DEPARTEMENT INFORMATIQUE - IUT 2 GRENOBLE

Année Universitaire 2008-2009
MEMOIRE DE STAGE JMMC-TRA-2910-0002

Développement d'application - Interface web d'applications scientifiques

LAOG

(Stage du 06 Avril au 26 Juin 2009)

Présenté par
Samuel PRETTE
Promotion 2A

Jury

IUT : Mr GEROT Cédric
IUT : Mr CHASTEL Frédéric
Société : Mr MELLA Guillaume

Table des matières

Introduction

Déjà deux années passés à l'IUT informatique de Grenoble et il faut maintenant découvrir le monde professionnel lors d'un stage de 10 semaines. C'est l'occasion de valider des acquis, de voir que se que l'on a appris sur des projets scolaires peut être réutilisé et amélioré dans le monde de l'Entreprise. J'ai particulièrement apprécié la démarche de recherche de stage, lors de laquelle il faut réussir à trouver un sujet motivant, mais aussi convaincre l'Entreprise qui le propose que l'on est capable de le mener à bien.

Tout commence par une séance d'introspection. Parmi tout les thèmes abordées à l'IUT, il faut choisir celui sur lequel je serrais intéressé de faire un stage. Lors de ma deuxième année à l'IUT, les deux projets qui m'ont le plus motivé ont pour dénominateur commun la réalisation d'une interface homme-machine en java, à savoir un jeux de poker en réseau et un chat. Je commence donc à chercher des offres portant sur cette thématique.

Dans la multitude d'offres de réalisation de site web, je trouve une annonce du laboratoire d'astrophysique de Grenoble (LAOG). Avec ce stage, j'aurais l'occasion de faire des interfaces java, mais avec une approche nouvelle plus accès sur la communication. Après un entretien avec Guillaume Mella me voilà parti pour 10 semaines au LAOG.

Le **L**aboratoire d'**A**strophysique de l'**O**bservatoire des sciences de l'univers de **G**renoble dédie ses recherches à la compréhension de l'univers. Ici sont rassemblées des compétentes diverses allant de l'astronomie, à l'électronique, la mécanique, la physique et bien sur l'informatique. Le laboratoire réalise des instruments dédiés à l'observation, développe des théories et les simule mais est aussi tourné vers le public pour lequel des missions de formation et de diffusion des compétences sont régulièrement organisées.

L'équipe du LAOG est très accueillante, il y a toujours une personne prête à vous aider ou simplement à vous faire découvrir ses travaux. Pour commencer,j'ai tout particulièrement travaillé avec Guillaume Mella mon tuteur qui n'a pas compté les heures pour m'aider. J'ai apprécié sa façon de donner des vecteurs d'information quand j'avais des questions et la manière avec laquelle il m'a fait comprendre les besoins auxquels je dois répondre en me donnant des domaines d'étude. Ca m'a permis d'attaquer le projet en ayant les idées claires sur les solutions techniques à utiliser. Sa connaissance du système Linux a aussi été très utile. De plus, j'ai aussi eu l'occasion de discuter de mes problématiques avec un collègue informaticien de Guillaume, Sylvain Lafrasse. Ensuite, pour comprendre les demandes des futurs utilisateurs de la solution mise en place, Gilles Duvert était mon interlocuteur. Grâce à ses compétences scientifiques et informatiques il a pu répondre à nombre de mes questions.

Pour organiser notre travail nous avions des habitudes simples. J'étais en stage avec un autre étudiant de l'IUT, Nicolas et nous avions tous les lundi matin une réunion qui rassemblait les membres de l'équipe technique: Guillaume, Sylvain, le directeur scientifique Gilles Duvert[1] (parfois le directeur du LAOG) pour discuter de nos avancées de la semaine passée, de nos difficultés et aussi de nos projets pour la semaine à venir. Cela permettait d'avoir les idées claires sur le déroulement du stage et son avancée. Guillaume étant très présent lors de nos phases de développement, avoir des personnes externes pour donner leur avis permet d'éviter l'effet tunnel en nous faisant garder en tête les objectifs. Si on ajoute à ces réunions les interventions de Guillaume chaque jour pour prendre connaissance de nos travaux en cours, on peux penser que notre méthode de travail ressemble au Scrum étudié en gestion de projets.

Voir annexe organigramme du JMMC page 34

Le stage ne s'est pas décomposé en phases d'études de développement et de test distincts. Pour chaque fonctionnalités et problématiques, Guillaume commençait par me donner des vecteurs d'information avec des test à faire sur différentes solutions techniques. Après chaque validation de fonctionnalité on passait à une autre suivant un modèle itératif. Le stage n'a donc pas été linéaire dans ses phases d'étude et de développement, j'ai d'ailleurs des recherches de solutions techniques à effectuer avant la fin de mon stage le 26 juin.

1 Gilles Duvert : Astronome - Directeur scientifique du LAOG.

I) Analyse de l'existant :

a) Cahier des charges détaillé

1) Sujet

Le JMMC dispose de logiciels scientifiques fonctionnant en mode client/serveur.
Pour améliorer la qualité de ses logiciels, le JMMC souhaite mettre à jour son mécanisme d'interfaces graphiques dynamiques. Le stage comportera une étude logicielle suivit d'une réalisation au sein de l'équipe technique du JMMC. Ce service devra être validé sur le logiciel de préparation d'observation du JMMC.

2) Planning

Semaine 1	Intégration : Étude des méthodes de développement du JMMC, installation des stations de travail Mandriva.
Semaine 2	Étude de l'existant : compréhension du fonctionnement d'ASPRO et étude de solutions techniques (CVS, analyseur syntaxique[2] xml, svg)
Semaine 3	Étude de l'existant : étude du comportement des logiciels du JMMC auquel ASPRO[3] doit correspondre
Semaine 4-5	Réalisation : modifications de l'interface ASPRO en accord avec la charte graphique du JMMC
Semaine 6	Réalisation : tests du bon fonctionnement des différentes modifications
Semaine 7	Réalisation : validation du travail effectué et améliorations éventuelles
Semaine 8	Documentation : finalisation de la rédaction de la documentation
Semaine 9	Réalisation : améliorations autres en accord avec la liste des demandes des utilisateurs.
Semaine 10	Réalisation et Documentation : test des dernières modifications et finalisation de la documentation

Les principales étapes du stage sont donc :

- Compréhension des méthodes de travail du JMMC
- Recherches et tests de solutions techniques propres à améliorer l'interface web d'ASPRO
- Discutions avec les utilisateurs pour connaître les attentes
- Travail sur différents points de la liste des demandes des utilisateurs.
- Validation du travail effectué

2 L'**analyse syntaxique** consiste à mettre en évidence la structure d'un texte, ici de l'XML.
3 **ASPRO** : The **A**stronomical **S**oftware to **PR**epare **O**bservations

b) ASPRO Web

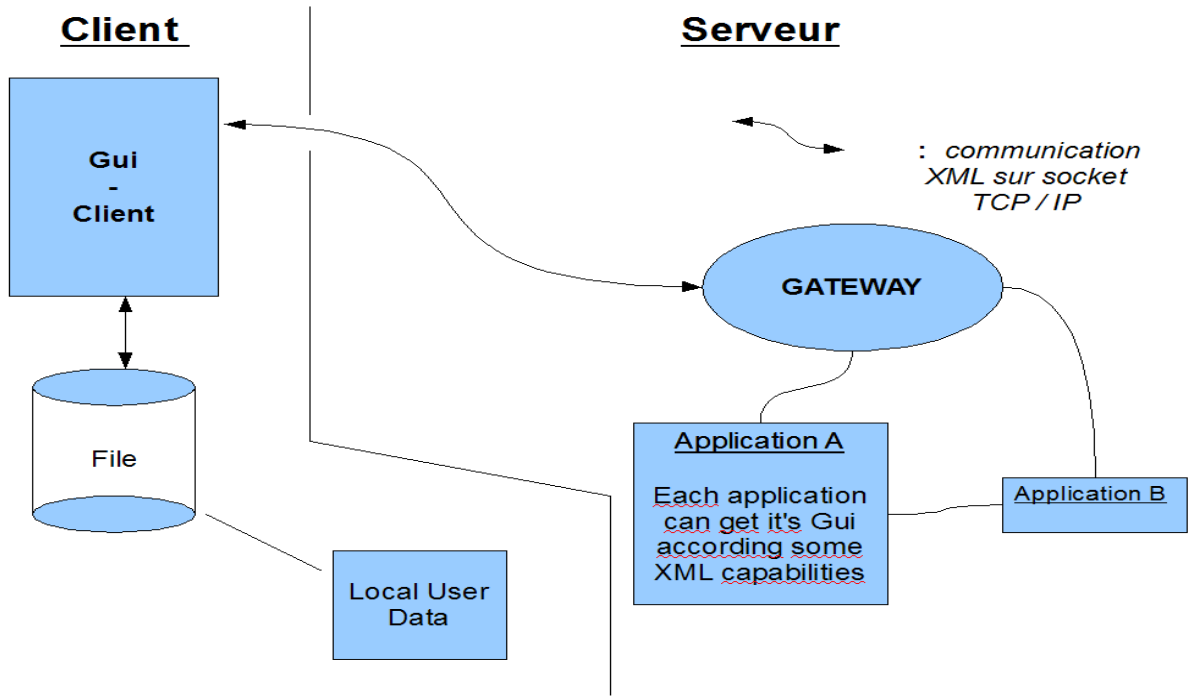
1) Description de l'architecture d'ASPRO Web

ASPRO comporte une partie client et serveur. Coté serveur, un service assure la communication réseau entre les différentes applications et le client. Les informations sont envoyées dans des sockets entre les applications ou du client vers le serveur et sont organisés au format XML[4].
Le client reçoit et émet donc des document XML qui peuvent être des descriptions d'image en SVG[5] (plot), de composants graphique java (widgets), des messages d'information, de configuration ou portant sur l'état de l'application.

