A partir d'ici nous pouvons reboucler sur le point 5.

Après le développement et les premiers tests, si une défaillance apparait (« bug »), la procédure consiste à créer un test qui permet de reproduire ce « bug ». Ensuite seulement, je modifie le code source pour appliquer un correctif.

Dans la mesure du possible, quelque soit le développement j'essaye de suivre une méthode agile. Présentons succinctement ces méthodes et plus particulièrement l'une d'entre elles, la méthode XP.

4.2.1 Méthodes agiles

Les méthodes de développements logiciels ditent Agiles sont apparues au cours des années 1990 et ont été mises en avant par la publication du « Manifeste Agile » en février 2001. Celui-ci vise à promouvoir une nouvelle approche du développement logiciel en s'attachant à délivrer de la meilleure façon possible ce qui a de la valeur pour le client. La ligne de conduite étant de viser la satisfaction réelle du besoin du client et non les termes d'un contrat de développement.[Balbous, 2008]

4.2.1.1 Méthode XP

La méthode de gestion de projets appelée « eXtreme Programing » ou XP est très adaptée aux petites équipes avec des projets ayant des besoins changeants. Elle est basée sur une

1. Les algorithmes sont en général développés en Matlab et/ou IDL.

série de valeurs : communication, simplicité, rétroaction, courage et respect. Concrètement elle s'appuie sur des pratiques interconnectées :

- client sur site ;
- séance de planification ;
- intégration continue ;
- livraisons fréquentes ;
- rythme soutenable ;
- tests de recette ;
- tests unitaires ;
- conception simple.

4.2.1.2 Pourquoi XP et la Recherche ?

Après avoir travaillé dans plusieurs laboratoires variés, mon expérience me permet de caractériser un projet de recherche par les points suivants :

- les spécifications ne sont jamais complètement définies ;
- le ou les utilisateurs sont en général disponibles et sur le même site géographique ;
- les utilisateurs voudraient avoir la prochaine version du logiciel immédiatement ;
- les équipes de développement sont peu nombreuses. Les projets sont souvent gérés par une seule personne ;
- la seule donnée fiable est que le système à concevoir et à réaliser va forcément changer.

De plus, comme j'ai eu l'occasion de pratiquer de façon intensive la méthode XP en milieu industriel cela m'offre un élément de comparaison. Mon opinion est que la méthode XP est particulièrement bien adaptée aux projets de recherche. Le client en recherche est sur le site car nous avons à faire dans la majorité des cas à des chercheurs du laboratoire. Les projets sont relativement changeants et c'est là une des difficultés. La formalisation des projets de recherche est trop sommaire et l'environnement des laboratoires est très créatifs. En conclusion, le cadre du projet a du mal à se stabiliser. Les budgets étant débloqués de plus en plus sur projet, de nombreux chercheurs ou équipes obtiennent des financements pour des projets ayant seulement un poste d'ingénieur. C'est l'élément le plus négatif, nous sommes dans une optique « un projet égal un ingénieur ». Pour conclure cette digression qui me semblait importante, une solution serait de mutualiser les ressources de développement. C'est-à-dire de confier plusieurs projets à une équipe minimum de quatre développeurs en fonctionnement « agile ».

4.3 Tests logiciels

Les méthodes agiles que nous avons rapidement présentées précédemment ne seraient rien sans les outils de gestion de tests. C'est dès le début du projet que j'ai décidé de déployer une architecture de test complète. Cette architecture est couramment appelée dans le cadre des méthodes agiles : « système d'intégration continue ».

4.3.1 Architecture de test

La conception de l'architecture de test fait suite à l'idée de déployer un système d'intégration continue et s'inscrit dans la démarche de suivre les recommandations des méthodes Agile. Celle-ci consiste en un ensemble de pratiques et d'outils qui permettent de vérifier à tout moment (typiquement à chaque « commit† ») que les modifications n'introduisent pas de régressions dans le logiciel en développement. J'ai donc décidé de mettre en place

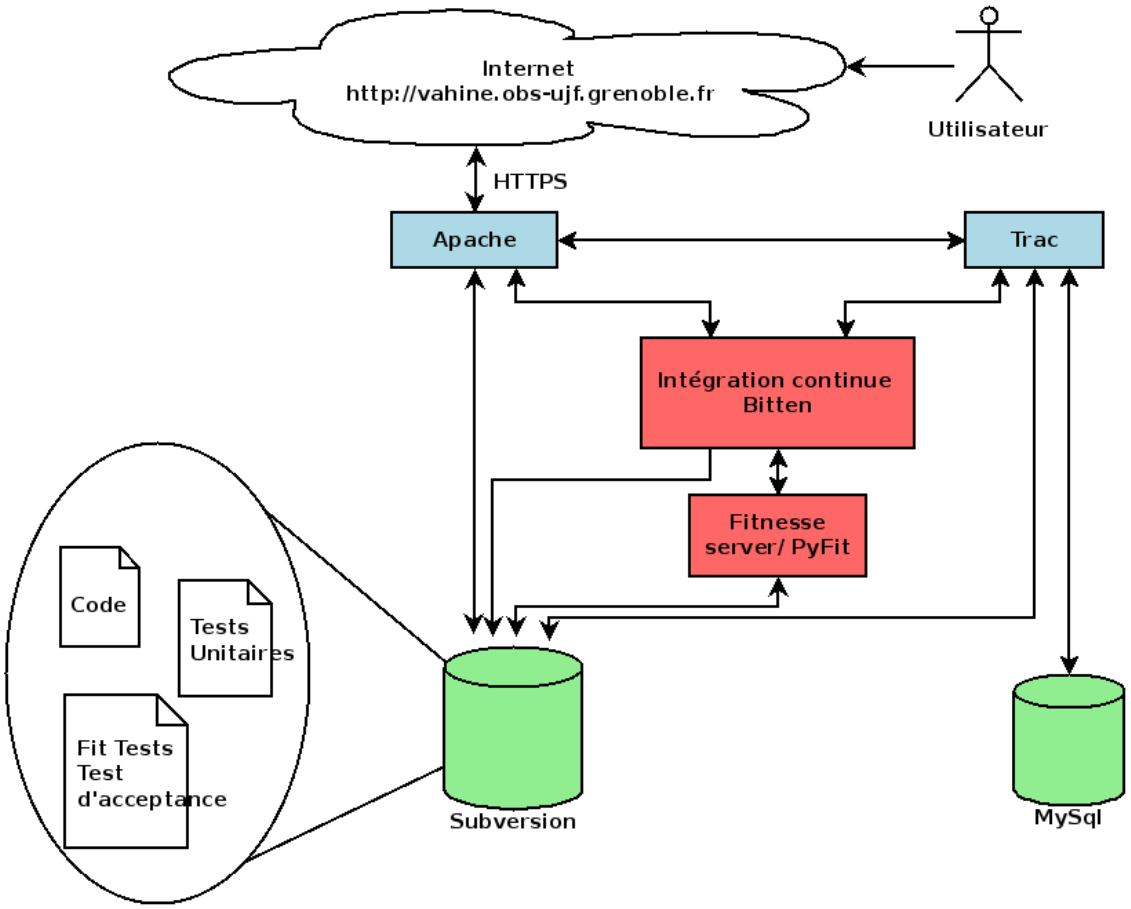

FIGURE 4.2 – Représentation de l'architecture prévue.

une architecture d'intégration continue représentée sur la figure 4.2. Un des critères est de pouvoir mutualiser ses efforts avec d'autres projets. Cette architecture s'articule autour de deux éléments : une gestionnaire de configuration (Subversion) et une plateforme web de suivi de projets (Trac).

La base de cette architecture repose sur un serveur de gestion de configuration subversion (voir a la section 4.3.4) et sur un système web de pilotage et de gestion de projets Trac (`http://trac.edgewall.org/`). Trac est un logiciel basé sur une interface web. Il se compose d'un wiki amélioré couplé avec un système de suivi de bugs et de tâches. Il fournit en outre une interface à Subversion et de nombreuses fonctionnalités sous forme de greffons. Une copie d'écran de notre instance de Trac est visible sur la figure 4.4.

Sur cette figure, apparaît un groupe de tests que je n'ai pas eu malheureusement le temps de développer, ce sont les tests d'acceptances[†]. L'ensemble des tests de bas niveau (les tests unitaires) concernant le code C++, sont réalisés par du code spécifique utilisant la bibliothèque Boost dont nous avons pu voir quelques exemples à la section 1.7.1.

4.3.2 Bibliothèque Boost

La bibliothèque de test Boost offre une boîte à outils pour faciliter la création et la maintenance de programmes de tests. Elle fournit comme on l'a vu précédemment un ensemble de Macro-définition qui ont la particularité en plus du test, de fournir une référence sur l'emplacement de l'erreur détectée. Un jeu de tests écrit avec cette bibliothèque est ainsi capable de renvoyer un rapport complet de tests au format XML avec des pointeurs sur les codes

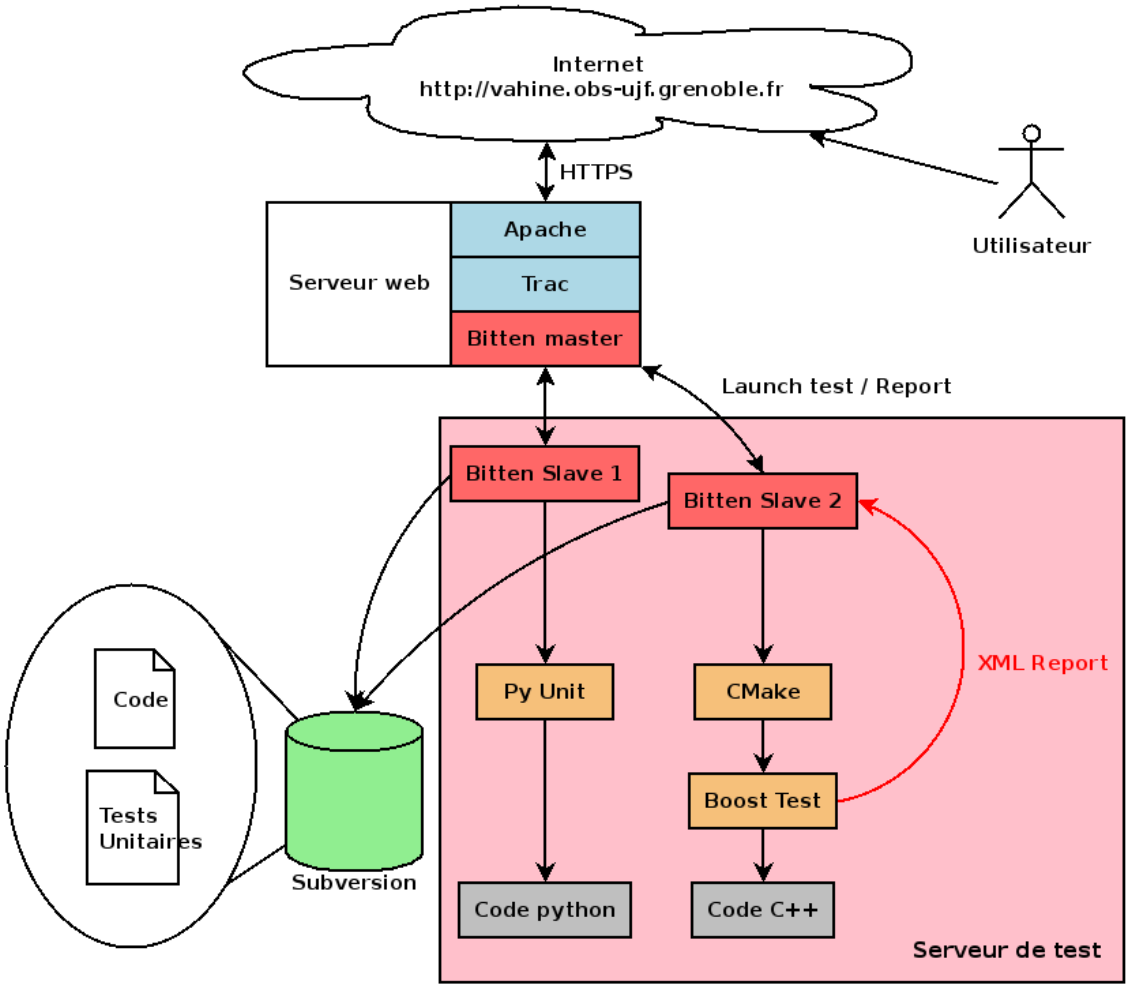

FIGURE 4.3 – Détail de l'architecture réalisée.

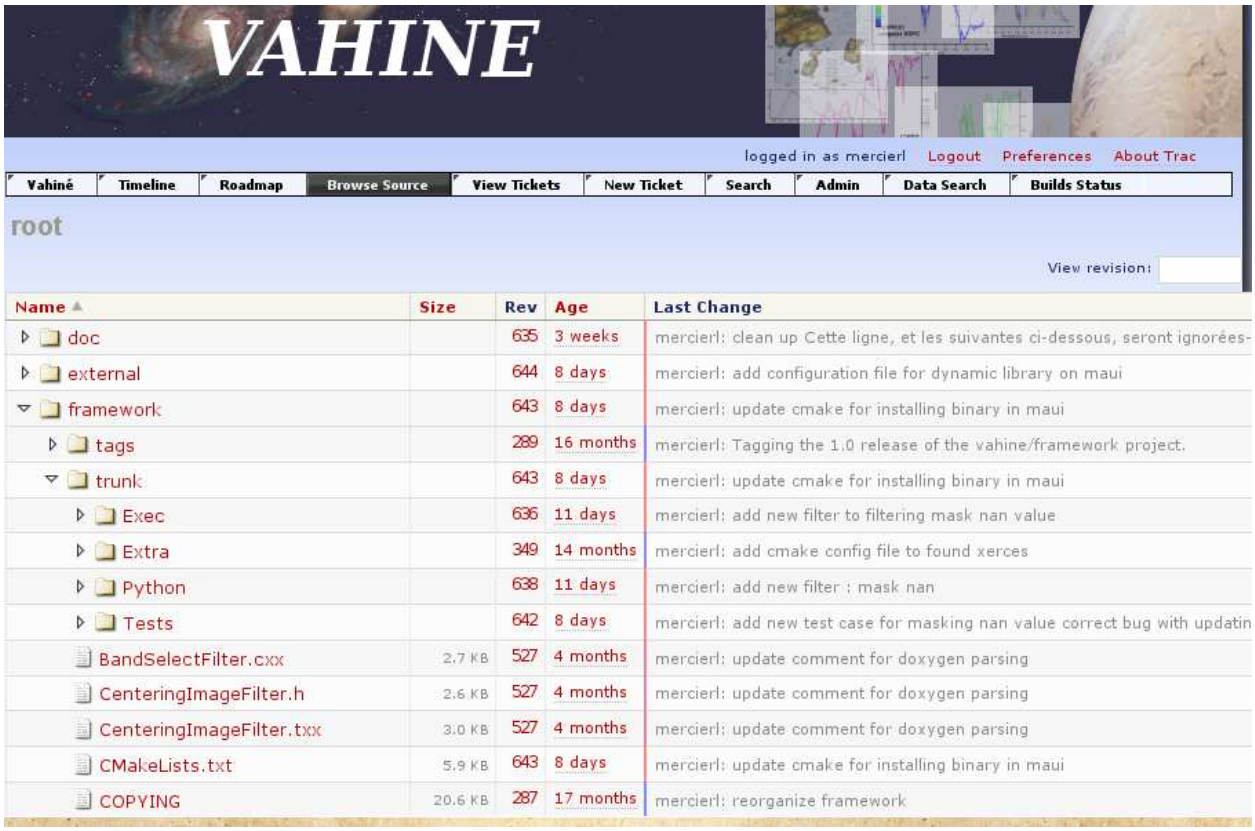

FIGURE 4.4 – Copie d'écran de l'instance de Trac pour Vahiné.

défaillants. Cela permet une intégration aisée dans des outils de supervision d'une chaîne d'intégration continue.

