Plugins audio multi-plateformes multi-standards

Rapport de projet de fin d'étude, INSA de Lyon, Génie électrique

—

IRCAM – Centre Pompidou

1, place Igor Stravinsky, 75004 Paris

Vincent Goudard et Rémy Muller

Vincent.Goudard@ircam.fr – vincent.goudard@insa-lyon.fr

Remy.Muller@ircam.fr – remy.muller@insa-lyon.fr

17 mars 2003 – 12 septembre 2003

Abstract

Ce document rend compte du travail effectué dans le cadre du projet de fin d'étude mené à l'IRCAM et intitulé : "Création d'un environnement de développement multi-plateformes multi-standards de plugins audio".

Nous remercions Norbert Schnell, notre maitre de stage qui nous a accueilli au sein de l'équipe "Applications Temps Réel" guidé et conseillé tout au long de ce projet, ainsi que François Hincker, notre tuteur à l'INSA. Nous remercions également l'équipe "Systèmes" qui nous a prêté sa mezzanine, nos voisins Thomas Pellegrini et Raphaël Duee, ainsi que toutes les personnes qui ont de près ou de loin contribué au bon déroulement de ce projet.

Contents

Chapter 1

Dossier de préparation

1.1 Situation du sujet

L'IRCAM (Institut de Recherche et Coordination Acoustique et Musique) a été crée en 1970 par Pierre Boulez autour de la complémentarité entre la recherche scientifique pour la musique et les artistes. Fort de ces années de recherche, de nombreux algorithmes d'analyse, synthèse et traitement musicaux ont été mis au point (synthèse de la voix chantée, modélisation physique d'instruments, suivi du fondamental et des partiels d'un son... etc).

Norbert Schnell:

> *Chaque algorithme temps-réel développé à l'IRCAM pourrait être implémenté pour divers logiciels musicaux supportant les plugins*[1]*. Et chaque logiciel représente un certain nombre d'utilisateurs. Par ailleurs, sur un marché aussi restreint que celui de l'audio temps-réel professionnel chaque utilisateur compte énormément et n'est pas à négliger. Le coût de développement lié au portage de plugins d'une plateforme ou d'un standard à un(e) autre est loin d'être négligeable en dépit des similitudes existant entre ceux-ci.*

1.2 Travail demandé

- Recenser tous les standards de plugins audio-numériques.
- Etablir un comparatif de ces standards.
- Créer un environnement de développement (multi-standards, multiplateformes) unifié.

[1] algorithmes externes permettant l'extension des fonctionnalités d'une application

1.3 Contexte

Ces 20 dernières années, l'informatique a connu un essor tel que de nombreuses applications musicales préalablement trop coûteuses du point de vu du temps de calcul ont pu voir le jour. On parle désormais de la notion de studio virtuel. Les nombreux travaux de recherche sur la modélisation des instruments, la synthèse...etc ont pu être implémenté en temps-réel et ont ouvert par la même occasion de nouvelles perspectives aux compositeurs, musiciens et ingénieurs du son. La plupart des applications musicales se sont dotées d'une architecture modulaire articulé autour des plugins permettant ainsi à différentes entreprises de fournir des algorithmes musicaux de très haute qualité.

L'IRCAM a toujours depuis sa création eté un moteur de cette révolution mais n'a pas encore décliné ses travaux de recherche sous la forme de plugins. Par ailleurs, ces dernières années, de nombreux standards ont vu le jour et il parait nécessaire de mener une étude afin dégager les formats les plus importants ainsi que les similitudes existants entre ceux-ci dans le but de pouvoir développer rapidement vers différentes architectures.