Le cœur des applications coté serveur est géré par gildas, un client peut d'ailleurs interagir directement avec le serveur en X11[6]. Le choix de laisser la possibilité aux clients de communiquer au moyen de messages XML est motivé par la volonté de préserver la bande passante qui est beaucoup plus sollicitée lors d'une session X11 que pour un fonctionnement basé sur des échanges XML. De plus, ces échanges XML permettent de laisser une grande liberté de développement à d'éventuels clients alternatifs tout en assurant une meilleur sécurité des serveurs qu'en mode X11.

L'objectif du stage est de rajeunir la partie Gui – Client qui est développée en java, en utilisant des solutions viables sur le long terme, d'améliorer l'ergonomie, de permettre des modifications et améliorations aisées pour l'équipe du JMMC en posant une base de développement stable pour le client.

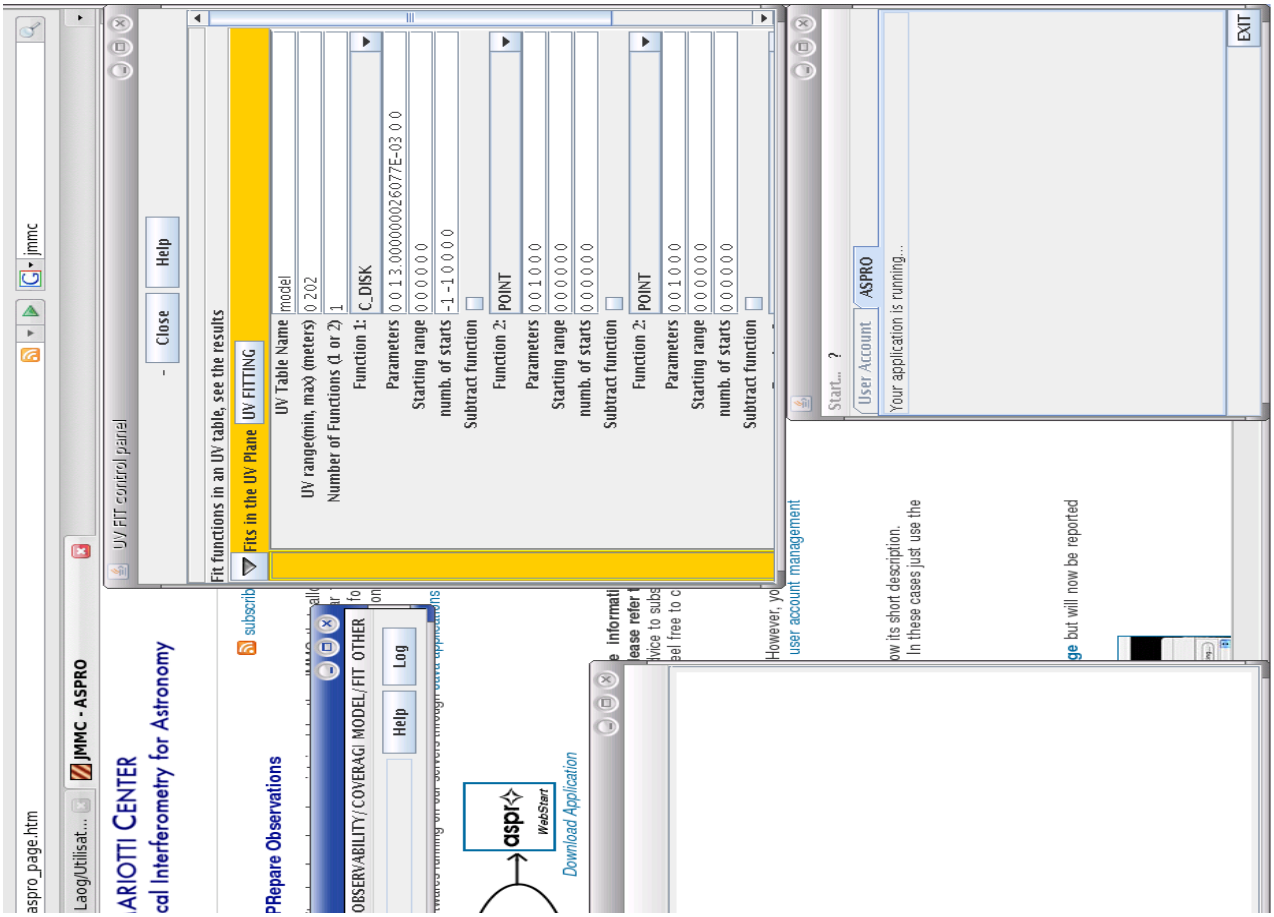
Synthèse du fonctionnement d'ASPRO

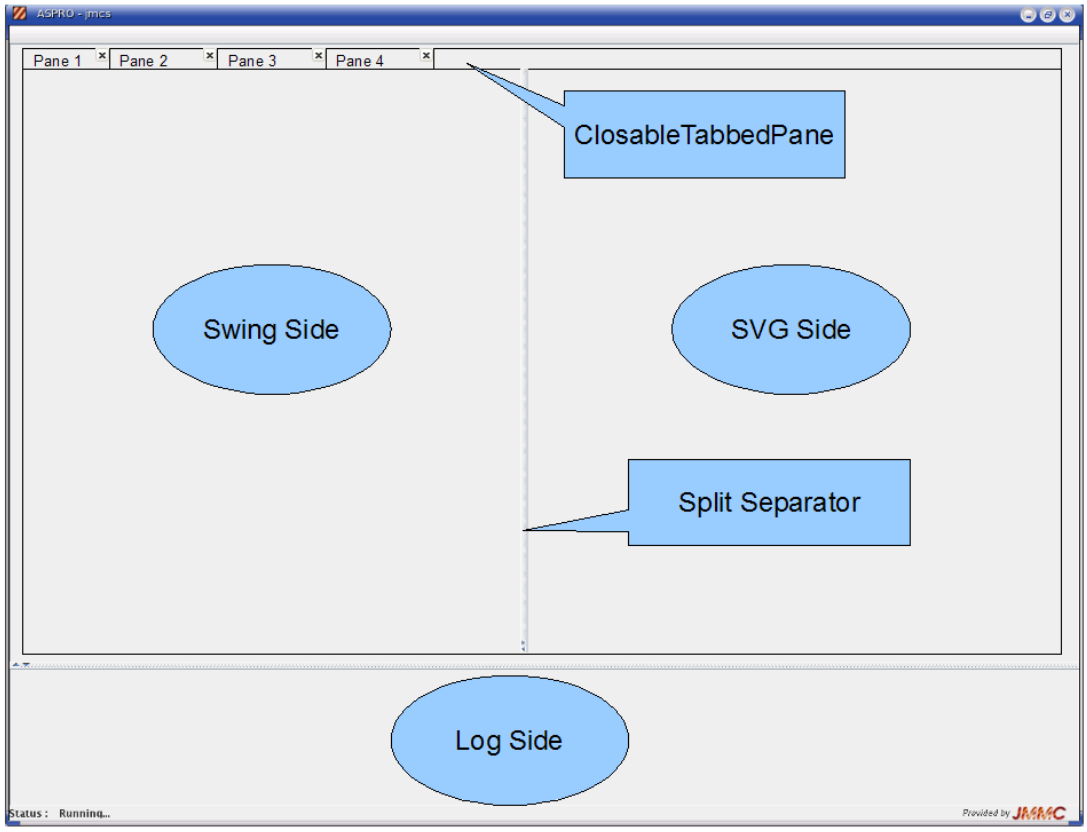


4 **XML** : « langage extensible de balisage ») est un langage informatique de balisage *générique*. Il sert essentiellement à stocker/transférer des données de type texte
5 ***Scalable Vector Graphics (SVG)*** : format de données conçu pour décrire des ensembles de graphiques vectoriels.
6 **X Window System** ou **X11** ou simplement **X** est une *interface utilisateur graphique* de type "fenêtré" qui gère l'interaction homme-machine par l'écran.

Pour l'évolution graphique du logiciel ASPRO, les utilisateurs veulent éviter d'avoir une multitude de fenêtres à gérer en même temps, pouvoir mettre en parallèle les données qu'ils ont entrées dans ASPRO et le graphique résultant de ces données. De plus, Gilles demande de pouvoir personnaliser l'affichage des différents types de données depuis le serveur par des « feuilles de style ».

Ancienne version d'ASPRO (4 fenêtres)





(IHM non finalisée lors de la rédaction du mémoire)

Lors d'une réunion avec Gilles, la représentation des données a été établie pour chaque Panel de la fenêtre à onglets suivant deux scénarios :

- Le scénario par défaut : Le serveur envoie séparément des documents XML à destination de la zone « Swing » s'il s'agit d'une description de widgets, de la zone « SVG » s'il s'agit d'une image svg ou de la zone texte de « Log »pour mettre à jour l'historique. La représentation des informations est alors statique et conforme à l'exemple ci-dessus.
- Le scénario avec feuille de style : Le serveur commence par envoyer une description de la disposition de différentes zones qu'il pourra remplir ultérieurement. Les informations pourront donc être représentées comme le serveur l'aura choisi et non plus statiquement comme pour le premier scénario.

