Rapport de Stage

Développement d'une interface logicielle

entre la SoundPalette

et le logiciel OpenMusic

Baptiste BUQLIER Master 2 ICHM	Responsable universitaire: René MANDIAU Responsable industriel: Jérome BARTHELEMY

Remerciements

En premier lieu, j'adresse mes plus vifs remerciements à Monsieur Hugues Vinet, directeur scientifique de l'IRCAM, pour m'avoir intégré dans l'une de ses équipes en qualité de stagiaire dans un domaine d'application en parfaite adéquation avec mes compétences.

Aussi, je tiens à remercier vivement Monsieur Jérôme Barthelemy pour m'avoir accueilli dans son équipe, et m'avoir proposé un stage dans un cadre aussi intéressant, et ce, bien au-delà de l'aspect purement informatique du sujet proposé. Je le remercie aussi pour sa cordialité et pour m'avoir soumis un projet qui par son originalité fut très riche d'enseignements et d'intérêts.

Je souhaite également remercier Monsieur Guillaume Boutard pour l'aide précieuse qu'il m'a apportée tout au long du stage, ainsi que pour son authentique et universel bon goût, riche de trésors subtilement dissimulés.

J'adresse aussi mes remerciements à Max Jacob, qui lors de ses passages à l'IRCAM, a toujours su se montrer disponible pour répondre à mes interrogations.

Parallèlement, je remercie les membres de l'équipe Représentations Musicales, et particulièrement Monsieur Gérard Assayag pour sa cordialité et ses suggestions toujours pertinentes, ainsi que Messieurs Carlos Agon Amado, et Jean Bresson pour avoir su m'orienter avec sympathie dans les dédales de LISP.

Enfin, je souhaite remercier toutes les personnes de l'IRCAM qui aussi bien directement qu'indirectement ont rendu ce stage si riche en découvertes multiples et avec qui j'ai pu partager ne serais-ce qu'une parcelle de ma passion.

Introduction

Le stage de fin d'études qui fait l'objet de ce rapport s'inscrit dans le cadre du Master 2 ICHM. Il était destiné à acquérir une expérience professionnelle par la mise en pratique des acquis dans les domaines abordés au cours de ma formation.

De façon plus personnelle, cette opportunité me donnait l'occasion de m'orienter vers un objectif professionnel précis. Passionné de musique, je souhaitais faire coïncider ma formation avec un domaine d'application lié à cette passion. J'ai alors orienté mes recherches dans ce sens, et c'est avec grand plaisir que j'ai accepté la proposition de Monsieur Jérôme Barthelemy, responsable de l'équipe Service en Ligne de l'IRCAM.

Le projet proposé visait à la conception et au développement d'une interface logicielle d'accès à une base de sons nommée *SoundPalette* depuis le logiciel *OpenMusic*.

Je me propose donc de présenter dans la suite de ce rapport les différents aspects relatifs à ce stage. Dans cette optique, nous aborderons dans un premier temps la présentation de l'IRCAM, nous nous focaliserons ensuite sur le contexte plus précis de mon stage, pour nous intéresser enfin au projet proprement dit et aux études menées pour le réaliser.

Table des matières

Remerciements 1

Introduction 4

Table des matières 5

I. Qu'est-ce que L'IRCAM 7

1. Recherche et Développement 7

2. Pédagogie 7

3. Création & Diffusion 8

4. Médiathèque 8

5. Relations Extérieures 8

6. Communication 9

II. Contexte du stage 9

1. L'équipe Service en Ligne 9

2. Le projet CUIDADO 10

3. Le projet Orchestration 10

4. L'équipe Représentation Musicale 11

III. Présentation du projet 11

1. Objectifs du projet 11

2. Environnement Logiciel 12

3. Compétences mises en oeuvre 12

IV. Etude préliminaire 13

1. Etat de l'existant – La *SoundPalette* 13

2. Recherche des solutions envisageables 14

3. Etude et Confrontation des solutions possibles 15

4. Conclusion de l'étude des solutions 18

V. Vers une solution logicielle 19

1. Définition d'une architecture adaptée 19

2. Définition des fonctionnalités de SP_OpenMusic 22

3. Définition d'un protocole de communication 22
   a. Format des échanges de LISP vers Java 23
   b. Format des échanges de Java vers LISP 23
   c. Protocole de communication 24

4. Principe des échanges par *Sockets* 25

Conclusion 27

Glossaire 28

Bibliographie 30

I. Qu'est-ce que L'IRCAM

L'Institut de Recherche et Coordination Acoustique/Musique fut fondé en 1969 par le compositeur et chef d'orchestre Pierre Boulez à la demande du Président Georges Pompidou. Dès sa création, l'institut visait à étendre les moyens d'expression de la musique par le développement et l'utilisation de nouvelles technologies.

L'institut ainsi créé fut associé au Centre Pompidou, et placé sous tutelle du ministère de la Culture et de la Communication. Il est actuellement dirigé par le philosophe Bernard Stiegler, et constitue un endroit unique au monde destiné à la recherche et la création musicale contemporaine.

L'étroite relation qui lie l'IRCAM et la création musicale s'exprime au travers de ses différents pôles d'activité. On peut ainsi distinguer les départements Recherche et Développement, Pédagogie, Création et diffusion, Médiathèque, Relations extérieures et Communication.

1. Recherche et Développement

Le département Recherche et Développement regroupe environ quatre-vingt-dix scientifiques de haut niveau placés sous la direction de Monsieur Hugues Vinet. Il rassemble différentes équipes dont les compétences recoupent des points de vue scientifiques variés. Les différents thèmes abordés sont répartis entre les équipes: acoustique instrumentale, acoustique des salles, design sonore, analyse-synthèse, représentations musicales, analyse des pratiques musicales, application temps réel, service en ligne, atelier mécanique. Les nombreuses collaborations tant à l'échelle nationale qu'internationale font de l'IRCAM un centre de recherche particulièrement reconnu pour ses compétences dans le domaine de la création musicale.

