Autonomie

« La haute technologie au service du handicap »

Programme du pôle de compétitivité Minalogic

SP4c – Gestionnaire de sons pour le guidage

Échéance du livrable : 31 Décembre 2011

Soumission du livrable : 31 Décembre 2011

Date de départ du projet : 1er Janvier 2009 Durée : 24 mois

Nom de l'organisation en charge du contrat pour ce livrable : INRIA

Projet subventionné par le FEDER	
Niveau de dissémination	
PU	Publique

Mots-clés : Audio interactif, guidage, spatialisation 3D, réalité augmentée audio, format

Partenaire responsable : INRIA

Historique des modifications			
Version	Date	État	Modifications faites par
1	04/18/11	Draft	Yohan Lasorsa

Responsable du livrable

Jacques Lemordant (INRIA)

Liste des contributeurs

Yohan Lasorsa

Liste des évaluateurs

TBD

Résumé

Ce document présente les résultats des développements effectués dans le cadre du sous-projet 4c. Il présente les différents aspects du format audio adapté à la navigation et au guidage qui a été défini, ainsi que son utilisation à travers un gestionnaire de sons qui fournira le rendu audio final. L'utilisation des différentes fonctionnalités du format ayant pour but la personnalisation du rendu audio de guidage est notamment mise en avant. L'évolution dans le langage utilisé pour ce format ainsi que le gestionnaire audio l'accompagnant est également abordé afin d'expliquer les raisons ayant motivés les divers choix et changements effectués.

TABLE DES MATIÈRES

1. Introduction

Autonomie se propose de développer des outils basés sur la haute technologie pour améliorer l'autonomie des personnes qui ont un handicap. Ces outils seront intégrés dans des appareils nomades tels que le téléphone mobile ou un boitier spécifique, pour permettre aux personnes handicapées de pallier à leurs déficiences et améliorer leur vie quotidienne principalement dans le domaine du déplacement. Ces appareils pourront accéder à des services dédiés à distance, pour apporter un complément d'aide ponctuelle tel que la localisation, le guidage, etc. Le projet se place donc résolument dans un contexte d'innovation technique et social, en essayant d'exploiter au mieux les technologies de l'embarqué pour permettre aux personnes handicapées de mieux vivre dans leur ville.

Ce projet est une première étape d'un projet plus vaste et devrait permettre de définir les étapes suivantes tant sur le plan technique que sur le plan des relations entre les partenaires non techniques comme les collectivités locales, les associations de personnes handicapés, les organismes officiels liés à la santé (par exemple, les mutuelles), etc.

Les outils existant aujourd'hui pour aider les personnes handicapées sont souvent coûteux parce qu'ils nécessitent un matériel spécifique. Notre projet souhaite se servir de plateformes disponibles dans le marché grand public et/ou développer des plateformes bas coût. Ces plateformes devront faire usage de ce qui se fait de mieux aujourd'hui dans le domaine des systèmes embarqués : caméras, audio, vidéo, GPS, accéléromètre, radiofréquences, etc. L'objectif étant de couvrir la plus large partie des usages et besoins du déplacement dans la ville des personnes malvoyantes ou non voyantes (principalement), mais également de créer une dynamique autour de ces plateformes pour attirer de nouveaux développements de produits

La thématique du projet est orientée sur le déplacement des personnes non voyantes ou malvoyantes avec 2 axes principaux :

- L'aide au déplacement a l'intérieur des bâtiments.
- L'aide au déplacement a l'extérieur des bâtiments en milieux urbain.

Le caractère exploratoire de ce projet par certains de ces aspects sera notamment encadré avec le suivi régulier des besoins auprès des associations et une revue formelle des objectifs à mi-parcours. L'extrême richesse du tissu de la région (PME, startups, laboratoires, universités) devrait nous permettre de créer une forte synergie entre toutes ces compétences au service de cette population, mais aussi de l'orienter résolument vers des cibles réalistes. Nous attendons des retombées du projet aussi bien au niveau commercial (mise sur le marché de nouveaux produits et services) que des avancées au niveau technique (ergonomie, nouvelles idées d'outils, nouvelles plateformes).

Représentant le cœur du projet dans globalité, le sous-projet SP4 a pour but de définir un langage de description de guides audio pour la navigation dans des bâtiments et de réaliser un navigateur embarqué pour la navigation dans ces bâtiments. L'objectif est de permettre la création d'un web de navigation indoor pour personnes en situation de handicap visuel, permettant ainsi de télécharger le guide audio d'un bâtiment avant de s'y rendre ou à l'entrée du dit bâtiment. On essayera d'être la plupart du temps indépendant d'une infrastructure pré-existante dans le bâtiment de façon à accroitre l'autonomie de la personne et l'usage du système. L'accent sera mis sur la réalisation d'un guidage audio-vocal de haute qualité et du type réalité augmentée. Le guidage sera interactif, permettant d'émettre des requêtes sur l'environnement proche ou les espaces constitutifs plus lointains.

Au sein de ce sous-projet, l'objectif de cette tâche est de construire un gestionnaire de sons

avec spatialisation 3D, mixage et rendu environnemental recevant des événements en provenance du gestionnaire de positions et d' interrogation de l'environnement. Ce gestionnaire lira des documents dans lesquels se trouveront les spécifications audio associées aux événements reçus. Ces documents seront personnalisables par l'utilisateur et ainsi adaptables aux besoins de chacun.

Ce document présente en particulier les résultats des développements effectués dans le cadre de ce sous-projet. Il présente les différents aspects du format audio adapté à la navigation et au guidage qui a été défini, ainsi que son utilisation à travers un gestionnaire de sons qui fournira le rendu audio final. L'utilisation des différentes fonctionnalités du format ayant pour but la personnalisation du rendu audio de guidage est notamment mise en avant, au travers de différents exemples d'utilisation du langage de description audio.

Des informations additionnelles ainsi qu'une partie des logiciels décrits dans ce document sont accessibles librement sur le web :

- Audio interactif, sur le site de l'équipe WAM :
  http://wam.inrialpes.fr/iaudio
- GForge des outils destinés à l'audio interactif :
  https://gforge.inria.fr/projects/iaudio

Le reste de ce document est organisé en 2 sous-sections, traitant chacune respectivement du format défini pour le guidage audio et de son implémentation au travers d'une libraire disponible notamment pour la plate-forme mobile iOS d'Apple[1].

2. Format pour le guidage audio

2.1. Motivations

La construction d'un système de navigation basé uniquement sur l'audio est une tâche complexe à mettre en œuvre : il faut pouvoir indiquer un nombre suffisant d'informations à l'utilisateur sans le noyer dans une foultitude de sons, être capable de fournir plusieurs types d'indices de guidage sans surcharger l'espace auditif, savoir trier l'information pour fournir l'information la plus pertinente à un instant donné et enfin pouvoir diriger l'usager de manière précise dans divers types d'environnements.

Un tel système comporte divers challenge à relever, et notamment le fait que pour être optimal, un tel système de guidage audio se doit d'être adaptable facilement aux besoins particuliers de la personne qui l'utilise. Dans cette optique d'adoption, un format d'échange devient alors indispensable pour faire le lien entre le gestionnaire de cartographie et de position de l'application de guidage et le gestionnaire de sons.

Si l'on analyse le problème de navigation audio d'un point de vue technique, il a toutes les caractéristiques d'une application de type audio interactif. D'un point de vue général, ce système a pour but de construire une bande-son dynamique capable de réagir à de multiples évènements au cours du temps, ce qui correspond bien à la définition d'audio interactif. Dans un premiers temps, nous allons donc nous attarder à analyser les formats déjà existants dans ce domaine et voir quels sont leurs points forts et leur points faibles dans une optique de guidage. Nous allons ensuite définir les objectifs d'un format audio destiné au guidage doit pouvoir atteindre, pour enfin décrire le format qui nous allons utiliser dans le cadre de ce sous-projet.

2.2. Audio Interactif et navigation

2.2.1. Formats existants et limitations

Dans le domaine de l'audio interactif, de nombreux travaux ont été effectués depuis des années, mais sans jamais vraiment aboutir à un format d'échange universel, reconnu et surtout implémenté et utilisé. En 1999, le groupe de réflexion *Interactive Audio Special Interest Group* (IASIG) a publié les bases attendues d'un système audio interactif à travers les recommandations *Interactive 3D Audio Rendering Guideline Level 2.0* (I3DL2)[2]. Ces spécifications précisent d'un manière généralise ce qui est conseillé d'implémenter dans un tel système et quels sont les méthodes de spatialisations globalement reconnues pour le rendu audio en 3D. Plus tard en 2005, la branche *Synchronized Multimedia Activity* du World Wide Web Consortium (W3C) a développé le format XML nommé *Synchronized Multimedia Integration Language* version 2.0 (SMIL 2.0)[3] pour la coordination de présentations multimédia ou l'audio, la vidéo, le texte et les images sont combinés en temps réel. Ce format a ainsi défini un standard en ce qui concerne la synchronisation d'éléments multimédia en langage XML. Enfin, plus de huit années après la publications des recommandations ID3L2, le groupe IASIG a annoncé l'achèvement d'un nouveau format de fichier pour l'audio interactif pour compléter les ID3L2. Ce nouveau format, basé sur le standard ouvert de format de fichier XMF, est appelé *Interactive XMF* (iXMF)[4]. Le but du groupe IASIG au travers du développement de ce format est de mettre le contrôle de la partie artistique aux mains des artistes, en évitant aux programmeurs à devoir prendre des décisions artistiques, d'éliminer le besoin de portage sur de nouvelles plates-formes et enfin de réduire les temps, coûts et stress de production.

Le format iXMF présente de nombreux atouts sur ses principes, mais souffre de plusieurs lacunes problématiques sur sa forme : étant un format de fichier binaire, il est n'est pas prévu pour être extensible et peut difficilement s'intégrer à d'autres documents. De plus, le format étant très complexe dans sa structure avec des possibilité de scripting audio avancées, il n'existe à ce jour aucune implémentation de référence, à fortiori sur plate-forme mobile. La seule implémentation connue à ce jour serait intégrée au SDK de la console PS3 de Sony selon une annonce officielle, mais aucune indication sur le degré d'implémentation du format n'est disponible, toutes les informations sur ce SDK étant protégées sous accord de non-divulgation.