A chaque Panel de la fenêtre à onglets correspond un scénario. L'utilisateur pourras donc mettre en parallèle les paramètres qu'il a entrés dans la partie widgets et le graphique qui en aura résulté.

L'objectif étant de rajeunir XmlBasedGui, il faut en premier lieux pouvoir reprendre toute les fonctionnalités de ce dernier avant d'en ajouter de nouvelles. J'avais créé un tableau récapitulatif pour en sauvegarder la liste sur le Twiki du LAOG :

Résumé des attentes

XmlBaseGui : *ancienne version d'ASPRO*

Xbg : *nouvelle version*

Reprise des ancienne fonctionnalités *Nouvelles fonctionnalités*

Fonctionnalité	XmlBaseGui	Xbg
SVG		
Affichage svg	✓	✓
Affichage svg composite	✓	✓
Zoom	✓	✗

Fonctionnalité	XmlBaseGui	Xbg
Widgets		
JLabel	✓	✓
JButton	✓	✓
JTextfield	✓	✓
JTextaera	✓	✓
JSlider	✓	✓
JCombobox	✓	✓
JCheckbox	✓	✓
Mise à jours des widgets	✓	✓

Fonctionnalité	XmlBaseGui	Xbg
Autre		
Barre de statut	✓	✓
Barre de menu	✓	✗
Panneau de login	✓	✓

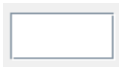
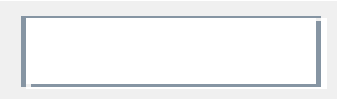
Nouvelles fonctionnalités	Xbg
Fenêtre unique	✓
Sauvegarde SVG en image	✗
Layout Swing décrits dans le XML	✓
Gestion des feuilles de style	✗
Format xml universel	✓ (cookswing)
Format svg universel	✓ (batik)
Lien vers la page http://apps.jmmc.fr/account pour la fenetre de login	✗

Ce graphique représente aussi l'avancée au moment de la rédaction du mémoire

- Dans les zones SVG il faut pouvoir afficher une image obtenue à partir du document XML envoyé par le serveur, l'image pouvant être transmise par un envoi ou plusieurs envois successifs. De plus, l'utilisateur peut être amené à devoir zoomer sur les graphiques reçus.
- La barre de menu de l'application est elle aussi envoyée par le serveur mais l'application en elle même a aussi besoin d'avoir certains menus: quitter le programme ou retourner à l'écran de connexion.

- Quand l'utilisateur change la valeur d'un composant éditable, il faut pouvoir repérer ce changement et émettre un message de mise à jour vers le serveur pour lui indiquer la nouvelle valeur prise par les composants en question.

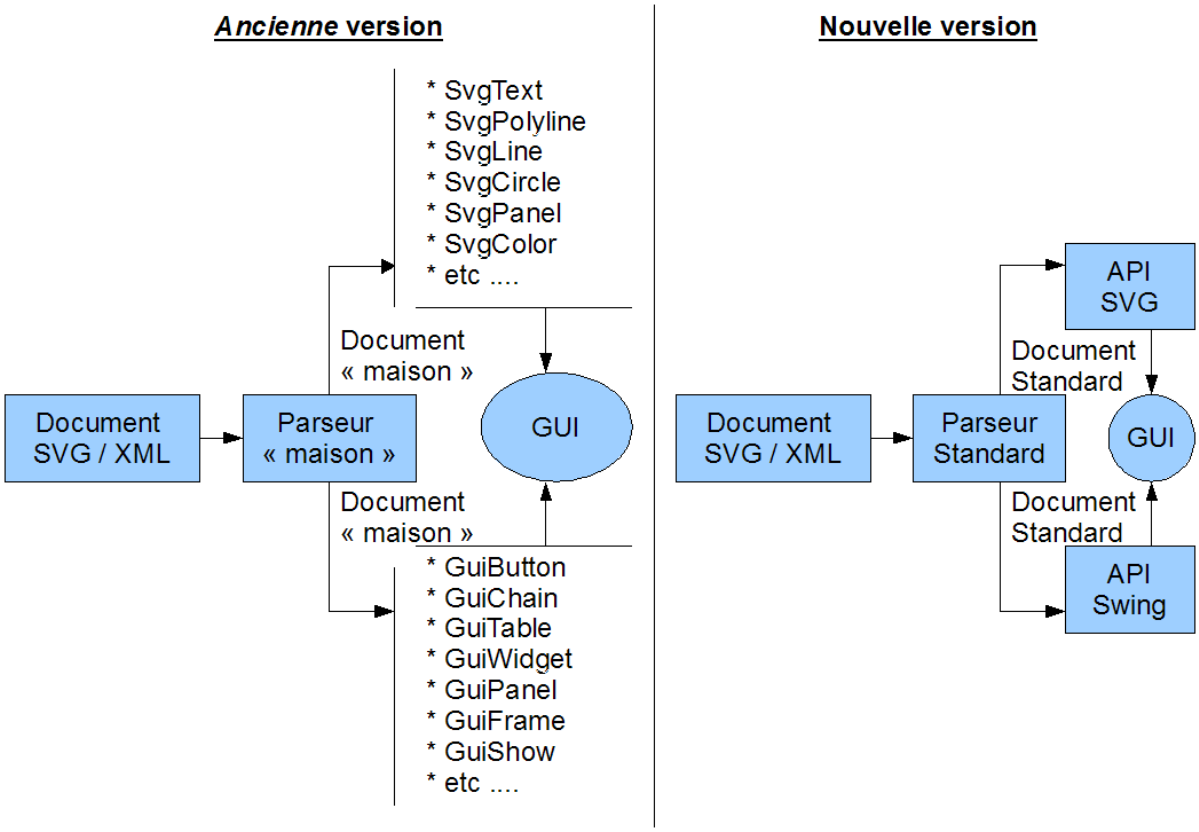
Dans les zones de composants Swing, il faut pouvoir afficher au minimum les composants que l'ancienne version d'ASPRO sait gérer, à savoir :

Nom standard	Nom XmlBased Gui	Description	Exemple
JLabel	CHAIN	Un JLabel permet d'afficher du texte non éditable ou une image dans une interface swing	
JButton	BUTTON	Un JButton est un composant avec lequel l'utilisateur peut interagir en cliquant	
JTextField	CHAIN (éditable)	Un JTextField permet à l'utilisateur de faire des entrées texte dans l'interface	
JTextAera	CHAIN	Un JTextAera affiche une zone de test éditable de plusieurs lignes	
JSlider	SLIDER	Un JSlider affiche une barre pondérée par des valeurs avec laquelle l'utilisateur peux choisir une nombre	
JComboBox	CHOICE	Un JComboBox est un menu déroulant avec lequel l'utilisateur peut faire une sélection parmi une liste définie ou entrer sa valeur si le composant est éditable	
JCheckBox	LOGIC	Un JCheckBox permet à l'utilisateur de choisir un élément ou non, et a une valeur binaire	

- Il a été demandé que tout ces composants puissent être disposés facilement dans la fenêtre par l'utilisation de layout swing.

4) Méthodes de travail

L'un des besoins exprimé est de reprendre toute les fonctionnalités d'ASPRO en utilisant des librairies pour remplacer différentes fonctionnalités pour lesquels il n'existait pas de librairies lors de la création d'ASPRO. Cela permettra une meilleur interopérabilité de la solution, un plus large panel de solutions techniques et un suivi des éventuelles évolutions de ces techniques par les équipes responsables de ces librairies.



XmlBasedGui fonctionne grâce à des classes qui gèrent chaque type de composants swing. L'un des intérêts de l'utilisation de librairies pour remplacer ces fonctionnalités est de simplifier le code en diminuant grandement le nombre de classes d'XmlBasedGui (ces classes étant remplacées par les librairies des API).

c)Recherche de solutions techniques

Pour fonctionner, XmlBasedGui a besoin de comprendre le XML qui lui est envoyé et, à partir de ces messages XML reçus, de dessiner des graphiques SVG ou des widgets swing.

Pour se faire, il a fallu rechercher 3 solutions techniques pour :

- gérer les échanges de messages
- transformer ces messages textuels en Objets Java
- utiliser ces Objets Java pour dessiner du SVG ou des composants Swing

1)Échange de messages

Pour transformer les lignes de texte reçues en document, ou DOM, il faut premièrement savoir

quand le serveur nous a envoyé un document XML entier. Pour se faire, une classe utilitaire XmlBuffer[7] est utilisée ; grâce à un StringBuffer et à des tests lui permettant de savoir quand la balise fermante d'un élément « root [8]» a été reçue, cette classe sait quand un document a fini d'être transmits.

Utilisation d'XmlBuffer

```
while ((line = in.readLine()) != null) {
    xmlBuffer.append(line + "\n");
    nbline++;

    while (xmlBuffer.containsOneWellFormedXmlDocument()) {
        GuiClient.processXml(xmlBuffer.getDocument());
        nbline = 0;
    }
```

2) Analyse du texte reçu

Deuxièmement, il faut pouvoir transformer ce document XML sous forme de String en document java ou DOM[9]. Pour se faire, il faut choisir quel analyseur syntaxique (parseur XML) utiliser ; il en existe plusieurs comme SAX ou le parseur java standard (javax.xml.parsers.DocumentBuilder).

Le parseur SAX est utile quand on veux pouvoir capturer des événements lors de l'analyse de notre document grâce à des handler[10].

On peux capturer un événement quand le logiciel passe sur un commentaire, le début ou la fin d'un élément ou sur un tous les caractères.