2. Pédagogie

Le département Pédagogie est orienté vers la transmission des savoirs et des technologies développés à l'IRCAM. Il vise aussi bien le monde professionnel et ses activités spécifiques tels que compositeur, chercheur, et designer sonore ; le monde universitaire au

travers des formations comme le DEA Acoustique Traitement du signal et Informatique Appliqués à la Musique, le DEA Musique Histoire et Sociétés, et le DESS Jeux Vidéos et Medias Interactifs ; ou le monde scolaire en s'adressant aux professeurs et élèves des lycées, collèges et conservatoires. Au-delà de ces activités d'enseignement, le département Pédagogie propose de nombreux stages et conférences destinés à des publics variés.

3. Création & Diffusion

La coordination entre les différentes activités de l'IRCAM, notamment celles liées à la Recherche et Développement et à la Pédagogie, est assurée par le département Création et Diffusion. Celui-ci est articulé autours des services *direction artistique* et *production*. Ce département englobe par ailleurs le *pôle de recherche sur les technologies pour le spectacle*, lequel s'appuie sur les compétences diverses de l'IRCAM afin de concrétiser des projets pluridisciplinaires qui concernent entre autres les domaines de la danse, du théâtre, et les installations nécessaires à la concrétisation des tels projets.

4. Médiathèque

Les activités de l'institut nécessitent la gestion d'un fonds documentaire et sonore. C'est donc dans un but d'actualisation et d'enrichissement de ce fonds, mais aussi pour rendre disponible en ligne son contenu qu'œuvre le département *Médiathèque*. Ses activités englobent par ailleurs le *bureau d'étude et méthodes* qui, en association avec le département des Relations Extérieures fait évoluer le site Web de l'IRCAM vers un système d'information intégré. Parallèlement à ces activités, le bureau *d'études et méthodes* se charge de la réalisation et du suivi technique des sites liés aux évènements initiés par l'IRCAM tels que le festival Agora, ou Résonances.

5. Relations Extérieures

Associé au département Communication, le département Relations Extérieures valorise les différents travaux de l'IRCAM. Son activité en relation avec l'industrie se concrétise par diverses publications reflétant les activités de l'IRCAM (publication d'articles et dépôt de brevets), et le montage de projets aussi bien nationaux qu'internationaux. Par ailleurs, cette entité assure, par l'intermédiaire du forum IRCAM, la diffusion des informations en direction des utilisateurs des logiciels développés et maintenus par l'IRCAM.

6. Communication

Le département Communication, en étroite collaboration avec les départements déjà cités, assure la gestion des divers aspects de la communication nécessaires au développement de l'institut. Cette activité englobe en particulier les relations avec la presse, la publication des diverses activités et représentations à venir, la communication interne, ainsi que les visites des locaux de l'IRCAM.

II. Contexte du stage

L'ensemble de mon stage s'est déroulé au sein du département Recherche et Développement de l'IRCAM, plus particulièrement dans l'équipe Service en Ligne. Aussi, la réalisation du sujet de ce stage s'est faite en collaboration avec l'équipe Représentation Musicale.

Afin de mieux comprendre les enjeux de ce projet, nous allons nous intéresser à ces deux équipes, ainsi qu'aux projets CUIDADO et Orchestration auxquels il est lié.

1. L'équipe Service en Ligne

L'équipe *Service en Ligne* est dirigée par Monsieur Jérôme Barthélémy responsable et chef de projet, en collaboration avec Messieurs Max Jacob et Guillaume Boutard, chargés de développement.

Les activités de l'équipe *Service en Ligne* sont orientées vers la conception et le développement de systèmes informatiques en ligne. Pour assurer sa mission, l'équipe SEL utilise l'ensembles des techniques du génie logiciel parmi lesquelles on peut distinguer: la conception de bases de données, la conception orientée objet, et la conception des interfaces homme-machine, ce qui cadre parfaitement avec les compétences acquises dans le cadre du Master 2 ICHM.

Depuis sa création en 2001, cette équipe, en collaboration avec d'autres services et partenaires, a mené à bien le développement des projets ECRINS et CUIDADO. Actuellement, ses activités sont tournées vers la réalisation du projet Semantic HIFI, la mise en place de nouveaux projets de développement, ainsi que la maintenance et l'évolution des éléments constitutifs de CUIDADO auquel prend part le projet réalisé au cours de mon stage.

2. Le projet CUIDADO

L'objectif de ce projet visait le stockage de nombreux échantillons sonores (près de 20000) ainsi que la génération automatique d'informations caractérisant ces échantillons appelés *descripteurs*. L'ensemble de ces éléments devait être disponible à des utilisateurs variés tels que chercheurs et compositeurs par l'intermédiaire d'outils de recherches adaptés à différents environnements logiciels. Ainsi ce projet rassemblait de nombreux domaines liés aux compétences de recherche et de développement de l'IRCAM.

Au-delà de l'atteinte des objectifs initiaux, la concrétisation du projet CUIDADO servira de base au projet Orchestration en matière de description de sons et d'apprentissage automatique.

3. Le projet Orchestration

Il s'agit d'un projet dont la mise en œuvre débutera cet été, et qui, sur la base du projet *CUIDADO*, rassemblera notamment les équipes Représentations Musicales et Analyse/Synthèse. Sont objectif vise à définir des méthodes pour assister les compositeurs dans les problèmes d'orchestration en les aidant à déterminer quelle combinaison d'instruments leur permettra d'aboutir à la texture sonore qu'ils désirent. Les méthodes employées seront basées sur l'exploitation de *descripteurs* sonores dont la *SoundPalette* assure la gestion. Cette exploitation se fera en partie à l'aide du logiciel *OpenMusic* développé par l'équipe Représentations Musicales, ce qui montre la nécessité d'un accès aux services offerts par la base de données développée par l'équipe Service En Ligne.

4. L'équipe Représentation Musicale

Cette équipe est menée par Monsieur Gérard Assayag (par ailleurs coordonnateur du DEA ATIAM), assisté dans le développement par Messieurs Carlos Agon Amado, et Jean Bresson. Elle est associée au CNRS par l'intermédiaire de Andreatta Moreno et Marc Chemiller, et accueille actuellement plusieurs doctorants.