Depuis 2009, un format XML pour l'audio interactif sur mobile est développé au sein de l'équipe WAM et se nomme *Advanced/Audio Mobile Markup Language* (A2ML)[5]. Il se base sur les principes de iXMF pour les structures audio et SMIL pour la synchronisation des divers éléments. L'un des avantages de l'utilisation d'un format XML pour définir de l'audio interactif est de pouvoir utiliser les nombreux outils existants pour transformer, éditer et adapter les documents XML. Un autre avantage est le développement d'éditeurs graphiques pour les compositeurs audio peut être réalisé très facilement, par l'utilisation du format XML comme un mécanisme de sérialisation. Une implémentation partielle d'une API utilisant ce format a été développée pour la plate-forme iOS, basée sur l'API audio bas niveau FMOD de Firelight Technologies[6]. Le format A2ML ainsi que sa librairie associée ont été utilisés dans un premier temps dans un but de prototypage pour l'application Autonomie. Cependant, ce format ayant été conçu initialement pour les applications de type jeux vidéos et paysage sonores, différents problèmes furent rapidement soulevés conduisant à la nécessité d'aménager le format pour les besoins spécifiques d'une application de guidage.

2.2.2. Objectifs d'un format adapté au guidage

Dans une optique d'un système de guidage audio, on retrouve tout d'abord tout ce qui constitue les bases de l'audio interactif : structuration audio, mixage des sons et synchronisation des éléments audio sur des évènements de l'application. Cependant, des besoins supplémentaires et particuliers à la construction d'un système de guidage sont également nécessaires : il faut que le langage fournisse un moyen d'organiser les sons à priori et dynamiquement durant la navigation pour que l'utilisateur puisse obtenir l'indication la plus pertinente à chaque instant, et adaptée à son handicap. De plus, la capacité d'intégration d'un flux d'informations audio d'une personne étant limitée, à fortiori quand elle reçoit de l'information en provenance de deux sources différentes (l'environnement réel et les indications de guidage audio), il indispensable de limiter la quantité d'information audio transmises simultanément et dans le temps afin de ne pas surcharger l'utilisateur.

En sus de fonctionnalités d'audio interactif, le format doit donc permettre une gestion de priorité des différentes indications, avec des mécanismes d'ordonnancement évolués permettants le filtrage suivant des paramètres multiples. On distingue ainsi pour le guidage trois paramètres indispensables pour le filtrage : la priorité d'une indication, ainsi que les distances temporelle et géographique par rapport à l'instant de déclenchement de cette indication. En effet, de par le jeu des priorités, lorsqu'une indication audio sera déclenchée, rien ne garantie que celle-ci sera jouée immédiatement, une ou plusieurs indications plus importantes pouvant passer avant. De ce fait, quelle sera la garantie que lorsque cette indication sera finalement donnée à l'utilisateur, elle sera encore pertinente ? C'est ici que l'on distingue plusieurs cas :

- Les indications pertinentes géographiquement uniquement (un trou dans la chaussée par exemple) qui doivent être éliminée dès que l'on s'éloigne de leur zone de pertinence.
- Les indications pertinentes dans le temps uniquement (comme le nom de la prochaine gare par exemple, si l'on se trouve dans un train) qui n'ont plus de sens si après qu'un certain temps se soit écoulé.
- Les indications pertinentes géographiquement et temporellement, qui regroupent les deux contraintes précédentes, comme la présence de travaux dans une bouche de métro par exemple.
- Les indications « toujours pertinentes » (dans le contexte de la navigation), comme le nom de la personne, tel que la météo du lieu de destination par exemple (on distingue bien ici la pertinence de l'indication en elle-même et non de son contenu).

Ces différents cas et leur paramètres de filtrage correspondants doivent donc pouvoir être configurés dans le format de document décrivant les informations de guidage audio.

2.3. Le format MAUDL

2.3.1. Description

Partant de ces différents objectifs, un format adapté à la conception d'un système de guidage a été défini et nommé *Mobile AUDio Language* (MAUDL). Ce format a été conçu avec le but d'être le plus léger, simple et direct possible tout aux répondant aux besoins exprimés. En effet, celui-ci a pour finalité d'être utilisé sur plate-forme mobile et d'être facilement adaptable et configurable par les utilisateurs pour convenir à tout type de handicap et préférences personnelles.

Ce format s'appuie sur les travaux réalisés sur le langage A2ML, notamment pour ce qui est des

structures audio et de la synchronisation, toujours basée sur un système similaire à SMIL, uniquement orienté évènements cependant. Des nouvelles fonctionnalités propres à la problématique de guidage ont ensuite été ajoutées sur ces bases pour obtenir la forme actuel du langage MAUDL (voir annexe 1 pour le schéma Relax-NG du format).

L'architecture d'un document MAUDL s'organise autour de 3 axes principaux (fig. 1) :

- La définition des sources sonores (*sounds*), qui permet d'effectuer des variations aléatoires ou définies d'une source à l'aide d'une hiérarchie simple (*sounds* → *subsounds*).
- La définition de files d'attente de sons (*queues*) à priorité ou non, qui permettent d'organiser la lecture de sources sonores ou autres files d'attente de manière à définir un enchaînement de celles-ci suivant plusieurs paramètres. Elles permettent notamment de jouer en priorité certaines sources sonores (ce qui est indispensable pour le guidage), et définir des paramètres de validité des sources, par rapport à une durée ou une distance parcourue depuis le moment où la source a été ajoutée à la file, par exemple.
- La création d'une hiérarchie de mixage libre, où plusieurs niveaux de mixeurs peuvent être définis afin de créer des groupes de mixages pour les sources sonores et files d'attente de sons.

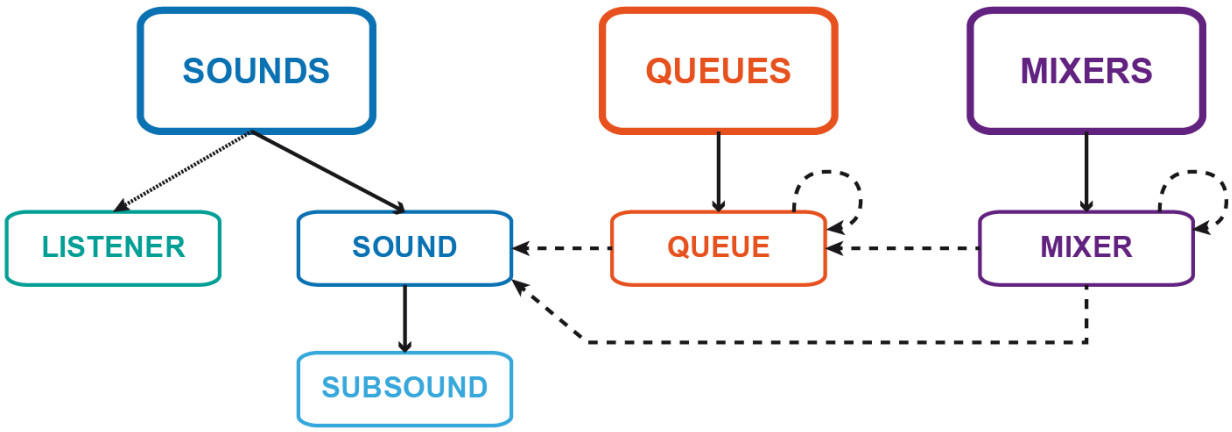

Figure 1: Architecture du langage MAUDL

Concernant la structuration audio, celle-ci reste minimaliste, dans un but de simplicité et flexibilité : on peut définir 2 types d'éléments, les sons (*sound*) qui peuvent éventuellement contenir plusieurs sous-sons (*subsound*). Un élément « son » est une méta-structure séquentielle liée ainsi à un ou plusieurs fichiers audio, avec des paramètres de mixage qui lui sont propres (volume, pan), de rendu (2D/3D), de synchronisation et éventuellement de position (pour le rendu 3D). Un son est toujours composé (implicitement ou explicitement) d'au moins un sous-son. Cette structuration permet d'introduire entre autres choses des variations d'une même icône audio : par exemple, pour sonoriser une porte, on peut choisir d'utiliser 3 enregistrements de bruits de porte différents, dont un seul sera choisi aléatoirement lors du déclenchement de la lecture du son (fig. 2).

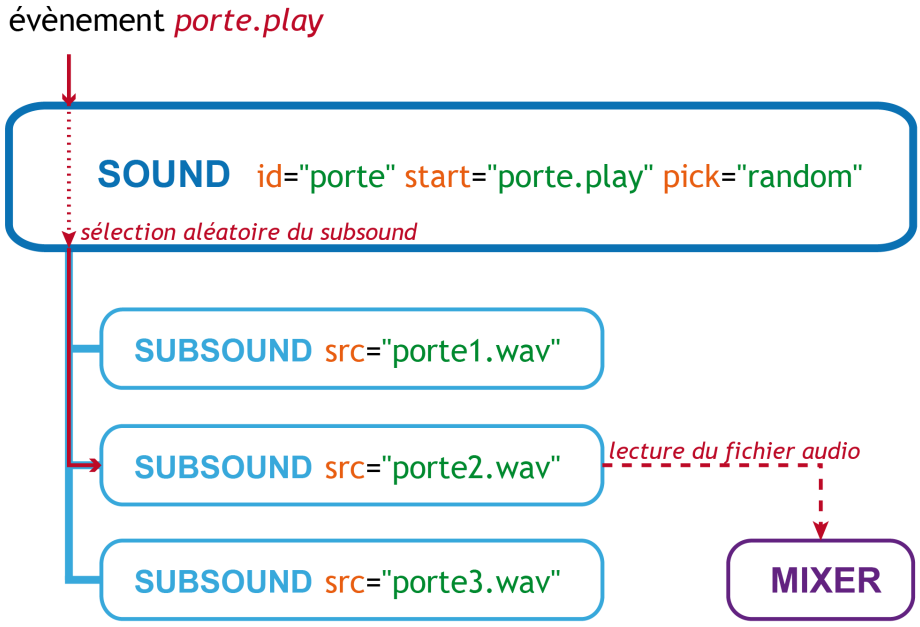

Figure 2: Exemple de structuration audio pour une icône audio de porte

Au niveau de la synchronisation, on distingue deux structures de média contrôlables : les sons et les files d'attentes de son. Concernant le contrôle de lecture de ces éléments, on peut les synchroniser sur un ou plusieurs évènements les contrôles suivants :

- Démarrage de la lecture du son / de la file d'attente (attribut *start*).
- Mise en pause de la lecture du son / de la file d'attente (attribut *pause*).
- Arrêt de la lecture du son / de la file d'attente (attribut *stop*).
- Réinitialisation du pointeur de lecture du son / du son courant d'une file d'attente au début du fichier audio (attribut *reset*).
- Ajout du son / de la file d'attente à une nouvelle file d'attente (attribut *enqueue*).