Voici la traduction d'un texte éménant de la MMA (Midi Manufacturer Association) résumant bien la situation actuelle:

> *Le marché audio professionnel offre une grande variété de standards concernant les plugins audio, pour certains matériels, pour d'autres logiciels et pour la plupart incompatibles entre eux. Citons VST/VSTi (Steinberg), DirectX (Microsoft), Audio units (apple), DXi (Cakewalk), Jack (Linux), LADSPA (linux), MAS (Motu), MFX (Cakewalk), RTAS (digidesign), TDM (digidesign), ReWire (propellerheads).*
>
> *Par abus de langage on appelle tous ces formats de plugins des standards alors que ce sont pour la plupart des formats propriétaires pour lesquels les entreprises responsables de leur développement doivent assurer un lourd suivi et une documentation vis-à-vis des développeurs. De plus, à la différence de réels standards tels que le MIDI, ces formats ne permettent pas une interopérabilité entre différents vendeurs, au contraire, ils fragmentent le marché en tribus adeptes de tel ou tel format.*
>
> *Le grand nombre de formats en competition met les developeurs de plugins audio face à un choix difficile: développer pour chaque format au prix de nombreuses heures de travail ou prendre le risque de se restreindre à une poignée de formats. Les développeurs d'applications hôtes se retrouvent confrontés au même dilemme lorsqu'ils doivent choisir les formats qui seront supportés par leur application.*

1.4 Objectifs – Cahier des charges

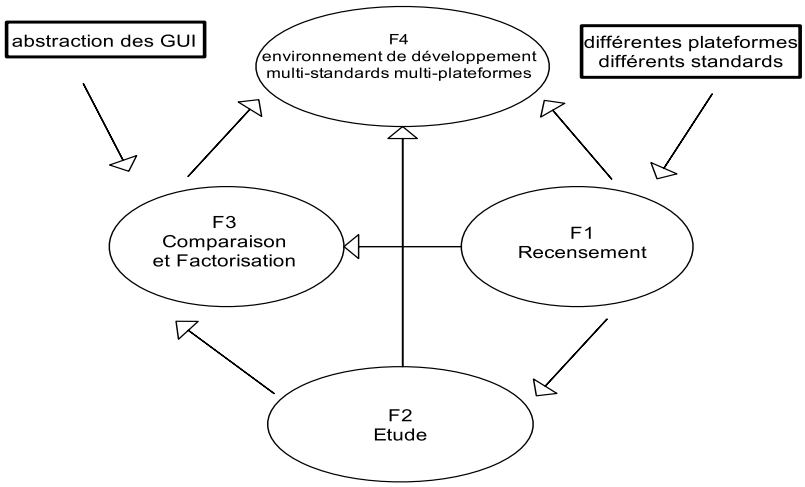

Figure 1.1: Découpage fonctionnel

Le découpage fonctionnel fait apparaitre 4 fonctions principales:

1. Le recensement d'un maximum de standards de plugins audio:
   Archivage de la documentation, des SDKs, des logiciels hôtes les plus connus.

2. L'étude des différents standards de plugins:
   Il s'agit de comprendre comment le plugin fonctionne dans son environnement, comprendre quelles formes ont les données musicales d'entrée et de sortie, les données de contrôle, quelle forme prend la communication avec l'application hôte.

3. Comparaison et factorisation
   Il s'agit de réaliser un bilan comparatif sur des critères fonctionnels précis, déjà en partie définis et qui se préciseront au cours de l'étude.
   La factorisation, basée sur ce bilan comparatif, consistera à considérer ce qu'il est possible/ impossible, de mettre en commun pour des plateformes et standards différents, et le cas échéant, définir des "familles" de standards pour lesquels la factorisation est possible.

4. Definition d'un framework unifié
   L'idée est encore assez vague, même si le but final est clair : minimiser le temps et le coût de développement des plugins audio. Des idées sont déjà là : SDK multi-standard multiplateforme, plugin templates, wrapper(s)...