4.3.3 Intégration continue

Un système d'intégration continue permet d'exécuter automatiquement des tests ou des portions de code du logiciel en développement à partir de certains évènements. Par exemple, les tests peuvent être exécutés après un changement qui a été archivé dans le système de gestion de versions, ou sans condition particulière mais à des intervalles spécifiques (typiquement des « nightly builds »). Le système d'intégration permet aussi la mise à disposition des fichiers de traces (ou fichier de « log ») ainsi que la possibilité de générer des paquets ou des archives automatiquement.

Le but étant de réaliser la construction de l'application dans un environnement dédié avec une copie de la dernière version du code à partir du référentiel. Notre chaîne d'intégration s'appuie sur le logiciel Bitten, greffon à la plateforme Trac.

4.3.3.1 Greffon Bitten

Bitten est un greffon développé en python permettant la collecte de données de différents logiciels et fournissant donc un mécanisme d'intégration continue. Il s'appuie sur la plateforme Trac pour fournir une interface utilisateur intégrée contrôlable via un navigateur Web. L'architecture de Bitten fonctionne en mode maître/esclave 4.3. Un maître déclenche l'exécution d'une suite d'instructions appelée « recette » (ou « recipe ») gérant un ensemble d'actions. La configuration d'une « recette » est faite par un fichier XML définissant l'ensemble des tâches. Un exemple est proposé sur le listing 4.1.

```
<build description="building gdal test"
  xmlns:sh="http://bitten.edgewall.org/tools/sh"
  xmlns:svn="http://bitten.edgewall.org/tools/svn"
  xmlns:x="http://bitten.edgewall.org/tools/xml">
  <step id="checkout" description="Test - checkout">
    <sh:exec file="echo" args="Checkout start"/>
    <svn:checkout url="https://svn-lpg.obs.ujf-grenoble.fr/vahine/external/"
    path="gdal-test" username="${svn.username}" password="${svn.password}"/>
    <sh:exec file="echo" args="Checkout complete"/>
  </step>
  <step id="compile-test">
    <sh:exec file="echo" args="Build start"/>
    <sh:exec file="cmake" args="."/>
    <sh:exec file="make" />
    <sh:exec file="echo" args="Build stop"/>
  </step>
  <step id="run-test">
    <sh:exec file="echo" args="Test start"/>
    <sh:exec file="mkdir" args="Results"/>
    <sh:exec file="run_gdal_pds_tests" output="test_results.xml" args="--log_level=nothing --report_format=XML --
        report_level=detailed" />
    <sh:exec file="echo" args="Test complete"/>
  </step>
  <step id="report-test">
    <x:transform src="test_results.xml" dest="test_report.xml" stylesheet="${slave.path}/boost2bitten.xsl" />
    <report category="test" file="test_report.xml"/>
  </step>
</build>
```

Listing 4.1 – Exemple de configuration du greffon Bitten pour Trac.

L'intégration de mes jeux de tests C++/Boost dans Bitten est réalisable via quelques subtilités. En effet le code de test développé avec Boost peut générer un fichier de résultat en XML. Bitten quand à lui gère en entrée du XML au niveau des processus esclaves. Ce XML est transmi au maître via une requête HTTP. Mais le XML n'est qu'un contenant ; pour passer du XML Boost au XML Bitten nous devons donc définir une transformation XSLT. L'exécution des tests de la bibliothèque Boost est déclenchée par la commande suivante :

```
run_gdal_tests --report_format=XML --report_level=detailed
```

Pour transformer le XML de sortie en un XML que sache interpréter Bitten, j'utilise le fichier XSL donné en annexe B.4. Notre « recette » Bitten qui correspond à la configuration de la partie maître située sur le serveur web va intégrer alors l'instruction suivante :

```
<x:transform src="test_results.xml" dest="test_report.xml" stylesheet="boost2bitten.xsl" />
```

Celle-ci va générer une transformation XSLT depuis le fichier généré par le code de test Boost.

Au final, nous pouvons avoir les résultats des étapes de test mis à jour automatiquement via l'interface de Trac comme le montrent les figures 4.5 et 4.6. La seconde copie d'écran représente l'exécution avec succès des différentes suites de tests sur mes modifications apportées à la bibliothèque GDAL.

Nous venons de voir une première approche d'une architecture d'intégration continue. Je note une première approche, car la mise en place et le test de toute la chaîne est complexe et prend beaucoup de temps. Une mutualisation est nécessaire et cela commence dans le cadre de notre nouvelle entité l'IPAG [2]. Ainsi, au sein de l'observatoire, nous avons déployé une interface Trac et un serveur de gestion de configurations commun Subversion. Le but ultime serait d'avoir des serveurs de tests pour chaque système d'exploitation pour lequel doivent être disponibles nos développements.

2. L'IPAG est la fusion du LPG et du LAOG (Laboratoire d'AstrOphysique de Grenoble).

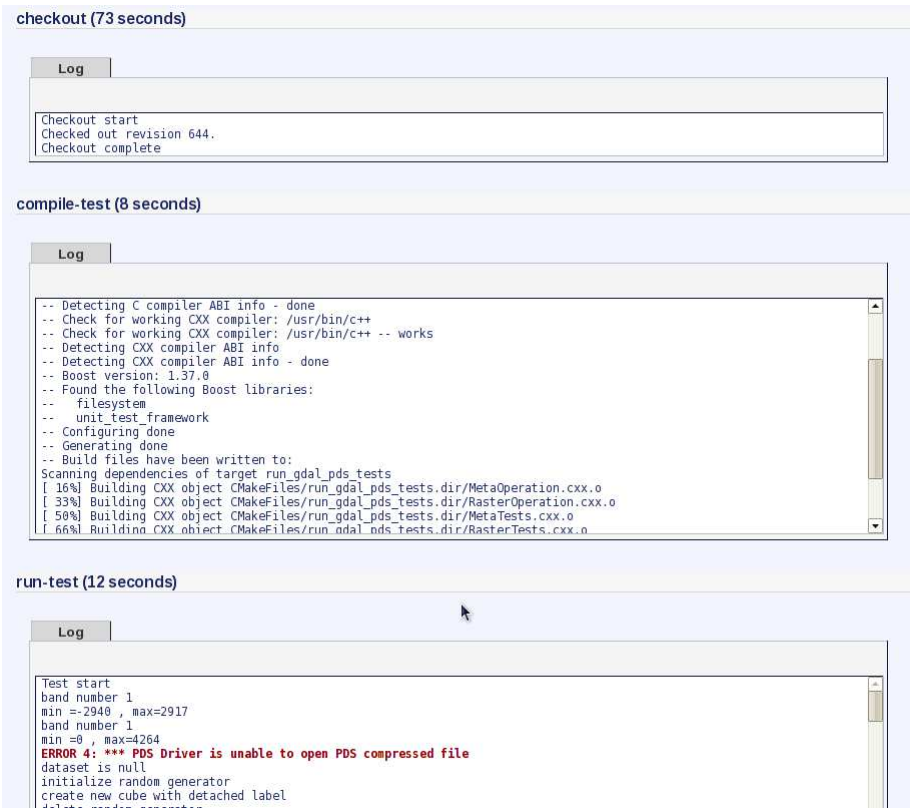

FIGURE 4.5 – Résultat d'une construction (« build ») et d'une exécution de tests avorté. Extrait de l'interface Trac/Bitten.

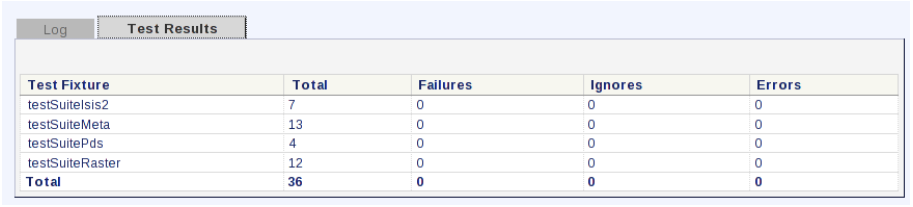

FIGURE 4.6 – Résultat d'une exécution de tests sans erreur. Extrait de l'interface Trac/Bitten.

4.3.4 Serveur de gestion de versions

Dans notre infrastructure, un élément central est le gestionnaire de versions qui permet, comme son nom l'indique, de gérer les différentes versions de tous les documents (fichier source ou documentation) d'un projet. Il est principalement utilisé pour maintenir le code source ou la documentation d'un logiciel.

J'utilise Apache Subversion (souvent abrégé en SVN d'après le nom de la commande). C'est un logiciel de gestion de versions et un système de contrôle de révisions crée en 2000 par CollabNet (http://subversion.tigris.org/). Ce logiciel est sous licence libre Apache. Il est le successeur du système CVS (Concurrent Versions System).

La communauté « open source » a largement utilisé Subversion. Il a été ou est encore utilisé par : la fondation Apache, FreeBSD, GCC, Django, Mono, SourceForge, etc. Le monde de l'entreprise a également adopté Subversion. Cependant, il est maintenant de moins en moins utilisé au profit de logiciels tel que Mercurial ou Git.

Dans notre cas nous l'avons déployé sur notre serveur web en parallèle à la plateforme Trac. Ces deux outils me permettent d'organiser notre projet en différents modules :

doc : la documentation de l'ensemble du projet, incluant les présentations et les rapports aux partenaires ;

external : regroupe les librairies que j'ai dû modifier comme GDAL, OTB ainsi que les configurations pour l'intégration sur MACOS et la construction des paquets RedHat (voir la section 4.4.2) ;

framework : le code des différents modules de calcul ;

prgm les codes initiaux en IDL ou Matlab des différents algorithmes ou simulation numérique (maquettes) ;

view : l'ensemble de nos développements concernant l'interface graphique ;

webservice : les développements ayant trait au système de recherche et de mise à disposition des méta-données et de données hyperspectrales. Ces développements ne sont pas abordés dans ce mémoire.

4.4 Environnement de développement

Quittons le registre méthodologique et les outils de gestion pour effectuer un tour d'horizon des principaux logiciels de développement qui interviennent dans le projet Vahiné.

J'utilise essentiellement l'environnement tout intégré Eclipse. Celui-ci se compose d'une IHM de base (éditeur, gestion de projet, ...) pouvant accueillir de nombreux greffons. Le projet Vahiné étant développé en C++ et Python, j'utilise les greffons CDT (C/C++ Development Tooling, http://www.eclipse.org/cdt/) pour le C++ et PyDev pour Python. Un greffon spécifique à Subversion (Subclipse) est disponible pour interagir avec notre serveur de gestion de versions depuis la vue projet d'éclipse.

Le développement de Vahiné s'effectue principalement avec le langage C++ qui est un langage compilé. Je fais donc un usage intensif du débuggeur pour traquer les disfonctionnement ou pour superviser certaines étapes du traitement numérique. Eclipse intègre gdb via son greffon dédié au langage C++, CDT. GDB (« GNU Project Debugger », http://www.gnu.org/software/gdb/) permet d'inspecter en détail le fonctionnement d'un morceau de code.

```
PROJECT(Vahine-Framework)
cmake_minimum_required(VERSION 2.4)

if(COMMAND cmake_policy)
  cmake_policy(SET CMP0003 NEW)
endif(COMMAND cmake_policy)

SET(CMAKE_VERBOSE_MAKEFILE ON)
SET(CMAKE_BUILD_TYPE DEBUG)

FIND_PACKAGE(Boost 1.34 REQUIRED)                                    ①
FIND_PACKAGE(OTB REQUIRED)
 ...

### elm vca filter ###
SET(VAHINE_ELMVCA_SRC
  ElmVcaFilter.cxx
  Exec/ExecElmVca.cxx
)

ADD_LIBRARY( VahineElmVca ${VAHINE_ELMVCA_SRC} )
TARGET_LINK_LIBRARIES( VahineElmVca OTBCommon OTBIO VahineVCA )

ADD_EXECUTABLE( elmvca ${VAHINE_ELMVCA_SRC} )
TARGET_LINK_LIBRARIES( elmvca OTBCommon OTBIO VahineElmVca VahineVCA )
```

Listing 4.2 – Extrait du fichier principal `CMakeLists.txt` du projet Vahiné

4.4.1 CMake

Pour la compilation, j'utilise une couche de plus haut niveau que les autotools[3] classiques sous Unix. CMake est un outil multiplateforme de construction de logiciels. Il est une surcouche alternative au triplé autoconf/automake/libtool pour générer les fichiers `Makefile` sous Unix. Mais il peut aussi générer des fichiers de projet Visual Studio sous Windows ou des fichiers de configuration pour l'environnement Eclipse.

Le processus de production CMake est contrôlé par des instructions situées dans les fichiers `CMakeLists.txt`. Chaque répertoire d'un projet CMake doit avoir un fichier `CMakeLists.txt`. Un point important est que les fichiers `CMakeLists.txt` dans un sous-répertoire héritent des propriétés définies dans le répertoire parent. Ce système permet d'éviter la duplication du code de construction.

CMake permet de générer rapidement un projet important car il dispose d'un grand nombre de commandes ou macros permettant par exemple d'effectuer la détection automatique des librairies nécessaires à la compilation. On peux ainsi voir au point ① du listing 4.2 l'utilisation de la macro `FIND` pour la recherche automatique de la bibliothèque Boost.

4.4.2 Mise en paquet

Pour faciliter la maintenance et l'installation de mes réalisations, j'ai décidé d'effectuer un important travail de mise en paquet. Nous discutons par la suite de cette technique.

Chaque système d'exploitation a ses spécificités et fournit donc son propre système de gestion de paquets. Les systèmes Debian utilisent APT (Advanced Package Tool) tandis que les systèmes à base RedHat (CentOs et Fedora) utilisent le gestionnaire RPM. Un système de gestion de paquets permet :

- de gérer la bonne cohérence des fichiers systèmes ;
- il s'oppose à une installation partielle ou erronée et permet de désinstaller l'ensemble des fichiers liés à une bibliothèque ou logiciel ;

3. Ensemble des outils de construction logiciel du projet GNU.

– il maintient aussi une liste de tous les paquets installés sur le système et facilite ainsi les mise à jour ;
– il gère les dépendances et prévient des conflits entre bibliothèques.