On retrouve donc l'ensemble des contrôles de lecture de média de base. Pour ce qui est du dernier cas, nous allons maintenant voir en détail comment fonctionnent les files d'attente de son.

Une file d'attente de son, telle que son nom l'indique, est une structure dans laquelle un ou plusieurs sons ordonnés attendent d'être joués. Un seul son est joué à la fois, et dès qu'un son a fini de jouer il est supprimé de la file d'attente. Dans son mode le plus basique, la file d'attente peut être vue comme un système d'enchaînement de sons. Elle peut être configurée pour démarrer automatiquement ou non la lecture dès qu'un son est ajouté (attribut *autoStart*), et une taille maximum peut être définie sous la forme du nombre maximum de sons qu'elle peut contenir (les sons de la file d'attente situés au delà de cette limite étant supprimés de la file).

De plus, la file d'attente peut également servir de système de tri et de filtrage des sons qui lui sont ajoutés : au sein d'une file d'attente, les sons peut être ordonnés selon leur ordre d'ajout ou leur priorité (attribut *sort*). Les sons peut être ensuite filtrés dynamiquement suivant des paramètres de temporalité et de proximité définis lors de leur ajout à la file d'attente. La valeur de l'attribut *enqueue* est ainsi construit de la forme, les deux derniers paramètres étant facultatifs (la valeur par défaut pour ceux-ci étant *-1*):

<évènement>:<nom de la file d'attente>[:<temps de validité ou -1>:[<distance de validité ou -1>]]

Le tri des sons et le filtrage de ceux-ci sera ensuite effectué à chaque fois qu'un son est ajouté à la file d'attente ou que le son courant a fini de jouer. Concernant le filtrage par temps de validité (en secondes), le temps de référence est compté à partir du moment où le son est ajouté à la file. Ensuite, le calcul pour déterminer si le son doit être supprimer de la file se base sur l'horloge de celle-ci qui peut être configurée de deux manière différentes (attribut *timebase*) :

- En mode temps réel (*realtime*), le temps s'écoule de manière continue sans aucune interruption.
- En mode temps de lecture (*playtime*), le temps ne s'écoule que lorsque la file d'attente est en train de jouer un son. Si elle est arrêtée ou en pause, le temps ne s'écoule pas.

Pour ce qui est de la distance de validité (exprimée en unités), la position de référence est celle de l'auditeur (*listener*) au moment où le son est ajouté à la file d'attente. De la même manière, le calcul de la distance pour déterminer ou non le filtrage sera basé sur la position actuelle de l'auditeur. Il revient à la charge de l'application de mettre à jour régulièrement cette position pour que le filtrage fonctionne comme attendu.

Enfin, nous arrivons à l'étape de mixage des différents sons. Dans le format MAUDL, il est possible de créer une hiérarchie libre de mixeurs, sans limites sur le nombre de niveaux. Ceci est très utile pour pouvoir réaliser des sous-groupes de mixage et ainsi agir facilement sur les différents niveaux. Par ailleurs, il est intéressant de noter que les sons se comportent comme des mixeurs de sous-sons, et que les files d'attentes comme des mixeurs de sons. De même, il existe toujours un mixeur global contrôlant l'ensemble des niveaux des éléments audio, même si il n'est pas déclaré explicitement dans le document. Voici par exemple l'organisation d'une hiérarchie de mixage possible, correspondant à l'exemple 5 de la section suivante :

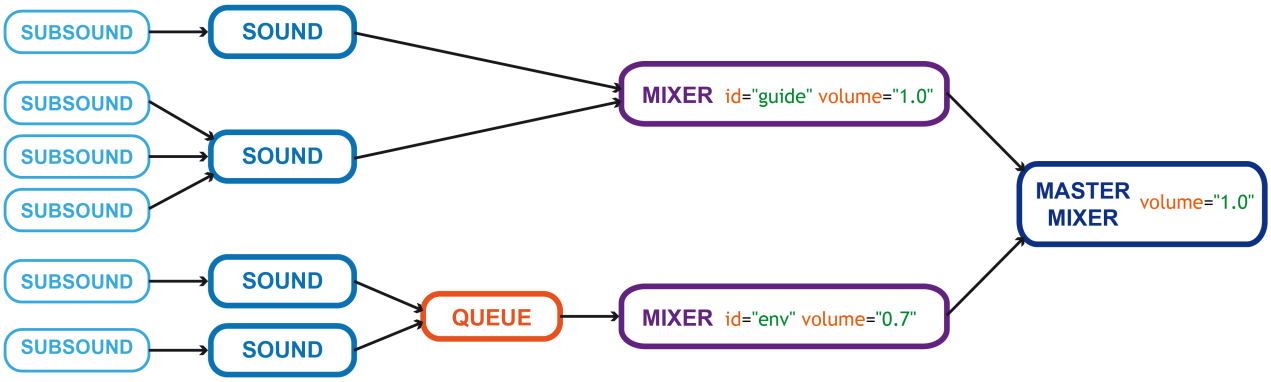

Figure 3: Exemple de hiérarchie de mixage

2.3.2. Utilisation et exemples

Afin d'illustrer plus en détail l'utilisation des différentes fonctionnalités du format, nous allons étudier différents exemples de cas d'utilisation du langage. A titre de référence, un exemple de document illustrant l'ensemble des éléments et attributs du langage est également disponible mais ne sera pas traité ici (voir annexe 2).

➔ Exemple 1 : structuration audio avec sélection aléatoire

Dans cet exemple nous allons voir comment créer une structure audio permettant d'introduire des variations aléatoires du son à chaque déclenchement. Pour illustrer cet exemple, nous allons prendre

le cas de le sonorisation de deux évènements simple, un bruit de pas et la signalisation de l'arrivée à la destination : à chaque pas que fait l'utilisateur, nous voulons qu'il reçoive un retour auditif, avec une variation du son joué à chaque fois pour éviter la monotonie. Une indication sonore simple sera également jouée pour signifier à l'utilisateur son arrivée à destination. Voici un exemple de document correspondant à ce cas :

```
<maudl>
  <sounds>
    <sound id="arrivée" src="arrivée.wav" play="navigation.arrivée"/>
    <sound id="pas" pick="omitMostRecent" play="navigation.pas">
      <subsound src="pas1.wav"/>
      <subsound src="pas2.wav"/>
      <subsound src="pas3.wav"/>
    </sound>
  </sounds>
</maudl>
```

On distingue ici deux types de son :

- Un son simple (arrivée), constitué d'un seul fichier audio. Le son sera joué dès que l'évènement *navigation.arrivée* sera envoyé au gestionnaire de son.
- Un son complexe (pas), constitué de 3 fichiers audio contenant différentes variations d'un bruit de pas. Le son sera joué dès que l'évènement *navigation.pas* sera envoyé au gestionnaire de son.

On note un attribut particulier sur le son « pas » : l'attribut *pick* spécifie la méthode de sélection du sous-son qui sera joué à chaque fois que la lecture du son est déclenchée. Il y a cinq valeurs possibles pour cet attribut :

- *ordered* : les sous-sons sont joués les uns après les autres, dans l'ordre dans lequel ils sont définis (valeur par défaut).
- *reversed* : les sous-sons sont joués les uns après les autres, dans l'ordre inverse dans lequel ils sont définis.
- *random* : un sous-son est choisi aléatoirement à chaque fois que le son est déclenché.
- *omitMostRecent* : un sous-son est choisi aléatoirement à chaque fois que le son est déclenché, à l'exclusion du dernier sous-son joué.
- *fixed* : le sous-son sélectionné ne change pas au déclenchement du son. La sélection d'un sous-son différent passe par l'utilisation de l'attribut *setNext* spécifiable dans un sous-son : celui-ci est sélectionné lorsque l'évènement spécifié dans cet attribut est déclenché.

Dans notre cas d'utilisation, nous avons utilisé la valeur *omitMostRecent*, puisque nous voulons un bruit de pas différent à chaque pas.