Fonction	Critère d'évaluation	flexibilité
F1	Documents reccueillis	pertinence exhaustivité
F2	fiches de spécification choix des critères d'étude	intéret fonctionnel pertinence
F3	Document(s) comparatif(s)	Utilité ergonomie esprit de synthèse
F4	Templates? API[2]? commune Wrappers[3]? Tutorials?	simplicité d'utilisation rapidité de développement faible coût en CPU
C1	Nombre de plateformes étudiées	le maximum
C2	Nombre de standards étudiés	le maximum

Table 1.1: Cahier des charges

1.5 Contraintes

1. trois environnements différents, dans leur ordre décroissant de priorité:
   - Mac OSX
   - Linux
   - Windows
2. Standards à envisager:
   AudioUnits, Vst, DirectX, MAS, RTAS, LADSPA, jMAX, MAX, PureData, TDM, EyesWeb, Buzz, Virtools.
3. Langage de programmation: C/C++
4. Pas de prise en considération des interfaces graphiques (GUI)

1.6 Organisation de l'équipe

Après tirage à pile ou face, nous avons officiellement proclamé à l'unanimité:

- Rémy Muller: co-chef de projet
- Vincent Goudard: co-chef de projet

Nous nous reservons tous droits quant à la modification de ces statuts.

Chapter 2

Organisation - Méthode

2.1 Planning - diagramme de Gantt

Nous avons découpé notre projet en deux phases principales:

- **Etude comparative**: Cette étude s'est déroulée en trois temps:
  - *La recherche documentaire:* Il s'agit de recenser tous les standards de plugin audio existants, et obtenir leur SDK[1].
  - *Les fiches comparatives:* Nous avons étudié l'implémentation de ces standards, leurs points communs, leurs différences, les compatibilités et incompatibilités... Cette comparaison s'est appuyée sur des fiches descriptives (voir Annexes).
  - *Le rapport "Architecture des plugins" [Goudard and Müller(2003c)]:* Enfin, nous avons rédigé un document synthétique récapitulant l'étude comparative, et dégageant la structure commune sur laquelle envisager une abstraction pour le développement.
- **Développement**: Le développement s'est déroulé en trois temps:
  - *La modélisation* consiste, à partir de l'étude précédemment menée, à modéliser et conçevoir une manière de décrire et d'implémenter des plugins audio, de manière unifiée et indépendante du standard et de la plateforme. Cela implique le choix des outils logiciels, ainsi que des langages à utiliser. Nous avons également réfléchi à la méthodologie à adopter pour travailler à deux (utilisation de CVS).
  - *L'implémentation* de l'environnement modélisé a été faite standard par standard: d'abord LADSPA et VST, puis PureData et AudioUnits.

[1] Software Development Kit

– *La documentation* En plus du rapport à rédiger pour l'INSA, une documentation était nécéssaire pour les utilisateurs de l'environnement développé, ainsi que pour les développeurs qui poursuivront éventuellement notre travail.

Le diagramme de Gantt (figure 2.1) montre comment nous avons réparti ces tâches sur la duréee de notre projet.

2.2 Découpage technique

La répartition du travail s'est largement basée sur le choix des standards. Vincent a travaillé sur un PC avec Windows et Linux et s'est tourné vers les standards utilisés sur ces plateformes; Rémy a travaillé sur un G4 avec Mac OSX et s'est donc davantage tourné vers les standards utilisés sous Mac.
Il nous est cependant beaucoup arrivé de travailler à deux sur la même chose, car notre but d'établir un rapport synthétique, puis un environnement de développement unifié exigeait que nous recoupions nos informations.

La figure 2.2 montre comment nous nous sommes réparti le travail.