J'ai élaboré une dizaine de fichiers de configuration (fichier « specs ») pour les bibliothèques de notre projet. Chaque paquet est créé grâce à l'utilitaire `rpmbuild` fournit en standard sous toute distribution linux RedHat. Celui-ci va utiliser un fichier « spec », qui définit comment créer le logiciel à empaqueter et la structure du paquet. Un exemple de fichier « spec » est présenté dans l'annexe B.5. Un fichier « spec » se compose essentiellement de mots clefs, qui dans la terminologie RPM, s'appelle un « tag ». Il y a deux formats possibles. Soit la notation en ligne : `tag:valeur` ou la notation en bloc : on commence par `%tag` suivi des lignes du champ et on termine par une ligne vide. Nos fichiers de configuration « spec » sont composés alors de tags obligatoires et optionnels suivant les actions à effectuer. Les tags obligatoires sont les suivants :

– Name : nom du paquet
– Version : version du logiciel
– Release : release
– Summary : description sommaire du logiciel packagé
– Group : pour classer le paquet
– Copyright : le type de licence
– %description : description détaillée
– %file : la liste des fichiers contenus dans le paquet

Le lecteur intéressé par cette technique trouvera un excellent guide dans le document [Streicher, 2010].

La mise en paquets des bibliothèques utilisées dans le projet Vahiné me permet de redéployer rapidement de façon « propre » les exécutables et leurs dépendances sur notre serveur de calcul qui est distinct de la machine de développement. Cela permettra aussi dans le futur de distribuer de façon aisée notre logiciel.

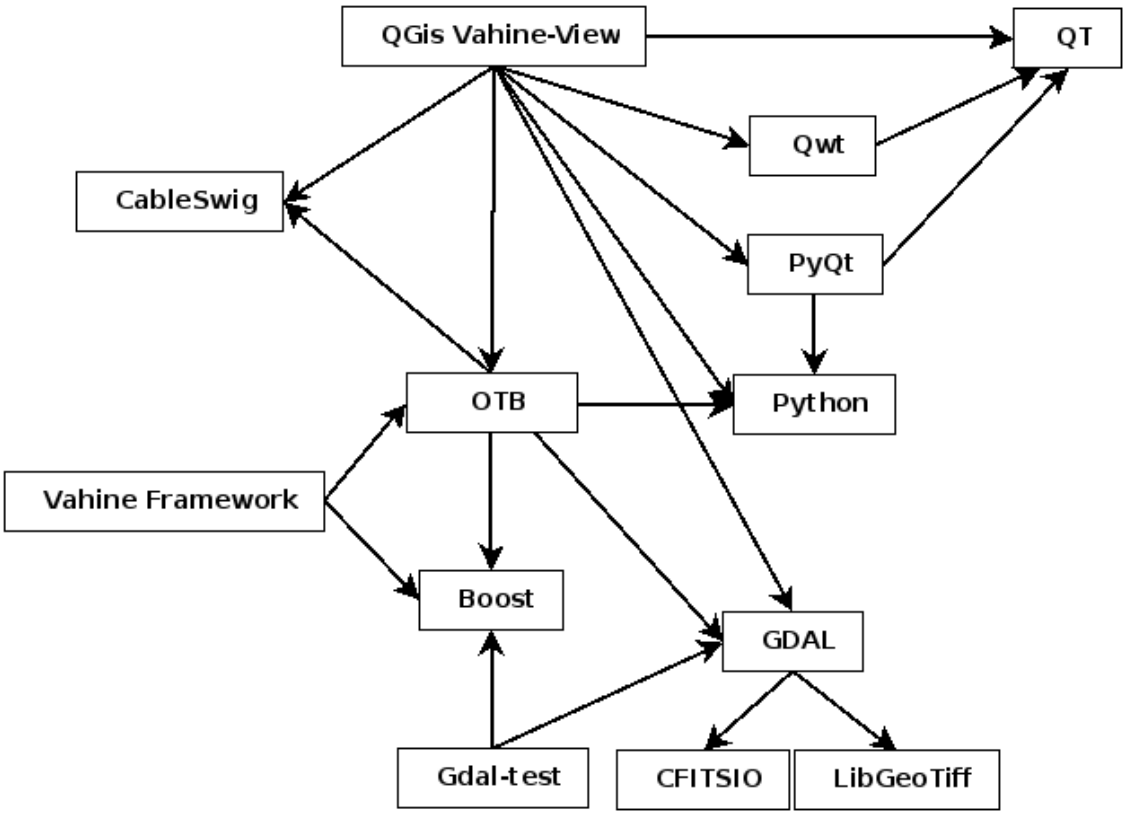

FIGURE 4.7 – Représentation des principaux paquets du projet Vahiné et de leurs dépendances.

Conclusion

Travaux effectués et synthèse

D'un point de vue général, j'ai déployé les bases d'un système d'intégration continue que nous essayons de mutualiser dans le cadre de l'OSUG. Ainsi actuellement, un serveur de gestion de versions Subversion et une plateforme de suivi de projets Trac ont été déployés au sein de l'Observatoire. Nous avons donc apporté un premier élément de rationalisation des moyens liés au développement logiciel.

Par rapport au projet Vahiné, j'ai réalisé un nouveau pilote GDAL pour permettre la lecture et l'écriture correctes d'images hyperspectrales au format PDS. Ceci constitue la couche basse de notre architecture. Après avoir pris en main la bibliothèque OTB, j'ai implémenté l'algorithme de séparation de sources ELM-VCA. Celui-ci est pour le moment piloté par un script en ligne de commande. J'ai ensuite réalisé sur la base du logiciel QGis le début de la plateforme de visualisation Vahiné. Celle-ci inclut un aspect multivue et bénéficie d'un ensemble d'outils d'analyses spectrales. L'IHM va demander encore quelques mois de travail à plein temps pour finaliser l'interaction avec les traitements numériques. Cela permettra dans un premier temps de passer ce logiciel en production au sein de l'équipe des planétologues de l'IPAG et ensuite d'effectuer sa diffusion dans la communauté scientifique.

Cette étude fut très intéressante pour ma part, mais assez difficile du fait de l'aspect mathématique et des nombreux points techniques à maîtriser. Pour la complexité mathématique, la solution est de « se plonger » dans les articles et interagir directement avec les chercheurs, auteurs des algorithmes. C'est un aspect qui sort des développements classiques (gestion, web) et qui se rapproche des développements numériques pointus. Cet aspect m'apporte une grande satisfaction intellectuelle et me permet de faire le lien avec ma formation initiale (DEA de physique).

Côté méthodologie, j'essaye d'inciter au sein de l'IPAG via les discussions et réunions entre membres de l'équipe technique à l'utilisation de méthodes agiles. Celles-ci bien plus souples que les cycles de développement classiques en V sont parfaitement adaptées au milieu de la Recherche. Cela, j'espère, contribuera à augmenter la qualité logiciel des futurs développements.

Au cours de ce mémoire, j'ai aussi eu l'occasion de recruter et d'encadrer un stagiaire de l'IUT2 de Grenoble. Le plus délicat fut de préparer le stage pour que le stagiaire ait toutes les informations pour travailler efficacement. La gestion du planning face aux aléas et aux difficultés de compréhension était aussi difficile. Ce fut une expérience très enrichissante que je n'hésiterai pas à reproduire.

Perspective pour le projet Vahiné

A la fin de ma période de mémoire, nous avons effectué un bilan des développements avec notre financeur principal, le CNES. La personne en charge du projet était satisfaite des développements et a donné son accord pour l'intégration de nos algorithmes dans la bibliothèque OTB. Suite à cela, le Dr S. Douté, mon responsable au sein du LPG/IPAG, a obtenu une prolongation du projet Vahiné au delà de juin 2011. Ce projet et plus particulièrement les développements associés sont prolongés jusqu'en décembre 2011. Cela nous permettra de livrer deux algorithmes de traitement supplémentaires : BPSS ainsi que Wavanglet-MSF. Les discussions en cours dans l'équipe QGis devraient en outre aboutir à une prochaine version de QGis, qui intégrera l'aspect multivue. Cela nous permettra de recentrer complètement nos développements sous la forme de plusieurs greffons Vahiné contenant les outils de manipulation d'images hyperspectrales.

Bilan et perspective personnelle

J'ai pu de par ma situation professionnelle effectuer mon stage du CNAM au sein du LPG. Humainement cela a été très enrichissant et assez particulier du fait de la fusion des laboratoires LPG et LAOG au cours de l'année 2010 pour donner naissance à l'IPAG. Cela a conduit à de nombreux changements organisationnels. Ce travail de mémoire a été aussi l'occasion d'enrichir mes connaissances dans le domaine de l'imagerie satellitaire et de nouer des contacts professionnels dans les différentes équipes de développement de la bibliothèque OTB et du logiciel QGis.

Dans l'optique de ma fin de contrat au LPG en juin 2011, l'obtention de mon diplôme d'ingénieur du CNAM est très important car cela me permettra un meilleur positionnement sur le marché de l'emploi. C'est aussi pour moi une étape personnelle emblématique par l'obtention d'un second diplôme de niveau bac+5. Cela validera pleinement ma reconversion dans le domaine du génie logiciel.

J'espère maintenant pouvoir valoriser au mieux mes compétences et mes acquis en proposant mes services en tant qu'indépendant au sein du tissu économique Grenoblois.

Appendices

Annexe A

Images hyperspectrales

A.1 Norme Planetary Data System

```
PDS_VERSION_ID                  = PDS3
LABEL_REVISION_NOTE             = "2004-11-02, FM"
RECORD_TYPE                     = FIXED_LENGTH
RECORD_BYTES                    =          512
FILE_RECORDS                    =        57124
LABEL_RECORDS                   =           36
FILE_NAME                       = "ORB0061_5.QUB"

^QUBE                           = ("ORB0061_5_V03.REF",1)

DATA_SET_ID                     = "MEX-M-OMEGA-2-EDR-FLIGHT-V1.0"
RELEASE_ID                      = 0001
REVISION_ID                     = 0000
PRODUCT_ID                      = ORB0061_5_DATA
PRODUCT_TYPE                    = EDR
STANDARD_DATA_PRODUCT_ID        = "OMEGA DATA"
PI_PDS_USER_ID                  = BIBRING
MISSION_NAME                    = "MARS EXPRESS"
MISSION_PHASE_NAME              = "MC PHASE 2"
INSTRUMENT_NAME                 = "Observatoire Mineralogie, Eau,
    Glaces, Activite"
PRODUCER_INSTITUTION_NAME       = "Laboratoire de Planetologie de
  Grenoble"
INSTRUMENT_ID                   = OMEGA
INSTRUMENT_TYPE                 = "IMAGING SPECTROMETER"
^INSTRUMENT_DESC                = "OMEGA_DESC.TXT"
^INSTRUMENT_CALIBRATION_DESC    = "OMEGA_CALIBRATION_DESC.TXT"
DATA_QUALITY_ID                 = 3
DATA_QUALITY_DESC               = " from 0 to 3 depending on
                        missing lines and compression errors"
MISSING_SCAN_LINES              =           0
```

Listing A.1 – Exemple de label PDS.

A.2 Spectro-imageur CRISM

Le spectro-imageur CRISM est un instrument permettant de faire des observations à haute résolution, CRISM fonctionne avec un mode dit « cible ». parmi ses multiples modes d'acquisition, CRISM offre la possibilité d'acquérir une image haute résolution sur une région d'environ 10 km x 10 km avec une résolution de 18 mètres par pixel, en 544 canaux couvrant la plage [0.36, 3.92] microns (en mode cible). Dix autres images sont prises avant et après l'image principale avec des géométries différentes en mode nadir.

La figure A.2 présente le principe d'acquisition d'une série d'images hyperspectrales multiangulaires (HMA). Dans la séquence précédant le survole de la cible, cinq balayages sont effectués au cours de laquelle les données hyperspectrales sont acquises (en violet). Puis, centrée sur la cible, un balayage est réalisé pour obtenir une image de haute résolution spatiale (vert). Enfin, cinq autres analyses sont effectuées après le survole de la cible (rouge). Le résultat est composé de 11 images, avec la 6^{e}qui est en haute résolution. Une fois projeté sur une carte, les empreintes des 11 images se chevauchent. C'est ce qui est présenté en bas de la figure A.2.

A.3 Cube synthétique et masquage spatiale

Les masques binaires (A.3) permettent la localisation dans l'image des diverses unités de terrains physiques (un masque par unité). Ils sont habituellement stockés en pile dans un cube PDS au format BSQ.

Les spectres synthétiques (A.4) $\mathbf{X}$ sont organisés dans un cube au format PDS de dimension $(n \times m \times d)$, n nombre de combinaisons, m nombre total de géométries et d nombre de spectels. A chaque ligne du cube est associée une occurence du vecteur des paramètres physiques $\mathbf{Y}$ et à chaque colonne une géométrie. La troisième dimension du cube reçoit les spectres $\mathbf{X}$ compatibles avec les caractéristiques instrumentales.

Le cube de spectres synthétiques est accompagné d'un cube au format PDS de données auxiliaires (A.5) de dimension $(n, m, p + q)$ qui donne pour chaque spectre de coordonnées (i, j) la valeur des q paramètres géométriques et la valeur des p paramètres physiques qui ont été utilisées pour son calcul.

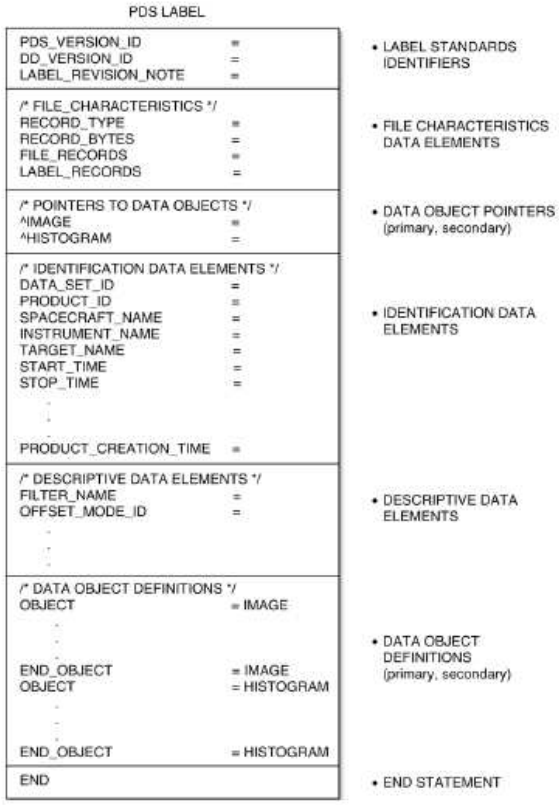

FIGURE A.1 – Structure d'un fichier PDS.