➔ *Exemple 2 : Spatialisation des sons*

Dans cet exemple, nous allons voir comment utiliser la spatialisation en 3D d'un son. Pour illustrer cet exemple, nous allons prendre le cas de la sonorisation d'une scène de concert de piano virtuelle au sein d'un théâtre. Cette scène sera constitué de deux éléments sonores : la musique du piano qui joue sur la scène, et la simulation de la réverbération de la salle de théâtre. Voici le document correspondant à ce cas :

```
<maudl>
   <sounds listenerPos="0.0 0.0 5.0" listenerLookAt="0.0 0.0 1.0" listenerRearAttenuation="0.5"
         listenerFrontAngle="180" rolloff="log" scale="1.0">
      <sound id="piano" src="piano.wav" loopCount="-1" render3D="true" pos="2.0 0.0 10.0" min="1"
         max="100" play="concert.start" stop="concert.stop"/>
      <sound id="reverb" src="reverb.wav" loopCount="-1" render3D="true" panFactor="0.0"
         pos="0.0 0.0 -10.0" min="1"  max="100" volume="0.2" play="concert.start" stop="concert.stop"/>
   </sounds>
</maudl>
```

Tout d'abord, une configuration de la scène audio est effectuée de manière globale au travers des divers attributs de l'élément *sounds*. Ces attributs définissent principalement les caractéristiques de l'auditeur (*listener*) et du modèle utilisé pour la spatialisation 3D (pour plus d'information sur les modèles existants et leur paramètres, voir la section 2.4.2) :

- L'attribut *listenerPos* permet de définir la position initiale de l'auditeur dans l'espace virtuel. Dans notre cas l'auditeur est positionné initialement face à la scène, assis au second rang.
- L'attribut *listenerLookAt* permet de définir la direction initiale dans lequel l'auditeur regarde dans l'espace virtuel. Dans notre cas l'auditeur regarde initialement en direction de la scène.
- L'attribut *listenerRearAttenuation* spécifie la quantité maximum d'atténuation arrière utilisée pour créer un effet de focus audio dans la direction regardée (voir fig. 11). Dans notre cas, l'atténuation maximale est fixée à la moitié du volume normal, pour focaliser l'attention de l'auditeur sur ce qui se trouve devant lui.
- L'attribut *listenerFrontAngle* spécifie la taille de l'angle définissant ce qui se trouve devant l'auditeur, et donc à partir duquel l'atténuation arrière entre en jeu (voir fig. 11). Dans notre cas nous avons conservé la valeur par défaut de 180°.
- L'attribut *rolloff* spécifie le modèle d'atténuation de volume des sources audio spatialisées par rapport à leur distance à l'auditeur qui sera utilisé. Cet attribut possède deux valeurs possibles : *log* pour un modèle d'atténuation logarithmique qui correspond au modèle physique réaliste, et *linear* pour un modèle d'atténuation linéaire. Dans notre cas, nous avons choisi le modèle réaliste.
- L'attribut *scale* spécifie le facteur d'échelle pour l'atténuation du volume des sources audio spatialisées. Un facteur 2 signifiera par exemple que l'atténuation sera 2 fois plus rapide qu'en temps normal.

Il est intéressant de noter que le positionnement d'objets dans l'espace virtuel n'utilise pas d'unité de mesure spécifique. Il revient à l'utilisateur du format de choisir sa propre unité de distance (cm, m, pieds...) en fonction de ses besoins.

Maintenant que l'espace sonore et l'auditeur sont définit, il convient de positionner notre source sonore (un piano en train de jouer) dans l'espace. Etudions maintenant en détail l'effet de certains attributs de cette source sonore :

- L'attribut *loopCount* permet de spécifier le nombre de fois que le son va jouer. Par défaut, le son joue une seule fois et s'arrête une fois le fichier audio terminé. Ici nous avons défini la valeur à *-1* qui indique que le son va jouer en boucle indéfiniment.
- L'attribut *render3D* spécifie si le son doit être spatialisé ou non. Dans notre cas, le piano le sera.
- L'attribut *pos* permet de définir la position initiale de la source sonore dans l'espace virtuel. Dans notre cas le piano est situé sur la scène devant l'auditeur, légèrement sur la droite.

- Les attributs *min* et *max* contrôlent l'atténuation du volume du son en fonction de la distance avec l'auditeur (voir fig. 5 section 2.4.2).

Concernant la réverbération du piano, elle sera simulée de manière simple : nous avons enregistré un fichier audio avec un effet de réverbération pré-calculé sur le même son qui est joué par le piano. Les deux sons (piano et reverb) seront démarré et arrêtés en même temps, en étant synchronisés sur les même évènements *concert.start* et *concert.stop*. Le volume est initialement réglé à 20% du son de piano, correspondant à une reverb audible classique. Avec d'ajuster dynamiquement la balance entre le son direct (piano) et la reverb, nous allons simplement positionner la reverb dans l'espace virtuel (activation du rendu 3D) et utiliser l'atténuation du volume pour jouer sur la balance. Cependant le son reverb étant sensé provenir de tous les cotés à la fois, nous choisissions d'ignorer l'effet du rendu 3D sur la panoramique du son en mettant l'attribut *panFactor* à 0. Cet attribut permet de régler la balance entre rendu 3D et rendu 2D classique sur la panoramique du son. Enfin, le son de reverb sera positionné au fond de la salle de théâtre, derrière l'auditeur : ainsi lorsque l'auditeur se rapproche du piano, la quantité de reverb diminue et le son direct augmente, et inversement lorsqu'il s'éloigne.

➔ *Exemple 3 : Synchronisation de sons avec interactions*

Dans cet exemple, nous allons voir comment tirer parti du système de synchronisation pour réaliser des interactions entre les divers éléments audio. Pour illustrer cet exemple, nous allons prendre le cas d'une visite guidée virtuelle, durant laquelle divers évènements ont lieu pour maintenir l'attention de l'utilisateur. La voix du guide parle en continu, mais peut s'interrompre brutalement pour vous dire de faire attention à un obstacle immédiat, comme une voiture ou un skateboard qui s'avance vers vous. Voici le document correspondant à ce cas :

```
<maudl>
  <sounds>
    <sound id="voix_visite" src="visite.wav" play="guide.parle" pause="attention.started"/>
    <sound id="voix_attention" src="attention.wav"
      play="danger_voiture.started danger_skate.started"/>
    <sound id="danger_voiture" src="voiture.wav" play="évènement.voiture"/>
    <sound id="danger_skate" src="skateboard.wav" play="évènement.skate"/>
  </sounds>
</maudl>
```

Prêtons attention aux divers évènements externe qui contrôlent cette scène audio :

- L'évènement *guide.parle* tout d'abord déclenche la voix du guide pour la visite.
- Les évènements *événement.voiture* et *évènement.skate* déclenchent respectivement les sons de voiture et de skateboard.

Ces trois évènements peuvent être envoyés par l'application au gestionnaire de sons. Mais il faut également noter que les sons en eux-mêmes génèrent des évènements internes qui peuvent être utilisés lors de la description d'une scène audio. Ces événement internes sont de la forme :

```
<id de l'élément ciblé>.<type d'évènement>
```

Voici les différents types d'évènements internes existants :

- *started* : la lecture du son ou de la file d'attente vient de démarrer.
- *ended* : la lecture du son ou de la file d'attente vient de terminer.
- *stopped* : la lecture du son ou de la file d'attente a été stoppée.

- *paused* : la lecture du son ou de la file d'attente a été mise en pause.
- *resumed* : la lecture du son ou de la file d'attente a été reprise après une pause.
- *reset* : la position de lecture du son ou du son courant de la file d'attente a été réinitialisée au début du fichier audio.
- *queued* : le son ou la file d'attente a été ajouté à une file d'attente.

Voyons maintenant comment ces évènements ont été mis à profit dans notre cas. Lorsque les sons *danger_voiture* et *danger_skate* vont commencer à jouer, le son *voix_attention* va également démarrer (utilisation des événements internes *danger_voiture.started* et *danger_skate.started*). De plus, lorsque le son *voix_attention* va démarrer, la voix de la visite sera automatiquement mise en pause (utilisation de l'évènement interne *voix_attention.started*).

L'utilisation d'évènements internes peut ainsi permettre des enchaînements complexe d'interactions entre sons. Dans ce cas volontairement simple, nous aurions bien sûr pu nous contenter d'utiliser uniquement des évènements externes. Mais dans des cas de scènes audio beaucoup plus complexes, si par exemple les son de voiture et de skate pouvaient être déclenchés par de multiples évènements externes différents, l'intérêt de l'utilisation d'évènements internes devient de suite plus évident et facilite grandement l'écriture de document représentant de telles scènes audio.

➔ *Exemple 4 : Utilisation d'une file d'attente*

Dans cet exemple, nous allons voir comment utiliser une file d'attente pour réaliser les opérations d'enchaînement, d'organisation et de filtrage de sons. Pour illustrer cet exemple, nous prenons le cas d'un système de navigation ayant différentes informations à fournir, de nature et de priorités variées. Voici le document correspondant :

```
<maudl>
   <sounds classOrder="danger obstacle info">
      <sound id="trou" src="trou.wav" enqueue="nav.trou:file_nav:-1:5" class="danger"/>
      <sound id="porte" src="porte.wav" enqueue="nav.porte:file_nav:-1:5" class="obstacle"/>
      <sound id="accueil" src="accueil.wav" enqueue="nav.accueil:file_nav:10" class="info"/>
      <sound id="toilettes" src="toilettes.wav" enqueue="nav.toilettes:file_nav:10" class="info"/>
   </sounds>
   <queues>
      <queue id="file_nav" autoPlay="true" timeBase="realTime" maxElements="-1" sort="class"/>
   </queues>
</maudl>
```

On peut voir que 4 sons son définis ici, organisés en 3 classes différentes : danger, obstacle et information. Ces classes sont organisées par ordre décroissant de priorité grâce à l'attribut *classOrder*. Une file d'attente *file_nav* a été déclarée dans le but de recevoir ces différents sons. Celle-ci est configurée pour jouer automatiquement dès qu'elle contient un son (attribut *autoPlay*) et trier les sons par ordre de priorité (attribut *sort*). Elle utilisera une horloge continue (attribut *timeBase*) et peut contenir un nombre illimité d'éléments (valeur *-1* pour l'attribut *maxElements*), puisque nous allons choisir de les filtrer suivant différent critères.

Lors du déclenchement des 4 sons que nous avons définis via les évènements *nav.**, ceux-ci seront ajoutés à la file d'attente *file_nav* avec divers paramètres de validités. Pour rappel, voici la structure de la valeur d'un attribut *enqueue* :

```
<évènement>:<nom de la file d'attente>[:<temps de validité ou -1>:[<distance de validité ou -1>]]
```

On voit dans notre exemple que les sons *trou* et *porte* ont tous deux un temps de validité illimité,

mais une distance de validité fixée à 5 unités (dans notre cas, considérons que ce sont des mètres). A l'opposé, les sons *accueil* et *toilettes* ont une durée de validité de 10 secondes, mais une distance de validité infinie

Lorsque ces sons seront déclenchés durant la navigation, il seront ainsi automatiquement ordonnés selon leur classe de priorité et enlevés de la file d'attente si l'un de leurs paramètres de validité a expiré.