L'objet d'étude de cette équipe concerne les paradigmes informatiques associés à la représentation symbolique des structures musicales tels que l'intelligence artificielle et l'apprentissage automatique. Les applications concernent aussi bien la Composition Assistée par Ordinateur que la musicologie computationnelle, et trouvent des applications au travers de l'implantation de modèles destinés aussi bien à la création qu'à l'analyse musicale. L'équipe est responsable du développement du logiciel *OpenMusic* (CAO), elle a notamment travaillé sur les projets européens *AGNULA* et *CUIDADO*. Son travail est actuellement orienté vers des sujets tels que la modélisation du style musical et de l'improvisation, ou l'extraction de structures musicales dans un contexte polyphonique, ainsi que le projet *Orchestration*.

III. Présentation du projet

1. Objectifs du projet

Le projet qui fut l'objet de mon stage s'inscrit dans la continuité du projet *CUIDADO* et vise à étendre les services proposés par la *SoundPalette* au logiciel *OpenMusic*. Aussi s'intègre-t-il comme élément indispensable à la mise en œuvre du projet *Orchestration* au travers du logiciel *OpenMusic*.

Afin de satisfaire à sa vocation, le programme à réaliser devait offrir des fonctionnalités similaires aux interfaces d'accès déjà réalisés pour les logiciels *MatLab* et *Max/MSP*, tout en s'adaptant aux exigences des langages nécessaires à son implantation dans *OpenMusic*.

L'objectif de ce projet était donc de réaliser une interface logicielle permettant de se connecter à la *SoundPalette* depuis le logiciel *OpenMusic*. Aussi, étant donné la différence des langages employés entre l'ensemble des librairies constitutives de la *SoundPalette* (Java) et *OpenMusic* (LISP), il était nécessaire de trouver un moyen de dialogue entre ces deux entités logicielles.

2. Environnement Logiciel

Les plateformes sur lesquelles le logiciel *OpenMusic* est implanté m'ont conduit à réaliser l'ensemble de mon travail en utilisant alternativement les systèmes d'exploitation Windows XP Professionnel, et Mac OS X.

En ce qui concerne le langage LISP, nous noterons l'utilisation d'environnements différents pour les systèmes Windows et Mac, pour des raisons liées à l'histoire du logiciel *OpenMusic*. Ainsi les environnement *Allegro Common LISP* (ACL) et *Digitool Mac Common LISP* (MCL) sont utilisés respectivement sous Windows XP et Mac OS X. Remarquons que ces deux environnements ont en commun les librairies relatives au *Common LISP* (CL), mais intègrent chacun des librairies différentes pour les même types de services, en particulier ceux relatifs à l'implémentation des *Sockets* et des *Streams*.

Pour l'édition des programmes Java, j'ai employé le logiciel Borland JBuilder 2005 Fondation. En ce qui concerne l'édition des programmes LISP, l'environnement d'édition utilisé dépend du système d'exploitation concerné. Ainsi, le développement a été réalisé avec l'environnement de développement intégré (IDE) Allegro sous Windows XP, et l'éditeur Fread de Digitool sous Mac OS X.

3. Compétences mises en oeuvre

En ce qui concerne les langages utilisés, j'ai pu mettre en application mes compétences en Java afin de m'adapter aux librairies *COSAL*. J'ai par ailleurs pu m'initier au langage LISP (LISt Processing) afin de proposer une librairie pour le logiciel *OpenMusic*, et ceci pour les deux systèmes d'exploitation visés.

Du point de vue des concepts mis en oeuvre, le projet réalisé m'a permis d'appliquer les connaissances acquises notamment cette année en matière de Conception Orientée Objet, Client/Serveur, Threads et Streams.

IV. Etude préliminaire

La première étape du projet a consisté en l'étude de l'architecture globale de la *SoundPalette*, pour ensuite aborder les moyens d'accès aux librairies nécessaires à la réalisation du projet depuis le logiciel *OpenMusic*. Cette démarche ma conduit à analyser les possibilités offertes par les environnements Allegro CL et Mac CL.

1. Etat de l'existant – La *SoundPalette*

La réalisation du projet *CUIDADO* a abouti à la création d'une base de données découpée en modules fonctionnels dédiés entre autres au stockage et à l'exploitation des échantillons (SOund FIle sErver) et des *descripteurs* numériques associés (MEtaData managEmEnt), mais aussi à la gestion des droits d'accès (Users MANagement). La gestion des données est intimement liée aux spécifications du schéma MPEG-7 (XML), et se fait par l'intermédiaire d'une base de données PostGre/SQL. Plusieurs couches logicielles Java (librairie COSAL) permettent l'accès aux données via une interface Web, une interface Java (Applet) ou d'autres logiciels tels que MatLab et Max/MSP. L'ensemble des éléments décrits constitue la *SoundPalette*.

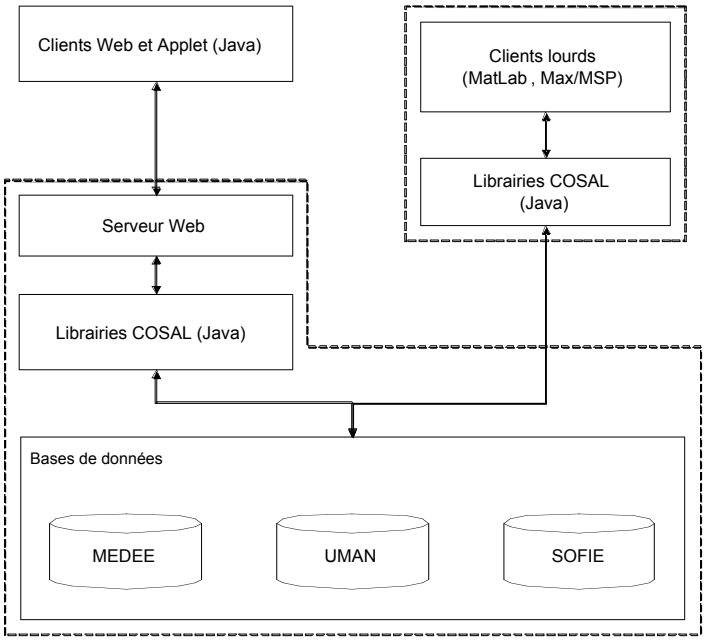

Figure 1 - architecture de la *SoundPalette*

En raison de l'architecture générale de la *SoundPalette*, le programme à réaliser pour *OpenMusic* devait offrir des fonctionnalités et un mode de fonctionnement similaires aux interfaces d'accès déjà réalisés pour les logiciels MatLab et Max/MSP, tout en s'adaptant aux contraintes du langage LISP.