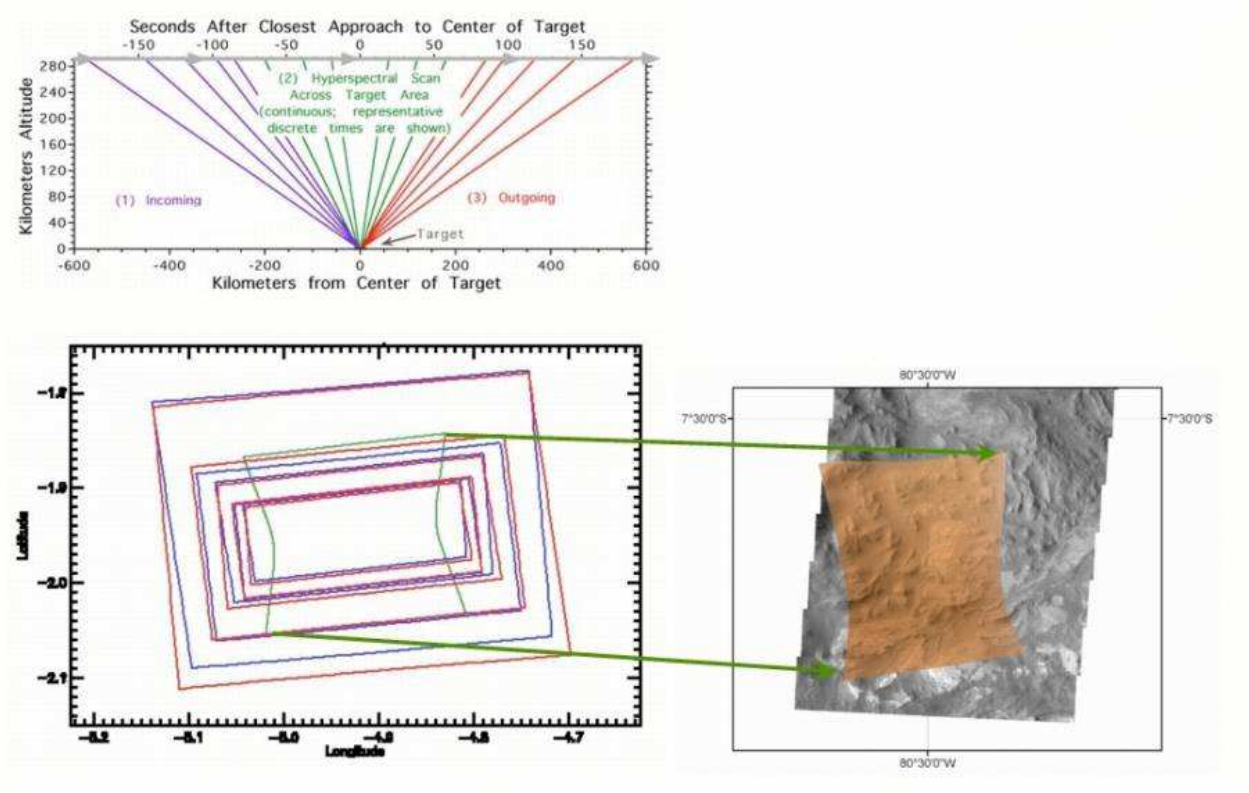

FIGURE A.2 – Principe de l'acquisition CRISM. Un spectro-imageur HMA.

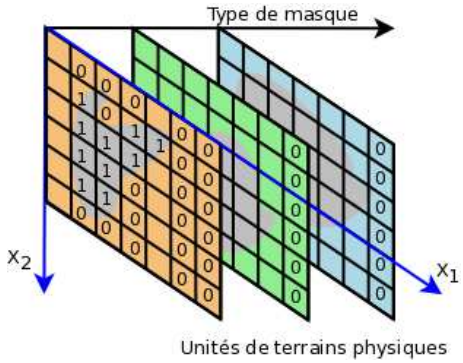

FIGURE A.3 – Structure des masques permettant de cibler la zone de validité d'un modèle physique.

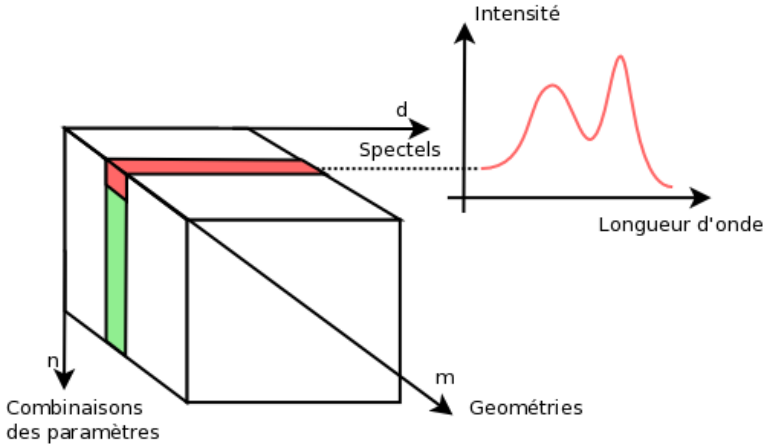

FIGURE A.4 – Structure d'un cube de spectres synthétiques.

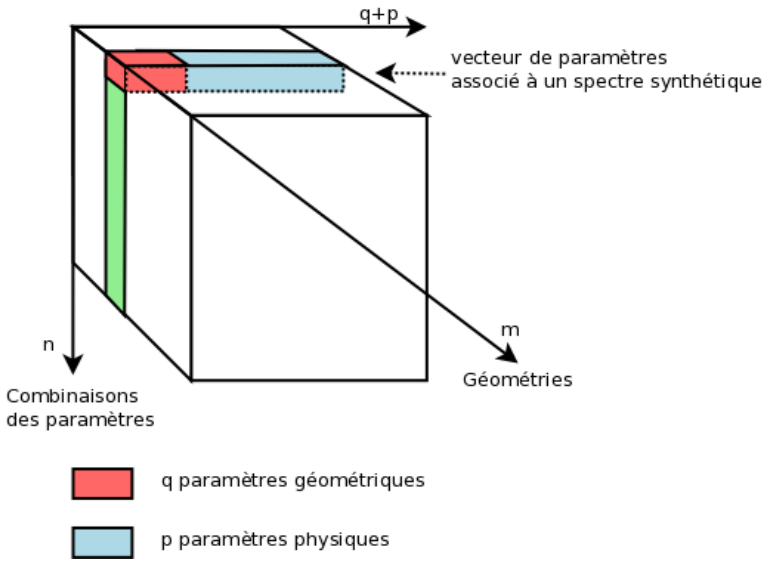

FIGURE A.5 – Structure d'un cube de données auxiliaires associées aux spectres synthétiques.

Annexe B

Code source

B.1 Pilote ISIS2

```
void ISIS2Dataset::WriteRaster(CPLString osFilename, bool includeLabel, GUIntBig iRecords, GUIntBig iLabelRecords,
      GDALDataType eType, const char * pszInterleaving) {
  GUIntBig nSize;
  GByte byZero(0);
  CPLString pszAccess("wb");
  if(includeLabel)
    pszAccess = "ab";

  FILE *fpBin = VSIFOpenL( osFilename, pszAccess.c_str() );
  if( fpBin == NULL ) {
    CPLError( CE_Failure, CPLE_FileIO,
        "Failed to create %s:\n%s",
        osFilename.c_str(), VSIStrerror( errno ) );
    throw ISIS2DatasetError();
  }

  nSize = iRecords * RECORD_SIZE;
  CPLDebug("ISIS2","nSize = %i", static_cast<int>(nSize));

  if(includeLabel)
   nSize = iLabelRecords * RECORD_SIZE + nSize;

  // write last byte
  if(VSIFSeekL( fpBin, nSize-1, SEEK_SET ) != 0 ||
      VSIFWriteL( &byZero, 1, 1, fpBin ) != 1){
    CPLError( CE_Failure, CPLE_FileIO,
        "Failed to write %s:\n%s",
        osFilename.c_str(), VSIStrerror( errno ) );
    throw ISIS2DatasetError();
  }
  VSIFCloseL( fpBin );
}
```

Listing B.1 – Méthode gérant l'écriture de la partie raster. Extrait du pilote ISIS2.

```
void ISIS2Dataset::WriteLabel(CPLString osFilename, CPLString osRasterFile,
    CPLString sObjectTag,
    unsigned int nXSize, unsigned int nYSize, unsigned int nBands, GDALDataType eType,
    GUIntBig iRecords, const char * pszInterleaving, GUIntBig & iLabelRecords, bool bRelaunch) {
  CPLDebug("ISIS2", "Write Label filename = %s, rasterfile = %s",osFilename.c_str(),osRasterFile.c_str());
  bool bAttachedLabel = EQUAL(osRasterFile, "");

  FILE *fpLabel = VSIFOpenL( osFilename, "w" );

  if( fpLabel == NULL ){
    CPLError( CE_Failure, CPLE_FileIO,
              "Failed to create %s:\n%s",
              osFilename.c_str(), VSIStrerror( errno ) );
    throw ISIS2DatasetError();
  }
```

```
    unsigned int iLevel(0);
    unsigned int nWritingBytes(0);

    /* write common header */
    nWritingBytes += ISIS2Dataset::WriteKeyword( fpLabel, iLevel, "PDS_VERSION_ID", "PDS3" );
    nWritingBytes += ISIS2Dataset::WriteFormatting( fpLabel, "");
    nWritingBytes += ISIS2Dataset::WriteFormatting( fpLabel, "/* File identification and structure */");
    nWritingBytes += ISIS2Dataset::WriteKeyword( fpLabel, iLevel, "RECORD_TYPE", "FIXED_LENGTH" );
    nWritingBytes += ISIS2Dataset::WriteKeyword( fpLabel, iLevel, "RECORD_BYTES", CPLString().Printf("%d",
        RECORD_SIZE));
    nWritingBytes += ISIS2Dataset::WriteKeyword( fpLabel, iLevel, "FILE_RECORDS", CPLString().Printf("%lu",(long
        unsigned int)(iRecords)));
    nWritingBytes += ISIS2Dataset::WriteKeyword( fpLabel, iLevel, "LABEL_RECORDS", CPLString().Printf("%llu",
        iLabelRecords));
    if(!bAttachedLabel){
      nWritingBytes += ISIS2Dataset::WriteKeyword( fpLabel, iLevel, "FILE_NAME", CPLGetFilename(osRasterFile));
    }
    nWritingBytes += ISIS2Dataset::WriteFormatting( fpLabel, "");

    nWritingBytes += ISIS2Dataset::WriteFormatting( fpLabel, "/* Pointers to Data Objects */");

    if(bAttachedLabel){
        nWritingBytes += ISIS2Dataset::WriteKeyword( fpLabel, iLevel, CPLString().Printf("^%s",sObjectTag.c_str()),
            CPLString().Printf("%llu",iLabelRecords+1));
    }else{
      nWritingBytes += ISIS2Dataset::WriteKeyword( fpLabel, iLevel, CPLString().Printf("^%s",sObjectTag.c_str()),
          CPLString().Printf("(\"%s\",1)",CPLGetFilename(osRasterFile)));
    }

    if(EQUAL(sObjectTag, "QUBE")){
      ISIS2Dataset::WriteQUBE_Information(fpLabel, iLevel, nWritingBytes, nXSize, nYSize, nBands, pszInterleaving);
    }

    nWritingBytes += ISIS2Dataset::WriteFormatting( fpLabel, "END");

    // check if file record is correct
    unsigned int q = nWritingBytes/RECORD_SIZE;
    if( q <= iLabelRecords){
      // correct we add space after the label end for complete from iLabelRecords
      unsigned int nSpaceBytesToWrite = iLabelRecords * RECORD_SIZE - nWritingBytes;
      VSIFPrintfL(fpLabel,"%*c", nSpaceBytesToWrite, ' ');
    }else{
      iLabelRecords = q+1;
      ISIS2Dataset::WriteLabel(osFilename, osRasterFile, sObjectTag, nXSize, nYSize, nBands, eType, iRecords,
          pszInterleaving, iLabelRecords);
    }
    VSIFCloseL( fpLabel );
}
```

Listing B.2 – Méthode gérant l'écriture de la partie label. Extrait du pilote ISIS2.

B.2 Traitements numériques

Cette section présente le code source complet du principal filtre que nous avons développé pendant ce travail de mémoire. Pour les détails de l'implémentation le lecteur se reportera à la section 2.4. Le code de ce filtre est découpé en deux fichiers : un fichier d'entête visible sur le listing B.3 et un fichier contenant l'implémentation présentée sur le listing B.4.

```
/**
 * This file is a part of Vahine Framework
 * Copyright (C) 2008 to 2010 Ludovic Leau-Mercier and Laboratoire de Planetologie de Grenoble
 * See LICENCE and COPYING for details.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * CeCILL License for more details and http://www.cecill.info
 *
 * \author ludovic leau-mercier
```

```
 */

#ifndef ELMVCAFILTER_H_
#define ELMVCAFILTER_H_

#include <limits>
#include <cmath>
#include <ctime>
#include <vector>
#include <list>
#include <otb/Utilities/ITK/Utilities/vxl/core/vnl/algo/vnl_symmetric_eigensystem.h>
#include <vcl_algorithm.h>
#include <vcl_vector.h>

#include <otb/BasicFilters/otbStreamingStatisticsVectorImageFilter.h>
#include <otb/BasicFilters/otbStreamingStatisticsImageFilter.h>
#include <otb/BasicFilters/otbVectorImageToImageListFilter.h>
#include <otb/Common/otbImageList.h>
#include <otb/IO/otbImage.h>
#include <otb/IO/otbVectorImage.h>

#include <otb/Utilities/ITK/Common/itkImageToImageFilter.h>
#include <itkImageRegionConstIterator.h>
#include <itkVariableLengthVector.h>
#include <itkCovarianceCalculator.h>
#include <itkImageToListAdaptor.h>
#include <itkListSample.h>
#include <itkArray.h>
#include "itkVariableSizeMatrix.h"

#include "VahineMacro.h"
#include "VcaFilter.h"
#include "GrsirFilter.h"

namespace otb {
  /**
   * Fill the array data with the pixel values from index offset.
   * Merge this with ManagedTypes::ToManagedVariableLengthVector of GrsirFilter class.
   * Why i can't use TPixel::RealValue instead of TInternalPixel
   */
    template <class TPixel, class TInternalPixel, class TVector> static inline void PixelToVnlVector(const TPixel &
        pixel, TVector & vector){
      unsigned int size = pixel.GetNumberOfElements();
      vnl_vector<TInternalPixel> pix(pixel.GetDataPointer(), size);
      vector.set_size(size);
      for (unsigned int i = 0; i<size; ++i){
        vector[i]=static_cast<typename TVector::element_type>(pix[i]);
      }
    }

  template <class TImage>
  class ITK_EXPORT VahineElmVcaFilter
    : public itk::ImageToImageFilter<TImage, TImage> {
    public:
      typedef TImage VectorImageType;

      typedef VahineElmVcaFilter Self;
      typedef itk::ImageToImageFilter<VectorImageType, VectorImageType> Superclass;
      typedef itk::SmartPointer<Self> Pointer;
      typedef itk::SmartPointer<const Self> ConstPointer;

      // creation of SmartPointer
      itkNewMacro(Self);
      // runtime type information
      itkTypeMacro(VahineElmVcaFilter, itk::ImageToImageFilter);

      // Input/Output Image
      typedef typename VectorImageType::Pointer VectorImagePointer;
      typedef typename VectorImageType::ConstPointer VectorImageConstPointer;
      typedef typename VectorImageType::PixelType PixelType;
      typedef typename VectorImageType::InternalPixelType InternalPixelType;
      typedef typename VectorImageType::RegionType RegionType;
      typedef typename VectorImageType::SizeType SizeType;
      typedef typename itk::NumericTraits<InternalPixelType>::RealType RealType;

