Orchidée

Serveur MATLAB© d'orchestration

Version 0.2.3

Grégoire Carpentier

6 juillet 2009

Table des matières

Introduction

Orchidée est une plateforme générique et extensible pour l'orchestration assistée par ordinateur. Elle permet de représenter des connaissances musicales à la fois symboliques (variables de l'écriture) et perceptives (descripteurs du signal), et d'utiliser ces connaissances pour rechercher des combinaisons instrumentales répondant à un ensemble de critères de timbre.

Orchidée se présente sous la forme d'une application serveur autonome, communicant avec ses environnements clients par l'intermédiaire d'une API OSC. Un patch d'exemple Max/MSP, fourni avec Orchidée, permet d'en explorer rapidement les fonctions. Côté OpenMusic, quelques éléments et interfaces de contrôle sont déjà disponibles dans la librairie `om-orchidee`. Orchidée a été testé sous Mac OS 10.4 (Tiger) et Mac OS 10.5 (Léopard).

Orchidée est développé en Matlab compilé. Il n'est pas nécessaire d'installer Matlab pour l'utiliser, seul X11 est requis, ainsi que la librairie `lo`, fournie avec la distribution. Le choix d'un serveur OSC en Matlab compilé se justifie par la flexibilité avec laquelle Orchidée peut être utilisé en production dans des environnements comme Max/MSP ou OpenMusic, ainsi que par la possibilité d'y intégrer facilement les recherches futures. Par rapport au prototype d'orchestration précédent, Orchidée dispose d'une connaissance instrumentale beaucoup plus vaste et plus représentative de l'orchestre symphonique. Cette connaissance peut en outre être étendue par l'utilisateur par ajout de nouveaux échantillons. Par ailleurs, Orchidée gère les hauteurs microtoniques jusqu'au 1/16 de ton, et intègre six descripteurs du timbre : centroïde spectral, étendue spectrale, principaux partiels résolus, spectre de mel, amplitude de la modulation d'énergie et temps d'attaque. En contrepartie, Orchidée ne dispose *d'aucune interface graphique*. Leur développement est laissé aux application clientes.

Le présent document décrit l'architecture logicielle d'Orchidée (chapitre 1), la structure et le contenu de la connaissance instrumentale (chapitre 2), ainsi que l'API OSC permettant le contrôle d'Orchidée depuis une application cliente (chapitre 3). Les éléments spécifiques au développement sont rassemblés dans le chapitre 4.

Chapitre 1

Architecture

L'architecture d'Orchidée répond au schéma général de la figure 1.1. Au niveau supérieur une première couche filtre les messages OSC entrants. Les messages élémentaires, en référence à l'état courant du serveur (voir section 3.2), comportent uniquement un chemin (`path`) OSC. Ils sont traités par la couche la plus externe du serveur. Tous les autres messages entrants sont parsés par une couche intermédiaire avant d'être traités. Ils doivent obligatoirement comporter un nombre entier comme premier élément du champ `data`. Ce nombre est à interpréter comme une valeur d'horodatage permettant d'identifier, à des fins de synchronisation, le message actuellement traité par le serveur.
Les messages OSC parsés par la couche intermédiaire se divisent à nouveau en deux catégories : ceux qui concernent la connaissance instrumentale interne d'Orchidée et ceux qui concernent le problème d'orchestration courant. Cette dichotomie reflète l'architecture d'Orchidée, organisée en deux parties :

- Une partie « statique », liée à la connaissance instrumentale. Celle-ci est intégralement contenue dans une instance de la classe `knowledge`, construite à partir d'un ensemble de fichiers XML de description (voir chapitre 2).
- Une partie « dynamique », liée au problème d'orchestration courant. Ce dernier est représenté par une instance de la classe `session`, elle-même composée des objets `target`, `orchestra`, `filter` et `searchstructure`.

En plus de ces cinq classes principales, Orchidée utilise également plusieurs « boîtes à outils », ou bibliothèques de fonctions. Dans la suite de ce chapitre, nous décrivons successivement chacune des classes et boîtes à outils d'Orchidée. Nous en donnons essentiellement une description fonctionnelle. Pour une description plus précise de chaque méthode ou fonction, on se rapportera directement au code source, richement commenté.

1.1 Classes

Classe `knwoledge`

La classe `knwoledge` est un objet permettant de représenter la connaissance instrumentale au sein du système. Elle se présente comme une base de données dont chaque entrée correspond à un échantillon sonore, et chaque champ un descripteur, symbolique ou perceptif. Toute instance de la classe `knwoledge`

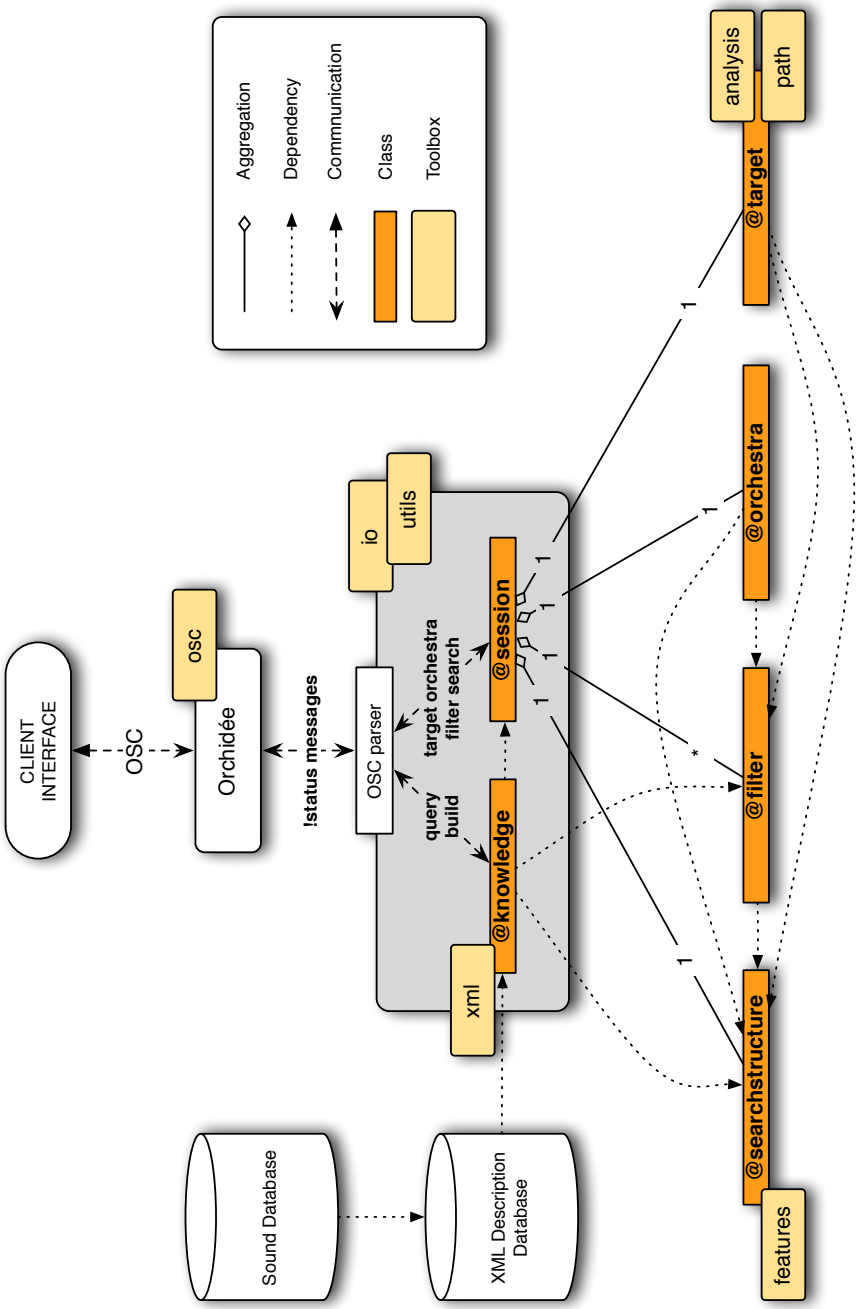

FIGURE 1.1 – Architecture générale du serveur d'orchestration Orchidée.

comprend un en-tête d'auto-description permettant d'interroger l'objet sur son contenu, ainsi qu'un ensemble de tables d'index permettant de faire des requêtes

efficaces (en $O(1)$ pour les requêtes élémentaires).