Notons par ailleurs que le logiciel *OpenMusic* a été développé de manière à disposer d'une librairie générique pour les plateformes Windows et Mac OS X. Ainsi, les librairies Windows et Mac OS X disposent d'une même interface, ce qui implique l'emploi d'une démarche similaire dans le développement de SP_OpenMusic.

2. Recherche des solutions envisageables

La première solution envisagée consistait en la réécriture des fonctions d'accès de la librairie *COSAL* en *LISP*. Le logiciel *OpenMusic* aurait ainsi profité d'une librairie spécialement dédiée, tous les appels de fonctions entre *OpenMusic* et la *SoundPalette* s'effectuant en LISP.

Une seconde solution consistait à profiter des possibilités de communications existantes entre *LISP* et C d'une part, C et Java d'autre part. L'idée était donc d'utiliser le langage C comme intermédiaire entre les langages *LISP* et Java.

Une autre possibilité devait être étudiée, et consistait en l'élaboration d'un protocole de communication entre *LISP* et Java par l'utilisation de *Sockets*. Le principe de cette communication consistait à échanger des messages en utilisant le concept de Client/Serveur. Pour ce faire, un processus (serveur) se met à l'écoute d'un port et attend la connexion d'un autre processus (client) sur ce même port. Appliqué à ce projet, cette idée permet à un processus Java et un processus *LISP* de communiquer de façon bidirectionnelle en échangeant des messages sous forme de chaînes de caractères. La mise en pratique de cette solution consiste donc en l'écriture d'un serveur Java chargé de gérer les appels provenant d'un client *LISP*, de les transmettre au programme Java concerné, puis de retourner les résultats vers le client *LISP*. L'application du principe de communication par *Sockets* implique le choix d'un protocole de communication entre les protocoles *TCP* et *UDP*. Dans le cas qui nous intéresse nous opterons pour le protocole *TCP* qui est un protocole fiable en mode connecté. En effet, le manque de fiabilité dans la transmission des messages et le caractère non connecté du protocole *UDP* l'excluaient d'emblé de la solution envisagée.

Enfin, une autre solution m'est apparue en étudiant les outils mis à disposition par *ACL* (Windows). En effet cette implantation de *LISP* propose un outil: *Allegro JLinker*, dont l'objet est justement de permettre une communication bidirectionnelle entre Java et *LISP*. Malheureusement l'étude de *MCL* a montré qu'il n'existe aucun équivalent sous cet environnement.

3. Etude et Confrontation des solutions possibles

Bien entendu, étant donné la complexité des librairies de *COSAL* vis-à-vis de la durée du stage, il était totalement exclu de réécrire ne serait-ce que partiellement ces librairies en *LISP*. Par ailleurs cette solution aurait empêché les avantages d'une solution modulaire en cas de modifications des librairies de *COSAL*. En effet, en cas de mise à jour de ces librairies, le portage des modifications sur *OpenMusic* aurait entraîné une multiplication par deux du temps de développement, sans compter les adaptations nécessaires à l'implantation des mêmes fonctionnalités dans des langages aux paradigmes différents.

Malgré son intérêt en matière de modularité, l'utilisation du langage C comme intermédiaire semblait peu efficace. En effet, chaque appel unidirectionnel entre *LISP* et Java aurait nécessité deux transferts d'informations et autant de conversions pour les types de données. Sachant qu'un appel nécessite une réponse, l'aboutissement de chaque requête aurait imposé quatre transferts d'informations et trans-typages, ce qui aurait eu un impact conséquent sur la rapidité d'exécution, et m'est apparu comme un facteur décisif dans l'exclusion de cette possibilité. Aussi, sans multiplier le temps de développement par trois, il est indéniable que ce choix ait affecté la réalisation du projet dans les temps prévus.

L'utilisation du module *JLinker* d'Allegro m'a semblé particulièrement intéressante, notamment en matière de temps de développement et d'efficacité dans la communication inter langages. En effet, cet outil permet d'utiliser directement les objets et méthodes Java depuis l'environnement *LISP* et assure dans une certaine mesure la correspondance entre les types Java et *LISP*. S'agissant d'une fonctionnalité spécialement conçue pour ce type d'utilisation, son fonctionnement s'en trouve optimisé. Comme le montre l'illustration suivante, la mise en œuvre de cet outil nécessite l'écriture d'un code *LISP* faisant directement appel aux classes et méthodes Java. *JLinker* propose alors deux modèles de construction "class model" et "funcall model". Le premier est typiquement orienté objet et permet d'écrire des classes *LISP* donnant automatiquement accès aux classes Java visées via *LISP*. Le deuxième modèle consiste en des appels isolés de fonctions Java, tous les appels étant transmis automatiquement aux classes Java concernées.

Code Java ciblé	Code LISP associé

package monpackage
 	
public class MaClasse{
 	
public String monChamp;
 	
public MaClasse (String nom){…}
 	
public maMethode (String param){…}
 	
}
	;;;class model
	
	(def-java-class
	(ma-classe "monpackage.MaClasse")
	()
	((mon-champ "monChamp") options])
	()
	())
	
	(def-java-constructor make-ma-classe
	(ma-classe "java.lang.String"))
	
	(def-java-method
	(ma-methode "maMethode")
	(ma-classe "java.lang.String"))
	
	;;;funcall model (appels directs des methodes java)

	(jcall (jconstructor "monpackage.MaClasse" java.lang.String") "toto") (jcall (jmethod "monpackage.MaClasse" "maMethode" 1) instance_De_ma-classe)

Figure 2 - correspondance entre code Java et code LISP en utilisant *JLinker*

Par ailleurs, l'utilisation de cet outil permet une organisation modulaire des éléments de programmes, qui, comme nous l'avons vu précédemment permet une plus grande évolutivité.