Voir l'annexe Test de SAX page 30

7 **XmlBuffer** : classe utilitaire développée par Guillaume Mella pour l'ancienne version d'ASPRO
8 **Élément « root »** : Première balise du document XML, mère de toute les autres
9 Le **Document Object Model** (ou **DOM**) est une recommandation du W3C qui décrit une interface permettant à des programmes informatiques d'accéder ou de mettre à jour le contenu, la structure ou le style de documents.
10 **Handler** : permet de manipuler des objets (to handle = manipuler).

Pour ASPRO, notre besoin est seulement de créer un document (objet java) à partir des messages texte envoyés par le serveur et ce, quelque soit le contenu de ce dernier car le document est ensuite traité suivant son type (svg ou description swing) dans des classe séparées.

Le parseur standard est donc préféré car il permet de faire le travail demandé sans alourdir le projet avec une nouvelle librairies.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder parser = factory.newDocumentBuilder();
doc = parser.parse(input);
```

3) Utilisation des documents

IHM

ASPRO communique avec son serveur avec des échanges XML. Ce dernier envoie des instructions permettant de créer des fenêtres dans un IHM java, en décrivant des composants Swing.

L'objectif de cette recherche est de standardiser ces échanges qui sont actuellement gérés par une librairie "maison" ; XmlBasedGui[11]. Il faut donc que la librairie sache faire au minimum ce qu'XmlBasedGui sait faire, la liste des fonctionnalités[12] ayant été documentées par Guillaume Mella.

La librairie recherchée devra être capable d'analyser l'XML, d'instancier les composants swing et de les afficher. De plus, cette librairie devra créer une instance des objets swing reçus permettant ainsi au client d'accéder facilement à ces derniers et d'en changer les propriétés. Dans cette même optique, la librairie devras offrir des outils type itérateur[13] permettant au développeur de parcourir

11 **XmlBasedGui** : code développé par Guillaume Mella en 2003 permettant de dessiner des composants Swing à partir d'XML.
12 **XmlBasedGui language** : http://jmmc.fr/~mella/REF_XML/
13 **Itérateur** : Un itérateur est un objet qui permet de parcourir tous les éléments contenus dans un autre objet, le plus

l'arbre constitué par les documents xml. La librairie devra implémenter les layout[14] swing et la majorité des paramètres existants sur chaque composant.

Enfin, il est souhaitable d'avoir une librairie finalisée, stable, déjà utilisée par le plus grand nombre pour garantir l'aspect universel et réutilisable de la solution mise en place tout en étant libre de droits.

La recherche se base sur une liste de librairies de java-source.net [15]:

SwiXml :

SwiXml crée bien des instances des objets Swing décrits dans les documents qu'on lui donne, et possède un itérateur permettant de faire facilement des traitements sur ces objets. Malheureusement, la librairie a une grosse limitation dans le sens ou il n'est pas possible de donner le contenu des JComboBox[16] directement dans les messages XML. De plus, elle fonctionne exclusivement avec des document JDOM (un autre analyseur syntaxique) ce qui multiplie les outils qui traitent des mêmes points et nous aurait contraint d'ajouter une librairie.

CookSwing :

Tout comme SwiXml, cette librairie permet de créer les instances attendues. Elle couvre l'intégralité des fonctionnalités de XmlBasedGui et en dispose d'autres. Elle comprend une dizaine de layout Swing. De plus elle est bien documentée.

Graphiques scientifiques

Le serveur envoie aussi des images SVG au client. Pour afficher du SVG dans un environnement, Java Batik est largement reconnu et utilisé par de nombreux projets d'envergure (The Apache Cocoon project, The Apache FOP project , Oracle Corp.'s JDeveloper10i ...). Il y a donc eu peu de recherche en se qui concerne la librairie gérant le SVG.

II) Solutions proposées :

a) CookSwing

L'utilisation de CookSwing est simple, il suffit de lui donner un document décrivant une interface Swing pour qu'il crée les instances java correspondantes. Le développeur a donc tout loisir d'utiliser ces instances par la suite.

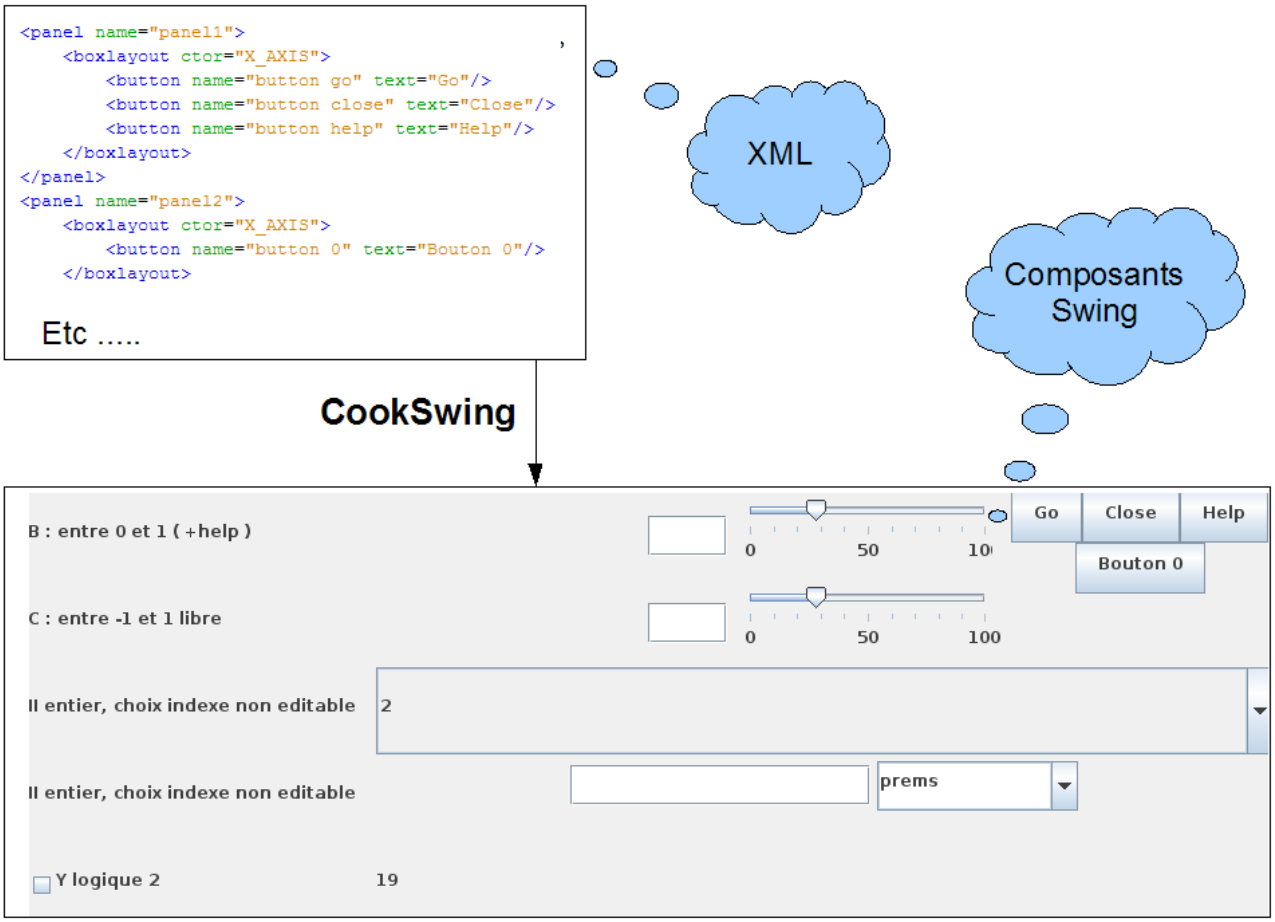
souvent un conteneur.

14 **Layout** : Un layout Swing permet de décrire la disposition des éléments dans une interface en swing

15 **Liste d'API** : http://java-source.net/open-source/xml-user-interface-toolkits

16 **JComboBox** : permet de faire apparaître une liste déroulante lorsque l'utilisateur clique sur l'objet, ainsi que de faire une sélection.

