HAL
open science

Indeterminate Adaptive Digital Audio for Games on Mobiles

Agnes Guerraz, Jacques Lemordant

To cite this version:

Agnes Guerraz, Jacques Lemordant. Indeterminate Adaptive Digital Audio for Games on Mobiles. Karen Collins. From Pac-Man to Pop Music, ashgate, 2008. inria-00191124

HAL Id: inria-00191124

https://inria.hal.science/inria-00191124v1

Submitted on 23 Nov 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Indeterminate Adaptive Digital Audio for Games on Mobiles

Agnès Guerraz, Jacques Lemordant
INRIA Rhône-Alpes, Projet WAM , Zirst - 655 avenue de l'Europe - Montbonnot - 38334 Saint Ismier Cedex - France
{Jacques.Lemordant, Agnes.Guerraz}@inrialpes.fr

1 Audio and Games on Mobiles

A mobile game is a video game played on a mobile phone. The game market for mobiles is clearly regarded as a market with a future [11], as the multiple investments carried out by the large world editors on this segment testify. The mobiles are true platforms of large and general public games: mobile games can be downloaded via the mobile operator's radio network, WLAN, bluetooth or USB connection. Mobiles phones give game developers a unique set of tools and behaviours and with a little creativity, game developers can make some really great games for this platform. The challenges posed by portable devices are numerous, but the biggest complaint of many in the industry is the lack of standards. It is necessary to adapt each game to the various models of existing terminals, which do not offer the same features (memory, processing power, keys...). Consequently, the number of different versions of a game rises to several hundreds. As we will see, Java Micro Edition (J2ME) attempts with some success to solve these problems.

The market for mobile audio products is exploding, creating opportunities for sound designers and content providers. With the mobile phone firmly established as a credible digital music player, mobile music-related applications are becoming an increasingly important component of mobile phone supplier's product offering. As end-users want access to digital entertainment on the move, building as much audio functionality as possible into the mobile phone is logical. Audio is a participatory medium, which actively engages the listener in the ongoing processing of aural information, and no game will be complete without sounds. Sounds constitute an essential part of any game and especially of a game on mobile helping in the immersion feeling of gamers. A convincing combination of sound and image is a key aspect in producing games. Spatialized sound sources, audio special effects, interactive audio, animated digital signal processing (DSP) parameters are the main ingredients for building such environments. However, their successful combination can only be achieved if they are correctly synchronized with the visual world: any interaction on one of the media must be immediately and relevantly associated with a corresponding interaction of the other. Audio technologies for mobile devices are being developed and deployed at an astonishing rate: sophisticated mixers for digital audio have begun to appear on the last generation of mobiles, opening the way to new kind of audio applications. DJ-style applications are now possible and the music experience on the phone is then much more than just playing music tracks. Audio technologies and interactive applications on mobiles will change the way music is conceived, designed, created, transmitted and experienced. We shall see in the

following, how game audio designers can now build complex interactive soundtracks for games on mobiles.

2 Digital Audio on Mobiles

Digital audio is the method of representing audio in digital form. An analogical signal is converted to a digital signal at a given sampling rate and bit resolution; it may contain multiple channels (two channels for stereo or more for surround sound). The digital signal is then compressed in a standard format like MP3, AAC, 3GPP. Digital audio is not the only way to represent audio on mobiles: MIDI is a communication protocol where music is represented as digital data "event messages" such as the pitch and intensity of musical notes to play, control signals for parameters such as volume, vibrato and panning, cues and clock signals to set the tempo. MIDI is notable for its success, both in its widespread adoption throughout the industry, and in remaining essentially unchanged in the face of technological developments since its introduction in 1983.