➔ *Exemple 5 : Création d'une hiérarchie de mixage*

Dans cet exemple, nous allons nous intéresser au mixage des différents éléments audio de notre document. Pour illustrer ce cas, nous allons reprendre l'exemple du système de navigation fournissant différentes informations de manière auditive. Voici le document que nous allons étudier :

```
<maudl>
   <sounds classOrder="danger info">
      <sound id="arrivée" src="arrivée.wav" play="nav.arrivée"/>
      <sound id="pas" pick="omitMostRecent" play="nav.pas">
         <subsound src="pas1.wav"/>
         <subsound src="pas2.wav"/>
         <subsound src="pas3.wav"/>
      </sound>
      <sound id="trou" src="trou.wav" enqueue="nav.trou:file_nav:-1:5" class="danger"/>
      <sound id="toilettes" src="toilettes.wav" enqueue="nav.toilettes:file_nav:10" class="info"/>
   </sounds>
   <queues>
      <queue id="file_nav" autoPlay="true" timeBase="realTime" maxElements="-1" sort="class"/>
   </queues>
   <mixers volume="1.0">
      <mixer id="guide" volume="1.0">arrivée, pas</mixer>
      <mixer id="env" volume="0.7">file_nav</mixer>
   </mixers>
</maudl>
```

Ce document est composé de 4 sons : 2 sont liés à la sonorisation du guidage de la personne (*arrivée* et *pas*) et 2 sont liés au rendu de l'environnement (*trou* et *toilettes*). Ces derniers seront joués par l'intermédiaire de la file d'attente *file_nav*, de la manière que dans l'exemple précédent.

Ce qui nous intéresse plus particulièrement dans cet exemple ce sont les élément *mixer* qui ont été ajoutés, pour créer une hiérarchie de mixage. Le volume du mixeur global est ici défini explicitement à 100% (attribut *volume* de l'élément *mixers*), pour illustrer la présence de ce mixeur qui sera dans tout les cas en bout de chaîne. Nous avons ensuite créé 2 sous-groupes de mixages *guide* et *env* qui contiennent respectivement les sons liés aux guidage et à l'environnement. Le mixeur des sons décrivant l'environnement a été configuré avec un volume plus faible que celui destiné au guidage, en raison de son importance moindre. La hiérarchie de mixage finale de cette configuration est représenté par la figure 3 de la section précédente.

Il est donc très facile d'organiser une hiérarchie de mixage complexe au travers de groupes de mixage séparés pour contrôler directement le volume de plusieurs catégories de sons de manière indépendante.

2.4. Pointeur audio 3D

2.4.1. Objectifs

La nécessité principale d'une application de navigation est d'indiquer l'itinéraire à suivre à la personne guidée. Pour se faire, il est donc indispensable de pouvoir lui communiquer la direction à suivre à un instant donné, et de pouvoir lui faire ajuster sa propre direction dynamiquement en cas d'erreur. Pour répondre à ce besoin, nous faisons donc l'usage d'un pointeur audio que nous spatialisons en 3D et positionnions dans la direction que l'utilisateur doit prendre.

L'utilisation d'un tel pointeur a déjà été expérimentée à de nombreuses reprises dans différentes configurations de guidage, notamment dans le cadre de visites virtuelles[7] ou encore pour localiser des objets en intérieur[8].

La qualité de la spatialisation 3D et surtout son effet perçu par l'utilisateur joue ici un grand rôle puisqu'elle sera le critère déterminant sur la précision du guidage.

2.4.2. Techniques de spatialisation 3D

Il existe différentes techniques pour simuler le positionnement 3D d'un son dans l'espace, qui s'appuient sur les différentes paramètres physiques et physiologiques qui permettent à notre oreille d'entendre notre environnement en 3D.

Ces paramètres sont de différentes nature :

- L'intensité du son, qui donne des indices sur distance d'une source sonore, ainsi que sur sa direction de par la différence d'intensité entre chaque oreille (IID, *Interaural Intensity Difference*).
- Le délai entre le son perçu par chaque oreille (ITD, *Interaural Time Difference*), qui donne un indice sur la direction d'une source sonore.
- Les fréquences perçue par chaque oreille, dont la fonction de filtrage se représente par une HRTF (*Head Related Tranfer Function*), liée à la forme du corps et de l'oreille et dont différente pour chaque personne. Celle-ci donne des indices sur la direction ainsi que le positionnement en hauteur d'une source sonore.
- Les premières réflexions (*early reflections*) qui correspondent aux rebonds immédiat du son sur les murs et les objets avoisinants la source sonore, donnent un indice sur la taille de la salle dans laquelle l'auditeur se situe ainsi que sur la position d'objets proches.
- Les réflexions tardives *(late reflection* ou *reverb wash*) correspondent à la conséquence de la réverbération d'une source sonore dans une salle, et donne un indice sur la distance de la source sonore et la taille de la salle.

On peut résumer ces paramètres et leurs effets psychoacoustiques dans le tableau suivant :

		Effets psychoacoustique perçus					
		Gauche/Droite	Devant/Derrière	Haut/Bas	Distance	Taille de la salle	Position d'objet proches
Paramètres	Intensité (IID)	●	(●)		●		
	Délai (ITD)	●	(●)				
	HRTF	●	●	●			
	Premières Réflexions					●	●
	Réflexions Tardives				●	●	

(●) : fournit un indice, mais paramètre non discriminant si utilisé seul

Figure 4: Effet psychoacoustique des divers paramètres audio de spatialisation

Dans le cadre de la réalisation d'un pointeur audio de guidage spatialisé, notre objectif est de pouvoir indiquer une direction à suivre à l'utilisateur. Nous allons donc nous intéresser principalement au rendu de cette direction, qui correspond à l'azimuth dans le plan utilisateur. Le but recherché ici se résumer donc à permettre à l'utilisateur de discerner efficacement la position gauche/droite et devant/derrière d'une source sonore, ainsi que sa distance par rapport à sa destination.

La technique la plus répandue et utilisée en spatialisation audio est le calcul de différence d'intensité entre chaque oreille : il revient à simplement positionner la source dans l'espace stéréo (canaux gauche/droite) en fonction orientation de l'auditeur et de l'angle à la source, puis calculer l'atténuation du volume en fonction de la distance à celle-ci. Cette technique étant simple à mettre en œuvre et peu couteuse en calculs, elle est très adaptée à une usage sur plate-formes mobiles, et est notamment utilisée dans l'API audio OpenAL[8] disponible entre autres sur l'iPhone.

Il existe différents modèles de calcul pour l'atténuation du volume d'une source audio en fonction de son éloignement par rapport à l'auditeur. Nous avons choisi d'utiliser le modèle IDCM (*Inverse Distance Clamped Rolloff Model*), étant celui qui se rapproche du modèle physique réel et recommandé par les *Interactive Audio Rendering Guidelines*[2].

Si l'on définit par *d* la distance, le gain de la source audio *Gain(d)* est calculé par dérivation (fig. 5.1)-(fig. 5.3) et est utilisé pour ajuster le signal audio et aboutir à une atténuation naturelle du volume (fig. 5). Le facteur d'atténuation *R* est utilisé pour augmenter ou diminuer l'effet d'atténuation, suivant les besoins.

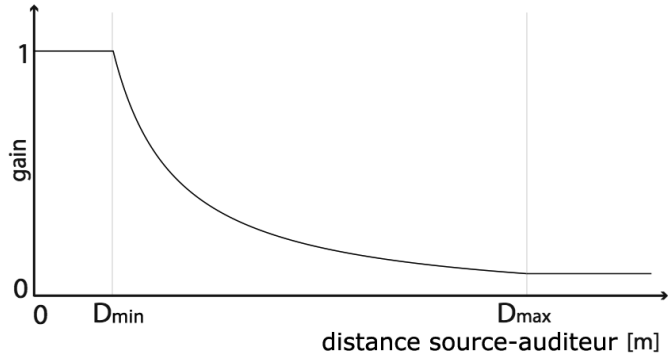

Figure 5: Gain d'une source calculé sur le modèle d'atténuation inverse de la distance

$$d = \max(d, D_{min}) \quad (5.1)$$

$$d = \min(d, D_{max}) \quad (5.2)$$

$$Gain(d) = \frac{D_{min}}{(D_{min} + R \times (d - D_{min}))} \quad (5.3)$$

Concernant le positionnement de la source dans l'espace stéréo, on effectue une simple balance entre les canaux gauche et droite en fonction de l'angle de la source par rapport à l'axe horizontal formé par l'alignement des oreilles de l'auditeur. Cette méthode présente néanmoins un problème : elle ne permet pas de distinguer si une source audio se situe devant ou derrière l'auditeur. En effet, deux source audio en provenance d'un angle identique par rapport à l'axe de l'auditeur forment ce que l'on appelle le cône de confusion (fig. 6). L'auditeur est ainsi incapable de déterminer si les sources formant ce cône se situent devant ou derrière lui. Ce problème est principalement du à la simplification du modèle de tête utilisé (une sphère), et peut se résoudre par une approche tenant compte des paramètres anatomique de la tête de l'utilisateur.