```
CookSwing cookSwing = new CookSwing();
Container container = cookSwing.render(document);
```



méthode auxiliaire pour créer des fonctions d'analyse des panneaux graphiques.

Voir l'annexe création d'un itérateur page 31

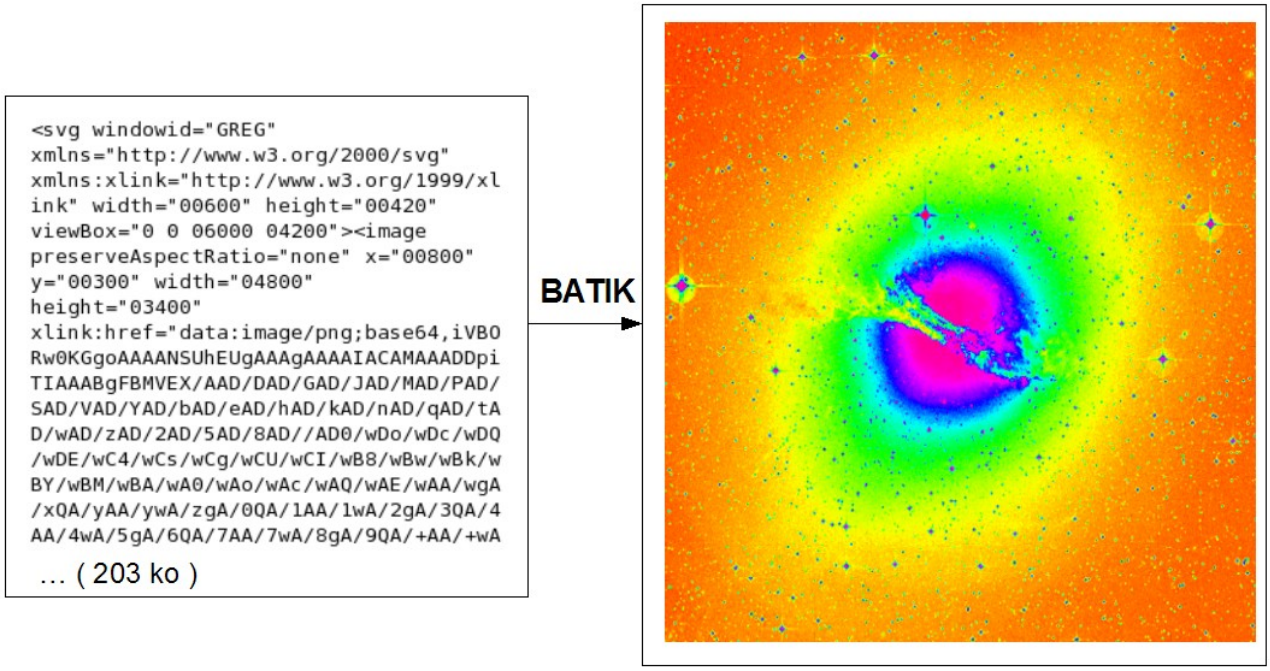
Grâce à une méthode récursive, on regroupe tous les enfants d'un container. L'itérateur ainsi produit permet de parcourir tout les éléments Swing contenu dans un container et d'effectuer des actions sur ces instances.

CookSwing ne pourra comprendre que les documents XML correspondant au formalisme défini dans sa documentation, le serveur va donc devoir subir des modifications. En effet, coté serveur, les documents XML sont créés par une couche écrite en C qui parcoure les widgets gildas et pour chacun d'eux donne la représentation XML définie par XmlBasedGui. Un chercheur de l'IRAM a accepté de modifier cette couche C du serveur à l'issue de mon stage, quand la documentation lui indiquera le fonctionnement du nouveau client et la syntaxe à respecter.

Voir annexe ancienne syntaxe et nouvelle syntaxe page 32

b) Gestion des graphiques scientifiques

Le cœur du serveur est écrit en Gildas qui est une boite à outils disposant d'un moteur de graphiques vectoriels. Gilles a implémenté un module lui permettant de produire du SVG. L'intérêt des graphiques vectoriels est qu'il est possible d'effectuer des zooms sans perdre en qualité de l'image.



JPanel" .

Il y a deux techniques pour afficher une image depuis un document SVG avec Batik :

- Créer un fichier svg temporaire et indiquer au composant JSVGCanvas[18] l'URI[19] de ce fichier. Cette technique n'a pas été retenue car elle pose des problèmes de sécurité et travaille sur le disque sans intérêt particulier.
- Créer un SVGDocument[20] avec un parser XML et ensuite associer le JSVGCanvas à ce SVGDocument. Cette méthode permet d'avoir un document sur lequel il est possible d'effectuer des recherches d'attributs. De plus, il est possible d'ajouter des nœuds à ce document si le serveur envoie de nouvelles instructions. Dans ce cas, l'image est mise à jour automatiquement par le JSVGCanvas.

17 **JPanel** : Container Swing pouvant contenir des widgets Swing et qui permet de les positionner dans la fenêtre (**JFrame**).
18 **JSVGCanvas** : Container défini par l'API Batik permettant de faire un render d'SVG à partir d'une URI ou d'un SVGDocument.
19 **URI** : courte chaîne de caractères identifiant une ressource sur un réseau.
20 **SVGDocument** : Type particulier de document contenant du SVG et que Batik comprend.

La principale difficulté pour répondre à ces contraintes a été de parser le SVG sous forme de String avec un parser qui ne vérifie pas la DTD[21] sur Internet puisque le LAOG a un proxy et que l'on veut éviter de faire cette opération qui n'a pas d'intérêt. Contrairement aux parsers JDOM, SAX ou standard, le parser de Batik répond à cette attente et a été préféré.

Nous voulions pouvoir donner directement à Batik un document standard (org.w3c.dom.document) mais nous n'avons par réussi à effectuer de render avec ce type de document. Il a donc fallu parser une deuxième fois notre document pour qu'il soit compris par Batik :

Voir annexe Utilisation de Batik page 33

Un JSVGCanvas est un widget Swing, il suffit donc de le placer dans un container dont le layout lui permet de prendre tout l'espace disponible (GridLayout(1,1) est utilisé) pour avoir l'image dans la JFrame.

III) Mise en place de la solution

a) Environnement de travail

Le LAOG a mis à disposition une machine sous Mandriva qui a été configurée dans les premiers jours avec l'aide de Guillaume.

Pour développer le projet, tester les librairies, partager le code, sauvegarder ses données, simuler un serveur et lui faire envoyer du XML, divers programmes et scripts ont étés utilisés.

NetBeans

Les avantages de Netbeans :

- Un gros gain de temps en automatisant de nombreuses tâches répétitives.
- Il supporte un grand nombre de langages et a un code couleur

léments pouvant apparaître et leur contenu, c'est-à-dire les sous-éléments et les

clair pour chacun d'eux.

- Compile les projets à la volée et crée un jar au besoin.

Le seul inconvénient que j'ai retenu dans son utilisation est qu'il faut avoir une machine récente pour permettre une utilisation fluide.

Guillaume avais mis au points un script de test que j'ai réutilisé pour envoyer des fichiers XML au client.

```
echo "<config entityName=\"tester2\" to=\"JavaGui\"></config> $(cat $1)" > /dev/tcp/$REMOTEHOST/$REMOTEPORT
```

Rsync

Pour sauvegarder nos données, nous avions accès à 4 GO d'espace disque sur un serveur donc le contenu est protégé, car sauvegardé tout les soirs. Pour faciliter les sauvegardes, Guillaume nous a écrit un script pour utiliser Rsync dont la fonction est de faire une copie des données en ne reprenant que les changements d'une sauvegarde à une autre.

```
rsync -v -e ssh --archive $HOME/ userlaog.obs.ujf-grenoble.fr:myRsync/$HOSTNAME/
```

CVS

Pour partager nos projets en cours et accéder à d'autres développées au JMMC, Guillaume nous à présenté CVS[22], un outils de travail collaboratif. Cet outil est directement intégré à NetBeans et m'a permit de partager les différentes versions de code en faisant un « commit ».

Ce système permet de visionner les changements d'une version à une autre, et d'expliquer les changements lors d'un commit avec un bref commentaire. Le code est alors directement commenté pour indiquer les différentes versions desquels ont résulté le code affiché en précisant le nom des personnes ayant effectuées les Commit.

Exemple de traces laissé par CVS