L'objet `knwoledge` est construit ou mis à jour à partir de fichiers XML de description. A chaque échantillon sonore de la base de données correspond un unique fichier de description. La structure des fichiers XML doit être conforme à structure Matlab stockée dans le fichier `metadata_tempate.mat`. Une commande de la bibliothèque `analysis` permet d'extraire les fichiers XML de description à partir des fichiers sons.

Au démarrage d'Orchidée, la connaissance instrumentale par défaut est chargée automatiquement depuis le fichier `default_knowledge.mat`.

Classe `session`

L'objet `session` rassemble toutes les données relatives au problème d'orchestration courant : cible, orchestre, filtres et structure de recherche. Les méthodes de la classe `session` font l'interface entre l'envoi et la réception de messages OSC d'une part et l'appel de méthodes spécifiques à chaque classe d'autre part.

Le problème d'orchestration courant est entièrement défini par la donnée d'un objet `target`, un objet `orchestra`, et un ensemble d'objets `filter`. Le contenus de ces objets peut être aisément modifié par l'utilisateur, via un ensemble de méthodes OSC ad-hoc. En revanche, la classe `searchstructure` doit être pensée comme privée. Elle ne peut pas être modifiée par l'utilisateur.

Classe `orchestra`

L'objet `orchestra` comprend deux champs, la composition de l'orchestre et la résolution microtonique. L'orchestre est pensé comme une liste d'instrumentistes plutôt qu'une liste d'instruments. Cela permet de spécifier qu'un même instrumentiste peut jouer alternativement plusieurs instruments de la même famille (c'est le cas notamment des trombonistes, clarinettistes, tubistes, saxophonistes ou flutistes).

La résolution microtonique correspond à la petitesse des intervalles de hauteurs que l'orchestre peut réaliser (demi-ton, quart de tons, huitième de ton ou seizième de ton).

Classe `target`

La classe `target` contient un chemin vers le fichier son de la cible d'orchestration, un ensemble de paramètres d'analyse, ainsi qu'une méthode permettant d'extraire les descripteurs audio de la cible.

Classe `filter`

L'objet `filter` permet de contraindre le domaine de recherche, en interdisant, autorisant ou encore forçant certaines valeurs d'un champ donné de la connaissance instrumentale. Via cet objet, l'utilisateur a la possibilité de restreindre la recherche à un ensemble de hauteurs donné, d'interdire certains modes de jeu,

etc. A la création d'un objet `session`, un filtre est instancié pour chaque champ indexé de la connaissance instrumentale, c'est-à-dire chaque champ pour lequel il existe une table d'index.

L'objet `filter` comporte un champ `mode` qui peut prendre quatre valeurs différentes :

- `free` : Toutes les valeurs du champ associé au filtre sont autorisées.
- `auto` : Les valeurs autorisées pour le champ champ associé au filtre sont calculées automatiquement en fonction du problème d'orchestration courant. C'est le mode par défaut à l'instanciation de l'objet. Par défaut également, le mode `auto` se confond avec le mode `free`, sauf si une méthode spécifique `apply_filter_*` est implémentée dans la classe.
- `inex` : Include/exclude - Certaines valeurs, spécifiées par l'utilisateur, sont inclues dans le domaines de recherche, d'autres en sont exclues.
- `force` : Contraint la recherche à un ensemble précis de valeurs.

Classe `searchstructure`

Cet objet n'est pas accessible directement à l'utilisateur. Il est construit progressivement à partir des objets `target`, `orchestra` et `filter`, au fur et à mesure de leur mise à jour. L'objet `searchstructure` comporte des structures de données optimisées pour la recherche combinatoire. Ses méthodes permettent notamment :

- de calculer, pour tout champ indexé de la connaissance, les valeurs autorisées dans la recherche, en fonction de l'état du filtre associé ;
- de calculer le domaine de chaque variable du domaine de recherche (chacune étant associée à un instrumentiste donné) en fonction des valeurs autorisées pour chaque champ ;
- de générer une matrice associée à chaque descripteur intervenant dans le processus optimisation, en tenant compte de la résolution microtonique de l'orchestre (les descripteurs des hauteurs microtoniques sont calculés en appliquant les méthodes de transposition de la boîte à outils `features` au descripteurs de la connaissance).

La mise à jour de l'objet `searchstructure` se fait automatiquement en fonction des actions de l'utilisateur pour définir le problème d'orchestration courant. Une fois cette mise à jour effectuée, l'algorithme d'orchestration, qui figure parmi les méthodes propres de la classe `searchstructure`, peut alors être exécuté.

1.2 Boîtes à outils

Répertoire `osc`

Cette boîte contient la fonction d'appel principale `oscserver`, ainsi que toutes les fonctions réalisant l'interface entre la communication externe OSC et les appels internes de fonctions. Elle comprend par ailleurs la librairie `osc-matlab` développée par Andy Schmeder au CNMAT.

Un fichier de préférences, définissant les paramètres OSC de la communication client/serveur, est créé à l'emplacement `~/Library/Preferences/IRCAM/orchidee/oscprefs`. Ce fichier est lu à chaque démarrage du serveur.

Répertoire `io`

Il s'agit d'un ensemble de fonctions de lecture/écriture de fichiers texte ou `AIFF`.

Répertoire `utils`

Il s'agit d'un ensemble de fonctions utilitaires, principalement de conversion et de dénombrement.

Répertoire `xml`

Ce répertoire comporte toutes les fonctions de recherche, lecture et écriture de fichiers XML, ainsi que la `xmltoolbox` pour Matlab développée à l'Université de Southampton par Marc Molinari.

Répertoire `features`

Ce répertoire contient l'ensemble des fonctions nécessaires pour l'estimation et la comparaison des descripteurs du timbre dans le processus de recherche d'orchestrations. Ces fonctions sont de quatre types :

- `compute` : Fonctions calculant les descripteurs d'une combinaison à partir des descripteurs de ses composantes.
- `compare` : Fonctions retournant la dissimilarité perceptive entre un descripteur de la cible et la valeur du même descripteur pour une combinaison.
- `neutral` : Fonctions générant les descripteurs de l'élément neutre, c'est-à-dire le silence musical.
- `transpose` : Fonctions générant les descripteurs associées aux hauteurs microtoniques en fonction des descripteurs des sons de demi-tons les plus proches.