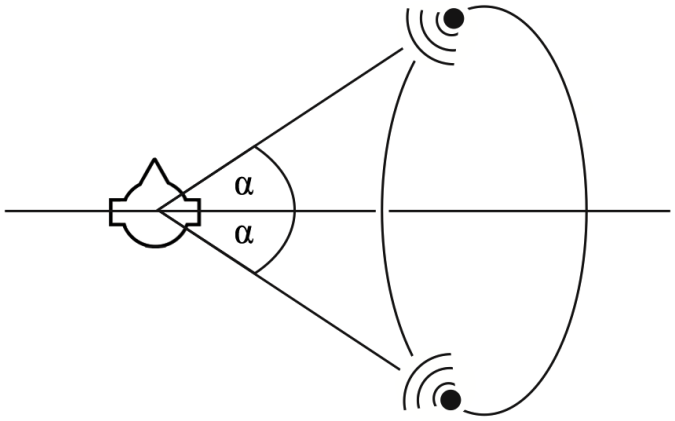

Figure 6: Deux sources avec un angle axe-oreille identique partagent le même cône de confusion

Pour effectuer une spatialisation tenant compte des paramètres anatomiques de la personne on utilise alors une HRTF pour réaliser la fonction de filtrage correspondant à l'auditeur. Le choix de la HRTF n'est cependant pas anodin : celle-ci étant liée à la morphologie d'une personne, cela implique que pour obtenir des résultats optimaux, cette personne devra se faire mesurer son HRTF personnelle à l'aide d'équipement spéciaux dans une chambre anéchoïde, telle que celle présente à l'IRCAM[10]. Il existe des solutions simples pour pallier à cette contrainte, comme par exemple d'utiliser des HRTF généralisée qui fonctionnent pour une tête « moyenne » (avec le désavantage d'être moins précise et de ne pas marcher chez certaines personnes) ou encore de piocher dans des bases de données de HRTF pré-existantes telles que *Listen* de l'IRCAM[11] par exemple, celle qui se rapproche le plus de la personne ciblée.

Cette contrainte mise à part, il existe un autre problème d'ordre technique cette fois concernant l'utilisation de HRTF : l'application de ce filtre nécessite des ressources de calcul importantes, ce qui est problématique dans le cas de notre application sur mobile, la puissance de calcul étant limitée. Une solution existe cependant pour pallier à ce problème : l'utilisation de fichiers audio dans lesquels on aura pré-calculé différentes positions dans l'espace et appliqué la

HRTF correspondante. C'est cette solution que nous avons mis en œuvre pour la création du pointeur audio de guidage.

Le principe est de diviser l'espace circulaire autour de l'auditeur en un nombre défini de points, et de pré-calculer la spatialisation 3D avec HRTF du pointeur audio pour chacun de ces points (fig. 7).

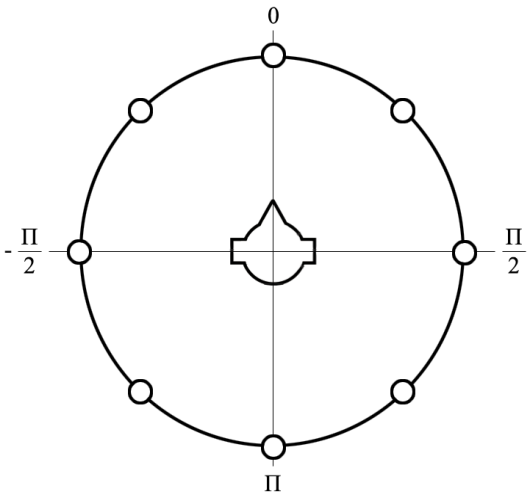

Figure 7: Pré-calcul du pointeur de guidage autour de l'auditeur sur 8 positions

Dans une optique de guidage, un minimum de 8 points sont nécessaires (4 points cardinaux + 4 diagonales). La granularité peut être augmentée pour plus de précision dans le guidage, au prix d'un coût en ressource mémoire plus important.

L'étape finale est ensuite d'effectuer le guidage à proprement parler, en utilisant les multiples échantillons audio. Deux méthodes sont alors envisageables, et nous avons implémenté chacune d'elles pour les expérimenter :

- En considérant que la durée du son du pointeur de guidage est relativement courte (1-2s), on peut se contenter de jouer les échantillons de guidage les uns à la suite des autre, en choisissant à chaque fois l'échantillon se rapprochant le plus de la direction à prendre à l'instant t où le son commence.
- En démarrant l'ensemble les échantillons en même temps, on joue sur le mixage de ses sources pour obtenir par interpolation la direction de guidage souhaitée, de manière continue. A chaque échantillon est affecté un poids P_{yi} correspondant à son volume de mixage (fig 8) et variant suivant l'angle d'orientation de la direction β relatif à la tête de l'auditeur (fig 9).

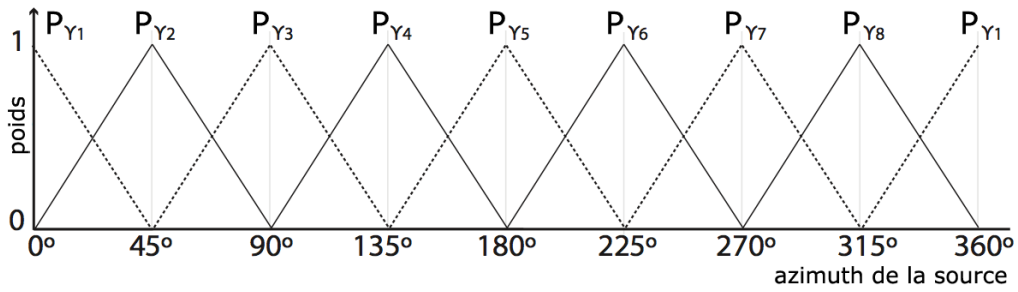

Figure 8: Poids des différents échantillons

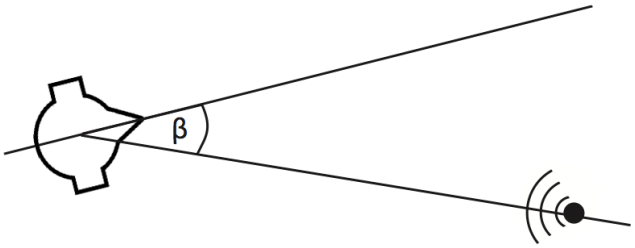

Figure 9: Angle d'orientation β de la source relatif à la tête de l'auditeur

Pour ce qui est du premier cas d'expérimentation, voici son écriture dans le langage MAUDL, à titre d'exemple (seuls les paramètres de synchronisation changent pour l'adapter au second cas) :

```
<maudl>
   <sounds>
      <sound id="pointer" loopCount="-1" pick="manual" render3D="false" play="nav.pointer.start"
        stop="nav.pointer.stop">
         <subsound src="pointer_p0.wav" setNext="pointer.p0"/>
         <subsound src="pointer_p1.wav" setNext="pointer.p1"/>
         <subsound src="pointer_p2.wav" setNext="pointer.p2"/>
         <subsound src="pointer_p3.wav" setNext="pointer.p3"/>
         <subsound src="pointer_p4.wav" setNext="pointer.p4"/>
         <subsound src="pointer_p5.wav" setNext="pointer.p5"/>
         <subsound src="pointer_p6.wav" setNext="pointer.p6"/>
         <subsound src="pointer_p7.wav" setNext="pointer.p7"/>
      </sound>
   </sounds>
</maudl>
```

Ici nous utilisons un seul bloc son nommé *pointer* dans lequel nous ajoutons les 8 échantillons correspondant aux 8 positions pré-calculées du pointeur sous forme de sous-sons. Il est configuré pour jouer en boucle (attribut *loopCount* à -1) avec une sélection manuelle des échantillons (attribut *pick*) et la avons désactivé le rendu 3D, celui-ci étant pré-calculé. Au cours du déplacement, le gestionnaire de position sélectionnera l'échantillon à jouer correspondant le mieux à la direction à prendre en envoyant aux gestionnaire audio un événement de la forme *pointer.p**.

2.4.3. Utilisation d'un head tracker pour améliorer la spatialisation

Au sein du système de navigation actuel, la direction indiquée par le pointeur audio de guidage considère que la tête de l'auditeur est orientée de la même manière que son corps. Si cette supposition fonctionne dans la plupart des cas, il est beaucoup plus naturel pour l'utilisateur de tourner sa tête sans bouger son corps dans une optique de recherche de direction.

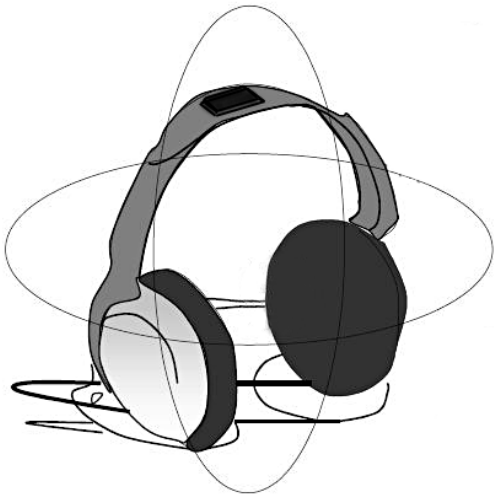

Figure 10: Exemple de dispositif de head tracking utilisant un gyroscope

Afin de compléter notre système de spatialisation, un dispositif de *head tracking* (fig. 10) a ainsi été développé en partenariat avec la société Resonance. Le module matériel se compose d'un accessoire à connecter à l'iPhone constitué de différents capteurs dont notamment un gyroscope et un magnétomètre. Le module, une fois positionné sur la tête de l'utilisateur, renvoie les valeurs valeurs de ces capteurs qui seront utilisées pour corriger l'orientation de la tête de l'auditeur dans la spatialisation : on stabilise ainsi l'espace sonore virtuel et la précision du pointeur de guidage s'en trouve grandement renforcée.

2.5. Rendu environnemental

Parallèlement à la tâche de guidage, l'utilisateur a également besoin de pouvoir connaître différentes informations sur l'environnement qui l'entoure au cours de son déplacement. Pour se faire, nous avons besoin de sonoriser les différents points d'intérêts (POI) signalés sur la carte durant un déplacement. Ces POIs seront ensuite mis en correspondance avec des icônes audio par l'intermédiaire d'une feuille de style audio décrite dans le langage MAUDL précédemment défini.

Il également nécessaire dans la plupart des cas de donner un positionnement dans l'espace à ces POI, et pour cela nous allons donc reprendre certaines techniques de spatialisation que nous avons vues dans la partie précédente. Dans un but de simplicité, pour économiser les ressources matérielles de la plate-forme mobile et ne nécessitant une localisation très précise de ces POI, nous avons choisi de nous contenter d'une spatialisation audio basée sur l'IID améliorée.