```
/*******************************************************************************
 * JMMC project
 *
 * "@(#) $Id: StatusBar.java,v 1.7 2009/04/20 08:22:33 lafrasse Exp $"
 *
 * History
 * -------
 * $Log: StatusBar.java,v $
 * Revision 1.7  2009/04/20 08:22:33  lafrasse
 * Optimized space between the JMMC logo and the resizing handle.
 *
 * Revision 1.6  2009/04/16 15:44:51  lafrasse
 * Jalopization.
 *
 * Revision 1.5  2009/04/15 11:55:24  mella
 * fix space
 *
 * Revision 1.4  2009/04/09 06:26:07  sprette
 * Change small jmmc logo name
 * Add space on the right bottom corner into the status bar (for Mac OS X)
```

22

Twiki

Nous avons utilisé un Twiki[23] pour stoker des références et aussi informer les membres du LAOG des travaux en cours.

b) Contraintes d'implémentation

Pour résumer, le client ne fait que réagir à des données que lui envoie le serveur en affichant des images SVG, des widgets Swing ou des informations textuelles et va ensuite avertir le serveur d'éventuelles modifications que l'utilisateur pourrait opérer sur des variables. L'objectif est de pouvoir garantir une logique entre les différentes données présentées à l'utilisateur mais aussi que le serveur et le client aient les mêmes valeurs pour chaque variable.

La première des difficultés est donc de gérer cette logique alors que le client ne peux pas présumer des données que va lui envoyer le serveur. Pour pouvoir gérer ces contraintes, le serveur doit donner le même nom à tout les widgets représentant la même valeur, quelque soit leur type. Par exemple, si un JTextField, un JLabel et un JSlider représentant la même variable (et donc la même valeur), il faut que ces trois widgets aient le même nom. De plus, si l'utilisateur modifie la variable au moyen de l'un de ces trois widgets (par exemple en faisant glisser la barre du JSlider), le client doit réagir instantanément en mettant à jour les deux autres widgets avec la nouvelle valeur. Cette mise à jour doit se faire sur les widgets du panneau en premier plan, mais aussi sur d'éventuels widgets liés à cette variable dans d'autres panneaux cachés(voir IHM I.b.2).

Une autre contrainte s'ajoute pour les JSlider. En effet, un JSlider a un minimum et un maximum et si ce dernier est lié à un composant dont l'utilisateur peux changer librement la valeur (un JTextField par exemple), l'utilisateur doit être averti s'il entre une valeur incohérente.

Deuxièmement, sur chaque panel de widgets envoyé par le serveur et dont l'utilisateur peux changer

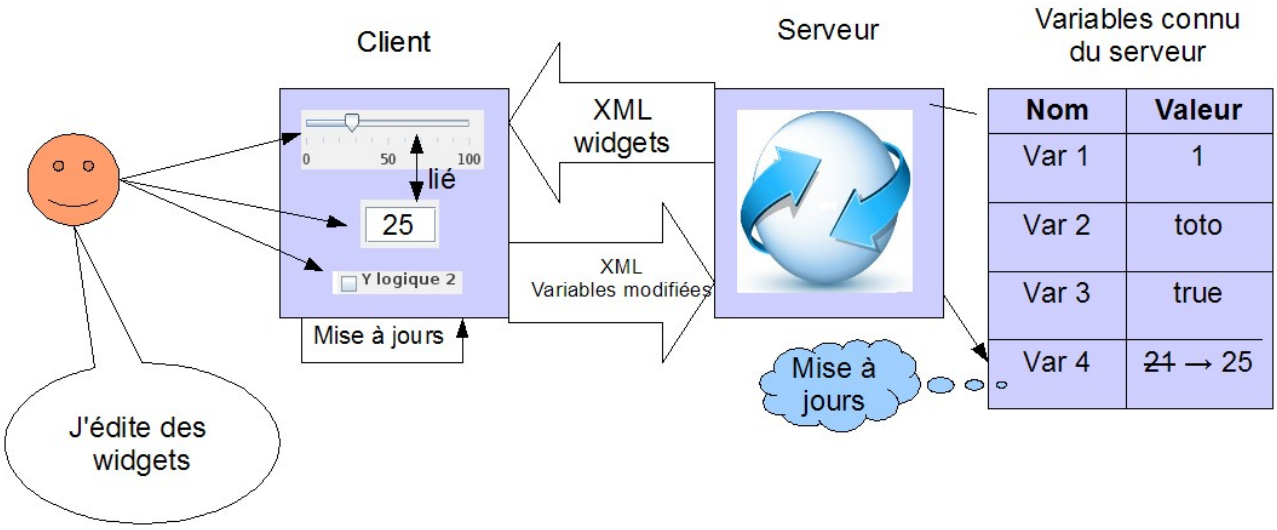
23 **Twiki** : http://www-laog.obs.ujf-grenoble.fr/twiki/ (toute les sections ne sont pas accessibles de l'extérieur).

la valeur de certains widgets, il y a un bouton qui permet à l'utilisateur de valider les nouvelles valeurs et de les envoyées au serveur. Lorsque le client clique sur ce bouton, il faut pouvoir construire un message XML à envoyer au serveur pour mettre à jour toute les variables qui ont été modifiées. Il faut donc pouvoir savoir quels variables ont étés modifiées par l'utilisateur en faisant un parallèle entre ce que le serveur a envoyé et ce que le client a au moment de la validation.

Troisièmement, le serveur peut envoyer des messages de mise à jour concernant des variables déjà représentées dans la fenêtre du client. Il faut alors mettre à jour tout les widgets représentant cette variable parmi les différents panneaux.

```
<gui_update>
    <var name="testupdate1" value="1"/>
    <var name="testupdate2" value="2"/>
    <var name="testupdate3" value="3"/>
    <var name="testupdate4" value="4"/>
</gui_update>
```

Remarque : dans un message de mise à jour, les seul informations représentées sont le nom de la variable suivi de sa nouvelle valeur. En effet il n'est pas nécessaire de préciser quel widget swing est à modifier puisqu'une même variable peut être représentée par une multitude de widgets différents.



c) Proposition d'implémentation

Dans cette partie, je vais présenter les différentes techniques utilisées pour répondre aux contraintes définies dans la partie précédente.

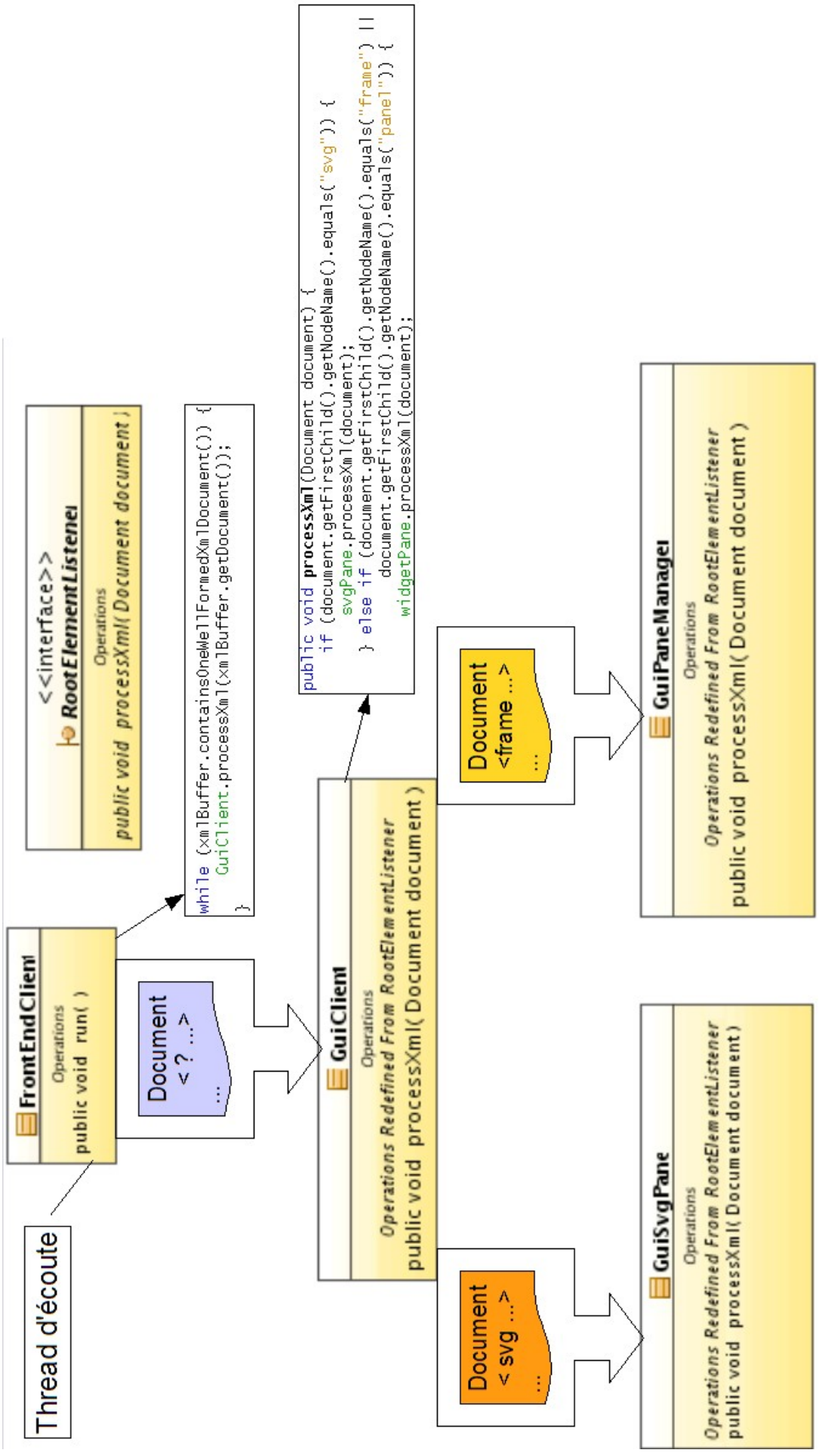
Le client possède un thread d'écoute qui reçoit les messages émis par le serveur. Comme expliqué dans la partie recherche des solutions techniques (I.c), on commence par détecter la fin de la réception d'un document XML bien formé pour ensuite créer une représentation java dont la manipulation est aisée ; le document.

Une fois que le thread d'écoute a créé un document, ce dernier est envoyé à la classe gérant son type. Pour que les document soit traités suivant leur type, les classes concernées répondent à une interface[24] qui indique qu'elles ont toute une méthode prévue pour traiter des documents.

```
public interface RootElementListener {

    void processXml(org.w3c.dom.Document document);
}
```

24 **Interface** : Une interface est constituée d'un ensemble de déclarations de méthodes sans implantation.


Thread d'écoute
FrontEndClient
Operations
public void run( )
Document
< ? ...>
...
<<interface>>
RootElementListener
Operations
public void processXml(Document document)
while (xmlBuffer.containsOneWellFormedXmlDocument()) {
GuiClient.processXml(xmlBuffer.getDocument());
}
GuiClient
Operations
Operations Redefined From RootElementListener
public void processXml(Document document)
public void processXml(Document document) {
if (document.getFirstChild().getNodeName().equals("svg")) {
svgPane.processXml(document);
} else if (document.getFirstChild().getNodeName().equals("frame") ||
document.getFirstChild().getNodeName().equals("panel")) {
widgetPane.processXml(document);
Document
< svg ...>
...
Document
<frame ...>
...
GuiSvgPane
Operations
Operations Redefined From RootElementListener
public void processXml(Document document)
GuiPaneManager
Operations
Operations Redefined From RootElementListener
public void processXml(Document document)

1) Gestion des variables liées

Comme présenté précédemment, des widgets ayant le même nom sont liés ; ils représentent tous une même variable de même valeur.

Pour répondre à cette contrainte, il faut commencer par savoir quand un widget est modifié par l'utilisateur, on utilise donc des Listeners[25] java pour détecter les actions utilisateur.

L'itérateur présenté en II.a trouve ici son utilité. En effet, CookSwing permet d'instancier tous les widgets décrits dans les documents XML reçus, il faut ensuite pouvoir atteindre toute ces instances pour leur ajouter des listeners.

Une méthode permet de faire passer une roulette sur chaque widgets présents dans un panel et d'y ajouter les bon listeners pour chaque type de widget. Cette méthode test chaque composant grâce à l'itérateur et regarde s'il s'agit d'un composant pour lequel il faut ajouter un listener.

Méthode d'ajout de listeners

```
Iterator iterator = SwingUtil.buildContainerIterator(contentPane);
while (iterator.hasNext()) {
    Container container = (Container) iterator.next();
    if (container instanceof JTextField) {
        JTextField textfield = (JTextField) container;
        textfield.addKeyListener(keyListener);
```

Dans l'exemple, un listener est ajouté à un JTextField, or comme expliqué dans les contraintes un JTextField peut être lié à un JSlider et donc avoir un minimum et un maximum. Grâce à ce « keyListener[26] » nous allons pouvoir savoir si l'utilisateur édite un JTextField contraint.