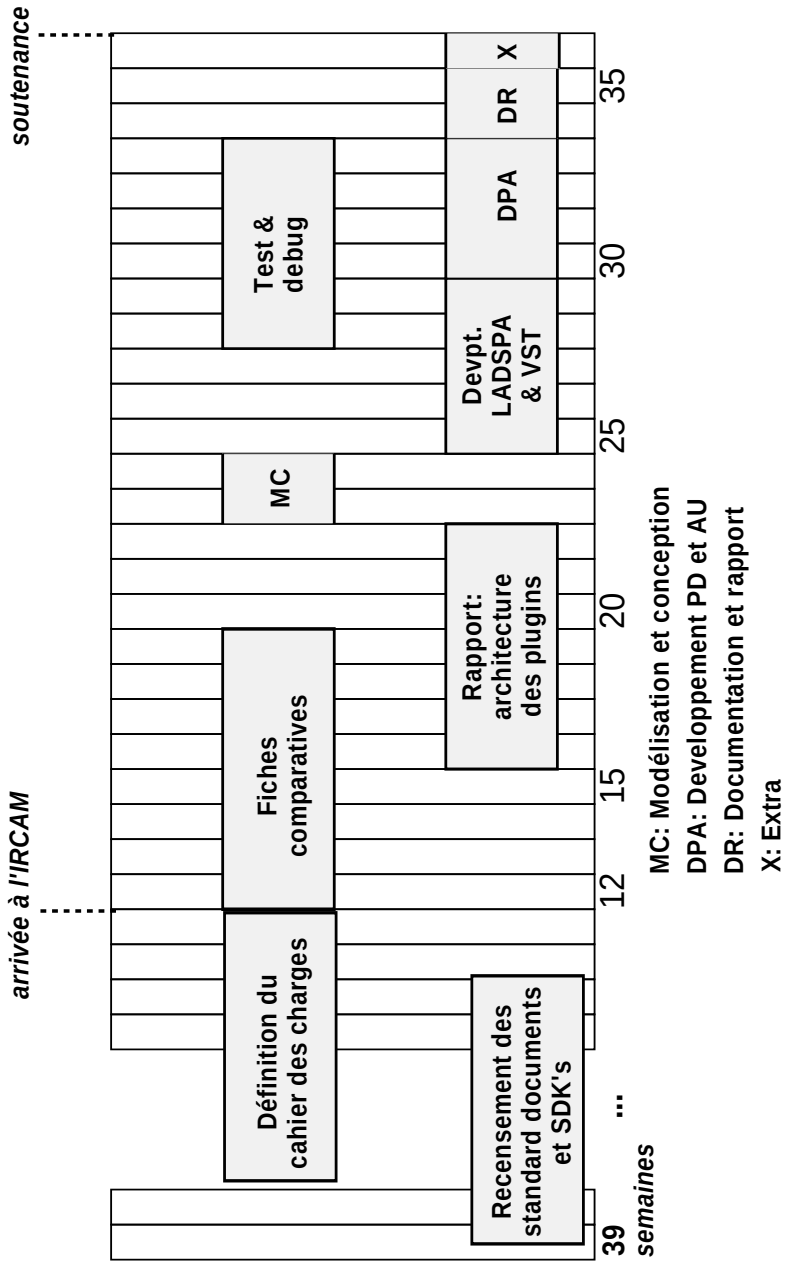

Figure 2.1: Phases du projet: Diagramme de Gantt

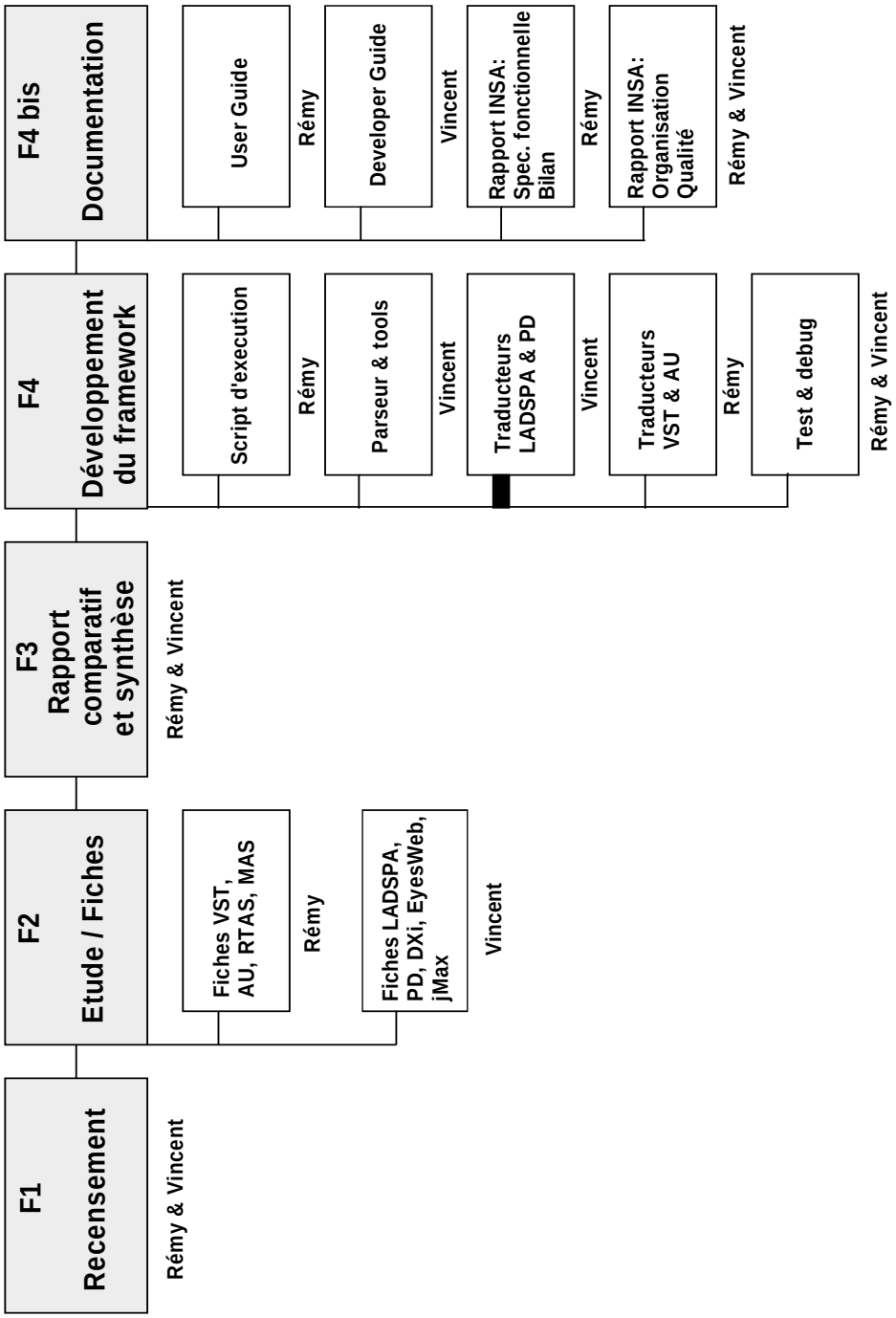

Figure 2.2: Découpage technique

Chapter 3

Qualité

3.1 Choix et méthode pour l'étude comparative

A notre arrivée à l'IRCAM, nous avons commencé par *rencontrer les personnes de l'IRCAM* développant des algorithmes de traitement du signal audio, mais également les personnes de la production, qui en sont les potentiels utilisateurs, afin de savoir quels étaient leurs besoins, leurs désirs, et ce que notre travail pouvait leur apporter.
Ce travail de prospection a également été largement nourris par les rencontres "virtuelles" de l'internet, dans les nombreux *forums de discussion de développeurs et utilisateurs* de logiciels audio. Cette prospection s'est en fait poursuivie tout au long de notre projet, notamment en suivant la discussion actuelle sur *GMPI*[1], un projet pour la création d'un véritable standard de plugin lancé par la MMA[2].

Nous avons commencé notre étude par une comparaison entre les différents standards sur la base de critères définis dans des fiches. Nous avons produits ces *fiches au format HTML* afin de faciliter leur diffusion sur internet; cela nous a permis de bénéficier de commentaires et de corrections de la part d'internautes intéressés par le sujet.