Le répertoire `features` comporte quatre méthodes génériques invoquant des fonctions spécifiques à chaque descripteur, situées dans les sous-répertoires `compute`, `compare`, `neutral` et `transpo`. L'implémentation d'un nouveau descripteur dans Orchidée nécessite donc, en plus de sa présence dans l'objet `knowledge` (et donc dans les fichiers XML), l'implémentation des quatre méthodes associées dans ces sous-répertoires.

Répertoire `analysis`

Ce répertoire contient les fonctions permettant d'extraire les descripteurs audio d'un son cible, ainsi que le fonctions permettant de générer, à partir d'un ensemble de fichiers sons, un ensemble de fichiers XML de description. Le sous répertoire `damien` contient un ensemble de fonctions d'analyse fournies par Damien Tardieu (ex-membre de l'équipe Analyse-synthèse). **Cette partie du code n'est pas commentée.**

Enfin, le répertoire `analysis` contient également les bibliothèques `ircamdescriptor` et `sdif-matlab` de l'équipe Analyse-synthèse.

Répertoire `genetic`

Ce répertoire contient les opérateurs génétiques de génération, sélection, évaluation, croisement et mutation de population, utilisés par l'algorithme d'orchestration.

Répertoire `multiobjective`

Ce répertoire contient l'ensemble des fonctions relatives à la recherche multicritère : test de dominance, extraction du front de Pareto, génération de poids, échantillonnage de l'espace de critères et méthode de préservation de la diversité.

Répertoire `path`

Ce répertoire contient les fonctions de recherche de librairie ou d'applications externes utilisée par Orchidée.

Chapitre 2

Connaissance instrumentale

Ce chapitre décrit la structure de la connaissance instrumentale « externe » fournie avec Orchidée (échantillons sonores et fichiers de descriptions associés).

2.1 Bases de données

La connaissance instrumentale dont dispose Orchidée provient intégralement de bases de données d'échantillons sonores. Ces échantillons doivent être nommés et organisés (en systèmes de répertoires) d'une façon précise pour pouvoir être exploités par Orchidée.

La figure 2.1 décrit cette organisation. Le répertoire `root` contient les différentes bases de données utilisées par Orchidée. C'est dans ce répertoire qu'il convient de placer la base de données `orchidee`, rassemblant plusieurs bases de sons disponibles à l'IRCAM, et dont la nomenclature est donnée au paragraphe 2.4. Les bases de données ultérieurement créées par l'utilisateurs seront également à placer dans le répertoire `root`, au même niveau que la base `orchidee`.

Chaque base de données s'organise en un système de répertoires et sous-répertoires selon la hiérarchie suivante (du niveau supérieur vers les niveaux inférieurs) :

Niveau 0 Répertoire racine de la base de données, portant son nom (par exemple `orchidee`).

Niveau 1 Familles d'instruments (par exemple `Strings`).

Niveau 2 Instruments (par exemple `Tenor-Trombone-cup`).

Niveau 3 Modes de jeux (par exemple `flatterzunge`).

Niveau 4 Echantillons sonores (par exemple `Ob-sfz-A#5-f.aif`).

La figure 2.2 est un exemple d'une hiérarchie d'échantillons sonores.

Les échantillons sonores doivent être des fichiers `WAV` ou `AIFF` échantillonnés à 11.025 kHz, 22.05 kHz ou 44.1 kHz avec une profondeur de 16 ou 24 bits. Il doivent par ailleurs être nommés selon la nomenclature de `SOL v.2` :
`[Instr]<+[Sourd]>-[modeDeJeu]-[Hauteur]-[Dyn]-<[Corde]c>`
Le respect de la hiérarchie de répertoires et de la nomenclature des noms de fichiers est indispensable à l'extraction d'informations symboliques lors de l'analyse de nouveaux échantillons. Par exemple, du fichier `ClBb-flatt-A#3-ff.wav`

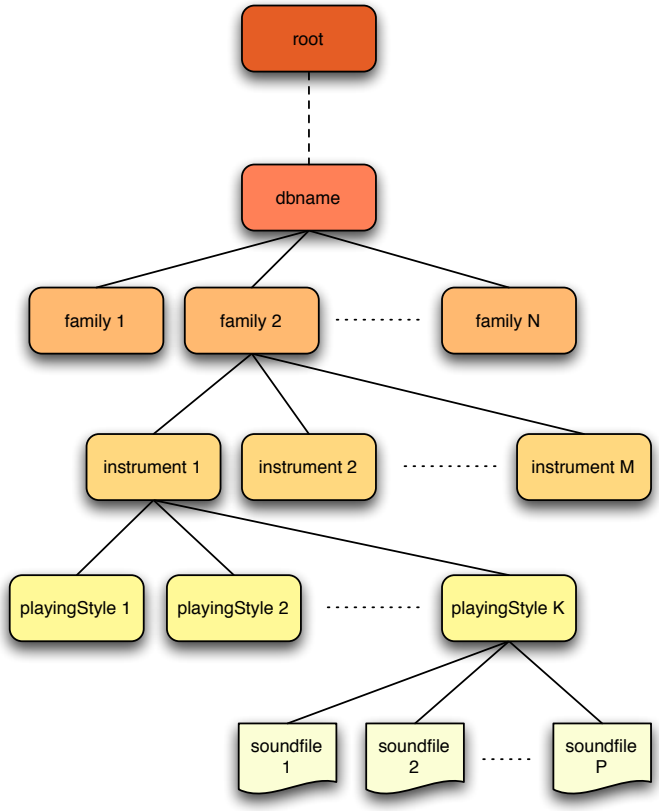

FIGURE 2.1 – Structure hiérarchique de la connaissance instrumentale.

de la figure 2.2 sont extraites, compte tenu de son nom et de son emplacement, les informations suivantes :

`uri`	ClBb-flatt-A#3-ff
`file`	ClBb-flatt-A#3-ff.wav
`dir`	/my_database/Clarinets/Clarinet-Bb/flatterzunge/
`dbname`	my_database
`source`	User
`instrument`	ClBb
`family`	Clarinets
`note`	A#3
`pitchClass`	A#
`octave`	3
`dynamics`	ff
`string`	0
`playingStyle`	flatt
`stringMute`	NA
`brassMute`	NA

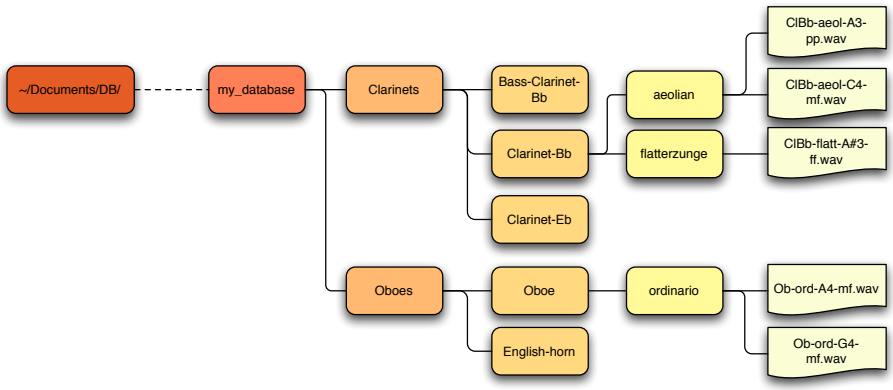

FIGURE 2.2 – Exemple de hiérarchie d'échantillons sonores, selon la structure `dbname/family/instrument/playingstyle/samples` explicitée figure 2.1.