```
KeyListener keyListener = new KeyListener() {

    public void keyReleased(KeyEvent e) {
          Code executé quand une touche est relachée ...
              }
         }
```

Comme le JTextField et le JSlider ont le même nom, on peux faire des tests dès que l'utilisateur édite le JTextField et l'avertir s'il entre une valeur incohérente.

Pour les Jslider, on utilise un ChangeListener[27] pour récupérer les événements de changement de

25 **Listeners** : « écouteur » d'action de l'utilisateur permettant de capturer des événements.
26 **KeyListener** : permet d'écouter les action clavier de l'utilisateur lorsqu'il édite un composant donné.
27 **ChangeListener** : permet de savoir quand l'utilisateur glisse la barre d'un JSlider et change ainsi sa valeur.

valeur :

Récupération d'un changement de valeur pour un JSlider

```
ChangeListener changeListener = new ChangeListener() {

    public void stateChanged(ChangeEvent e) {
        if (e.getSource() instanceof JSlider) {
            JSlider slider = (JSlider) e.getSource();
            manager.updateAllWidgetPane(slider.getName(), String.valueOf(slider.getValue()));
        }
    }
};
```

Une fois qu'on a récupéré cet événement, on peux demander à tous les widgets représentant la même variable que le JSlider en question mette à jour leur valeur :

Méthode de mise à jour d'une variable

```
public void updateAllWidgetPane(String name, String value) {

    Enumeration<GuiPane> widgetPaneEnum = widgetPaneStorage.elements();
    while (widgetPaneEnum.hasMoreElements()) {
        widgetPaneEnum.nextElement().updatePaneContainers(name, value);
    }
}
```

Avec cette méthode, toute les variables d'un nom donné sont mises à jour, quelque soit le panel qui les contient.

Conclusion

Ce stage de 10 semaines permet de découvrir de nombreux aspects du travail d'informaticien que l'on ne peux pas complètement comprendre dans un milieux scolaire.
Il y a en premier lieu le sentiment de s'inscrire dans un ensemble de talents et d'expériences qui ont des objectifs communs. Ici, la capacité à faire partager ses difficultés ou ses réussites a autant d'importance voir plus que le simple faite d'abattre ses propres objectifs individuels puisque la valeur que l'on crée est à l'échelle d'un groupe.
Dans un milieux scolaire il me semble plus difficile de reproduire et de valoriser cette notion de travail d'équipe de part la limitation en nombre de projets de groupe et de part leur ampleur moindre.
Le plus grand apport que je pense tirer de se stage est donc de l'expérience sur les notions de travail en collaboration et d'expression en groupe. En effet cela m'a permis de mieux comprendre les notions de gestion de projets et d'entreprise dont j'ai pu suivre l'enseignement théorique, l'exemple typhique étant la pause café lors de laquelle on trouve souvent des solutions en discutant avec des personnes externes au projet.

Sur le thème de la communication, l'anglais est une nouvelle contrainte car je n'avais pas l'habitude d'écrire et de commenter mon code en anglais jusqu'à maintenant. Guillaume a insisté sur la lisibilité et l'importance de donner des nom compréhensibles à mes méthodes, classes et variables pour qu'une personne tierce comprenne plus rapidement le code.

L'intégration a été la première et je pense la phase qui a le plus d'importance. Guillaume a grandement facilité cet aspect de part sa motivation et son dynamisme. Mon tuteur est sensible à la notion d'équipe, lors de mon stage il a notamment mis en place un repas d'informaticiens par semaine pour aider à la diffusion des expériences. A l'échelle du LAOG, le partage est encouragé au travers de nombreux séminaires se déroulant pour la plupart le jeudi après-midi et auxquels je suis parfois allé. De plus, des activités sont organisées en dehors des heures de travail comme le jeudi midi ou les sportifs vont taper la balle.

Le bilan technique est aussi positif sur plusieurs points. Premièrement les séances de recherche de solution techniques ont eu comme effet d'élargir mon champ de vision sur plusieurs domaines comme le fonctionnement et l'utilité de l'XML. J'avais eu l'occasion de travailler sur des protocoles de communication lors du projet de réseau du quatrième semestre ainsi que lors du projet de deuxième année, ce travail sur une interface de communication en XML me permet d'intégrer de nombreux outils qui me seront utiles lors de mes prochains projets me demandant de mettre en place un protocole.

Ensuite, j'avais eu l'occasion de réaliser des interfaces homme-machine au sein de l'IUT, mais ce stage m'a fait découvrir une toute autre approche en injectant la composant dynamique. En effet jusqu'à maintenant j'avais réalisé des interfaces dont chaque scénario d'utilisation était codé en dur alors que ce sujet m'a demandé de pouvoir réagir et adapter une interface en fonction de signaux venant parfois de l'extérieur (serveur) et des actions utilisateur.

Remerciements

Je tiens à remercier Guillaume Mella pour la qualité de son soutien tout au long du stage, sa patience, son aide et sa motivation à me faire découvrir un large panel des activités du LAOG.

De même, je remercie Sylvain Lafrasse et Gilles Duvert pour le temps qu'ils ont passé à me conseiller et répondre à mes questions ainsi que Françoise Roch pour la visite du cluster.

Plus largement, je remercie les membres du LAOG qui m'ont tous bien accueillis et informés au besoin. Grâce à eux et à leur enthousiasme à me présenter leur travail j'ai pu comprendre quelques ficelles du fonctionnement d'un laboratoire de recherche.

Glossaire

Analyseur syntaxique : L'**analyse syntaxique** consiste à mettre en évidence la structure d'un texte, ici de l'XML.

ASPRO : The **A**stronomical **S**oftware to **PR**epare **O**bservations.

API : Application programming interface.

CVS (***C**oncurrent **V**ersions **S**ystem*) : système permettant la gestion de versions concurrentes.

ChangeListener : permet de savoir quand l'utilisateur glisse la barre d'un JSlider et change ainsi sa valeur.

DTD : indique les noms des éléments pouvant apparaître et leur contenu, c'est-à-dire les sous-éléments et les attributs.

DOM : Le **Document Object Model** (ou **DOM**) est une recommandation du W3C qui décrit une interface permettant à des programmes informatiques d'accéder ou de mettre à jour le contenu, la structure ou le style de documents.

Élément « root » : Première balise du document XML, mère de toute les autres.

Handler : permet de manipuler des objets (to handle = manipuler).

Itérateur : Un itérateur est un objet qui permet de parcourir tous les éléments contenus dans un autre objet, le plus souvent un conteneur.

Interface : Une interface est constituée d'un ensemble de déclarations de méthodes sans implantation.

JComboBox : permet de faire apparaître une liste déroulante lorsque l'utilisateur clique sur l'objet, ainsi que de faire une sélection.

JPanel : Container Swing pouvant contenir des widgets Swing et qui permet de les positionner dans la fenêtre (**JFrame**).

JSVGCanvas : Container défini par l'API Batik permettant de faire un render d'SVG à partir d'une URI ou d'un SVGDocument.

KeyListener : permet d'écouter les actions clavier de l'utilisateur lorsqu'il édite un composant donné.

Layout : Un layout Swing permet de décrire la disposition des éléments dans une interface en swing.

Liste d'API : http://java-source.net/open-source/xml-user-interface-toolkits.

Listeners : « écouteur » d'action de l'utilisateur permettant de capturer des événements.

Render : le but des **render** est d'avoir une image prête à intégrer dans un décor (ici une **JFrame**).

SVGDocument : Type particulier de document contenant du SVG et que Batik comprend.

Twiki : http://www-laog.obs.ujf-grenoble.fr/twiki/ (toute les sections ne sont pas accessibles de l'extérieur).

URI : courte chaîne de caractères identifiant une ressource sur un réseau.

XML : « langage extensible de balisage ») est un langage informatique de balisage *générique*. Il sert essentiellement à stocker/transférer des données de type texte.

Scalable Vector Graphics (SVG) : format de données conçu pour décrire des ensembles de graphiques vectoriels.

X Window System ou **X11** ou simplement **X** est une *interface utilisateur graphique* de type "fenêtré" qui gère l'interaction homme-machine par l'écran.

Gilles Duvert : Astronome - Directeur scientifique du LAOG.

XmlBuffer : classe utilitaire développée par Guillaume Mella pour l'ancienne version d'ASPRO.

XmlBasedGui : code développé par Guillaume Mella en 2003 permettant de dessiner des composants Swing à partir d'XML.

XmlBasedGui language : http://jmmc.fr/~mella/REF_XML/.

Webographie

LAOG : *www-**laog**.obs.ujf-grenoble.fr/*

JMMC : http://www.jmmc.fr/

Netbeans : *www.**netbeans**.org*

CookSwing : http://cookxml.yuanheng.org/cookswing/

SwiXml : http://www.swixml.org/

Batik : http://xmlgraphics.apache.org/Batik/

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

Java source (recherche de librairies): http://java-source.net/open-source/xml-user-interface-toolkits

Exemple dépôt (aide au débuggage): http://www.exampledepot.com/

Annexe

Test du parseur SAX

avec récupération d'événements