Nous avons ensuite rédigé un rapport de synthèse en *LaTeX* sur cette étude préalable. En effet, ce language permet de générer une documentation claire au format Portable Document Format (pdf), PostScript(ps) ou Hyper Text Markup Language (html) conforme aux règles de typographie pour la publication, et facilement diffusable sur l'internet. Dans un même soucis de diffusion de ce document, nous avons choisi de rédiger ce rapport *en anglais*.

[1] Generalized Musical Plugin Interface
[2] MIDI Manufacturer Association

3.2 Choix pour la réalisation logicielle

Suite à l'étude menée durant les trois premiers mois sur l'état de l'art en matière de plugins audio, nous avons du faire des choix concernant la réalisation de l'environnement. Ces choix concernaient:

- La modélisation et les structures de données à adopter.
- Les outils logiciels pour le développement de l'environnement.
- Les outils logiciels pour la documentation.
- Les standards de plugins visés.

3.2.1 Modélisation

L'étude menée durant la remière partie du projet [Goudard and Müller(2003c)] nous a conduit à déduire un modèle faisant abstraction du standard et de la plateforme.

3.2.2 Outils logiciels pour le développment

XML et C/C++

Nous avons choisi d'implémenter notre modélisation dans une syntaxe XML [Walsh(1998)], qui est un langage à balises dont le HTML est une variante. Le choix du langage XML a été motivé par:

- La souplesse: XML est un langage permettant de manier des données structurées, qui convient à notre objectif de modélisation. Il est d'ailleurs couramment employé pour enregistrer des "patchs" dans les logiciels modulaires.
- La popularité: XML est un langage de plus en plus utilisé à cause du besoin grandissant de structuration de données sur l'internet et dans les applications, nous laissant la garantie qu'il ne deviendra pas obsolète dans un futur proche.
- La simplicité: Les documents XML sont très simple à écrire et peuvent éventuellement être construits dans un environnement graphique.

Python

Le parseur assurant la traduction depuis le document XML vers les sources en C/C++ est écrit en Python[van Rossum and col.(2003)]. Le choix d'utiliser Python a été motivé par:

- *Sa simplicité:* Python, de façon similaire à java, est un langage interprété qui permet de programmer aussi bien des scripts que des modules. Il est beaucoup plus intuitif et haut niveau que les langages C/C++, d'où sa simplicité d'utilisation. De plus, le fait que le language soir interprété et non compilé accélère considérablement le temps de développement.

- *Sa license:* Python est un logiciel Open Source – à la différence de Java – et ne pose pas de problème pour que nous puissions diffuser notre logiciel sous une license libre.

- *Sa popularité:* Python bénéficie d'une importante communauté d'utilisateurs, qui l'enrichissent de nouveaux modules, et qui permet de facilement trouver de l'aide quand on en a besoin. Nous avons ainsi pu nous sevrir directement de PyXML, une librairie dédiée au traitement de documents XML dans Python.

CVS

Etant donné que nous travaillons à deux, et malgré la séparation des tâches que nous nous sommes fixés, il nous arrive de devoir travailler sur les mêmes documents en même temps.
CVS: "Concurrent Version System" [Fogel and Bar(2000)] est un outil efficace permettant de gérer les problèmes de version de documents, dans le cadre de travail de groupe sur des documents informatiques.

3.2.3 Outils pour la documentation

De même que pour l'étude comparative, nous avons utilisé LaTeXpour rédiger la documentation de notre réalisation, consistant en un "guide du développeur" [Goudard and Müller(2003a)] pour ceux qui vont améliorer cet environnement de développement, et un "guide de l'utilisateur" [Goudard and Müller(2003b)] destiné à ceux qui vont utiliser l'environnement que nous avons crée. Ces deux documents ont également été rédigés *en anglais* pour leur diffusion.