2.2 Fichiers XML de description

La connaissance instrumentale interne dans Orchidée est un ensemble de descripteurs symboliques et numériques. Ils sont extraits à partir du signal audio et de l'emplacement des échantillons sonores dans la hiérarchie décrite à la section précédente. La construction de la connaissance se fait en deux étapes :

1. Une étape d'analyse à la suite de laquelle un ensemble de fichiers XML de description sont crés, selon une hiérarchie identique à celle des échantillons sonores.
2. Une étape d'import qui consiste à parcourir l'ensemble des fichiers XML de description, et à ajouter les nouveaux éléments dans l'objet interne.

Ce processus offre de nombreux avantages :

1. La connaissance instrumentale n'est pas « prisonnière » d'Orchidée, on peut y accéder à partir d'autres applications etl'utiliser à d'autres fins que l'orchestration.
2. Le choix d'un standard comme XML garantit la pérennité des informations extraites des échantillons, quel que soit l'avenir Orchidée.
3. Il est possible de définir un *template* externe de description par l'intermédiaire d'un schéma XML.
4. L'utilisation d'Orchidée pour l'étape d'analyse des échantillons n'est pas nécessaire. L'utilisateur peut s'il le souhaite recourir à ses propres méthodes de calcul de descripteurs.

2.3 Liste des descripteurs

Il existe trois catégories de descripteurs :

- Descripteurs informatifs : il s'agit du nom, de l'origine et de la localisation des échantillons, ainsi que de quelques informations utiles pour la simulation des orchestrations.

- Descripteurs symboliques : ce sont avant tout les variables de l'écriture musicale, ainsi que la taxonomie des modes de jeu instrumentaux de Damien Tardieu.

- Descripteurs du signal : valeurs numériques extraites directement du signal audio, caractérisant le timbre sur le plan perceptif.

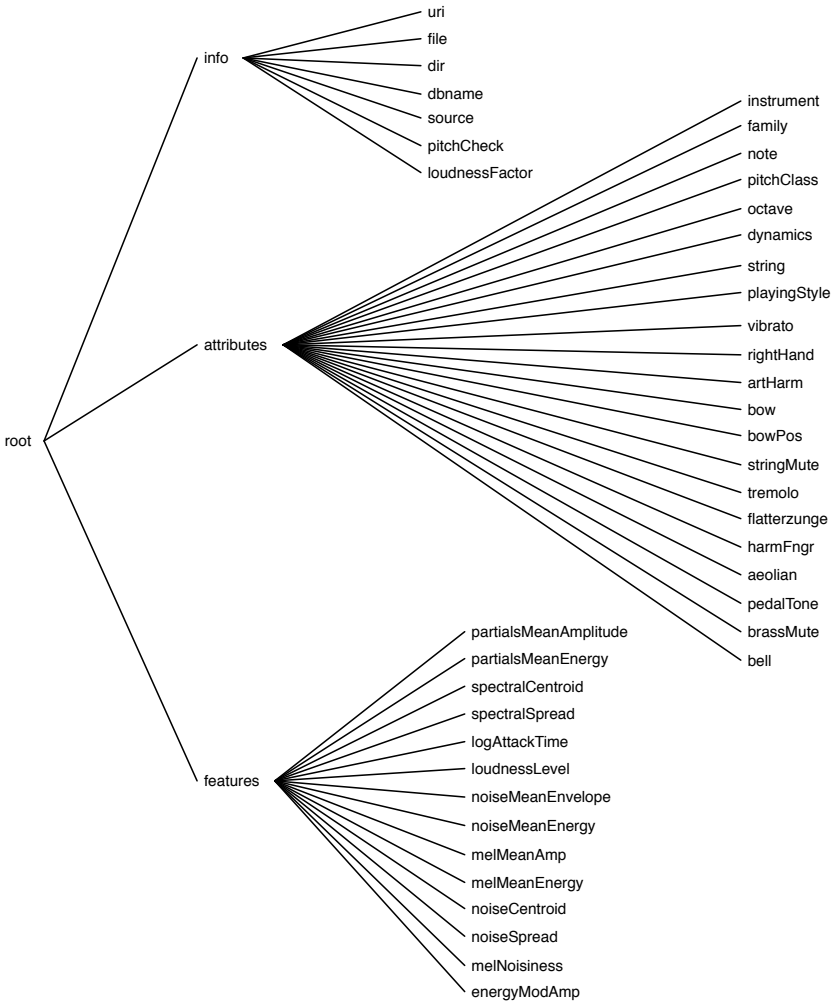

FIGURE 2.3 – Structure des fichiers XML de description.

Ces catégories se retrouvent dans la structure même des fichiers XML de descriptions, comme le montre la figure 2.3.

Descripteurs informatifs

`uri`	Identifiant unique de l'échantillon
`file`	Nom du fichier
`dir`	Chemin relatif dans la hiérarchie
`dbname`	Nom du répertoire racine de la hiérarchie
`source`	Origine de l'échantillon
`pitchCheck`	Justesse de l'échantillon vérifiée ?
`loudnessFactor`	Facteur de correction d'amplitude

Descripteurs symboliques

`instrument`	Symbole de l'instrument
`family`	Nome de la famille instrumentale
`note`	Hauteur
`pitchClass`	Classe de hauteur
`octave`	Numéro d'octave
`dynamics`	Dynamique
`string`	Numéro de corde (pour les cordes)
`playingStyle`	Nom du mode de jeu
`vibrato`	Présence de vibrato ?
`rightHand`	Action de la main droite (pour les cordes)
`artHarm`	Harmonique articficielle ?
`bow`	Action de l'archet (pour les cordes)
`bowPos`	Position de l'archet (pour les cordes)
`stringMute`	Sourdine (pour les cordes)
`tremolo`	Présence de trémolo ?
`flatterzunge`	Présence de flatterzunge ?
`harmFngr`	Doigté harmnique (pour les bois) ?
`aeolian`	Mode de jeu aéolien (pour les bois) ?
`pedalTone`	Note pédale (pour les cuivres) ?
`brassMute`	Sourdine (pour les cuivres)
`bell`	Action au pavillon (pour les sourdines wah)

N.B. Bien que présents dans les fichiers XML de description, les descripteurs `vibrato`, `rightHand`, `artHarm`, `bow`, `bowPos`, `tremolo`, `flatterzunge`, `harmFngr`, `aeolian`, `pedalTone` et `bell` ne sont pas utilisés dans la version actuelle d'Orchidée.

Descripteurs du signal

`partialsMeanAmplitude`	Amplitude moyenne des 100 premiers partiels
`partialsMeanEnergy`	Energie totale moyenne des 100 premiers partiels
`spectralCentroid`	Centroïde du spectre d'energie
`spectralSpread`	Etendue du spectre d'energie
`logAttackTime`	Logarithme du temps d'attaque
`loudnessLevel`	Niveau de sonie
`noiseMeanEnvelope`	Enveloppe de bruit moyenne
`noiseMeanEnergy`	Energie moyenne de l'enveloppe de bruit
`melMeanAmp`	Spectre de mel moyen
`melMeanEnergy`	Energie moyenne du spectre de mel
`noiseCentroid`	Centroïde du spectre de bruit
`noiseSpread`	Etendue du spectre de bruit
`melNoisiness`	Niveau de bruit
`energyModAmp`	Amplitude de la modulation d'énergie

Note 1. Les descripteurs vectoriels (à savoir `partialsMeanAmplitude`, `noiseMeanEnvelope`, et `melMeanAmp`) sont normalisés. Cela facilite leur utilisation dans des tâches d'apprentissage. Les valeurs objectives peuvent être retrouvées à l'aide des descripteurs d'énergie associés : `partialsMeanEnergy`, `noiseMeanEnergy`, et `melMeanEnergy`.