Concernant le trans-typage entre les types de données Java et les types de données *LISP*, *JLinker* assure automatiquement la correspondance suivante:

LISP type ->	**Java type ->**	**Lisp type**
Nil	Null	Nil
Character	Char	Character
None	Byte	Integer
None	short	Integer
integer	Int	Integer
None	Long	Integer
None	float	double-float
single-float	double	double-float
double-float	double	double-float
None	boolean	t or nil
string	String	String
None	Byte[]	(array (signed-byte 32) *)
None	short[]	(array (signed-byte 32) *)
(array (signed-byte 32) (*))	int[]	(array (signed-byte 32) *)
None	float[]	(array double-float *)
(array (double-float 32) (*))	double[]	(array double-float *)
None	String[]	(array t *)
None	Object	tran-struct
symbol	TranStruct	Symbol
Lisp object	TranStruct	Lisp object

Figure 3 – conversions assurées par *JLinker* entre types Java et types LISP

En outre, l'absence d'équivalent dans l'environnement *MCL* laissait un certain doute quand à l'adoption de cette solution.

La solution mettant en œuvre un système de communication par *Sockets TCP* m'a semblé intéressante à plusieurs égards. Elle offrait la possibilité d'opter pour une structuration modulaire de l'ensemble de la librairie visée (Java + *LISP*), en permettant l'écriture en Java de la librairie d'accès aux fonctionnalités de *COSAL*. De ce fait elle offrait nombre d'avantages en matière d'évolutivité. En revanche, en considérant l'absence de compatibilité entre les librairies *Sockets* de *ACL* et *MCL*, cette solution imposait l'écriture de deux clients *LISP* différents, ce qui, ajouté à la programmation d'un serveur pour les transferts d'informations vers les librairies *COSAL*, pouvait affecter le temps de développement de façon difficilement prévisible. Néanmoins, l'efficacité d'une communication directe par *Sockets* semblait plus avantageuse que la solution impliquant l'utilisation du langage C comme intermédiaire entre Java et *LISP*.

4. Conclusion de l'étude des solutions

Comme nous l'avons vu précédemment, la solution proposant la réécriture complète ou même partielle des librairies COSAL n'aura pas été retenue. Il en est de même pour l'utilisation du langage C comme intermédiaire entre *LISP* et Java.

Contrairement à la solution employant l'outil *JLinker*, l'utilisation d'une communication par *Sockets TCP* semblait plus appropriée. En particulier grâce sa possibilité d'implantation sur les systèmes Windows et Mac, mais aussi pour la structuration modulaire des éléments logiciels qu'elle implique. Le seul inconvénient étant lié à la nécessité d'implanter un client sur chaque système d'exploitation d'après des bibliothèques différentes. Ce constat m'a conduit à étudier plus avant cette solution.

V. Vers une solution logicielle

Afin d'envisager l'implantation d'une solution adaptée au regard des critères de modularité, de temps de développement, et d'efficacité dans la communication entre langages, il était nécessaire de comparer les différentes architectures possibles.

Après avoir opté pour une communication par *Sockets TCP*, et en gardant à l'esprit les avantages offerts par l'utilisation de l'outil *Allegro JLinker* pour Windows, j'ai procédé au fractionnement du projet en modules fonctionnels distincts.

1. Définition d'une architecture adaptée

Pour répondre aux exigences du projet dans le cadre de la solution retenue, il était nécessaire de différencier plusieurs entités fonctionnelles. Ainsi, hors considération du système d'exploitation et de la localisation de ces entités, on obtient une solution comprenant:

- Un module Java d'accès aux services des librairies de *COSAL*
- Un module *LISP* offrant au logiciel *OpenMusic* les services du module Java
- Un module de communication entre les deux modules précédents

La localisation des modules peut être sujette à discussion. En effet, sachant que la base de données de la *SoundPalette* et le logiciel *OpenMusic* seront situés respectivement sur un serveur et sur une machine locale, plusieurs dispositions de ces modules entre machine locale et serveur étaient envisageables. Afin de correspondre au fonctionnement de la librairie *COSAL* (librairie locale dans le cadre de clients lourds) et des autres applications déjà développées par l'équipe SEL telles que SP_Matlab et SP_Max, j'ai opté pour la solution d'implantation des librairies que j'ai nommé SP_OpenMusic, illustrée par le schéma suivant:

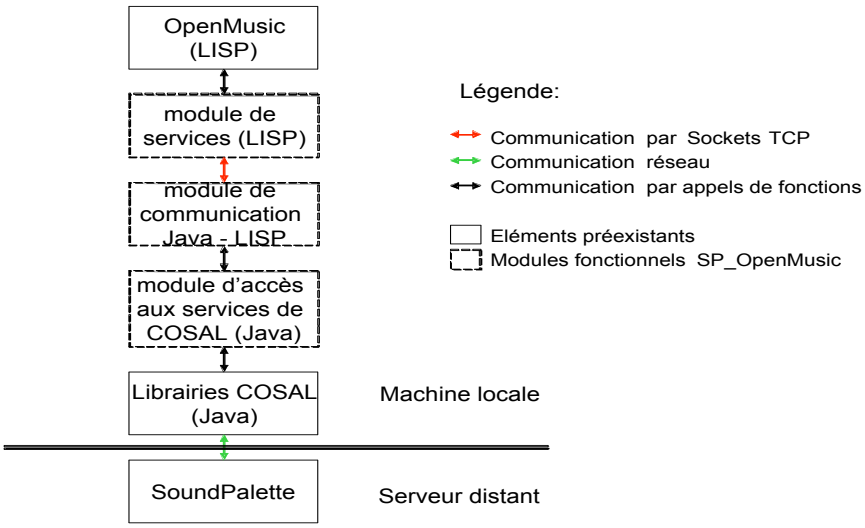

Figure 4 - architecture générale des modules de SP_OpenMusic

Ayant gardé à l'esprit l'intérêt de la solution proposée par l'outil *Allegro JLinker*, je me suis intéressé à l'architecture nécessaire à son utilisation, ainsi qu'à ses implications dans le cadre du projet en cours. J'ai alors pu en déduire l'équivalence suivante vis-à-vis des modules de SP_OpenMusic:

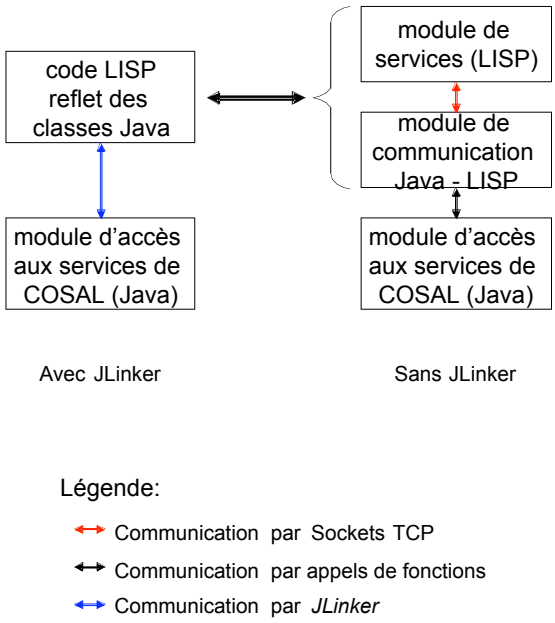

Figure 5 - Equivalence entre JLinker et communication par Sockets

Par conséquent, nous constatons la possibilité d'utiliser de façon équivalente l'outil *JLinker* dans la version Windows et une communication par *Sockets TCP* dans la version Mac.

La solution résultante gagne alors en temps de développement puisqu'elle substitue de simples classes *LISP* à l'implantation d'un client sous *ACL*. Ce qui évite l'apprentissage des librairies *Sockets* et *Streams* d'*ACL* et les erreurs de programmation consécutives à leur utilisation.

Cette démarche d'analyse a donc conduit à l'élaboration de l'architecture suivante:

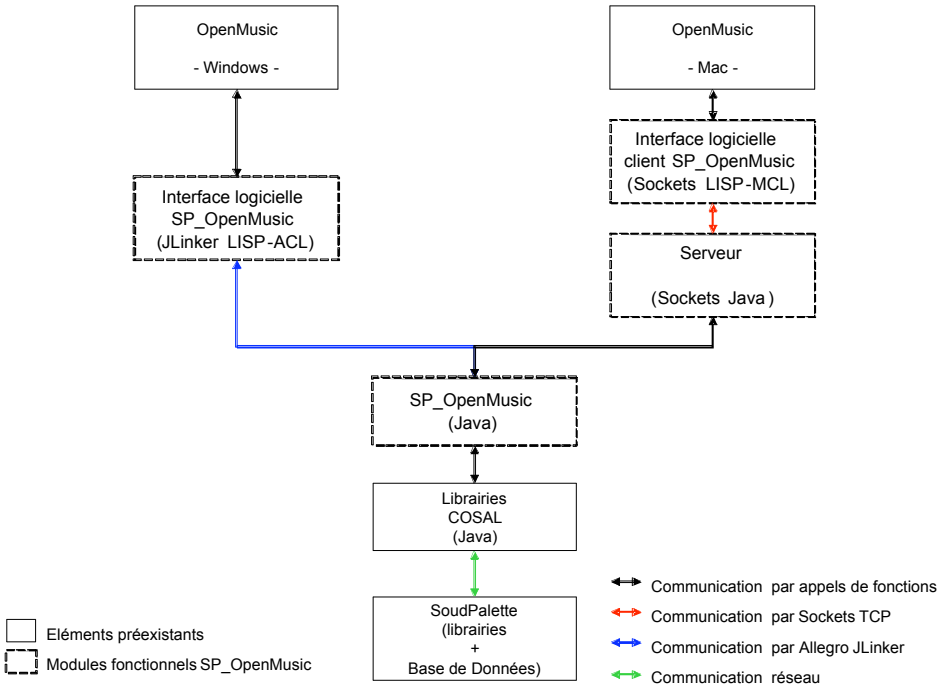

Figure 6 - Architecture globale de SP_OpenMusic (Windows - Mac)

Suite à cette étude, nous disposons d'une architecture adaptée aux spécifications définies par l'analyse du projet. Nous allons dorénavant nous intéresser aux divers aspects liés à l'implantation d'une telle architecture.

2. Définition des fonctionnalités de SP_OpenMusic

Afin d'offrir les services correspondants aux possibilités de la *SoundPalette*, une étude des fonctionnalités de *SP_MatLab* et *SP_Max* s'est avérée nécessaire. Ceci m'a permis de définir une interface logicielle pour *SP_OpenMusic* en accord avec les besoins exprimés pour le projet Orchestration. On peut alors distinguer les services suivants:

- Connexion et Déconnexion à la SoundPalette moyennant un login et un mot de passe.
- Listage des collections disponibles et des Uris associées
- Listage des instruments disponibles dans une collection
- Listage des libellés des *descripteurs* et de leurs sous-types
- Réception des libellés et des identifiants des samples correspondants à une requête
- Réception de la (des) valeur(s) d'un descripteur pour un sample donné
- Réception du chemin d'accès à un sample d'après son identifiant

L'établissement de ces fonctionnalités m'a permis par la suite de procéder à leur implémentation dans classe SP_OpenMusic du package fr.ircam.cuidado.spopenmusic. Différents tests m'ont permis de vérifier le bon fonctionnement de l'ensemble des méthodes écrites, ainsi que l'intégrité des résultats obtenus.

L'étape suivante a consisté en l'écriture des classes *LISP* utilisant l'outil *JLinker*, suivie des tests nécessaires aux vérifications des spécifications définies précédemment.

Une fois l'implantation de SP_OpenMusic sous Windows terminée, il était nécessaire de permettre la communication entre Java et *LISP* pour la version Macintosh.

3. Définition d'un protocole de communication

La problématique principale soulevée par les échanges entre *LISP* et Java concerne le format des messages dans ces échanges. Pour cela, il était nécessaire de distinguer d'une part les échanges de *LISP* vers Java, puis de Java vers *LISP*.