Le défaut principal de cette technique étant liée à la confusion devant/derrière des sources sonores, nous allons modifier la courbe de réponse en volume pour y intégrer une atténuation arrière, créant ainsi un effet de focus sur les sources audio positionnés devant l'auditeur (fig. 11).

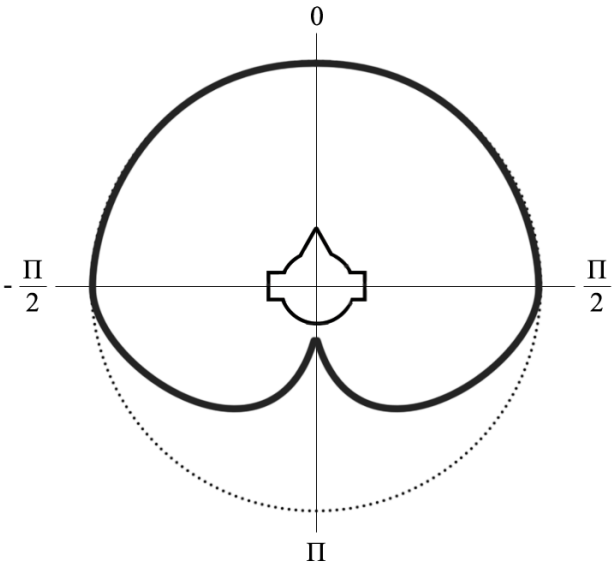

Figure 11: Courbe d'atténuation suivant l'orientation de la source

L'attention de l'utilisateur est ainsi orientée instinctivement vers les POIs sonorisés se trouvant devant lui tout en discriminant par effet de masquage ceux situés derrière.

Concernant l'association des icônes audio aux POIs proches, celle-ci se fera par l'intermédiaire d'évènements nommés qui seront déclenchés par le gestionnaire de localisation. La feuille de style audio est ensuite définie de la manière suivant pour sonoriser cet événement :

```
<maudl>
   <sounds classOrder="danger obstacle info">
      <sound id="hole" src="icon/hole.wav" render3D="true" enqueue="nav.poi.hole:audio_icons:-1:5"
        class="danger"/>
      <sound id="door" src="icon/door.wav" render3D="true" enqueue="nav.poi.door:audio_icons:-1:5"
        class="obstacle"/>
      <sound id="lift" src="icon/lift.wav" render3D="true" enqueue="nav.poi.lift:audio_icons:10"
        class="info"/>
      <sound id="toilets" src="icon/toilets.wav" render3D="true" enqueue="nav.poi.toilets:audio_icons:10"
        class="info"/>
   </sounds>
   <queues>
      <queue id="audio_icons" autoPlay="true" timeBase="realTime" maxElements="-1" sort="class"/>
   </queues>
</maudl>
```

Dans cet exemple nous avons défini 3 classes de priorité pour les icônes audio, ordonné par ordre décroissant d'importance :

- *danger*, qui correspond à l'indication d'un danger immédiat.
- *obstacle*, qui correspond à l'annonce d'un obstacle proche.
- *info*, qui correspond à une indication uniquement informative.

Ces icônes audio sont automatiquement ajoutées à la file d'attente audio_icons dès lors que le gestionnaire de position signale le passage de l'utilisateur à proximité d'un POI correspondant par l'intermédiaire d'un événement de la forme *nav.poi.**. Grâce à la spécification de priorité de ces classes (attribut *classOrder*) et de par la configuration du tri des sons par classe de priorité spécifié

dans la file d'attente (attribut *sort*), les objets audio seront ordonné automatiquement par ordre d'importance dans la file.

Nous avons également utilisé ici les attributs de validité d'un son dans la file d'attente pour configurer le comportement de ceux-ci : pour les POIs de type danger ou obstacle, tant que l'utilisateur se trouve à proximité il a besoin d'en être informé, c'est pour cela que la durée de validité est infinie (-1) et la distance de validité est fixée à 5m. Concernant les POIs de type information, la distance importe moins par contre la priorité étant de délivrer les icônes audio prioritaires en premier, si le l'information tarde trop à venir elle sera évincée (durée de validité fixée à 10s) pour ne pas encombrer la file d'attente audio. Etant d'importance moindre, si ces icônes audio ne sont pas jouées cela n'influera pas sur le guidage ou la sécurité de l'usager. De plus, le gestionnaire de position peut de toutes façons être interrogé à la demande si l'usager souhaite absolument obtenir cette information et qu'elle a été évincer par le déclenchement automatique.

Concernant la file audio, elle est configurée ici de manière standard, pour jouer automatiquement les sons dès leur ajout à celle-ci, fonctionnant en temps réel pour ce qui est du filtrage suivant la durée de validité des sons ajoutés et pouvant contenir un nombre illimité de sons (-1), ceux étant configurés pour être de toutes façons éliminés au bout d'une certaine durée ou distance parcourue.

3. Gestionnaire de sons

3.1. Présentation et organisation

Sdf

bla

3.2. La librairie *libmaud*

3.2.1. Fonctionnement de l'API

sdf

3.2.2. Utilisation et exemples

sdf

3.2.3. Dépendances et limitations actuelles

sdf

3.3. Integration avec les autres gestionnaires

3.3.1. Envoi d'évènements par messagerie différée

sdf

3.3.2. Synchronisation alternée

sdf

4. Références

[1] iOS, mobile operation system by Apple, http://www.apple.com/ios/

[2] Interactive Audio Rendering Guidelines Level 2, *Interactive Audio SIG*, http://www.iasig.org

[3] Synchronized Multimedia Integration Language, *W3C*, http://www.w3.org/TR/SMIL2/

[4] Interactive XMF, *Interactive Audio Special Interest Group*, http://www.iasig.org./wg/ixwg/

[5] *An Interactive Audio System for Mobile*, Y. Lasorsa, J. Lemordant, 127th AES Convention.

[6] FMOD, *Firelight Technologies Pty, Ltd.*, http://www.fmod.org

[7] Thomas Knott. CORONA - *Implementation and Evaluation of Continuous Virtual Audio Spaces for Interactive Exhibits. Master's thesis*, RWTH Aachen University, 2009.

[8] S. Sandberg, C. Håkansson, N. Elmqvist, P. Tsigas, F. Chen. Using 3D Audio Guidance to Locate Indoor Static Objects. In *Proceedings of the Human Factors and Ergonomics Society 50th Annual Meeting*, pp. 1581-1584, 2006.

[9] OpenAL (*Open Audio Layer*), http://connect.creativelabs.com/openal/default.aspx

[10] Measuring HRIR, *Listen HRTF Database*, IRCAM, http://recherche.ircam.fr/equipes/salles/listen/system_protocol.html

[11] Listen HRTF Database, IRCAM, http://recherche.ircam.fr/equipes/salles/listen/

OpenSLES, http://www.khronos.org/opensles/

Evaluation of Spatial Displays for Navigation without Sight, J. Marston, J. Loomis and all, ACM Transactions on Applied Perception, V3, N2, 2006.

5. Annexes

5.1. Schéma de validation Relax-NG du format MAUDL

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
   Relax-NG validation schema for the Mobile Audio Language [maudl]
   - Version 1.1
   - Last modification: 02/02/2011
   - Author: Lasorsa Yohan, INRIA Research Center, Team WAM
-->
<grammar ns="maudl" xmlns="http://relaxng.org/ns/structure/1.0"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
   <start>
      <element name="maudl">
```

```
        <!--
            Sounds are the playable audio elements.
            The sounds container can configure the properties of the 3D rendering for the sounds
        -->
        <optional>
            <element name="sounds">
                <!-- The global 3D settings. -->
                <ref name="global3DSettings"/>
                <ref name="prioritySettings"/>
                <zeroOrMore>
                    <ref name="sound"/>
                </zeroOrMore>
            </element>
        </optional>
        <!--
            Queues allow to play an ordered list of sounds or queues, one after the other.
        -->
        <optional>
            <element name="queues">
                <zeroOrMore>
                    <ref name="queue"/>
                </zeroOrMore>
            </element>
        </optional>
        <!--
            Mixers allow to perform the submix of a set of sounds, queues and mixers.
            A sound or mixer can only have one parent mixer.
        -->
        <optional>
            <element name="mixers">
                <!-- Master mix settings. -->
                <ref name="mixAttributes"/>
                <zeroOrMore>
                    <ref name="mixer"/>
                </zeroOrMore>
            </element>
        </optional>
    </element>
</start>

<!-- Global 3D settings -->
<define name="global3DSettings">
    <!-- The listener 3D position. -->
    <optional>
        <attribute name="listenerPos">
            <list>
                <data type="double"/>    <!-- x, default = 0.0 -->
                <data type="double"/>    <!-- y, default = 0.0 -->
                <data type="double"/>    <!-- z, default = 0.0 -->
            </list>
        </attribute>
    </optional>
    <!-- The listener 3D orientation (the direction he is looking at). -->
    <optional>
        <attribute name="listenerLookAt">
            <list>
                <data type="double"/>    <!-- vx, default = 0.0 -->
```

```
            <data type="double"/>    <!-- vy, default = 0.0 -->
            <data type="double"/>    <!-- vz, default = 1.0 -->
         </list>
      </attribute>
   </optional>
   <!-- The rolloff model for the volume used in 3D rendering. -->
   <optional>
      <attribute name="rolloff">
         <choice>
            <value>log</value>      <!-- default realistic logarithmic attenuation model -->
            <value>linear</value>   <!-- linear attenuation model -->
         </choice>
      </attribute>
   </optional>
   <!-- The rolloff attenuation scale factor. -->
   <optional>
      <attribute name="scale">
         <data type="double"/>    <!-- default = 1.0 -->
      </attribute>
   </optional>
</define>