```

```
    // type for computation
    const static unsigned int Dimension = 2;
    typedef Image<InternalPixelType, Dimension> BandType;

    typedef VahineVCAFilter<VectorImageType> VCAFilterType;
    typedef typename VCAFilterType::Pointer VCAFilterPointer;
    typedef typename std::vector<unsigned int> IntegerList;

    typedef vnl_vector<double> VnlVector;

    typedef typename itk::VariableSizeMatrix<RealType> EndMembersMatrixType;

    bool GetDebug(){ return true; };
    itkGetMacro( NbComponents, unsigned int );
    itkSetMacro( NbComponents, unsigned int );
    IntegerList  GetMaxLikeHood();
    /**
     * MVSAFilter produces an image with same resolution that input image but
     * with a different number of bands. As such, this filter needs to provide an
     * implementation for GenerateOutputInformation() in order
     * to inform the pipeline execution model.
     */
    virtual void GenerateOutputInformation();

    void SetEndmembers();
    itkGetMacro( EndMembers, EndMembersMatrixType* );
    itkGetMacro( Likelihood, VnlVector);
    // for testing
    friend class ElmVcaFilterFixture;

protected:
    VahineElmVcaFilter();
    virtual ~VahineElmVcaFilter();

    void PrintSelf(std::ostream& os, itk::Indent indent) const;
    void GenerateData();
    unsigned int m_NumberOfBand; // protected for testing

private:
    // internal type for computation
    typedef vnl_matrix<double> VnlMatrix;
    typedef vnl_vector<InternalPixelType> InternalVnlVector;
    typedef vnl_matrix<InternalPixelType> InternalVnlMatrix;

    typedef itk::ImageRegionConstIterator<VectorImageType> ConstIteratorType;
    typedef itk::ImageRegionIterator<VectorImageType> IteratorType;
    typedef otb::StreamingStatisticsVectorImageFilter<VectorImageType> StatsFilterType;
    typedef typename StatsFilterType::Pointer StatsFilterPointer;

    typedef ImageList<BandType> ImageListType;
    typedef typename ImageListType::Pointer ImageListPointer;
    typedef typename ImageListType::ConstIterator ConstListIteratorType;
    typedef VectorImageToImageListFilter<VectorImageType, ImageListType> DecomposeFilterType;
    typedef typename DecomposeFilterType::Pointer DecomposeFilterPointer;
    typedef StreamingStatisticsImageFilter<BandType> BandStatsFilterType;
    typedef typename BandStatsFilterType::Pointer BandStatsFilterPointer;

    /** Typedef for statistic computing. */
    typedef StreamingStatisticsVectorImageFilter<VectorImageType> StreamingStatisticsVectorImageFilterType;
    typedef typename StreamingStatisticsVectorImageFilterType::MatrixType MatrixType;

    typedef typename itk::Statistics::ListSample< itk::VariableLengthVector< InternalPixelType > > SampleType;
    typedef typename itk::Statistics::CovarianceCalculator< SampleType > CovarianceAlgorithmType;
    typedef typename CovarianceAlgorithmType::Pointer CovarianceAlgorithmPointer;

    std::numeric_limits<InternalPixelType> m_RealLimits;
    unsigned int m_NbComponents;
    IntegerList  m_MaxLikeHood;
    VnlVector m_Mean;
    VnlVector m_Likelihood;
    MatrixType *m_EndMembers;

    typename VCAFilterType::IndexArray m_PurePixelsIndexArray;

    VnlMatrix myCovariance(VectorImagePointer image, unsigned int nbObserv);
```

```
        VnlMatrix streamingCovariance(VectorImagePointer image, unsigned int nbObserv);

        unsigned int Demelange(unsigned int);
        void LikelihoodLocalMaximum(VnlVector & likeHood);
        void SpectralMean(VnlVector & mean);
        void LikelihoodLog(VnlVector & eigenR, VnlVector & eigenK, unsigned int N);

        VahineElmVcaFilter(const Self&); //not implemented
        void UpdateNumberOfComponents();
        void operator=(const Self&); //not implemented

        static double mysquare(const double x){return x*x;}
        static double mylog(const double x){return log(x);}

      };

} // end namesapce otb


#include "ElmVcaFilter.txx"

#endif
```

Listing B.3 – Code source du filtre OTB/Vahiné ElmVca. Fichier d'entête.

```
/**
 * \file  ElmVcaFilter.txx
 *
 * This file is a part of Vahine Framework
 * Copyright (C) 2008 to 2010 Ludovic Leau-Mercier and Laboratoire de Planetologie de Grenoble
 * See LICENCE and COPYING for details.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * CeCILL License for more details and http://www.cecill.info
 *
 * \author ludovic leau-mercier
 *
 */

#ifndef _VAHINEELMVCAFILTER_TXX
#define _VAHINEELMVCAFILTER_TXX

#include "ElmVcaFilter.h"


namespace otb {

  template<class TImage>
  VahineElmVcaFilter<TImage>::VahineElmVcaFilter() :
  m_NbComponents(0) {

  }

  template<class TImage>
  VahineElmVcaFilter<TImage>::~VahineElmVcaFilter(){

  }

  template<class TImage>
  void VahineElmVcaFilter<TImage>::UpdateNumberOfComponents(){
    RegionType imageRegion = this->GetInput()->GetLargestPossibleRegion();
    SizeType size = imageRegion.GetSize();
    unsigned int NObserv = size[0]*size[1];
    m_NumberOfBand = this->GetInput()->GetNumberOfComponentsPerPixel();
    if(m_NbComponents == 0){
      m_NbComponents = Demelange(NObserv);
    }
  }

  template<class TImage>
  void VahineElmVcaFilter<TImage>::GenerateData(){
    RegionType imageRegion = this->GetInput()->GetLargestPossibleRegion();
```

```
50
    VCAFilterPointer vcaFilter = VCAFilterType::New();
    vcaFilter->SetInput(this->GetInput());
    vcaFilter->SetNumberEndMember(m_NbComponents);
    vcaFilter->SetDebug(false);
55  vcaFilter->Update();

    m_PurePixelsIndexArray = vcaFilter->GetPurePixelsIndex();

    VnlMatrix mixingMatrix = vcaFilter->GetMixingMatrix();
60  vahineDebug("mixingMatrix=\n");//<<mixingMatrix);
    vnl_matrix_inverse<double> PseudoInv(mixingMatrix);
    VnlMatrix pinv = PseudoInv.pinverse();
    vahineDebug("pinv=\n");//<<pinv);

65  VectorImagePointer output = VectorImageType::New();
    output->SetRegions(imageRegion);
    output->SetNumberOfComponentsPerPixel(m_NbComponents);
    output->Allocate();

70  // fill output
    ConstIteratorType itInput( this->GetInput(), imageRegion );
    IteratorType itOutput( output, imageRegion );

    for(itInput.GoToBegin(), itOutput.GoToBegin(); !itInput.IsAtEnd(); ++itInput, ++itOutput){
75    PixelType pixelIn = itInput.Get();
      PixelType pixelOut = itOutput.Get();
      VnlVector vectorPixel;
      PixelToVnlVector<PixelType, InternalPixelType, VnlVector>(pixelIn, vectorPixel);
      for(unsigned int i=0; i<pixelOut.Size(); ++i){
80      VnlVector currentRow = pinv.get_row(i);
        // we convert in out type in latest moment
        pixelOut[i] = static_cast<InternalPixelType>(dot_product(currentRow, vectorPixel));
      }
      itOutput.Set(pixelOut);
85  }

    SetEndmembers();

    vahineDebug("end_output");
90  this->GraftOutput(output);
  }

  /**
   * Compute the mean value of each spectral band
95 */
  template<class TImage>
  void VahineElmVcaFilter<TImage>::SpectralMean(VnlVector & mean){
    mean.set_size(m_NumberOfBand);
    VectorImagePointer image = const_cast<VectorImageType *>(this->GetInput());
100 DecomposeFilterPointer decomposer = DecomposeFilterType::New();
    decomposer->SetInput(image);
    ImageListPointer imageList = decomposer->GetOutput();
    imageList->Update();
    unsigned int i = 0;
105 for (ConstListIteratorType it = imageList->Begin(); it!=imageList->End();++it){
      // Compute the mean for each band with the statistical filter
      BandStatsFilterPointer meanFilter = BandStatsFilterType::New();
      meanFilter->SetInput(it.Get());
      meanFilter->Update();
110   mean(i) = static_cast<double>(meanFilter->GetMean());
      ++i;
    }
    vahineDebug("mean = "<<mean);
  }
115
  template<class TImage>
  typename VahineElmVcaFilter<TImage>::VnlMatrix VahineElmVcaFilter<TImage>::myCovariance(VectorImagePointer
        image, unsigned int nbObserv){
    image->Update();
    typename SampleType::Pointer sample = SampleType::New();
120 RegionType imageRegion = this->GetInput()->GetLargestPossibleRegion();
    ConstIteratorType it( this->GetInput(), imageRegion );
    for(it.GoToBegin(); !it.IsAtEnd(); ++it){
      sample->PushBack( it.Get() );
```

```
//      vahineDebug("it "<<it.Get());
    }

    CovarianceAlgorithmPointer covarianceAlgorithm = CovarianceAlgorithmType::New();
    covarianceAlgorithm->SetInputSample(sample);
    covarianceAlgorithm->SetMean(0);
    covarianceAlgorithm->Update();
    const itk::VariableSizeMatrix<double> *covarianceVSMat = covarianceAlgorithm->GetOutput();
    VnlMatrix covarianceMatrix = covarianceVSMat->GetVnlMatrix();
    typename CovarianceAlgorithmType::MeanType *mean = covarianceAlgorithm->GetMean();
    m_Mean.set_size(mean->size());
    m_Mean.copy_in(mean->data_block());
    return covarianceMatrix;
  }

  template<class TImage>
  typename VahineElmVcaFilter<TImage>::VnlMatrix VahineElmVcaFilter<TImage>::streamingCovariance(
      VectorImagePointer image, unsigned int nbObserv){
    image->Update();
    typename StreamingStatisticsVectorImageFilterType::Pointer covComputefilter =
        StreamingStatisticsVectorImageFilterType::New();
    covComputefilter->SetInput(image);
    covComputefilter->Update();
    VnlMatrix covarianceMatrix = (covComputefilter->GetCovariance()).GetVnlMatrix();
    PixelType mean = covComputefilter->GetMean();
    m_Mean.set_size(mean.Size());
    for(unsigned int i=0; i< mean.Size(); ++i){
      m_Mean[i] = mean[i];
    }
    double regul = static_cast<double>(nbObserv)/(nbObserv-1);
    covarianceMatrix *= regul;
    return covarianceMatrix;
  }


  /**
   * Find the number of element by calculation of :
   * covariance and correlation matrix
   * take care of correlation calculation, the matrix is normalize
   * with the number of observations.
   */
  template<class TImage>
  unsigned int VahineElmVcaFilter<TImage>::Demelange(unsigned int nbObserv){
    std::time_t start = time(NULL);
    vahineDebug("statistic calculation start");
    VectorImagePointer image = const_cast<VectorImageType *>(this->GetInput());
    VnlMatrix covarianceMatrix;
    covarianceMatrix = myCovariance(image, nbObserv);
    //covarianceMatrix = streamingCovariance(image, nbObserv);
    //covarianceMatrix = threadedCovariance(image, nbObserv);
    std::time_t end = time(NULL);
    vahineDebug("statistic calculation ended");
    vahineDebug("covariance calculation :"<< end - start << " seconds.");

    vahineDebug("covariance matrix\n");//<<covarianceMatrix);

    unsigned int dim = covarianceMatrix.cols();

    VnlMatrix meanProduct(dim, dim);
    for(unsigned int j=0; j<dim; ++j){
      VnlVector currentColumn(dim, m_Mean[j]);
      meanProduct.set_column(j, currentColumn);
    }

    VnlMatrix correlationMatrix(dim, dim);
    for(unsigned int i=0; i<dim; ++i){
      meanProduct.set_row(i,meanProduct.get_row(i)*m_Mean[i]);
    }
    vahineDebug("meanProduct\n");//<<meanProduct);
    correlationMatrix = covarianceMatrix + meanProduct;
    vahineDebug("correlation matrix\n");//<<correlationMatrix);

    vnl_symmetric_eigensystem<double> eigenK(covarianceMatrix);
    VnlVector eigenCovariance = eigenK.D.diagonal();
    vcl_sort (eigenCovariance.begin(),eigenCovariance.end());
```

```
    vnl_symmetric_eigensystem<double> eigenR(correlationMatrix);
    VnlVector eigenCorrelation = eigenR.D.diagonal();
    vcl_sort (eigenCorrelation.begin(),eigenCorrelation.end());
    eigenCorrelation.flip();
    eigenCovariance.flip();

    vahineDebug("eigen covariance =" <<eigenCovariance);
    vahineDebug("eigen correlation =" <<eigenCorrelation);

    LikelihoodLog(eigenCorrelation, eigenCovariance, nbObserv);
    vahineDebug("likeHood=" <<m_Likelihood);
    LikelihoodLocalMaximum(m_Likelihood);
    for(unsigned int i=0; i<m_MaxLikeHood.size(); ++i){
      if(m_MaxLikeHood[i]>=2)
        return m_MaxLikeHood[i];
    }
    // if maxlikehood is empty, normally never in real case
    // we search maximum value
    // this is only for test with a small image
    double max = m_Likelihood.max_value();
    for(unsigned int i=0; i<m_Likelihood.size(); ++i){
      if(abs(m_Likelihood[i] - max) < m_RealLimits.epsilon()){
        return i;
      }
    }
    // if we return 0 it's a bug !
    // TODO throw exception
    return 0;
  }

  /**
   * save all local maximum index
   * min that the index start to 0
   */
  template<class TImage>
  void VahineElmVcaFilter<TImage>::LikelihoodLocalMaximum(VnlVector & likeHood){
    unsigned int n = likeHood.size()-1;
    VnlVector df(n), fi(n), fii(n);
    fi.copy_in(likeHood.data_block());
    fii.copy_in(likeHood.data_block()+1);
    df = fii-fi;
    for(unsigned int i=0; i<df.size()-1; ++i){
      if(df(i)>=0 && df(i+1)<0){
        m_MaxLikeHood.push_back(i+1);
      }
    }
  }

  /**
   * calculate the likelihodd function of eigen vector
   * and the logarithme of this function.
   */
  template<class TImage>
  void VahineElmVcaFilter<TImage>::LikelihoodLog(VnlVector & eigenR, VnlVector & eigenK, unsigned int NObserv){
    unsigned int Ns = eigenR.size();
    m_Likelihood.set_size(Ns);
    double coef = 2.0/NObserv;
    for(unsigned int i=0; i<Ns; ++i){
      unsigned int Nl = Ns - i;
      VnlVector r(Nl), k(Nl), ZSquare(Nl), Sigma(Nl), T(Nl);
      r.copy_in(&eigenR.data_block()[i]);
      k.copy_in(&eigenK.data_block()[i]);
      ZSquare = (r-k).apply(&mysquare);
      Sigma = r.apply(&mysquare) + k.apply(&mysquare);
      Sigma *= coef;
      VnlVector::iterator itSigma = Sigma.begin();
      VnlVector::iterator itZSquare = ZSquare.begin();
      VnlVector::iterator itT = T.begin();
      while(itSigma != Sigma.end()){
        *itT = (*itZSquare)/(*itSigma);
        itSigma++;
        itZSquare++;
        itT++;
      }
      Sigma=Sigma.apply(&log);
```

```
      m_Likelihood(i) = -0.5*T.sum() - 0.5*Sigma.sum();
    }
  }

  template<class TImage>
  typename VahineElmVcaFilter<TImage>::IntegerList VahineElmVcaFilter<TImage>::GetMaxLikeHood(){
    return m_MaxLikeHood;
  }

  /**
   * Standard "PrintSelf" method
   */
  template<class TImage>
  void VahineElmVcaFilter<TImage>::PrintSelf(std::ostream& os, itk::Indent indent) const {
    Superclass::PrintSelf( os, indent );
  }

  template<class TImage>
  void VahineElmVcaFilter<TImage>::GenerateOutputInformation(){
    Superclass::GenerateOutputInformation();
    // get pointers to the input and output
    VectorImageConstPointer inputPtr = this->GetInput();
    VectorImagePointer outputPtr = this->GetOutput();

    if ( !inputPtr || !outputPtr ){
      return;
    }
    // we need to compute the output spacing, the output image size, and the
    // output image start index
    const SizeType& inputSize = inputPtr->GetLargestPossibleRegion().GetSize();
    RegionType outputLargestPossibleRegion;
    outputLargestPossibleRegion.SetSize( inputSize );
    outputPtr->SetLargestPossibleRegion( outputLargestPossibleRegion );
    UpdateNumberOfComponents();
    outputPtr->SetNumberOfComponentsPerPixel(m_NbComponents);
  }

  // TODO refactor this method for improve data access
  template<class TImage>
  void VahineElmVcaFilter<TImage>::SetEndmembers(){
    VectorImageConstPointer inputPtr = this->GetInput();
    unsigned int nbEndMembers = m_PurePixelsIndexArray.size();
    unsigned int nbBands = inputPtr->GetNumberOfComponentsPerPixel();

    m_EndMembers = new EndMembersMatrixType();
    m_EndMembers->SetSize(nbEndMembers, nbBands);

    for(unsigned int i=0; i<nbEndMembers; i++){
      PixelType pixel = inputPtr->GetPixel(m_PurePixelsIndexArray[i]);
      for(unsigned int j=0; j<nbBands; j++){
        (*m_EndMembers)(i,j) = pixel[j];
      }
    }
  }

} // end namespace otb

#endif
```