Note 2. Bien que présents dans les fichiers XML de description, les descripteurs `noiseMeanEnvelope`, `noiseMeanEnergy`, `noiseCentroid` et `noiseSpread` ne sont pas utilisés dans la version actuelle d'Orchidée.

2.4 Nomenclature

Instruments

ASax	Alto-Sax
BFl	Bass-Flute
BTb	Bass-Tuba
BTbn	Bass-Trombone
BClBb	Bass-Clarinet-Bb
CbClBb	Contrabass-Clarinet-Bb
ClBb	Clarinet-Bb
Bn	Bassoon
TpC	Trumpet-C
Cb	Contrabass
CbFl	Contrabass-Flute
CbTb	Contrabass-Tuba
EH	English-Horn
ClEb	Clarinet-Eb
Fl	Flute
Hn	Horn
Ob	Oboe
Picc	Piccolo
TTbn	Tenor-Trombone
Va	Viola
Vc	Violoncello
Vn	Violin

Sourdines

C	cup	TpC, TTbn
H	harmon	TpC, TTbn
P	plunger	TTbn
S	sordina	BTb, Hn, Va, Vn, Vc, Cb
S	straight	TpC, TTbn
SP	sordina piombo	Va, Vn, Vc
W	wah	TpC, TTbn
Whsp	whisper	TpC

Modes de jeu

`aeol`	aeolian
`aeol+ord`	aeolian-and-ordinario
`aeol-flatt`	aeolian-flatterzunge
`art-harm`	artificial-harmonic
`art-harm-trem`	artificial-harmonic-tremolo
`brassy`	brassy
`flatt`	flatterzunge
`flatt-closed`	flatterzunge-closed
`flatt-hi-reg`	flatterzunge-high-register
`flatt-open`	flatterzunge-open
`flatt-stopped`	flatterzunge-stopped
`fp`	fortepiano
`harm-fngr`	harmonic-fingering
`key-cl`	key-click
`legno-batt`	col-legno-battuto
`legno-tratto`	col-legno-tratto
`legno-tratto-pont`	col-legno-tratto-ponticello
`legno-tratto-tasto`	col-legno-tratto-tasto
`nonvib`	non-vibrato
`ord`	ordinario
`ord-closed`	ordinario-closed
`ord-hi-reg`	ordinario-high-register
`ord-open`	ordinario-open
`pdl-tone`	pedal-tone
`pizz`	pizzicato
`pizz-bartok`	pizzicato-bartok
`pizz-lv`	pizzicato-l-vib
`pizz-sec`	pizzicato-secco
`pont`	sul-ponticello
`pont-trem`	sul-ponticello-tremolo
`sfz`	sforzando
`stacc`	staccato
`stopped`	stopped
`tasto`	sul-tasto
`tasto-trem`	sul-tasto-tremolo
`tng-ram`	tongue-ram
`trem`	tremolo
`vib`	vibrato

Chapitre 3

API OSC

Ce chapitre décrit l'intégralité des fonctions de l'API OSC d'Orchidée. Les messages entrants sont affichés en rouge (→ **/inputmessage**). Un patch d'exemple Max/MSP, fourni avec Orchidée, permet d'expérimenter une grande part des fonctions listées ici.

3.1 Erreurs

← /error

Message d'erreur suite à l'exécution d'une tâche ou d'un calcul, commandé par un message entrant d'identification `<id>`.

```
path: /error
data: <id> <errortext>
```

Exemple : `/error 112 "in main.m line 36: Can't open soundfile."`

3.2 Etat du serveur

Les messages d'état permettent de renseigner l'interface cliente sur l'état courant du serveur. Ils ont la particularité de ne nécessiter aucun argument, pas même un `<id>`. Ils sont traités par la couche la plus périphérique d'Orchidée et, à la différence de tous les autres messages entrant, ne sont pas parsés.

→ /isready

Vérifie la disponibilité du serveur. S'il est libre, il répond aussitôt par ← **/ready**.

```
path: /isready
data: ∅
```

← /ready

Le serveur indique qu'il est prêt à recevoir un message. Peut être envoyé après un → **/isready** ou après l'exécution d'un calcul.

```
path: /ready
data: ∅
```

→ /version

Interroge le serveur sur la version courante d'Orchidée.

```
path: /version
data: ∅
```

← /version

En réponse à → **/version**, retourne la version courante d'Orchidée.

```
path: /version
data: [version-message]
```

← /busy

Le serveur indique qu'il débute un calcul. Il sera indisponible jusqu'au prochain ← **/ready**. De manière optionnelle, on peut envoyer un flottant `progress` (entre 0 et 1) afin de renseigner une waitbar, ainsi qu'un message renseignant le client sur l'opération en cours.

```
path: /busy
data: [progress] [message]
```

→ /quit

Termine la communication OSC avec le serveur, avant que celui-ci ne quitte automatiquement. Il faudra alors relancer le serveur pour ouvrir une nouvelle communication.

```
path: /quit
data: ∅
```

← /quit

Dernier message envoyé par le serveur avant de quitter.

```
path: /quit
data: ∅
```

← /acknowledge

Message retourné par le serveur pour accuser réception d'un message entrant dont le champ `data` contient une identification `<id>`. Un tel message entrant correspond alors à une tâche à accomplir ou un calcul à effectuer. **← /acknowledge** est donc aussitôt suivi d'un **← /busy** indiquant le début de l'exécution de la tâche.

```
path: /acknowledge
data: <id>
```

3.3 Gestion de la connaissance instrumentale

A l'exception de **→ /dbsave**, les messages de gestion de la connaissance instrumentale ne peuvent pas être envoyés si une session de travail est ouverte (voir sect. 3.5).

→ /dbmake

Lit l'ensemble des fichiers XML de description instrumentale contenus dans les sous répertoires du répertoire `<xml-db-root>`. Chaque fichier de description doit être conforme au patron de description `metadata_template.mat` situé à la racine de l'application. La connaissance est alors convertie en une instance de la classe `knowledge`.

```
path: /dbmake
data: <id> <xml-db-root>
```

Exemple : `/dbmake 111 ~/Orchidee/DB/xml/Horns/`

→ /dbupdate

Actualise la connaissance instrumentale interne (stockée dans un objet `knowledge`) en parcourant les fichiers XML de description instrumentale contenus dans les sous répertoires du répertoire `<xml-db-root>`. Seuls les fichiers dont l'URI est absent de la connaissance ou dont la date de création est plus récente que la dernière mise à jour sont importés.

```
path: /dbupdate
data: <id> <xml-db-root>
```

Exemple : `/dbupdate 112 ~/Orchidee/DB/xml/Saxophones/`

→ /dbsave

Sauve l'objet `knowledge` courant dans un fichier `<db-filepath>` (celui-ci doit comporter l'extension `.mat`.

```
path: /dbsave
data: <id> <db-filepath>
```

Exemple : `/dbsave 113 ~/Documents/Orchidee/db/database.mat`

→ /dbload

Charge un objet `knowledge` depuis un fichier `<db-filepath>` (celui-ci doit comporter l'extension `.mat`.

```
path: /dbload
data: <id> <db-filepath>
```

Exemple : `/dbload 114 ~/Documents/Orchidee/db/mydb.mat`

→ /dbreset

Réinitialise l'objet `knowledge` à son état par défaut lors du lancement du serveur.

```
path: /dbreset
data: <id>
```

→ /dbanalyzesamples

Analyse de nouveaux échantillons sonores et écrit leurs fichiers XML de description. Afin que les métadonnées relatives aux attributs symboliques soient

correctement extraites, les échantillons à analyser doivent répondre aux deux exigences suivantes :

1. être organisés selon la hiérarchie `dbname/family/instrument/playing-style/samples` explicitée figure 2.1 ;
2. être nommé selon la syntaxe `instrument-playingstyle-note-dynamic-(string)` détaillée à la section 2.1.

```
path: /dbanalyzesamples
data: <id> <sound-db-rootdir> <xml-db-rootdir>
```

→ **/dbanalyzesamples** cherche dans le répertoire racine `sound-db-rootdir` les sons qui n'ont pas encore de fichier XML de description dans `xml-db-rootdir`, et y rajoute les descriptions associées, selon la même hiérarchie de fichiers. Les nouveaux fichiers XML peuvent alors compléter la connaissance instrumentale grâce à → **/dbupdate**.