Generating music by using digital audio chunks is a quite new way to build a soundtrack, which has started to be used for games on console and is now possible on mobile due to the presence of powerful mixers for digital audio. First experiences in generating music from digital audio chunks can be trace back to early 2004 when Kirschner [13], a New-York based composer, used Flash to compose ever-changing pieces of digital music. These compositions generally consist of a number of digital audio files that are randomly layered simultaneously, and that can play for as long as the listener desires. A piece of music that is indeterminate has no inherent end and is therefore well suited for games, which can drive it to give the appearance that no matter what the player does, the music is appropriately supporting the action. Indeterminate adaptive digital audio meets the requirements of mobile phones with limited memory and digital audio mixer. We believe that the experiences of minimalism and contemporary musical forms will be the fundamental building to allow rich audio for multimedia graphics applications (games, navigation and guidance applications…) that can be developed on mobiles.

Most of mobiles are now shipped with a Java™ virtual machine and a java programming environment call Java Micro Edition (J2ME). The Java Community Process or JCP, established in 1998, is a formalized process, which allows interested parties to be involved in the definition of future versions and features of the Java platform. The JCP involves the use of Java Specification Requests (JSR), which are formal documents that describe proposed specifications and technologies to be added to the Java platform. Formal public reviews of JSRs are conducted before the JSR becomes final and is voted on by the JCP Executive Committee. A final JSR provides a reference implementation, which provides an implementation of the technology in source code. There are over 300 JSRs, the more visible JSRs for our concern are:

- Java Advanced Multimedia Supplements API called JSR 234,
- Java Mobile Media API (MMAPI) for Java ME called JSR 135,
- Scalable 2D Vector Graphics API for J2ME called JSR 226,
- Mobile 3D Graphics API for Java ME 1.0 and 1.1 called JSR 184.

An application-programming interface (API) is a code interface that a program library provides in order to support requests for services asked by a computer program. The software that provides the functionality described by an API is said to be an implementation of the API. The API itself is abstract, in that it specifies an interface and does not get involved with implementation details. An API particularly interesting for audio has been defined in a JSR. This API is called Advanced Multimedia Supplements or Java Specification Request (JSR–234). The audio part of JSR-234 is a well designed object oriented framework which enhances the audio support on mobile devices by adding rendering features like 3D audio, special audio effects and virtual acoustics. We will explain how this API can be used to drive the game's music at runtime, according to a set of rules established by the composer/editor. As there is no specific support in JSR-234 for interactive audio, we will have to define new objects to manage time, synchronization and animation of DSP parameters.

3 Content Development for Games

Content development is the process of designing, writing, gathering, organizing, and editing content that may consist of graphics, pictures, recordings, or other media assets that could be used on mobile devices. Content development is the responsibility of audio designers, visual designers and software developers. Audio content development for game must take into account the non-linear nature of games which means that game play length is indeterminate. In [12], Collins discusses in depth the implications of the non-linearity of game audio and music on their aesthetics and functions in the context of games. In addition to spatial acoustics helping to create an environment, the music, dialogue and sound effects help to represent and reinforce a sense of location in terms of cultural, physical, social, or historical environments.

At this stage we can make an analogy with the web at its beginning in 80's and today mobile applications, especially mobile gaming. When the World Wide Web began, web developers either generated content themselves, or took existing documents and coded them into hypertext markup language (HTML). In time, the field of web site development came to encompass many technologies, so it became difficult for web site developers to maintain so many different skills. Content developers are specialized web site developers who have mastered content generation skills. They can integrate content into new or existing web sites, but they may not have skills such as script language programming, database programming and graphic design. As shown in Figure 1, a usual approach to game development takes into account the visual content with graphic animated objects, the audio content with 3D audio sources and sound effects, and the software development linking the audio and graphical objects with game events and rules. Letting the audio designers produce animated audio sources and sound effects and giving them tools to construct these animations is one of the key motivations for designing formats for interactive audio.