3.2.4 Standards visés

Nous avons limité notre travail à un nombre restreint de standards de plugins, car le temps imparti pour notre projet ne nous permettait pas de balayer l'ensemble des standards existants. Le choix des standards dépendait de différents facteurs:

- *La disponibilité du SDK*[3] *et de la documentation* Il nous a été difficile d'obtenir de la documentation pour certains standards. Cela est dû au

[3]SDK: Software Development Kit

fait que tous les standards de plugins ne sont pas ouverts au public. Ainsi, nous avons du laisser de coté les standards RTAS et TDM de Digidesign.

- *Le public du standard*: Tous les standards n'ont pas le même succès et les mêmes utilisateurs. Ainsi, les standards OPT, et Buzz sont moins couramment utilisés et ont été laissés de coté.

- *La complexité de l'implémentation*: Certains standards comme MAS de MOTU, ou DXi de Cakewalk étaitent plus complexes et ont été écartés pour le moment. Ils ne fournissaient notamment pas d'interface graphique générique; hors le cahier des charges avait exclu les interfaces graphiques du champ de notre travail.

Ainsi, nous avons implémenté les quatre standards suivants:

- *VST:* est le standard le plus répandu et le plus populaire. Il est relativement simple et multi-plateformes, fonctionnant sous Windows, Mac OSX, BeOS, Irix.

- *LADSPA:* est le standard sous Linux. Il a été pensé pour être très simple.

- *Audio Unit:* est le standard sous Mac OSX. C'est le standard le plus récent, et l'utilisation des Mac est très courante en performance live, et en général dans le domaine musical.

- *Pure Data:* est un logiciel modulaire de la même famille que les logiciels MAX/MSP et jMax, crées à l'IRCAM. Nous avons choisi Pure Data, car notre tuteur, Norbert Schnell, dévelopant des objets pour MAX/MSP et jMax, pouvait aisément implémenter ces standards à partir de ce qui a été fait pour Pure Data. Pure Data est aussi multi-plateformes, fonctionnant sous Linux, Windows, Mac OSX, et Irix.

Chapter 4

Bilan

4.1 Moyens

Notre projet étant exclusivement basé sur du développement logiciel, les moyens nécessaires à la réalisation de ce projet n'ont pas été très importants.

4.1.1 Moyens humains

Notre PFE s'est déroulé à plein temps à partir du 15 mars jusqu'au 12 septembre. Le décompte des heures est récapitulé dans le tableau 4.1.

4.1.2 Moyens Matériels

- un PC portable windows / linux
- un power Mac G4 – Mac OSX
- connection internet / intranet haut-débit
- accès à la médiathèque de l'IRCAM

Periode	heures/semaine	nombre de semaines	nombre d'heures
30/09 – 14/03	2	15	30
10/02 – 17/03	8	4	32
17/03 – 12/09	35	23	805
TOTAL		42	867

Table 4.1: Nombre d'heures de travail

4.1.3 Moyens logiciels

Pour le développpement:

- Emacs (un éditeur de texte très performant connaissant la syntaxe de Python, C, C++, XML et bien d'autres encore)
- Python et le package PyXML
- LaTeX
- SDK's: VST, LADSPA, AudioUnit, PureData, DXi, EyesWeb, MAS, jMax, MAX
- IDE[1] sous OSX: Project Builder
- Compilateur sous Linux: cc et gcc
- IDE sous Windows: Visual C++

Pour les tests sous OSX:

- Bidule
- Rax
- SynthTest
- Metro
- Digital Perfomer

Pour les tests sous Linux:

- Audacity
- PureData
- AlsaModularSynth
- jMax

Pour les tests sous Windows:

- PureData

[1]Integrated Developement Environement

4.1.4 Moyens Financiers

Indemnité de stage de 152,45 euros par personne pour 2 personnes pendant 6 mois, soit 1829,4 euros.