Une autre problématique soulevée par le concept de Client/Serveur inclus dans la partie Mac de l'architecture choisie, concerne la définition d'un protocole de communication entre client et serveur.

a. Format des échanges de LISP vers Java

Puisque *JLinker* gère automatiquement les appels de fonction Java, les échanges de *LISP* vers Java ne concernent que la partie Mac (*LISP MCL*). Pour cette communication par *Sockets*, chaque message sous forme de chaîne de caractères doit comporter un ou plusieurs éléments:

- Le nom de la fonction appelée (obligatoire)
- Un nombre variable de paramètres (de 0 à n)

Le nom de la fonction étant indispensable, il sera placé en début de chaîne. Il faut alors définir un séparateur pour différencier les éventuels paramètres qui suivent. La syntaxe des messages envoyés suivra donc le modèle suivant (pour plus de lisibilité nous utiliserons un "_" pour différencier les éléments de l'appel):

Nom-de-fonction_[Separateur_Paramètre-1]*

b. Format des échanges de Java vers LISP

L'examen des fonctionnalités requises pour SP_OpenMusic montre que les réponses attendues peuvent comporter plusieurs éléments, il était donc nécessaire de trouver une solution permettant de transmettre plusieurs éléments distincts de Java vers *LISP*.

En étudiant les caractéristiques de trans-typage induites par l'utilisation de *JLinker*, et en considérant l'utilisation de *MCL* dans la partie Mac de SP_OpenMusic, on constate la nécessité d'une convention commune aux deux implémentations concernant l'utilisation des types de retour des données en direction de LISP. En effet, d'après le tableau présenté à la page 15, le type "liste" très lié à l'utilisation de LISP n'a pas d'équivalents en Java, et aucun trans-typage ne permet à *JLinker* de retourner des listes à partir d'un type Java. En conséquence et pour des raisons d'homogénéité, il sera nécessaire de trouver un type de données facilement utilisable depuis les versions Mac et Windows.

En considérant les librairies relatives à l'utilisation des *Streams* liés aux *Sockets* sous *MCL*, on constate l'existence de la fonction "read" qui permet de convertir directement en objets *LISP* une chaîne syntaxiquement correcte. La problématique reposait donc sur le choix

d'un type de données correspondant aux possibilités de *JLinker*, une syntaxe appropriée des résultats permettant d'obtenir le même type de données sous *MCL*.

Devant la nécessité de retourner des résultats multiples, le type de données tableau de chaînes de caractères semblait le plus approprié. Cette solution ne posant aucun problème vis-à-vis de *JLinker* et *MCL*, l'échange de données se fera donc par tableaux de type String en se conformant à la norme suivante:

Indice 0	Indice 1	Indice 2	…	Indice n
Type de réponse	Taille du tableau	Elément 1	…	Elément n

Figure 7 - Norme de retour des valeurs

L'indice 0 est un entier sous forme de chaîne de caractère. Il permet de déterminer si la réponse retournée contient: un résultat (valeur "0"), un message d'erreur (valeur "-1"), ou un message d'information (valeur "1").

c. Protocole de communication

Afin de permettre la communication par *Sockets TCP* entre la partie LISP et la partie Java de Mac, il était nécessaire de définir un protocole de communication. La définition de ce protocole permet d'assurer une logique constante de fonctionnement entre les parties client (LISP) et serveur (Java).

Le schéma de communication établi est le suivant:

1. Serveur (Java): Démarrage
    a. ouverture d'un *Socket*
    b. mise à l'écoute du serveur sur ce *Socket*
2. Client (LISP): Démarrage
    a. ouverture d'un *Socket* client
    b. envoi du message "make-sp-openmusic" pour créer l'objet Java SP_OpenMusic ou "sp-shutdown" pour stopper le serveur.
    c. attente de la réponse du serveur (puis fermeture du *Socket* si l'envoi précédent était "sp-shutdown")
3. Serveur: Traitement du message par le serveur:

    a. Tout autre message que "make-sp-openmusic" ou "sp-shutdown" donnera lieu à un message d'erreur.
    b. Envoi de la réponse ou du message d'erreur
4. Client: envoi d'un appel de fonction et attente de la réponse
5. Serveur: traitement de l'appel de fonction et renvoi du résultat ou du message d'erreur
6. répétition des étapes 4 & 5 autant de fois que nécessaire
7. Client: Terminaison de la communication par l'envoi de "sp-shutdown"
8. Serveur: envoi de la réponse puis fin d'exécution de la Machine Virtuelle Java

4. Principe des échanges par *Sockets*

L'écriture des classes nécessaires à la communication entre un client LISP et le Serveur Java s'est donc faite d'après ce protocole. Une autre classe Java (SP_OpenMusic_ActionHandler) prend alors en charge le décodage des messages, les échanges en direction de la classe SP_OpenMusic, ainsi que le formatage des réponses conformément à la syntaxe LISP par l'intermédiaire de la méthode "transmit". L'ensemble des échanges de messages entre le client LISP et la classe SP_OpenMusic est schématisé par le diagramme de séquence ci-dessous. Notons que pour des raisons de simplification ce diagramme ne répond pas rigoureusement aux normes définies par UML.