<!-- 3D settings of an audio source -->
<define name="mix3DSettings">
   <!-- The audio source 3D position. -->
   <optional>
      <attribute name="pos">
         <list>
            <data type="double"/>    <!-- x, default = 0.0 -->
            <data type="double"/>    <!-- y, default = 0.0 -->
            <data type="double"/>    <!-- z, default = 0.0 -->
         </list>
      </attribute>
   </optional>
   <!-- Render the audio source in 3D (if disabled, the audio source will be rendered using the standard
      2D model). -->
   <optional>
      <attribute name="render3D">
         <data type="boolean"/>    <!-- default = false -->
      </attribute>
   </optional>
   <!-- The audio source's minimum volume distance. When using the standard log rolloff model, this will
      define how loud is the sound in the 3D space. -->
   <optional>
      <attribute name="min">
         <data type="double"/>    <!-- default = 1.0 -->
      </attribute>
   </optional>
   <!-- The audio source's maximum volume distance after which the volume stops attenuating. -->
   <optional>
      <attribute name="max">
         <data type="double"/>    <!-- default = 100.0 -->
      </attribute>
   </optional>
</define>

<!-- Priority settings -->
```

```
<define name="prioritySettings">
   <!-- The priority classes. -->
   <optional>
      <attribute name="classOrder">
         <list>
            <data type="string"/>    <!-- Names of classes ordered by highest to lowest priority. -->
         </list>
      </attribute>
   </optional>
</define>

<!-- Attributes of a mixable audio object -->
<define name="mixAttributes">
   <!-- The volume of the audio object, from 0.0 (silence) to 1.0 (full volume). -->
   <optional>
      <attribute name="volume">
         <data type="double"/>   <!-- default = 1.0 -->
      </attribute>
   </optional>
   <!-- The pan of the audio object, from -1.0 (full left) to 1.0 (full right). Ignored when 3D rendering is
      used. -->
   <optional>
      <attribute name="pan">
         <data type="double"/>   <!-- default = 0.0 -->
      </attribute>
   </optional>
</define>

<!-- Attributes of a playable audio object -->
<define name="playableAttributes">
   <!-- The priority class of the audio object. If the class does not appear in the classOrder attribute, its
      priority is considered as the lowest, equally with the others of its kind. -->
   <optional>
      <attribute name="class">
         <data type="string"/>
      </attribute>
   </optional>
      <!-- Determines the behavior of the audio object when it is already playing and a new play request
      has been received. -->
      <optional>
         <attribute name="replay">
            <choice>
               <value>restart</value>        <!-- Restart playing the audio object from the beginning
                                                 (default value). -->
               <value>ignore</value>         <!-- Ignore the new play request. -->
            </choice>
         </attribute>
      </optional>
   <!-- Space-separated list of events that will start the playback of the audio object. -->
   <optional>
      <attribute name="play">
         <ref name="eventList"/>
      </attribute>
   </optional>
   <!-- Space-separated list of events that will pause the playback of the audio object. -->
   <optional>
      <attribute name="pause">
```

```
            <ref name="eventList"/>
        </attribute>
    </optional>
    <!-- Space-separated list of events that will reset the playback of the audio object. -->
    <optional>
        <attribute name="reset">
            <ref name="eventList"/>
        </attribute>
    </optional>
    <!-- Space-separated list of events that will stop the playback of the audio object. -->
    <optional>
        <attribute name="stop">
            <ref name="eventList"/>
        </attribute>
    </optional>
    <!--
        Space-separated list of events that will enqueue the audio object.
        The events have additional parameters as follows:
       <event>:<queue_id>[:<max_delay>[:<max_distance>]]
        - event        : the event that will trigger the enqueue action
        - queue_id     : the id of the target queue
        - max_delay    : the maximum acceptable time before playing the audio object, in seconds (default
       = -1, which means indefinite time)
        - max_distance : the maximum acceptable distance from from initial position at the time of the
       event the listener has moved before playing the audio object, in distance unit (default = -1, which
       means indefinite distance)
    -->
    <optional>
        <attribute name="enqueue">
            <ref name="eventList"/>
        </attribute>
    </optional>
</define>

<!-- A sound object -->
<define name="sound">
    <!--
        A sound is an implicit mixer for 1 or more subsounds.
        Subsounds are introduced to allow for sound variations, but are only seen as being part of one
       sound.
    -->
    <element name="sound">
        <!-- The unique ID of the sound. An ID must be defined for event interactions and explicit mixing of
       this sound. -->
        <optional>
            <attribute name="id">
                <data type="ID"/>
            </attribute>
        </optional>
        <!-- Number of times the sound should repeat playback. -->
        <optional>
            <attribute name="loopCount">
                <data type="int"/>      <!-- Default = 0. If set to -1, the sound will repeat undefinitely. -->
            </attribute>
        </optional>
        <!-- Determines how the subsounds are picked. -->
        <optional>
```

```
            <attribute name="pick">
               <choice>
                  <value>ordered</value>        <!-- Subsounds are played in order, one after the other
                                                (default value). -->
                  <value>reversed</value>       <!-- Subsounds are played in reverse order. -->
                  <value>random</value>         <!-- Subsounds are played in random order. -->
                  <value>omitMostRecent</value>       <!-- Subsounds are played in reverse order,
                                                          excluding the most recently played subsound. -->
                  <value>fixed</value>          <!-- The currently active subsound does not change (unless
                                                explicit active request from a subsound). -->
               </choice>
            </attribute>
         </optional>
         <!-- The playable attributes of the sound. -->
         <ref name="playableAttributes"/>
         <!-- The mix attributes of the sound. -->
         <ref name="mixAttributes"/>
         <!-- The 3D mix setting of the sound. -->
         <ref name="mix3DSettings"/>
         <!-- The audio source file of the implicit first subsound. -->
         <optional>
            <attribute name="src"/>
         </optional>
         <zeroOrMore>
            <ref name="subsound"/>
         </zeroOrMore>
      </element>
   </define>

   <!-- Subsound object -->
   <define name="subsound">
      <element name="subsound">
         <!-- The unique ID of the subsound. An ID must be defined for event interactions with the
         subsound. -->
         <optional>
            <attribute name="id">
               <data type="ID"/>
            </attribute>
         </optional>
         <!-- The audio source file. -->
         <attribute name="src"/>
         <!-- Space-separated list of events that will set the current sound as the active one. -->
         <optional>
            <attribute name="setNext">
               <ref name="eventList"/>
            </attribute>
         </optional>
         <!-- The mix attributes of the subsound. -->
         <ref name="mixAttributes"/>
      </element>
   </define>

   <!-- Queue object -->
   <define name="queue">
      <element name="queue">
         <!-- The unique ID of the queue. -->
         <attribute name="id">
```

```
            <data type="ID"/>
         </attribute>
         <!-- Time base of the queue. -->
         <optional>
            <attribute name="timeBase">
               <choice>
                  <value>playtime</value>      <!-- The time is based on the addition of the duration of the played audio objects, which means pausing the queue also pause the time for the queue (default value). -->
                  <value>realtime</value>      <!-- The time is based on the continuous absolute time, which means pausing the queue has no effect on the time for the queue. -->
               </choice>
            </attribute>
         </optional>
         <!-- Sorting of the audio objects. -->
         <optional>
            <attribute name="sort">
               <choice>
                  <value>class</value>         <!-- Sort the incoming audio objects by their priority class (default value). -->
                  <value>fifo</value>          <!-- Do no sort the incoming audio objects, the oldest entered audio object will always be played first. -->
               </choice>
            </attribute>
         </optional>
         <!-- Automatically starts playing the queue as new audio objects come in. -->
         <optional>
            <attribute name="autoPlay">
               <data type="boolean"/>          <!-- Default value is false. -->
            </attribute>
         </optional>
         <!-- Maximum number of audio objects allowed in the queue. -->
         <optional>
            <attribute name="maxElements">
               <data type="int"/>              <!-- Values <= 0 means no limit (default = 0). -->
            </attribute>
         </optional>
         <!-- The mix attributes of the subsound. -->
         <ref name="playableAttributes"/>
      </element>
   </define>

   <!-- Mixer object -->
   <define name="mixer">
      <element name="mixer">
         <!-- The unique ID of the mixer. An ID must be defined for using this mixer as an input of another mixer. -->
         <optional>
            <attribute name="id">
               <data type="ID"/>
            </attribute>
         </optional>
         <!-- The mix attributes. -->
         <ref name="mixAttributes"/>
         <!--
            Coma-separated list of ID references of sounds or mixers to input in this mixer.
            An audio source (sound or mixer) can only have 1 mixer parent.
         -->
```

```
        <text/>
      </element>
    </define>

    <!-- A list of events. -->
    <define name="eventList">
      <list>
        <oneOrMore>
          <choice>
            <data type="string"/>
          </choice>
        </oneOrMore>
      </list>
    </define>

</grammar>
```

5.2. Exemple de document audio MAUDL

```
<?xml version="1.0" encoding="UTF-8"?>
<maudl xmlns="http://gforge.inria.fr/projects/iaudio/maudl/1.1">

        <sounds listenerPos="1.0 0.0 0.0" listenerLookAt="0.0 1.0 0.0" listenerRearAttenuation="0.5"
        listenerFrontAngle="180" rolloff="log" scale="1.0" classOrder="high low">
                <sound id="music" src="bleep.wav" loopCount="-1" pan="0.0" volume="1.0"
                play="app.start_music" stop="app.stop_music" reset="app.reset_music" replay="ignore"/>
                <sound id="sfx" src="bleep.wav" pick="random" render3D="true" panFactor="1.0"
                pos="1.0 0.0 0.0" play="app.sfx">
                        <subsound src="bleep2.wav" pan="1.0" volume="0.7"/>
                        <subsound src="bleep3.wav" setNext="event.useThisBleep"/>
                </sound>
                <sound id="voice1" src="voice1.wav" loopCount="0" pan="0.0" volume="1.0"
                enqueue="app.voice1:ui_voice:5:-1" class="high"/>
                <sound id="voice2" src="voice2.wav" loopCount="0" pan="0.0" volume="1.0"
                enqueue="app.voice2:ui_voice" class="low"/>
        </sounds>

        <queues>
                <queue id="ui_voice" autoPlay="true" timeBase="playTime" maxElements="5"
                sort="class"/>
        </queues>

        <mixers volume="0.7">
                <mixer id="submix" volume="0.7">music, sfx</mixer>
                <mixer id="premix">submix</mixer>
        </mixers>

</maudl>
```