Listing B.4 – Code source du filtre OTB/Vahiné ElmVca. Fichier d'implémentation.

B.3 Script de pilotage des traitements numériques

```
#!/usr/bin/python
#
# author ludovic mercier, ludovic.mercier@obs.ujf-grenoble.fr
# copyright laboratoire de planetologie 2010
#
import sys, os, getopt
import re
```

```
import xml.dom.minidom

#bindir = "/home/zone12/spectro/Programmes/bin/vahine/"
bindir = "/home/mercierl/workspace/vahineFramework/Debug/"
#bindir = "/home/mercierl/workspace/vahineFramework-trunk/Debug/"
#workdir = "/home/mercierl/workspace/vahineFramework-trunk/Tests/"

class Process:
    __vahineProcess__ = {
                        'VahineSyntheticBaseGeometricFilter': 'synthetic',
                        'VahineFlatReshapeFilter': 'flatreshape',
                        'VahineCovPixelSelectFilter': 'covpixel',
                        'VahineMaskUpdateFilter': 'maskupdate',
                        'VahineGrsirLearning' : 'grsirLearning',
                        'VahineGrsirInverse' : ' grsirInverse ',
                        'VahineBandSelectFilter' : 'bandSelect',
                        'VahineVcaFilter' : 'vca',
                        'VahineElmVcaFilter' : 'elmvca',
                        }


    def __init__(self):
        self.name = None
        self.run = False
        self.input = []
        self.ouput = None
        self.parameter = []
        self.flat = True

    def execute(self):
        if self.run:
            mode = 'Execute'
            if self.simulate:
                mode = 'Simulate'
            print mode, ' ', self.name
            cmd = bindir,self.__vahineProcess__[self.name],' ',' '.join(self.input),' ', self.output
            cmd = ''.join(cmd)
            cmd = cmd + ' ' + ' '.join(['%s' % x for x in self.parameter])
            print 'cmd = ', cmd
            if not self.simulate:
                os.system(cmd)
```

Listing B.5 – La classe de gestion des processus. Extrait du script de pilotage `vahine.py` des traitement numérique vahiné.

```
class Configuration:

    __currentNode__ = None
    __processList__ = None

    def readXml(self, file):
        try:
            self.doc = xml.dom.minidom.parse(file)
        except Exception:
            print "Oops!  That was no valid xml file.  Try again..."

    def getRootElement(self):
        if self.__currentNode__ == None:
            self.__currentNode__ = self.doc.documentElement
        return self.__currentNode__

    def getInOut(self, tagList):
        l = []
        for tag in tagList:
            try:
                l.append(tag.attributes["value"].value)
            except:
                print 'Invalid input/output description'
        return l

    def getParameter(self, pname, tagList):
        if pname == 'VahineFlatReshapeFilter':
            return self.getParameterFlat(tagList)
```

```
        elif pname == 'VahineMaskUpdateFilter':
            return self.getParameterMask(tagList)
        elif pname in ['VahineGrsirLearning', 'VahineGrsirInverse']:
            return self.getParameterGrsir(pname, tagList)
        elif pname == 'VahineBandSelectFilter':
            return self.getParameterBandSelect(tagList)
        elif pname == 'VahineElmVcaFilter':
            return self.getParameterElmVca(tagList)
        else:
            return []

    def getParameterElmVca(self, tagList):
        print 'getparameterElmVca'
        l = []
        try:
            tag = tagList[0]
            l.append(tag.attributes['endmembers'].value)
        except Exception, e:
            print 'Invalid elmvca parameter, ',e
        return l

    def getParameterBandSelect(self, tagList):
        print 'getparameterBandSelect'
        l = []
        index = []
        indexInitial = []
        keep = None
        myreg = re.compile("(?P<startidx>\d+)(,(?P<endidx>\d+)?)?")
        for tag in tagList:
            startidx = None
            endidx = None
            try:
                seq = tag.attributes["band"].value
                currentKeep = tag.attributes["keep"].value == "true"
                obj = myreg.match(seq)
                startidx = obj.group("startidx")
                endidx = obj.group("endidx")
            except Exception, e:
                print 'Invalid band mask parameter, ',e
            if keep is None:
                if obj is not None and startidx is not None and endidx is not None :
                    if len(index) == 0:
                        index = set(range(int(startidx),int(endidx)+1))
                        indexInitial = index
                        keep = currentKeep
            else:
                if endidx is None:
                    current = set([int(startidx)])
                else:
                    current = set(range(int(startidx),int(endidx)+1))
                if keep == currentKeep:
                    index = index | current
                else:
                    index = index - current

        if keep is not None:
            # prepare list of all masking band
            if keep:
                l = list(indexInitial - index)
            else:
                l = list(index)
        return l


    def getParameterMask(self, tagList):
        l = []
        try:
            for tag in tagList:
                val = int(tag.attributes["maskband"].value)
                l.append(val)
        except Exception, e:
            print 'Invalid parameter description, ',e
        return l

    def getParameterFlat(self, tagList):
```

```
        l = []
        tag = None
        try:
            tag = tagList[0]
        except:
            print 'VahineFlatReshapeFilter : Invalid or no parameter tag'
            return l
        try:
            if tag.attributes["convert"].value == 'flat':
                l.append('flat')
            else:
                l.append('unflat')
            l.append(int(tag.attributes["maskband"].value))
        except Exception, e:
            print 'Invalid parameter description, ', e
        return l

    def getParameterGrsir(self, pname, tagList):
        l = []
        index = 1
        for tag in tagList:
            try:
                if tag.attributes["activ"].value == 'on':
                    if pname == 'VahineGrsirLearning':
                        l.append(index)
                    if pname == 'VahineGrsirInverse':
                        if tag.attributes["regulation"].value == 'best':
                            l.append(3)
                        elif tag.attributes["regulation"].value == 'all':
                            l.append(7)
                        else:
                            try:
                                val = int(tag.attributes["regulation"].value)
                                val = val << 3
                                val = val | 5
                                l.append(val)
                            except Exception, e:
                                print 'Invalid parameter description, ',e
                else:
                    if pname == 'VahineGrsirInverse':
                        l.append(0)

                index = index + 1
            except Exception, e:
                print 'Invalid parameter description, ',e
        return l

    def getProcess(self):
        if self.__processList__ != None:
            return
        self.__processList__ = []
        for process in self.getRootElement().getElementsByTagName("process"):
            if process.nodeType == process.ELEMENT_NODE:
                p = Process()
                try:
                    p.name = process.attributes["name"].value
                    p.run = process.attributes["run"].value == 'True'
                    p.input = self.getInOut(process.getElementsByTagName("input"))
                    p.output = process.getElementsByTagName("output")[0].attributes["value"].value
                    p.regul = process.getElementsByTagName("regulation")
                    p.parameter = self.getParameter(p.name, process.getElementsByTagName("parameter"))
                except Exception, e:
                    print 'Invalid process description, ', p.name,' :', e
                self.__processList__.append(p)
        return self.__processList__
```

Listing B.6 – La classe de gestion de la configuration de chaque processus. Extrait du script de pilotage `vahine.py` des traitement numérique vahiné.

```
def usage():
    print "usage : vahine [parameters]"
    print "using the following parameters:\n"
    print "  -h --help: this message\n"
    print "  -c --configuration: xml configuration file\n"
```

```
    print "  -e --execute: execute all process, default is simulate\n"
    print "  c, are mandatory \n"

def main():                                                                    ①
    """main program loop"""
    try:
        opts, args = getopt.getopt( sys.argv[1:],
                                    "hc:e",
                                    [ "help", "configuration=" , "execute"] )

    except getopt.GetoptError, err:
        print str(err)
        sys.exit( 2 )

    #
    # process options
    #
    configuration = None
    simulate = True
    for opt in opts:
        if opt[0] in ( "-h", "--help" ):
            usage()
            sys.exit( 0 )

        if opt[0] in ( "-c", "--configuration" ):
            configuration = opt[1]

        if opt[0] in ( "-e", "--execute" ):
            simulate = False
    if configuration is None:
        usage()
        sys.exit(1)

    config = Configuration()
    config.readXml(configuration)
    processList = config.getProcess()
    currentProcess = ''
    try:
        for process in processList:
            currentProcess = process.name
            process.simulate = simulate
            process.execute()
    except Exception, e:
        print "Error in process ", currentProcess, e


if __name__ == '__main__':
    main()
```

Listing B.7 – La fonction principale extraite du script de pilotage `vahine.py` des traitement numérique vahiné.

B.4 Fichier de conversion XSLT

Dans le cadre d'une chaîne d'intégration continue utilisant Trac, Bitten et Boost nous avons besoin d'effectuer une conversion XSLT. Celle-ci est utilisée pour formater la sortie des tests Boost dans le format d'entrée de Bitten.

B.5 Fichier de spécification pour la mise en paquet

Le listing B.9 représente le fichier de spécification pour la création du paquet contenant la bibliothèque `OrfeoToolBox`.

```
<xsl:stylesheet  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>
 <xsl:template match="/">
    <unit_tests>
       <xsl:apply-templates/>
    </unit_tests>
 </xsl:template>
 <xsl:template match="TestSuite/TestCase">
     <test>
        <status>
    <xsl:if  test="@result = 'passed'">success</xsl:if>
    <xsl:if  test="@result != 'passed'">failure</xsl:if>
  </status>
        <fixture>
    <xsl:value-of select="../@name" />
        </fixture>
        <type>test</type>
        <name>
    <xsl:value-of select="@name" />
        </name>
     </test>
  </xsl:template>
</xsl:stylesheet>
```

Listing B.8 – Fichier XML/XSL `boost2bitten.xsl` de conversion pour l'interprétation des rapports BoostTest par Bitten.

```
Name: OrfeoToolbox
Version: 3.4.1
Release: 1%{?dist}
Summary: OTB, the ORFEO Toolbox is a library of image processing algorithms developed by CNES
Group: System Environment/Libraries
License: Cecill
URL: http://www.orfeo-toolbox.org
Source0: OrfeoToolbox-%{version}.tgz
Patch0: %{name}-%{version}-lpg-598.patch
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)

BuildRequires: cmake gdal-devel libgeotiff-devel cableswig boost
Requires: gdal libgeotiff

# don't build debug package
#%define debug_package %{nil}

%description
OTB, the %{name} is a library of image processing algorithms developed by CNES in the frame of the ORFEO
      Accompaniment Program. OTB
is based on the medical image processing library ITK, http://www.itk.org, and offers particular functionalities
for remote sensing image processing in general and for high spatial resolution images in particular.

%package devel
Summary: Development Libraries for the %{name} library
Group: Development/Libraries
Requires: pkgconfig
Requires: libgeotiff -devel cableswig
Requires: %{name} = %{version}-%{release}

%description devel
Header files for the %{name} library. The %{name} is a library of
image processing algorithms developed by CNES in the frame of the ORFEO Accompaniment
Program. OTB is based on the medical image processing library ITK, http://www.itk.org, and
offers particular functionalities for remote sensing image processing in general and for
high spatial resolution images in particular.

%prep
%setup -q
%patch0 -p1 -b .lpg~

%build
%cmake . -DBUILD_EXAMPLES:BOOL=OFF \
        -DBUILD_SHARED_LIBS:BOOL=ON \
        -DBUILD_TESTING:BOOL=OFF \
        -DBUILD_OTB_USE_EXTERNAL_ITK:BOOL=OFF \
        -DBUILD_ITK_CSWIG_JAVA:BOOL=OFF \
        -DBUILD_ITK_CSWIG_PYTHON:BOOL=ON \
        -DBUILD_ITK_CSWIG_TCL:BOOL=OFF \
        -DBUILD_OTB_USE_VISU_GUI=OFF \
        -DGEOTIFF_INCLUDE_DIRS:STRING=/usr/include/libgeotiff \
        -DOTB_INSTALL_LIB_DIR:STRING=/lib64/otb \
        -DITK_INSTALL_LIB_DIR:STRING=/lib64/otb \
        -DOTB_USE_VISU_GUI:BOOL=OFF \
        -DCMAKE_C_FLAGS_DEBUG:STRING="-pg -g" \
        -DCMAKE_CXX_FLAGS_DEBUG:STRING="-pg -g" \
        -DCMAKE_EXE_LINKER_FLAGS_DEBUG:STRING="-pg" \
        -DCMAKE_SHARED_LINKER_FLAGS_DEBUG:STRING="-pg" \
        -DCMAKE_MODULE_LINKER_FLAGS_DEBUG:STRING="-pg"

make VERBOSE=1 %{?_smp_mflags}
%install
rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$RPM_BUILD_ROOT

%clean
rm -rf $RPM_BUILD_ROOT
%post -p /sbin/ldconfig
%postun -p /sbin/ldconfig

%files
%defattr(-,root,root,-)
%{_bindir}/
%{_libdir}/otb/lib*
```

Listing B.9 – Exemple de fichier spec.