4.2 Résultats

4.2.1 Etude comparative

Durant l'étude comparative, nous avons rédigé des fiches pour chaque standard, qui nous ont permis de les examiner et des les comparer suivant les critères que nous nous étions fixés[2]. Ces fiches seront rendues publiques, et pourront servir à quiconque souhaite avoir un aperçu synthétique des possibilités qu'offre chaque standard afin d'en cibler certains plus que d'autres.
Il s'en est suivi un document[3] visant à faire une synthèse de ces différents formats de plugins, afin d'en extraire les fonctions principales et récurrentes ainsi que les différences fondamentales. Ce document sera également rendu public sur le serveur de l'IRCAM, en tant que resource à la fois interne et externe pour quiconque souhaite aborder le sujet des "plugins audio".

4.2.2 Environnement de dévelopement XSPIF

Nous avons créé un environnement pour faciliter le développement rapide de différents formats de plugins à partir d'une même description: un *meta-plugin*.

A titre indicatif, nous avons mentionné le nombres de lignes de code nécessaires pour décrire un plugin dans chaque format ainsi que le nombre de lignes nécessaire pour le décrire grace à notre environnement. (cf. Table 4.2)
L'example cité ici est *lowpass.xspif* un plugin passe-bas du premier ordre avec un seul paramètre. Le gain moyen en nombre de lignes est approximativement de 464%.
Cette implémentation synthétique permet également de s'affranchir des problèmes de version d'un standard: le meta-plugin restera identique, et il faudra uniquement mettre le traducteur à jour. Ceci présente naturellement un intéret quand le nombre de plugins à mettre à jour est important.

Le meta-plugin permet également à qui veut comprendre la topologie du plugin et son fonctionnement, d'éviter d'avoir à "fouiller" dans les lignes de code: le meta-plugin est une version épurée, qui cache tout le code relatif à l'API de

[2] cf Annexes
[3] [Goudard and Müller(2003c)]

l'application hôte, ou à la GUI[4].

Cet outil permet un prototypage d'algorithmes de traitement du signal audio rapide et efficace, afin d'apprécier leur rendu sonore en temps-réel et également afin de pouvoir démarcher les entreprises spécialisées dans la création et la distribution de plugins audio, pour leur vendre certaines technologies IRCAM[5].

Par ailleurs afin d'illustrer notre outil d'exemples pour les utilisateurs et de mettre nos connaissance en traitement du signal en pratique, nous avons implémentés 4 effets audio:

- *lowpass.xspif* Un filtre passe-bas IIR du premier ordre.
- *XspifDelay.xspif* Une ligne à retard non fractionnaire.
- *EnvFollower.xspif* Un suiveur d'enveloppe utilisant une moyenne mobile
- *compressor.xspif* Un traitement non-linéaire de compression de la dynamique du signal également appelé AGC (Automatique Gain Control)

[4]Graphical User Interface

[5]Detection du fondamentale d'un son (breveté), suivi des partiels d'un son, synthèse de la voix chantée, PSOLA, PAGS/SINOLA, Chant, PAF, Additive...

Standard	nombre de lignes	taille fichier(s)
VST	415	9504
PD	370	8957
AU	263	7845
LADSPA	437	11950
XSPIF	80	2078

Table 4.2: gain en nombre de lignes

Bibliography

[Fogel and Bar(2000)] Karl Fogel and Moshe Bar. *Open Source Development with CVS*, 2000. URL `http://cvsbook.red-bean.com/`.

[Goudard and Müller(2003a)] Vincent Goudard and Remy Müller. *XSPIF developer guide*, 2003a.

[Goudard and Müller(2003b)] Vincent Goudard and Remy Müller. *XSPIF user guide*, 2003b.

[Goudard and Müller(2003c)] Vincent Goudard and Rémy Müller. Real-time audio plugin architectures. Technical report, 2003c. URL `http://mdsp2.free.fr/ircam/pluginarch.pdf`.

[van Rossum and col.(2003)] Guido van Rossum and col. Python, 2003. URL `http://www.python.org/`.

[Walsh(1998)] Norman Walsh. A technical introduction to xml. October 1998. URL `http://www.xml.com/pub/a/98/10/guide0.html`.