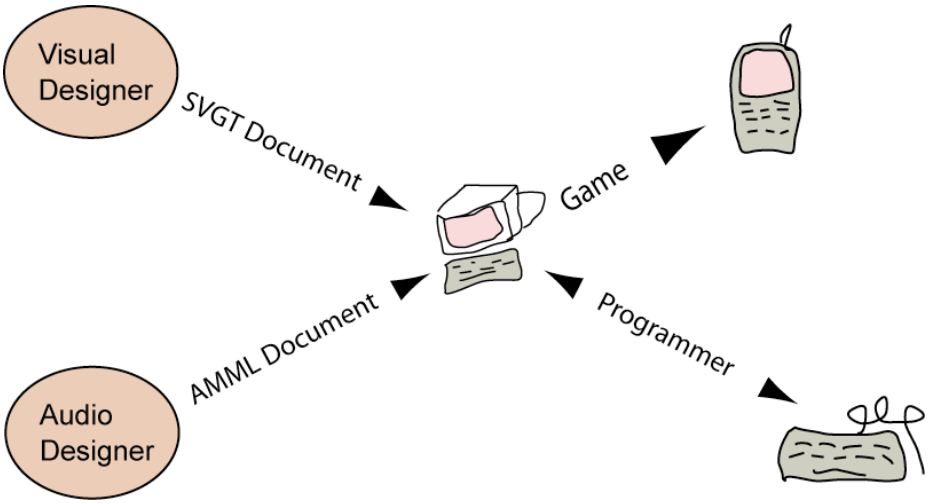

Fig1: Content development.

Audio designer

By audio designers we mean composers and/or sound designers, depending on the application and the content to be built. In the case of a game development, both are needed to achieve a complete audio rendering in the game. Audio designers maintain direct control over the composition and presentation of an interactive game soundtrack. The audio designer recognizes the limitations of the medium and strives to engage interaction between the sound stimulus and the listener's interpretive ability. Frequently, the perception of a message is greatly influenced by the listener's ability to create multi-sensory imagery within the mind. These mental images are formed in response to an analysis of the signal received, and the personal experience background the listener has with the subject or content. In effect, each individual fills-in details beyond the limited audio information provided. The ultimate objective of all the games is to reproduce specific conditions so as to make users feel as if they are in as real a situation as is possible. Sound immersion, especially using headphones, is really amazing. The sound is really all over, from right to left, back to front.

Visual designer

The visual designer conceives, creates assets, and refines the look and feel for the application. This includes, but is not limited to, the components of the application, user interface controls, iconography, and so forth. The visual designer is responsible for providing the conceptual design including typography, colour, branding and graphics. While visual designers may relay concepts through documents, mood boards, and other artefacts, their primary contribution to the development process is graphic assets.

Software developer

The software developer works with the live tree obtained by parsing the documents produced by audio and visual designers. He has to write the logic of the application, to detect collision between visual objects, to get input from the user and to actualize the graph of audio and visual object inside the main loop at the right frequency.

4 Audio Formats and API

There are numerous formats available to make audio contents with more or less music and sound functionalities. A series of documents regarding existing and potential formats for audio content are developed by groups such as the Interactive Audio special interest group (IAsig)[1].

In 1999, the Interactive Audio special interest group published an Interactive 3D Audio Rendering Guideline Level 2.0 (I3DL2) and in 2005, the Synchronized Multimedia Activity of the World Wide Web Consortium (W3C) designed the Synchronized Multimedia Integration Language version 2.0 (SMIL 2.0) for choreographing multimedia presentations where audio, video, text and graphics are combined in real time[4].

As a result, there have been series of important advances and suggestions for overcoming the difficulties involved in creating multimedia content for games. SMIL and I3DL2 can be joined as shown by Kari Pihkala and Tapio Lokki [5]. In their approach, visual and audio elements are defined in a same SMIL document.

More than eight years after the completion of the I3DL2 guidelines, IASIG should announce the completion of a new interactive audio file format to complement I3DL2. This new format, based on the open-standard XMF file format, will be called Interactive XMF (iXMF). The goal of the IASIG in designing this format is to put artistic control into the hands of the artists, keep programmers from having to make artistic decisions, eliminate rework for porting to new platforms, and reduce production time, cost, and stress. The main objects in IXMF are cue events. A cue can be defined as a symbolic name associated to a graph of audio elements producing a continuous soundtrack from discrete media chunks. Clearly, iXMF with its 4-level hierarchical model for mixing and muting (track, chunk, cue, and mixgroups) is a complex low-level file format, which has not been designed with mobiles in mind.