Annexe C

La méthode GRSIR

C.1 Généralités

La méthode GRSIR est une méthode statistique pour évaluer les propriétés physiques des matériaux de surface à partir d'images hyperspectrales. L'approche développée est basée sur l'estimation de la relation fonctionnelle entre certains paramètres physiques et les spectres observés. A cet effet, une base de données de spectres synthétiques est générée par un modèle physique de transfert radiatif et utilisée pour estimer la fonctionnelle.

Cette méthode se décompose en deux phases : une d'apprentissage ayant pour but d'établir la fonctionnelle et une phase d'inversion. Cette dernière peut alors se faire massivement sur un grand nombre d'images. La figure C.3 montre l'ensemble des étapes et les flux de données associés.

La méthode GRSIR est basée sur la méthode SIR (Sliced Inverse Regression) qui est une méthode de régression semi-paramétrique ayant un temps de calcul court par rapport aux autres méthodes de régression semi-paramétriques.

Nous donnons ici quelques notions théoriques indispensables pour comprendre l'essence de ces méthodes.

C.2 Notions théoriques sur la régression inverse par tranches (SIR)

La méthode SIR ne nécessite aucune spécification de la distribution de l'erreur ϵ et de la fonction de lien f. Elle repose sur le postulat fondamental que l'essentiel de l'information de la régression de X sur Y se trouve dans un sous-espace $\beta = (\beta_1, ..., \beta_k)$ avec $k < d$ tel que :

$$Y = g(\beta_1 X, ..., \beta_k X, \epsilon)$$

Rappelons que d est ici le nombre de bandes spectrales.

Définition : Etant donnée une transformation monotone T, on appelle courbe de régression inverse standardisée la fonction de $\Re$ dans $\Re^p$ définie par : $y \longrightarrow E[z|T(y)]$.

Définition : Le sous-espace linéaire de $\Re^p$ de dimension k engendré par les β_k est appelé "espace EDR" ("effective dimension reduction").

Théorème : Soit $\Gamma = Cov(E[z|T(y)])$ la matrice de covariance de la courbe de régression inverse. Celle-ci est dégénérée dans chaque direction orthogonale aux directions de l'EDR. Un corollaire montre que ces directions de l'EDR peuvent être dé-

duites des vecteurs propres associés aux K plus grandes valeurs propres de $\Sigma^{-1}\Gamma$ où $\Gamma = Cov(E[X - \bar{X}|T(Y)])$ et où Σ est la matrice de covariance de la variable X.

C.3 Principales étapes de SIR

La méthode SIR se décline en 5 étapes détaillées ci-dessous :

centrage des données : $Z_i = X_i - \bar{X}$ avec $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$

tri des paramètres : On trie chaque paramètre $(Y_1, \ldots, Y_p)$ dans un ordre croissant et on divise l'intervalle de variation en N tranches notées S_h, $h = 1, ..., N$.

calcul des moyennes « intra » tranche : $\bar{m_h} = \frac{1}{n\bar{p_h}}\sum_{i=1}^{n} Z_i I_{Y_i \in S_h}$ où $\bar{p_h} = \frac{1}{n}\sum_{i=1}^{n} I_{Y_i \in S_h}$ est la proportion des spectres synthétiques qui tombent dans la tranche S_h

calcul des moyennes « inter » tranche : On calcule la matrice de covariance des moyennes "inter" tranche défini par : $\bar{\Gamma} = \sum_{h=1}^{N} \bar{p_h}\bar{m_h}\bar{m_h}^t$

estimation de l'EDR : On calcule les vecteurs propres $\beta_1, ..., \beta_k$ de l'estimation $\bar{\Sigma}^{-1}\bar{\Gamma}$ avec $\bar{\Sigma}$ l'estimation de la covariance des variables centrées tel que :

$$\bar{\Sigma} = \frac{1}{n}\sum_{i=1}^{n} Z_i Z_i^t$$

l'estimation de la covariance des variables X centrées.

C.4 Régularisation gaussienne

Dans la méthode GRSIR nous appliquons la méthode de réduction de dimensionalité SIR décrite précédemment avec une étape supplémentaire : « la régularisation ». En effet l'estimation des axes EDR par diagonalisation de $\bar{\Sigma}^{-1}\bar{\Gamma}$ n'est pas stable dès que les spectres X sont bruités.

Pour éviter l'instabilité nous utilisons dans GRSIR une méthode de régularisation inspirée de Tikhonov résoudre un problème mal conditionné, l'idée est de perturber légèrement les données : un changement minimum dans les données initiales devra impliquer un changement minimum dans les solutions.

L'inversion GRSIR consiste à remplacer l'inverse de Σ : Σ^{-1} dans la méthode SIR par $(\Sigma^2 + \delta I_p)^{-1}\Sigma\Gamma$ comme le suggère Tikhonov. Le paramètre δ est libre. Pour résoudre notre problème nous calculons alors les k premiers vecteurs propres correspondants aux k plus grandes valeurs propres de :

$$(\Sigma^2 + \delta I_p)^{-1}\Sigma\Gamma$$

Les spectres X de la base d'apprentissage sont bruités par un bruit gaussien de moyenne nulle ayant la même matrice de covariance que le bruit instrumental (estimée par ailleurs). Ils constituent alors la base test. La valeur optimale du paramètre de régularisation δ est celle qui va minimiser la différence entre valeurs initiales des paramètres physiques et valeurs estimées après régression GRSIR. Cette adéquation est quantifiée par le paramètre normalisé NRMSE est défini par :

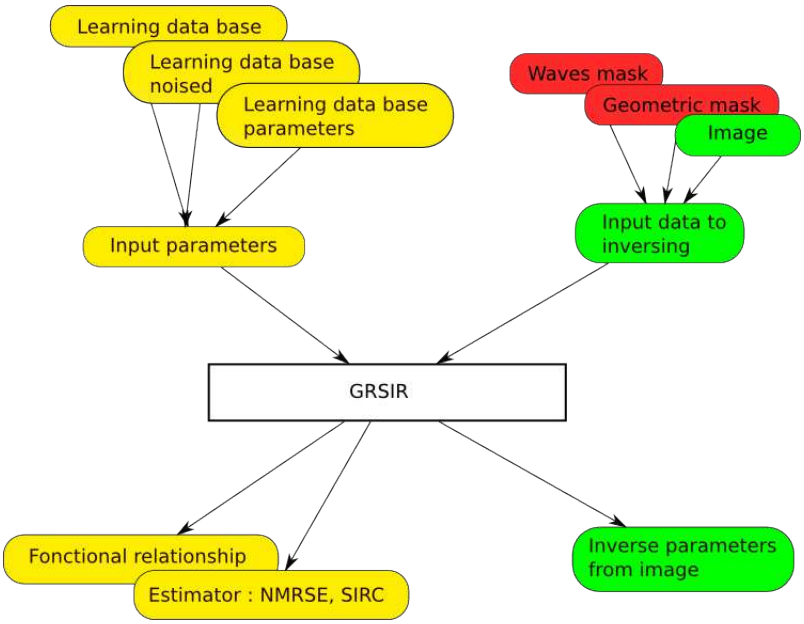

FIGURE C.1 – flot de données de GRSIR

$$NRMSE = \sqrt{\frac{\sum(\hat{y}_i - y_i)^2}{\sum(y_i - \bar{y})^2}}$$

avec

$$\bar{y} = \frac{1}{n}\sum y_i$$

C.5 Implémentation OTB de GRSIR

L'implémentation de l'algorithme GRSIR avec la bibliothèque OTB peut être résumé par les figures C.1 et C.2. Celles-ci montrent les paramètres d'entrées, comme par exemple les bases d'apprentissages et les masques. Le résultat de l'algorithme est une image hyperspectrale dite inversée. La deuxième figure (C.2) montre le découpage de l'algorithme en blocs fonctionnels.

Enfin, pour avoir un aperçu des classes implémentées le lecteur se reportera au diagramme de la figure C.3. Celui-ci détaille notamment les différents filtres OTB implémentés.

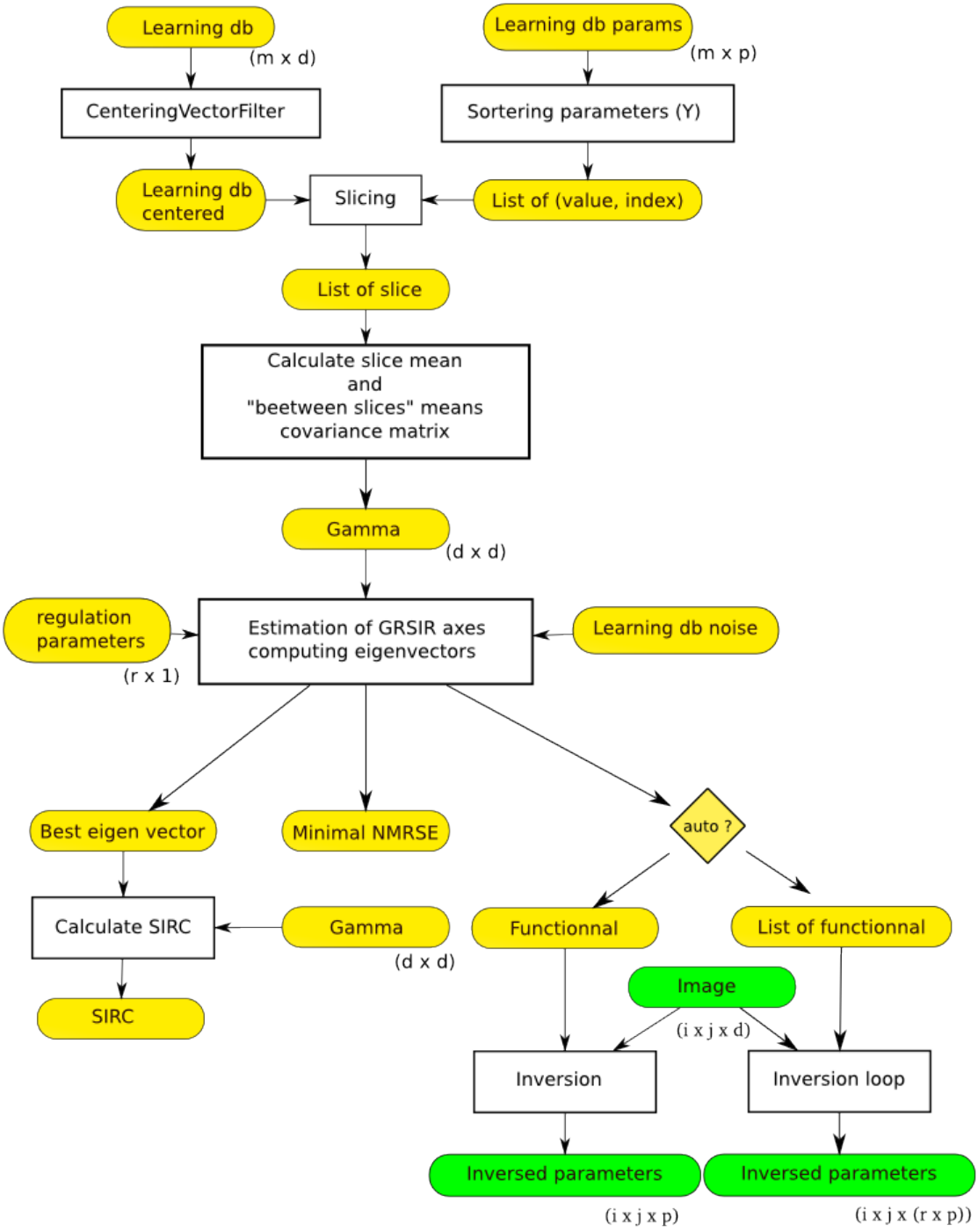

FIGURE C.2 – Détail de l'implémentation de GRSIR

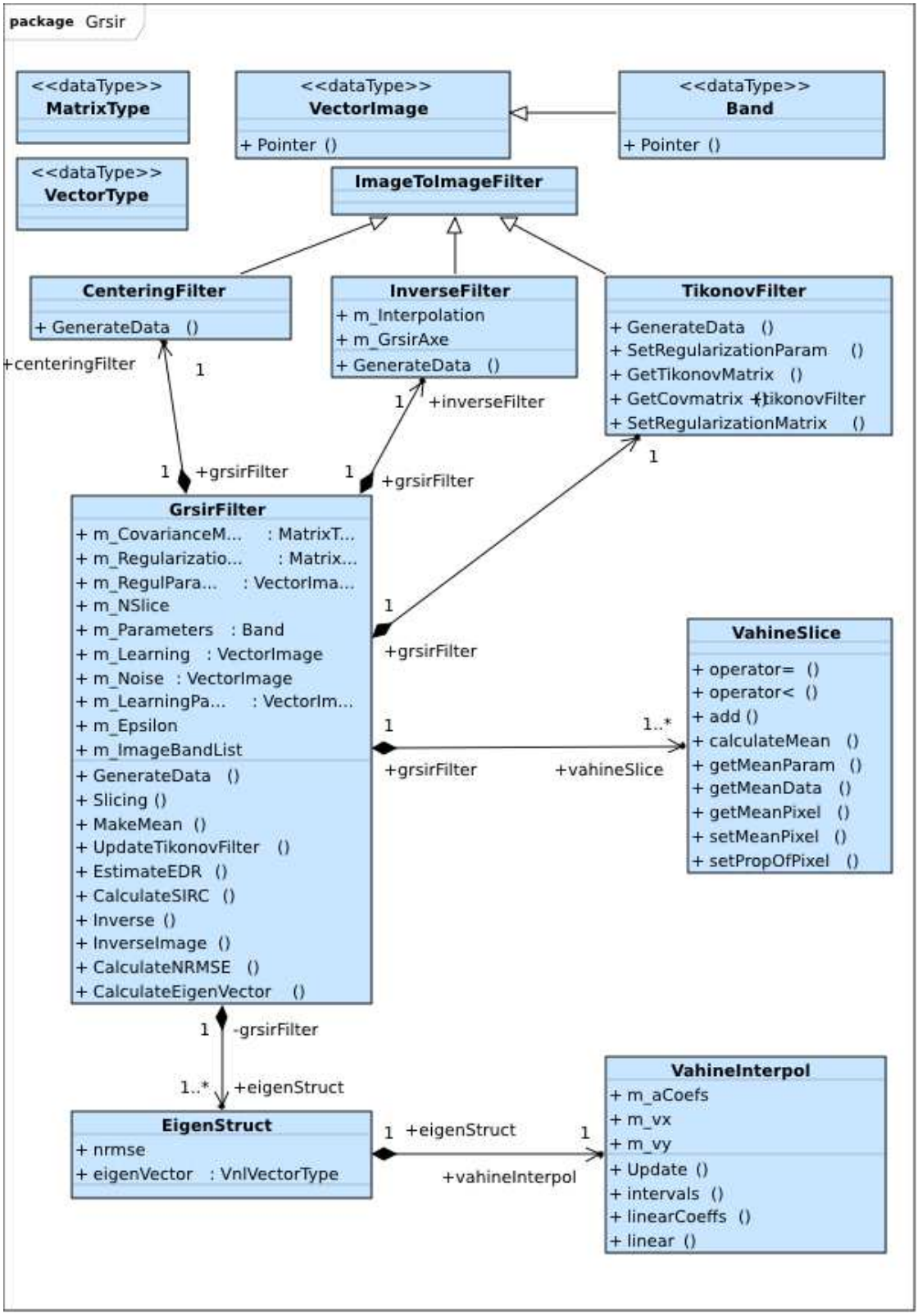

FIGURE C.3 – Schéma structurel de GRSIR

Glossaire

Acceptance (tests d') : *Voir Validation.*

Canevas : Elément de la bilbiothèque Qt utilisé pour afficher les graphiques 2D.