Exemple : `/dbanalyzesamples 543 ~/orch/db/sounds/ ~/orch/db/xml/`

3.4 Requêtes sur la connaissance

Tous les messages de requête ont pour premier argument une une identification `<id>` et pour second une sortie `<output>` vers laquelle envoyer le résultat de la requête. Si le champ une identification `<output>` contient le mot `message`, alors le résultat de la requête est envoyé dans un message OSC. Dans tous les autres cas, `<output>` est interprêté comme le chemin absolu vers un fichier texte. L'envoi de résultats sous forme de message OSC n'est pas possible pour les requêtes → **/dbgetfieldvalues** et → **/dbquery**.

→ /dbgetfields

Demande la liste des champs disponibles dans la connaissance instrumentale.

```
path: /dbgetfields
data: <id> <output>
```

Exemple : `/dbgetfields 234 /tmp/dbfields.txt`
Exemple : `/dbgetfields 235 message`

← /dbfields

Retourne la liste des champs disponibles dans la connaissance instrumentale.

```
path: /dbfields
data: <id> <output>
      <id> [field list]
```

Exemple : `/dbfields 234 /tmp/dbfields.txt`
Exemple : `/dbfields 234 uri file dir originalDB pitchCheck ... loudnessFactor instrument family note octave dynamics string ... playingStyle brassMute stringMute partialsMeanAmplitude ... spectralCentroid spectralSpread partialsMeanEnergy melMeanEnergy`

→ /dbgetqueryfields

Demande la liste des champs de la connaissance instrumentale pour lesquels une table d'index est disponible et sur lesquels on peut exécuter des requêtes (voir **→ /dbquery**).

```
path: /getdbqueryfields
data: <id> <output>
```

Exemple : `/dbgetqueryfields 235 /tmp/dbqueryfields.txt`
Exemple : `/dbgetqueryfields 235 message`

← /dbqueryfields

Retourne la liste des champs de la connaissance instrumentale pour lesquels une table d'index est disponible et sur lesquels on peut exécuter des requêtes.

```
path: /dbqueryfields
data: <id> <output>
      <id> [field list]
```

Exemple : `/dbqueryfields 235 /tmp/dbqueryfields.txt`
Exemple : `/dbqueryfields 235 dir originalDB pitchCheck ... instrument family note octave dynamics string playingStyle ... brassMute stringMute`

→ /dbquery

Effectue une requête sur un ou plusieurs champs indexés de la connaissance instrumentale. La liste des champs indexés peut être obtenue à l'aide de **→ /getdbqueryfields**. Une requêtes passée à l'aide de **→ /dbquery** est une succession de couples attributs/valeurs. Est retournée dans un fichier texte les index dans la base de données correspondant aux sons qui satisfont les paramètres de la requête.

```
path: /dbquery
data: <id> <output> [field_A] [value_1/value_2/value_3 ... ] ...
                    [field_B] [value_1/value_2/value_3 ... ] ...
                    ...
```

Exemple : `/dbquery 247 /tmp/dbquery.txt stringMute SP instrument Vn` écrit dans le fichier `/tmp/dbquery.txt` les index dans la base de données correspondant aux sons de violon avec sourdine de plomb.
Exemple : `/dbquery2 248 /tmp/dbquery.txt family Flutes note C3/G3/D4` écrit dans le fichier `/tmp/dbquery2.txt` les index des sons appartenant à la famille des flûtes de hauteur C3, G3 ou D4.

→ /dbgetfieldvalues

Lit un fichier d'index créé par → **/dbquery** et demande les valeurs d'un champ donné pour chaque index du fichier d'entrée.

```
path: /dbgetfieldvalues
data: <id> <output> <attribute> <input>
```

Exemple : `/dbgetfieldvalues 311 /tmp/uris.txt uri /tmp/indices.txt` retourne les URIs associés aux index lus dans le fichier `/tmp/indices.txt`.
Exemple : `/dbgetfieldvalues 312 /tmp/amps.txt partialsMeanAmplitude ... /tmp/indices.txt` retourne la matrice d'amplitude de partiels associée aux index lus dans le fichier `/tmp/indices.txt`.

← /dbfieldvalues

Retourne dans un fichier texte les valeurs d'un champ donné pour chaque index d'un fichier d'entrée passé à → **/dbgetfieldvalues**.

```
path: /dbfieldvalues
data: <id> <output>
```

Exemple : `/dbfieldvalues 311 /tmp/uris.txt`

→ /dbgetfieldvaluelist

Lit un fichier d'index créé par → **/dbquery** et demande l'ensemble des valeurs (sans doublon) prises par un attribut donné au sein des index du fichier d'entrée.

```
path: /dbgetfieldvaluelist
data: <id> <output> <attribute> <input>
```

Exemple : `/dbgetfieldvaluelist 321 message instrument /tmp/tdx.txt` retourne sous forme de message OSC la liste des instruments présents dans le fichier d'index `/tmp/idx.txt`.

← /dbfieldvaluelist

Retourne l'ensemble des valeurs (sans doublon) prises par un attribut donné au sein des index du fichier d'entrée.

```
path: /dbfieldvaluelist
data: <id> <output>
data: <id> <attribute> [value#1] [value#1] ...
```

Exemple : `/dbgetfieldvalues 311 /tmp/uris.txt`
Exemple : `/dbgetfieldvalues 312 note C1 C2 C3 C4`

3.5 Gestion de session

→ /newsession

Ouvre une nouvelle session de travail. Les messages de gestion de la connaissance instrumentale (voir sect. 3.3) ne pourront plus être envoyés (à l'exception de → **/dbsave**) tant que la session restera ouverte. → **/newsession** retourne également les valeurs par défaut des paramètres d'analyse (voir → **/gettargetparameters**).

```
path: \newsession
data: <id>
```

→ /closesession

Ferme la session courante de travail. Cible, orchestre, et filtres sont perdus. Les messages de gestion de la connaissance instrumentale (voir sect. 3.3) peuvent de nouveau être envoyés.

```
path: \closesession
data: <id>
```

3.6 Spécification de problème

→ /setorchestra

Définit la constitution de l'orchestre avec lequel reproduire la cible. Il est possible d'autoriser plusieurs instruments différents pour le même instrumentiste, en les séparant par le caractère `"/"` (cf. exemples).

```
path: /setorchestra
data: <id> [instsymbol#11/instsymbol#12/...] ...
           [instsymbol#21/instsymbol#22/...] ...
           ...
```

Exemple : `/setorchestra 324 Vn Vn Va Vc` construit un quatuor à cordes.
Exemple : `/setorchestra 325 Fl TTbn/BTbn Vn ClBb/BClBb/ClEb TpC Vc` construit un petit ensemble dans lequel le tromboniste peut jouer du trombone ténor ou basse, le clarinettiste de la clarinette en Si bémol, Mi bémol, ou de la clarinette basse en Si bémol.