The Synchronised Multimedia Integration Language (SMIL) has been recommended by W3C and allows time based multimedia delivery on the web. It offers a way to synchronise media, text, video, graphics, audio and vector-based animation based on a timeline. SMIL-based timing and synchronization and SMIL-based declarative animation have not been used by the IASIG to define the iXMF format. We think that for mobile (and probably not only for mobile) a format for interactive audio can be defined by extending with SMIL modules the audio object-oriented format (close to I3DL2) specified in JSR 234: SMIL-based timing and synchronization and SMIL-based declarative animation for DSP parameters can be joined with the audio part of JSR-234 to give something with capabilities similar to iXMF and easier to implement on mobile phones.

3D Audio & JSR-234

If we want audio to be a participatory medium, we have to make sounds appear to be located in a three-dimensional space around the user and we have to produce audio effects such as hard sound effects, background sound effects, and collision sound effects. The presence of 3D audio on a device can expand the user's experience beyond the physical dimensions of the device. 3D audio allows the user to hear sounds not just way beyond the left and right sides of the device, but in front and behind the user, and even above and below him.

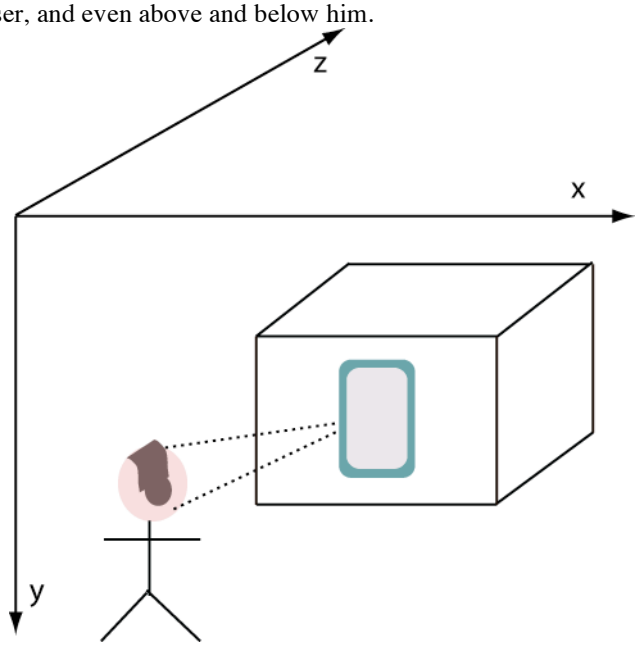

Fig. 2: 3D Audio Rendering

A mobile device can support both stereo microspeakers and stereo headphones, see Figure 2, so that the users can choose how they listen to their spatialized audio (the way the microspeakers are mounted in a mobile device is critical to ensuring a good quality 3D audio [3]). JSR 234, Advanced Multimedia Supplements [2], extends the Mobile Media API (MMAPI, JSR 135) with additional controls for three-dimensional sound and audio effects. This API allows for a kind of effects network of a limited complexity to be built from Java, allowing real time submixing (with various effects being applied to each input before the submix and after the submix). JSR-234 has been designed in a modular fashion and is based on JSR-135's principle of creating a player for a particular piece of media, adding the player to a group or module and then obtaining controls from that module. The JSR 135 is the Mobile Media API. This specifies a multimedia API for Java Micro Edition (J2ME), allowing simple, easy

access and control of basic audio and multimedia resources while also addressing scalability and support of more sophisticated features.

3D effects are accomplished via SoundSource3D objects, which specify a 3D position and other spatial properties. Adding a player to a SoundSource3D object causes its audio content to be spatialized accordingly. Sequences of insert effects and submix groups all require processing power, which at present is still limited on mobile devices. Applications such as mixing, karaoke software, and immersive games with dynamic 3D sounds, will be opened up with JSR 234 API. The content would usually be prepared as a standard monophonic format file (MP3, AAC, 3GPP,…) and the positioning applied at run-time under programmatic control from Java.