```
public static void parseXmlFile(String filename, DefaultHandler handler) {
    System.out.println("Parsing document:" + filename);
    try {
        // Create a builder factory
        SAXParserFactory factory = SAXParserFactory.newInstance();
        // Create the builder and parse the file
        SAXParser parser = factory.newSAXParser();
        parser.getXMLReader().setProperty(
                "http://xml.org/sax/properties/lexical-handler",
                handler);
        parser.parse(new File(filename), handler);
        System.out.println("-----------------------------------------------------");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
    // DefaultHandler contain no-op implementations for all SAX events
    static class MyHandler extends DefaultHandler implements LexicalHandler {

public void comment(char[] ch, int start, int length) throws SAXException {
    String data = new String(ch, start, length);
    System.out.print(data);
}
public void startElement(String uri, String localName, String qName, Attributes attrs) throws SAXException {
    System.out.print(localName);
}
public void endElement(String namespaceURI, String localName, String qName) throws SAXException {
    System.out.print(localName);
}
public void characters(char[] ch, int start, int length) throws SAXException {
    String data = new String(ch, start, length);
    System.out.print(data);
}
```

Création d'un itérateur

```
public static Iterator<Component> buildContainerIterator(Container container) {
    Collection<Component> collection = new ArrayList();
    traverse(container, collection);
    return collection.iterator();
}
```

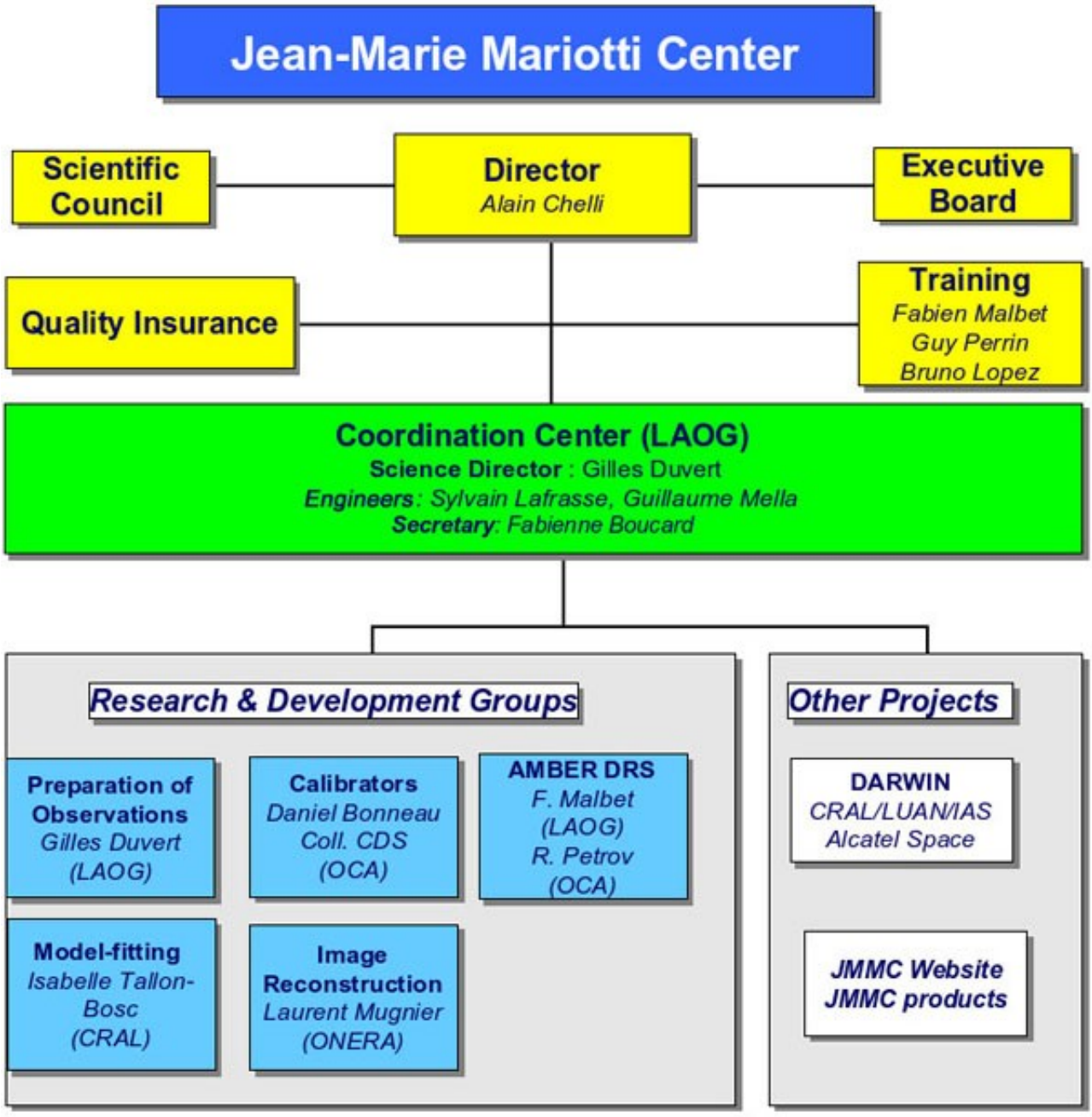
```
protected static void traverse(final Component c, Collection<Component> collection) {
    if (c != null) {
        collection.add(c);
        if (c instanceof JMenu) {
            final JMenu m = (JMenu) c;
            final int k = m.getItemCount();
            for (int i = 0; i < k; i++) {
                traverse(m.getItem(i), collection);
            }
        } else if (c instanceof Container) {
            final Component[] s = ((Container) c).getComponents();
            for (Component value : s) {
                traverse(value, collection);
            }
        }
    }
}
```

```
    <gui_desc title="TEST Panel" command="CONTINUE">
        <CHAIN label="A: reel normal" variable="A
    <SLIDER label="B entre 0 et 1:" variable="B
    with other lines
    11   ...
    12...   ...
    13   ...
    14"/>
    <SLIDER label="C entre -1 et 1, libre:" variable="C
    <CHOICE  label="II entier, choix indexe" variable="II
        <ITEM value="Prem"/>
        <ITEM value="Deuz"/>
        <ITEM value="Troiz"/>
        <ITEM value="Quatrez"/>
    </CHOICE>
    <CHAIN label="LL entier, Choix impossible (visu seule)" variable="JJ
    </CHAIN>
    <CHOICE  label="CA string, Choix normal" variable="CA
    </CHOICE>
3:  <CHOICE  label="CB string, Choix + Libre" variable="CB
    </CHOICE>
    <LOGIC  label="Y logique" variable="Y
    </LOGIC>
    <BUTTON command="SAY Button 1 Pressed" title="Bouton 1" showlength="0">
    </BUTTON>
```

Nouvelle syntaxe

```
<borderlayout>
    <constraint location="North">
        <menubar>
            <menu text="File">
                <menuitem text="menu 1" actionlistener="buttonAction" />
                <menusplit />
                <menuitem text="menu 5" actionlistener="buttonAction" />
            </menu>
            <checkboxmenuitem id="menu_2" text="menu 2" actionlistener="buttonAction" />
            <radiobuttonmenuitem id="menu_3" text="menu 3" actionlistener="buttonAction" />
            <checkboxmenuitem id="menu_4" text="menu 4" actionlistener="buttonAction" />
            <menu text="help">
                <menuitem text="About" actionlistener="aboutAction" />
            </menu>
        </menubar>
    </constraint>
    <constraint location="Center">
        <panel>
            <borderlayout>
                <constraint location="North">
                    <toolbar>
                        <button text="button 1" actionlistener="buttonAction" />
                        <checkbox id="button_2" text="button 2" actionlistener="buttonAction" />
                        <radiobutton id="button_3" text="button 3" actionlistener="buttonAction" />
                        <radiobutton id="button_4" text="button 4" actionlistener="buttonAction" />
                        <toolbar-separator separatorsize="50,1" />
                        <button text="about" actionlistener="aboutAction">
                            <icon setas="icon" ui="OptionPane.informationIcon" />
                        </button>
                    </toolbar>
                </constraint>
            </borderlayout>
        </panel>
    </constraint>
</borderlayout>
```

Organigramme du JMMC



Abstract

For my work placement, I have spent ten week in LAOG, the Laboratory of Observatory in Grenoble with Guillaume Mella. There are many kind of engineer and researcher in this Laboratory because you need mechanics, mathematical and astronomical capabilities to be able to have a look at some distant star and to understand the information you got from those observation. During my

training period many researcher explained to me their work and it was fascinating. My job was to develop an interface between a graphical client and a server to allow them to communicate with XML (*Extensible Markup Language*), and then to improve the look and feel of the client side. I used many tools I never had like CVS to help many people at working for the same project, rsync to save my work safely. The language I used was Java, I already had used this one for some projects during my study mainly during the second year. I also used Java to build an interface allowing user to send information to the server and to get graphical information in return. I already have done those kind of work but not in this way, so it was an important improvement for my IT culture. I had to write my software in English so that the moment to see that I can explain what I want in this language.