→ /setresolution

Définit la précision microtonique à laquelle l'orchestre peut jouer. Celle-ci est donnée en fraction de demi-tons. Les valeurs autorisées sont 1, 2, 4 et 8 (soit respectivement une précision au 1/2 ton, 1/4 ton, 1/8 ton et 1/16 ton.

```
path: /setresolution
data: <id> <resolution>
```

Exemple : `/setresolution 765 4` autorise l'orchestre à jouer des 1/8 tons.
Exemple : `/setresolution 766 1` restreint l'orchestre à un jeu en demi-tons.

→ /settargetparameters

Définit les paramètres d'analyse du son cible. Ceux-ci sont passés sous forme d'une liste de couples paramètre/valeur.

```
path: /settargetparameters
data: <id> [param#1] [value#1] ...
           [param#2] [value#2] ...
           ...
```

Exemple : `/setanalyseparams 325 t1 0.5 t2 1.1 fmin 80 npartials 15`

Paramètres disponibles :

- `t1` : début de la fenêtre d'analyse – par défaut 0.3333 ms

- `t2` : fin de la fenêtre d'analyse – par défaut 0.6667 ms
- `fmin` : résolution fréquentielle – par défaut 50 Hz
- `nparitals` : nombre de partiels à extraire – par défaut 25
- `delta` : tolérance d'inharmonicité – par défaut 0.015
- `autodelta` : valeur de `delta` fixée automatiquement – par défaut 1

→ /gettargetparameters

Interroge le serveur sur les valeurs courantes des paramètres d'analyse du son cible.

```
path: /gettargetparameters
data: <id>
```

Exemple : `/targetparameters 456`

← /targetparameters

Retourne les valeurs courantes des paramètres d'analyse du son cible.

```
path: /targetparameters
data: <id> [param#1] [value#1] ...
           [param#2] [value#2] ...
           ...
```

Exemple : `/gettargetparameters 456 t1 0.5 t2 1.1 fmin 80 ... npartials 15 delta 0.0141 autodelta 1`

→ /setsoundfile

Spécifie un fichier son à analyser. Les formats supportés sont WAV et AIFF.

```
path: /setsoundfile
data: <id> <filename>
```

Exemple : `/setsoundfile 326 /System/Users/Orchestration/sound.aif`

→ /setfilter

Définit un filtre sur un champ de la connaissance afin de limiter le domaine de recherche. On peut définir un filtre sur n'importe quel champ indexé (voir

→ **/dbgetqueryfields**). Cela permet par exemple d'inclure certaines hauteurs dans le domaine de recherche, d'exclure certains modes de jeu, etc.

```
path: /setfilter
data: <id> <attribute> <mode> [value#1] [value#2] ...
```

Cinq modes sont possibles :

1. `free`, le filtre est inactif ;
2. `auto`, laisse le serveur définir automatiquement l'ensemble des valeurs du filtre ;
3. `include`, spécifie un ensemble minimal de valeurs de filtre (le serveur peut éventuellement compléter cet ensemble) ;
4. `exclude`, complémentaire de `include` ;
5. `force`, impose un ensemble de valeurs de filtre que le serveur ne peut pas modifier.

Par défaut, les filtres sont instanciés en mode `auto`, équivalent au mode `free` pour l'ensemble des champs à l'exception des hauteurs (champ `note`).

Exemple : `/setfilter 432 note free` : le domaine de recherche n'est pas limité sur les hauteurs.
Exemple : `/setfilter 432 note auto` : laisse au serveur le choix des hauteurs à retenir dans la recherche.
Exemple : `/setfilter 433 note include A2 D3 C4 B5+1.4` : ajoute tous les sons de hauteur A2, D3, C4 ou B5+1.4 au domaine de recherche.
Exemple : `/setfilter 434 note force A2 D3 C4 B5+1.4 C6+1.8 D6 F#7+3.8` : limite le domaine de recherche exclusivement aux sons de hauteur A2 D3 C4 B5+1.4 C6+1.8 D6 F#7+3.8.
Exemple : `/setfilter 435 playingStyle exclude tremolo` : exclut du domaine de recherche tous les sons tremolo.

→ /getcriterialist

Permet d'accéder à la liste des critères possibles pour la recherche d'orchestrations. Ces critères sont les champs de la connaissance pour lesquels des méthodes d'agrégation, de comparaison et de transposition aux hauteurs microtoniques sont implémentées dans la toolbox `features`.

Exemple : `/getcriterialist 555`

← /criterialist

Retourne la liste des critères possibles pour la recherche d'orchestrations.

Exemple : `/criterialist 555 partialsMeanAmplitude spectralCentroid`

→ /setcriteria

Spécifie un sous-ensemble de critères pour la recherche d'orchestrations. Ces critères doivent faire parie de la liste renvoyée par ← **/criterialist**.

Exemple : `/setcriteria 556 spectralCentroid spectralSpread`

3.7 Espaces de recherche, espaces de solutions

→ /getattributedomain

Interroge la structure de recherche (objet `searchstructure`) de la session courante sur l'ensemble des valeurs pouvant être prises par un attribut, i.e. tout champ indexé de la connaissance (cf. → **/dbgetqueryfield**). Si cela n'a pas encore été fait, → **/getattributedomain** déclenche l'analyse de la cible et met à jour la structure de recherche (domaines des variables et matrice de descripteurs). Le domaine d'un attribut dépend de la cible, de l'orchestre, de la résolution microtonique, de la connaissance instrumentale et des filtres. Le calcul peut donc prendre un peu de temps. A l'instar des messages de requêtes sur la connaissance (voir sect. 3.4), la sortie peut être dirigée dans un message OSC ou dans un fichier texte.

```
path: /getattributedomain
data: <id> <ouput> <attribute>
```

Exemple : `/getattributedomain 648 message note` retourne sous forme de message OSC la liste des hauteurs possibles pour la recherche d'orchestration.
Exemple : `/getattributedomain 649 ~/Desktop/domain.txt playingStyle` écrit dans un fichier texte la liste des modes de jeu.