Interactive audio and iXMF

iXMF is a public standard structured audio file format that supports cross-platform interchange of advanced interactive audio soundtracks. It uses a queue-oriented model, is programming-neutral, and can be used without license agreements or royalty payments. iXMF is a draft specification currently being finalized in the IASIG and should be ready for public release in 2007. The main concept in iXMF is that of a cue, see Figure 3.

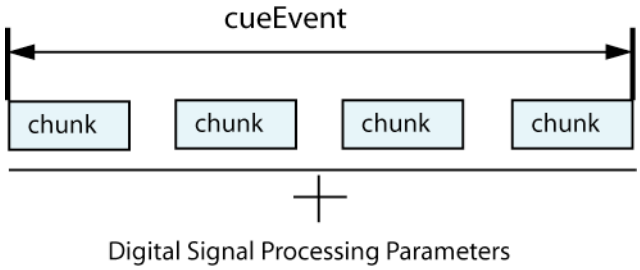

Fig. 3: iXMF Model

Some terms that are used in conjunction with iXMF should be defined. These are "media chunk", "cue request", and "cue". A media chunk is any piece of playable media data. It can be an entire audio file, or a defined contiguous region of an audio file, or either an entire MIDI file or a defined contiguous region within a MIDI file. The continuous soundtrack is built by stringing media chunks together, and sometimes by layering them. A cue request is an event that the game signals to the soundtrack manager, and to which the soundtrack manager responds with a corresponding action designed by the audio artist at the time of authoring. That action is called a cue. A cue can contain any combination of services or operations that the soundtrack manager can perform. In most cases a cue will contain a playable soundtrack element but it may also be used to perform other soundtrack manager functions that do not result in something audible, such as setting a variable, loading media, or executing a callback to the game.

Synchronisation and DSP Animation with SMIL

SMIL is an XML-based language and is a recommendation of the W3C consortium [4]. SMIL does not embed media objects (images, video/audio clips, text...etc): instead, a SMIL presentation consists of at least one XML file and some external media files (which can be distant or available locally). SMIL is composed of many modules, the following two are of particular interest: timing + synchronization, and animation. SMIL is a complex language and requires editing tools that can handle various aspects of the authoring process: layout design, timing specification, encoding support, document preview, and so on. An authoring application for SMIL, called LimSee, has been developed by the WAM team (Web Adaptation Multimedia) at INRIA [6].

5. AMML: A Format for 3D and Interactive Audio

The advantage of using an XML format to specify 3D and interactive audio, is that we have many tools to transform and edit XML documents. Moreover, an XML format can be used as a generic design format as we can generate from it, Java code for JSR-234 or C code for another interactive audio API like Fmod[9]. Another advantage is that a graphical editor for music composers and its associated soundtrack manager can be constructed more easily, by using an XML format as a serialisation mechanism. A soundtrack manager will control the audio playback and will provide the following functionality in response to cue requests:

- Responding to game sound requests by playing appropriate sound media, sometimes influenced by game state;
- Constructing continuous soundtrack elements from discrete media chunks, whether via static play lists or dynamic rules;
- Dynamically ordering or selecting which media chunks get played, sometimes influenced by game state, sometimes to reduce repetition;
- Mixing and/or muting parallel tracks within media chunks;
- Providing continuous, dynamic control of DSP parameters such as volume, pan, and 3D spatial position, sometimes influenced by game state, sometimes to reduce repetition.

In this paragraph, we will show how an XML format for 3D and interactive audio on mobiles can be designed. This format will be called AMML for Advanced/Audio Multimedia Markup Language and the grammar of the format will be control by an XML-Schema. If mobiles are the main target, the best is to have a format the closest possible from the JSR-234 Advanced Multimedia API for 3D audio and the closest possible to SMIL timing and synchronization modules for its interactive audio capabilities.

AMML and 3D Audio

The best way to derive a format for 3D audio from the JSR-234 is through an XML serialization of the JSR-234 API objects. This will have two advantages:

- It will be easier to build a graphical authoring system allowing an audio-designer to produce AMML documents by embedding the JSR-234 engine in the authoring system.
- It will be easier for an application programmer to parse an AMML instance document and map it on the JSR-234 API's objects.