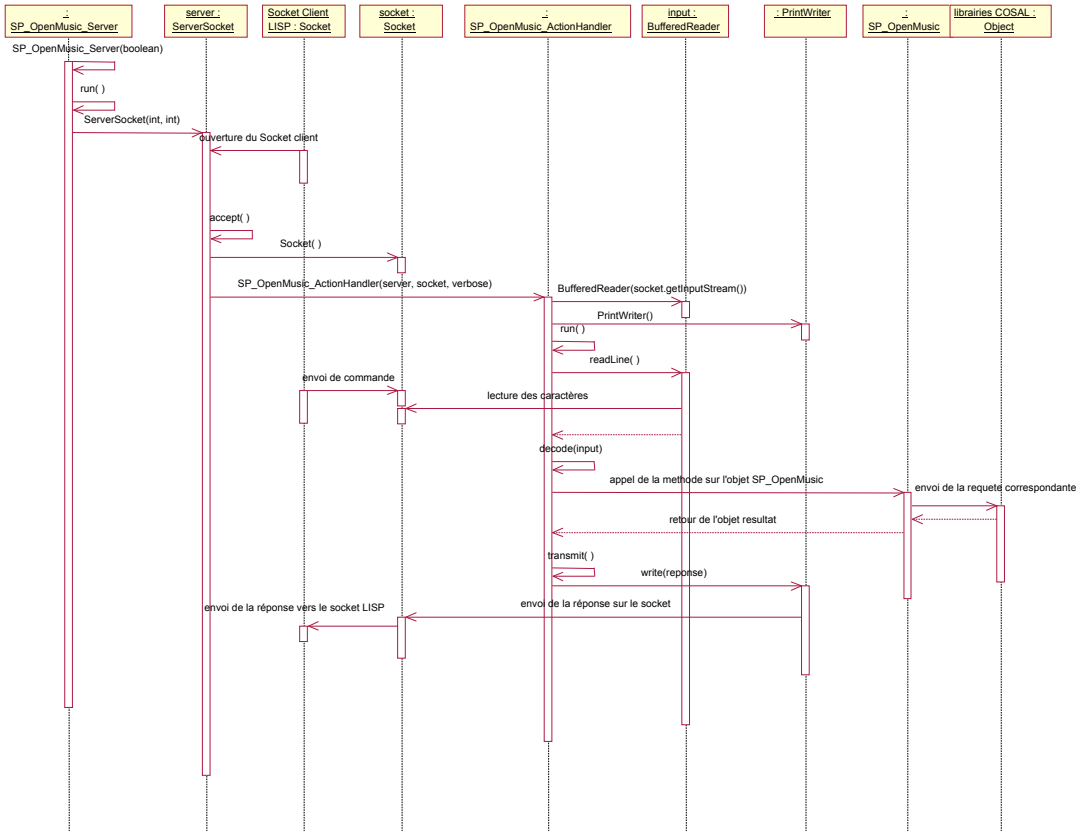

Figure 8 - diagramme représentatif des échanges entre LISP et les librairies COSAL

L'utilisation de *JLinker* sous Windows permettant l'appel des classes Java depuis LISP de façon transparente, le déroulement des appels s'effectue donc de façon similaire à l'exécution d'un programme exclusivement écrit en LISP.

Conclusion

Ces trois mois passés au sein de l'IRCAM ont été d'une grande richesse. En effet, ils m'ont permis de conforter mes acquis en informatique en étant confronté à une situation concrète de développement.

Le projet ainsi réalisé répond de manière relativement originale aux objectifs fixés, et saura, je l'espère, être apprécié des utilisateurs de OpenMusic. Malheureusement il n'a pu être complété par une interface graphique adaptée. En effet, le projet Orchestration ne débutant que peu de temps après la fin de mon stage le contexte d'utilisation n'a pu être définis.

D'un point de vue technique, mis à part son intérêt certain, en partie dû à son originalité, l'ensemble du projet accomplis a été une excellente occasion de mettre en œuvre mes connaissances, notamment en Conception Orientée Objet et en Java, ainsi qu'en Client/Serveur. Aussi, la réalisation de ce projet m'a sensibilisé à la programmation en LISP, et m'a apporté nombre de connaissances supplémentaires au cours de mes recherches sur ce langage.

La réalisation de ce projet m'a par ailleurs permis de me familiariser avec l'environnement Mac OS X, ce qui, en considérant son utilisation très répandue dans le milieu musical, constitue un avantage certain pour mon avenir professionnel.

D'autre part, ce stage a été pour moi l'occasion de découvrir le fonctionnement et les exigences liées au développement informatique dans le cadre d'un centre de Recherche et Développement. Ainsi cette expérience m'a-t-elle conforté dans ma volonté d'investissement futur dans ce contexte professionnel, si possible en rapport avec la musique.

Enfin, et d'un point de vue personnel, ce stage fut pour moi un moment très riche de découvertes, notamment musicales, en particulier au travers des différentes conférences et représentations auxquelles j'ai eu la chance d'assister, ainsi qu'aux nombreuses rencontres que j'ai eu l'occasion de faire.

Glossaire

Allegro Common LISP (ACL):

Il s'agit de la version de LISP implémentée par Franz Inc. Elle est utilisée pour l'implémentation du logiciel OpenMusic sous Windows.

Common LISP (CL):

LISP est un langage de programmation fonctionnel dont les nombreuses implémentations se basent sur des spécifications communes (Common LISP). Notons qu'il existe une extension objet de Common LISP nommée CLOS (Common LISP Object System).

COSAL:

Ce terme englobe l'ensemble des librairies Java offrant une interface d'accès aux données de la SoundPalette.

CUIDADO:

Nom du projet ayant abouti au développement de la SoundPalette.

Descripteurs:

Les descripteurs sont des informations destinées à caractériser les samples stockés dans la *SoundPalette*, ils sont le plus souvent générés de façon automatique.

JLinker:

JLinker est un outil proposé par Allegro Common LISP, dont le but est de permettre l'utilisation de classes Java depuis l'environnement ACL moyennant une syntaxe spécifique.

LISP:

Voir "Common LISP"

Mac Common LISP (MCL):

Désigne la version de LISP implémentée par Digitool Inc. pour les systèmes Macintosh.

OpenMusic:

OpenMusic constitue un environnement de programmation visuel pour la création d'applications de composition assistée par ordinateur. Il est disponible pour les systèmes Windows, Mac OS 9, et Mac OS X.

Orchestration:

Projet dont la réalisation débutera cet été. Il sera destiné à l'aide à l'orchestration.

Sockets:

Les sockets constituent les extrémités d'une connexion entre processus, et permettent à deux processus de communiquer entre eux.

SoundPalette:

Le terme SoundPalette défini l'ensemble des éléments de l'interface d'accès aux contenus sonores. Elle est issue entre autres du projet CUIDADO.

Streams:

Ce terme désigne les flux de données associés notamment aux Sockets.

TCP (Transmission Control Protocol):

Il s'agit d'un protocole fiable de communication en mode connecté.

UDP (User Datagram Protocol):

Protocole de communication non fiable en mode non connecté.

Bibliographie

Documents issus d'internet:

- Introduction à la programmation en Common LISP (Francis Leboutte)
- On LISP (Paul Graham)
- Basic Lisp Techniques (David J. Cooper, Jr.)
- ANSI Common LISP - Paul Graham, Prentice Hall

Ouvrages:

- Common LISP THE LANGUAGE (2nd edition) - Guy L. Stelle jr.

Autres:

- Documentation d'Allegro Common LISP (Franz Inc.)