Classification : opération algorithmique qui consiste à attribuer une classe ou catégorie à chaque objet (ou individu), à classer, en se basant sur des données statistiques. Voir classification non-supervisée ou classification supervisée.

Classification non-supervisée : opération de classification ne nécessitant pas d'autres informations que les données brutes.

Classification supervisée : classification qui nécessite des connaissances préalables.

Commit : action d'envoyer des modifications locales vers le serveur de gestion de versions.

Design pattern : ou patron de conception est un concept de génie logiciel destiné à résoudre les problèmes récurrents en utilisant la conception objet.

Directive (préprocesseur) : instruction qui conduit à une transformation du code source (souvent un remplacement) avant l'étape de compilation.

Distance : ressemblance entre deux objets au sens mathématiques.

Endianness : ce terme désigne le fait qu'en informatique les données qui peuvent être représentées sur plusieurs octets peuvent être organisées de différentes façons. Deux orientations sont disponibles « big-endian » et « little-endian » . Par exemple, dans le premier cas, l'octet de poids le plus fort est enregistré à l'adresse mémoire la plus petite, l'octet de poids inférieur est enregistré à l'adresse mémoire suivante et ainsi de suite. A l'opposé dans le cas « little-endian » c'est l'octet de poids le plus faible qui est enregistré en premier.

Fixture : définit l'ensemble des opérations nécessaires à l'établissement et à la terminaison de l'environnement de test. Typiquement l'allocation de ressources externes comme la connexion à une base de données. [Niemi, 2004]

Fonction de vraisemblance : *Voir vraisemblance.*

Géoréférencement : action qui consiste à relier un objet à sa position dans l'espace par rapport à un système de coordonnées géographiques.

Ingénierie inverse : c'est la traduction de l'expression anglaise « Reverse Enginnering ». C'est l'art d'étudier un objet afin d'en découvrir les mécanismes et les finalités. Appliquée à l'informatique, cette science vise à observer et étudier un programme pour comprendre les algorithmes qu'il utilise. Cela peut se faire dans le but de les reproduire ou de les améliorer, ou simplement pour la compréhension.

Instance : copie d'un objet appartenant à une classe donnée, qui a ses données propres et se comporte indépendamment des autres objets de la même classe.

Itérateur : un itérateur est un objet associé à un conteneur, qui va permettre de balayer l'ensemble des objets se trouvant dans le conteneur, et ceci sans avoir aucune idée de la structure de données sous-jacentes.

Libre (logiciel, licence) : un logiciel libre est un logiciel dont l'utilisation, l'étude, la modification et la duplication en vue de sa diffusion sont permises.

Monte-Carlo : méthode de calcul numérique itérative, basée sur la convergence statistique.

Macro-définition : association d'un texte de remplacement à un identificateur, tel que l'identificateur est remplacé par le texte dans tout usage ultérieur.

Masque (spatial) : objet de même dimension spatiale que l'image mais contenant une seule bande généralement binaire définissant une zone d'intérêt de l'image par la valeur des ses pixels a 1.

Matricielle (image) : image définie par un tableau de pixels.

Non supervisé (algorithme) : algorithme qui ne nécessite aucune connaissance à part les donnés brutes.

Open-source (logiciel) : *Voir Libre.*

Pixel : élément discret de la dimension spatiale.

Pixel pur : pixel qui ne contient qu'un seul type de matériau ou unité de terrain chimiquement homogène.

Pôle : *Voir Spectre de référence.*

Prédicat : propriété d'un objet ou de plusieurs objets.

Raster (image) : C'est un terme générique et un anglicisme couramment utilisé pour désigner une image matricielle. Cette dernière est une image numérique dans un format de données composé d'une juxtaposition de pixels.

Régression : Un modèle de régression décrit les relations entre une variable à expliquer Y et une variable explicative X sous la forme : $Y = f(X, \epsilon)$.

RVB (image) : type de codage pour une image matricielle ou l'espace de couleur est représenté à partir des couleurs primaires rouge, vert, bleu. Une image de ce type est donc définie par trois bandes spectrales.

Simplex : c'est l'enveloppe convexe d'un ensemble de points. C'est la généralisation du triangle pour une dimension quelconque.

Source (spectrale) : *Voir Spectre de référence.*

Spectel : élément discret de la dimension spectrale.

Spectre : flux d'énergie électromagnétique décomposé en fréquences.

Spectre de référence : spectre représentatif d'un type de terrain (caractérisé par la présence d'une espèce chimique ou d'un mélange particulier).

Spectre source : signature spectrale des pôles. *Voir Spectre de référence.*

Spectre synthètique : spectre généré par un algorithme typiquement de transfert radiatif.

Spectro-imageur : appareil à base de caméra CCD permettant l'acquisition d'une image sur plusieurs bandes spectrales.

Supervisé (algorithme) algorithme qui nécessite des connaissances préalables.

Template : patron définissant un modèle de fonction ou de classe dont certaines parties sont des paramètres.

Transfert radiatif Le transfert radiatif est le domaine de la physique décrivant l'interaction du rayonnement électromagnétique et de la matière. Cette discipline permet notamment l'analyse de la propagation lumineuse à travers un milieu gazeux.

Tuile : portion d'une image.

Validation (test de) : Les tests de validation permettent de vérifier si toutes les exigences client décrites dans les documents de spécification sont respectées.

Vectorielle (Image) : Une image vectorielle est décrite à l'aide de courbes et d'équations mathématiques.

Vraisemblance (Fonction de) : La fonction de vraisemblance souvent noté $L(x_1, \ldots, x_n|\theta_1, \ldots, \theta_k)$ est une fonction de probabilités conditionnelles qui décrit les paramètres θ_j d'une loi statistique en fonction des valeurs des variables aléatoire x_i supposées connues. Elle s'exprime à partir de la fonction de densité $f(x|\theta)$ par $L(x|\theta) = \prod_{i=1}^{n} f(x_i, \theta)$.

Index

Bibliographie

[Balbous, 2008] Balbous, T. (2008). Les méthodes agiles. Agile Gardener. URL http://www.agilegardener.com.

[Bernard-Michel *et al.*, 2007] Bernard-Michel, C., Douté, S., Gardes, L. et Girard, S. (2007). Estimation of mars surface physical properties from hyperspectral images using sliced inverse regression. Rapport technique inria-00187444-v2, MISTIS (INRIA Rhône-Alpes / LJK Laboratoire Jean Kuntzmann).

[Bernard-Michel *et al.*, 2009a] Bernard-Michel, C., Gardes, L. et Girard, S. (2009a). Gaussian regularized sliced inverse regression. *Statistics and Computing*, 19(1):85–98. URL http://dx.doi.org/10.1007/s11222-008-9073-z.

[Bernard-Michel *et al.*, 2009b] Bernard-Michel, C., Gardes, L. et Girard, S. (2009b). Gaussian regularized sliced inverse regression. *Statistics and Computing*, 19(1):85–98.

[Bibring *et al.*, 2004] Bibring, J.-P., Soufflot, A., Berthé, andLangevin, M., Y., Gondet, B., Drossart, P., Bouyé, andCombes, M., M., Puget, P., Semery, A., Bellucci, andFormisano, G., V., Moroz, V., Kottsov, V., Bonello, andErard, G., S., Forni, O., Gendrin, A., Manaud, andPoulet, N., F., Poulleau, G., Encrenaz, T., Fouchet, andMelchiori, T., R., Altieri, F., Ignatiev, N., Titov, andZasova, D., L., Coradini, A., Capacionni, F., Cerroni, andFonti, P., S., Mangold, N., Pinet, P., Schmitt, andSotin, B., C., Hauber, E., Hoffmann, H., Jaumann, andKeller, R., U., Arvidson, R., Mustard, J., Forget et F. (2004). *OMEGA : Observatoire pour la Minéralogie, l'Eau, les Glaces et l'Activité*, pages 37–49. ESA SP-1240 : Mars Express : the Scientific Payload.

[Boost, 2010] Boost (2010). Boost c++ libraries. URL http://www.boost.org/.

[CNES, 2009] CNES (2009). Système dual d'observation optique de résolution métrique. URL http://smsc.cnes.fr/PLEIADES/Fr/.

[Dobigeaon *et al.*, 2009] Dobigeaon, N., Moussaoui, S., Tourneret, J. Y. et Carteret, C. (2009). Bayesian separation of spectral sources under non-negativity and full additivity constraints. *signal processing*, 89:2657–2669.

[Douté *et al.*, 2011] Douté, S., Ceamanos, X., Luo, B., Schmidt, F., Jouannic, G. et Chanussot, J. (2011). Intercomparison and validation of techniques for spectral unmixing of hyperspectral images : A planetary case study. *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING*.

[ESA, 2009] ESA (2009). Mars express orbiter instruments. URL http://www.esa.int/SPECIALS/Mars_Express/SEMUC75V9ED_0.html.

[Freeman et Freeman, 2005] Freeman, E. et Freeman, E. (2005). *Tête la première : Design Patterns*. O'REILLY.

[GDAL, 2010a] GDAL (2010a). Gdal data model. URL www.gdal.org/gdal_datamodel.html.

[GDAL, 2010b] GDAL (2010b). Gdal driver implementation tutorial. URL www.gdal.org/gdal_drivertut.html.

[Luo *et al.*, 2009] Luo, B., J., C., S., D. et X., C. (2009). Unsupervised endmember extraction of martian hyperspectral images. *In Hyperspectral Image and Signal Processing : Evolution in Remote Sensing, 2009. WHISPERS '09*, pages 1–4. IEEE.

[Mercier, 2008] Mercier, L. (2008). Veille technique en matière d'analyse et de visualisation d'images astrophysiques. Rapport technique, Laboratoire de Planétologie de Grenoble.

[NASA, 2010] NASA (2010). Pds : The planetary data system. URL http://pds.jpl.nasa.gov/about/about.shtml.

[Nascimento et Dias, 2004] Nascimento, J. M. P. et Dias, J. M. B. (2004). Vertex component analysis : A fast algorithm to unmix hyperspectral data. *IEEE TRANS. GEOSCI. REM. SENS*, 43:898—910. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.7329.

[Niemi, 2004] Niemi, I. (2004). Better unit tests using correct test fixtures. *In* SPRING, éditeur : *SEMINAR IN SOFTWARE ENGINEERING*, volume 76, page 650.

[Schmidt, 2007] Schmidt, F. (2007). *Classification de la surface de Mars par imagerie hyperspectrale OMEGA. Suivi spatio-temporel et étde des dépots saisonniers de CO2 et H2O.* Thèse de doctorat, Université Joseph Fourier Grenoble 1.

[Streicher, 2010] Streicher, M. (2010). Packaging software with rpm. URL http://www.ibm.com/developerworks/library/l-rpm1/.

[Wikipedia, 2011] Wikipedia (2011). Arbre couvrant de poids minimal. URL http://fr.wikipedia.org/wiki/Arbre_couvrant_de_poids_minimal.

MEMOIRE D'INGENIEUR C.N.A.M. en INFORMATIQUE

Système d'analyse et de visualisation d'images hyperspectrales appliqué aux sciences planétaires.

Ludovic Mercier Grenoble le 8 avril 2011

Résumé

L'imagerie hyperspectrale est une technique clef pour l'étude des planètes et de l'univers. Elle est utilisée pour conduire une grande variété d'investigations tel que la télédétection et la caractérisation des surfaces planétaires. Le nombre de satellites et de sondes spatiales équipés de spectro-imageurs augmente régulièrement. La communauté scientifique fait donc face à une augmentation exponentielle de la quantité et de la taille des données. Ces images requièrent des algorithmes d'analyse de plus en plus sophistiqués et performants pour faciliter le traitement de ces images.

C'est dans ce contexte qu'est né le projet Vahiné et c'est dans ce projet que s'inscrit ce mémoire. Le but étant de mettre à la disposition de la communauté un logiciel libre permettant la manipulation et le traitement de ces images. La réalisation de ce logiciel a porté sur l'implémentation d'algorithmes de traitements d'images (classification, inversion, ...) mais a aussi comporté la réalisation d'une interface de visualisation.

Mots clefs : algorithme, image hyperspectrale, GDAL, Orfeotoolbox, séparation de sources, télédétection, visualisation.

Abstract

Hyperspectral imaging is used for studying planets and the universe. It is used to drive a large variety of investigations such as remote sensing and characterizing planetary surfaces. The number of satellites and space probes equipped by imaging spectrometer is increasing. The scientific community therefore faces an exponential increase of the number of datasets and their size. These images require sophisticated and efficient analysis algorithms for their treatment.

This is the context in which the Vahine project is borned and it is in this project that this report was realized. The goal is to provide the community with an open source software allowing handling and processing of these images. This software require the development of treatment algorithms (classification, inversion, ...) but also the realization of a visualization interface.

Key word : algorithm, hyperspectral image, GDAL, Orfeotoolbox, remote sensing, source separation, visualization.