← /attributedomain

Retourne l'ensemble des valeurs pouvant être prises par un attribut.

```
path: /attributedomain
data: <id> <ouput>
data: <id> <attribute> [value#1] [value#1] ...
```

Exemple : `/attributedomain 648 note A2 A3 E4 A4 C#5 A5 B6 D6+1.4`
Exemple : `/attributedomain 649 ~/Desktop/domain.txt`

→ /orchestrate

Lance l'algorithme d'orchestration.

```
path: /orchestrate
data: <id>
```

→ /exportsolutions

Ecrit dans un fichier texte l'ensemble courant des solutions d'orchestration. Le fichier exporté contient le maximum d'informations nécessaires à la navigation dans l'espace des solutions.

```
path: /exportsolutions
data: <id> <export-text-file>
```

Le fichier de solutions a la structure suivante :

```
<number-of-solutions>
<number-of-instruments>
<number-of-criteria>
<criterion#1-name>
<criterion#2-name>
...
<solution-id>
<instrument#1> <note#1> <playing-style#1> <dynamics#1> <mute#1> <string#1> ...
          <path#2> <transpose#1> <gain#1>
<instrument#2> <note#2> <playing-style#2> <dynamics#2> <mute#2> <string#2> ...
          <path#2> <transpose#2> <gain#2>
...
<feature#1-value>
<feature#2-value>
...
<criterion#1-value>
<criterion#2-value>
...
<solution-id>
<instrument#1> <note#1> <playing-style#1> <dynamics#1> <mute#1> <string#1> ...
          <path#2> <transpose#1> <gain#1>
<instrument#2> <note#2> <playing-style#2> <dynamics#2> <mute#2> <string#2> ...
          <path#2> <transpose#2> <gain#2>
...
<feature#1-value>
<feature#2-value>
...
<criterion#1-value>
<criterion#2-value>
...
...
```

Chaque solution d'orchestration (comprise entre deux valeurs de `solution-id`) est successivement décrite par sa composition instrumentale (autant de lignes que de d'instruments dans l'orchestre), ses valeurs de descripteurs audio (autant de lignes que de critères), et ses valeurs de critères (dissimilarités perceptives avec la cible). Chaque ligne de la composition instrumentale répond à la structure suivante :

- `<instrument>` Nom de l'instrument.
- `<note>` Hauteur.
- `<playing-style>` Mode de jeu.
- `<dynamics>` Dynamique.
- `<mute>` Sourdine (prend la valeur `NA` si le champ n'est pas pertinent).
- `<string>` Numéro de corde (prend la valeur `0` si le champ n'est pas pertinent).
- `<path>` Chemin du fichier son à utiliser pour la simulation.
- `<transpose>` Facteur de transposition à appliquer sur le fichier son pour la simulation.
- `<gain>` Facteur d'amplitude à appliquer sur le fichier son pour la simulation.

Chapitre 4

Le coin des développeurs

Sont rassemblées dans ce chapitre toutes les informations techniques utiles aux futurs développeurs d'Orchidée. Rappelons que le présent document n'a pas vocation à décrire *précisément* le contenu et la fonction de chaque méthode, classe, ou fonction. On se rapportera pour cela directement au code source, richement commenté.

4.1 Configuration requise

Orchidée est une application en Matlab compilé est nécessite donc l'installation préalable de X11. L'installation de Matlab nest pas nécessaire.

Orchidée fait appel à la libraire pm2, noyau de AudioSculpt. AudioSculpt doit donc être installé (et autorisé) dans le dossier `/Applications/` (obligatoirement), sans quoi Orchidée ne pourra pas utiliser pm2.

Orchidée utilise la librairie `osc-matlab` de Andy Schmeder (CNMAT). La librairie `lo` doit donc être préalablement copiée dans `/usr/lib/` ou `/usr/local/lib/`. Elle est distribuée dans chaque release d'Orchidée, et se trouve également dans le répertoire `osc/lib/intel/`.

Orchidée est développé pour Mac Intel et a été testé sous Mac OS 10.4 (Tiger) et Mac OS 10.5 (Leopard).

4.2 Gestion des répertoires du projet

Voir l'aide en ligne du fichier `handle_paths` pour l'ajout ou la suppression des répertoires du code source dans le `path` de Matlab.

4.3 Code externe

Orchidée utilise les éléments de code Matlab externe suivants :

- 2 fonctions de la librairie `miditoolbox` de Eerola et Toiviainen, Copyright © 2004, University of Jyvaskyla. La `miditoolbox` est distribuée sous licence GNU/GPL, consultable en ligne : http://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/miditoolbox/.
- La librairie `ircamdescriptor` (p-code Matlab) de Geoffroy Peeters (équipe Analyse-synthèse, IRCAM).
- La librairie `sdif-matlab` de l'équipe Analyse-synthèse.
- La fonction `aiffread` développée par Axel Roebel (Analyse-synthèse).
- La librairie `osc-matlab` développée par Andy Schmeder, disponible en ligne ici : http://opensoundcontrol.org/implementation/matlab-osc. Attention : certains éléments du code source de `osc-matlab` ont été réécrits afin de pouvoir supporter des chaînes de caractères de taille supérieure à 31 caractères (Orchidée ne fonctionne pas avec la version originale distribuée en ligne).
- Un ensemble de fonctions Matlab fournies par Damien Tardieu (ex-membre de l'équipe Analyse-synthèse). Cette partie du code *n'est pas* commentée, ou alors l'est *seulement de manière partielle*.
- La `xmltoolbox` pour `Matlab` développée à l'Université de Southampton par Marc Molinari : http://mathworks.com/matlabcentral/fileexchange/4278.

4.4 Compilation

La compilation du serveur se fait via un makefile (fichier `make.m`) à la racine du code source. La compilation fait appel au compilateur `mcc` de Matlab (version sans Java), qui devra être préalablement configuré à l'aide des commandes `mcc -setup` et `buildmcr`. Les instructions de compilation, de déploiement et de paquetage sont accessibles via la commande `help make`.

N.B Les fichiers suivants doivent impérativement demeurer dans le repertoire `Sources/` :

```
- default_knowledge.mat
- list_descriptors.mat
- list_descriptors.txt
- make.m
- metadata_template.mat
```

4.5 Bogues connus

Segmentation faults

Sous Mac OS 10.5 (Leopard), une `segmentation fault` survient juste après l'arrêt du serveur (consécutif à un message → **/quit** de l'interface cliente). Un fichier `matlab_crash_dump` est alors créé à la racine du compte utilisateur. Le problème vient manifestement d'une mauvaise gestion de mémoire entre la

librairie `sdif-matlab` et la la version Runtime de Matlab. Le problème ne survient pas lorsque le serveur est lancé directement depuis Matlab (en version non compilée). Dans tous les cas, il ne remet aucunement en cause l'utilisation d'Orchidée.

4.6 Améliorations possibles à court terme

Template de description

Idéalement, la taille des descripteurs (en plus du type) devrait être spécifiée dans le modèle de description (fichier `metadata_template.mat`). Ainsi, la vérification du contenu d'un fichier XML pourrait être effecutée intégralement au moment de sa lecture, et non plus en deux temps comme c'est le cas actuellement (vérification de type à la lecture du fichier, vérification de taille à la construction d'un objet `knowledge`).

Méthode `ismember`

Pour vérifier la précence éventuelle de plusieurs éléments dans la connaissance instrumentale, il serait préférable d'utiliser la méthode `ismember` définie pour les objets de type `cell` plutôt que la méthode surchargée dans la classe `knowledge`.

Transposition du spectre de mel

La méthode actuelle de transposition (vers les hauteurs microtoniques) du descripteur de spectre de mel est très approximative. Elle ne permet pas à Orchidée de différencier un son natif de la base de données (en demi-tons) de n'importe laquelle de ses transpositions microtoniques. Elle pourrait être grandement améliorée en calculant les spectres transposés par interpolation entre deux hauteurs.

Justesse des échantillons

La justesse des échantillons de la base de données n'a pas été vérifiée. Il conviendra de le faire, pour le réalisme des simulations.

Fréquence d'échantillonnage

Orchidée ne permet de traiter que des sons échantillonnés à une fréquence de 11.025 kHz, 22.05 kHz ou 44.100 kHz. Cette contrainte est imposée par la librairie `ircamdescriptor` (dans sa version Matlab). L'accès aux sons échantillonnés à 48 kHz (indispensable) nécessitera une nouvelle version de `ircamdescriptor`.