In Fig. 5, we have an UML view with inheritance of the AMML XML-Schema driving the AMML format. The light grey part of the schema corresponds to the serialization of the JSR-234 API and consequently to 3D audio. A cueEvent is the main object of the AMML format. CueEvents are of two types like in JSR-234, *SoundSources3D* for 3D audio rendering and *EffectModule* for audio special effects. The position of the spectator is controlled in a 3D-space and is detained by the *amml* root object. Global environment parameters are also under control of the *amml* root object.

AMML and Interactive Audio

Structuration

The South-East part of the schema, shown in Figure 5, supports the interactive capabilities of the format and is borrowed from the SMIL timing and synchronization modules. We have defined three new types of objects, layer, chunk and chunkExcl (see Figure 4). These three objects are containers helping to structure the data: a cueEvent aggregates layers, and a layer aggregate audio chunks of type Chunk or ChunkExcl. A ChunkExcl is itself a container starting and stopping in random order its players. This brings in the API a 6-level hierarchical model for mixing, muting and rendering composed of the following objects: amml, cueEvents, layers, chunks or chunkExcls, players and tracks.

Synchronization

In the AMML model for interactive audio, the component synchronization is made as in SMIL, with the help of sequential compositions, parallel compositions and timing attributes (*begin*, *end*, *dur*, *repeatCount*, *repeatDur*, …).

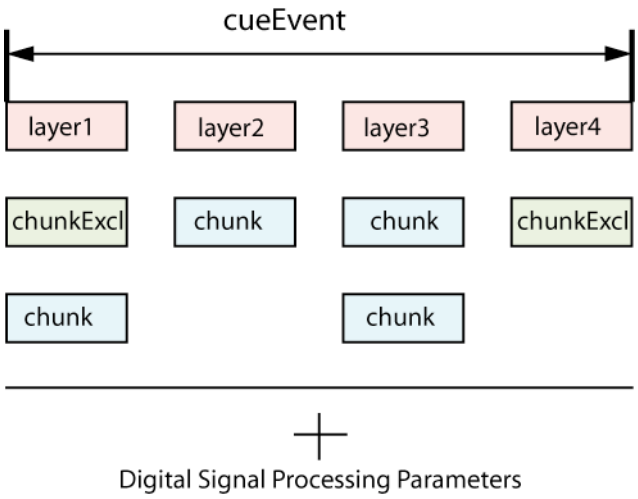

Fig. 4: AMML Interactive Audio Model

The temporal relationships among media elements are determined by time elements, which are of three kinds:

- *CueEvent* elements execute their children in parallel.
- *Layer* elements execute their children in sequence.
- *Chunk* elements of type *ChunkExcl*, execute one child at a time like an iPod in shuffle mode. In fact, an object of type *ChunkExcl* is more than a state machine: although only one child may play at a time, several other children may be queued. SMIL offers an attractive conceptual framework called *priority classes* for doing so. Each priority class is characterized by a *peers* attribute to determine how members of the class would interrupt each other. The possible values are *defer* (don't interrupt, but put the interrupting peer into the queue), *pause*, *never* (don't start the interrupting peer), and *stop* (stop the current peer right away).

Animation of DSP parameters

In order to keep the diagram (Figure 6) readable, we have not shown the important part corresponding to the animation of the DSP parameters. Animation of the DSP parameters is done using XML elements and attributes defined in the SMIL animation module. The example given in paragraph 6 shows how the animation of the reverberation time is specified in the AMML language:

```
<reverbControl>
        <animate attributeName="reverbTime" begin="1" end="3" from="120" to="412"/>
</reverbControl>
```

AMML Schema

An XML-schema can be defined to control and validate AMML documents. The hierarchy of the types comprising this schema is shown using an UML diagram (see Figure 5). One part of the schema is dedicated to 3D audio and the other part (South-East) is dedicated to interactive audio. Extra types, not shown on the figure, are needed to control the animation of DSP parameters

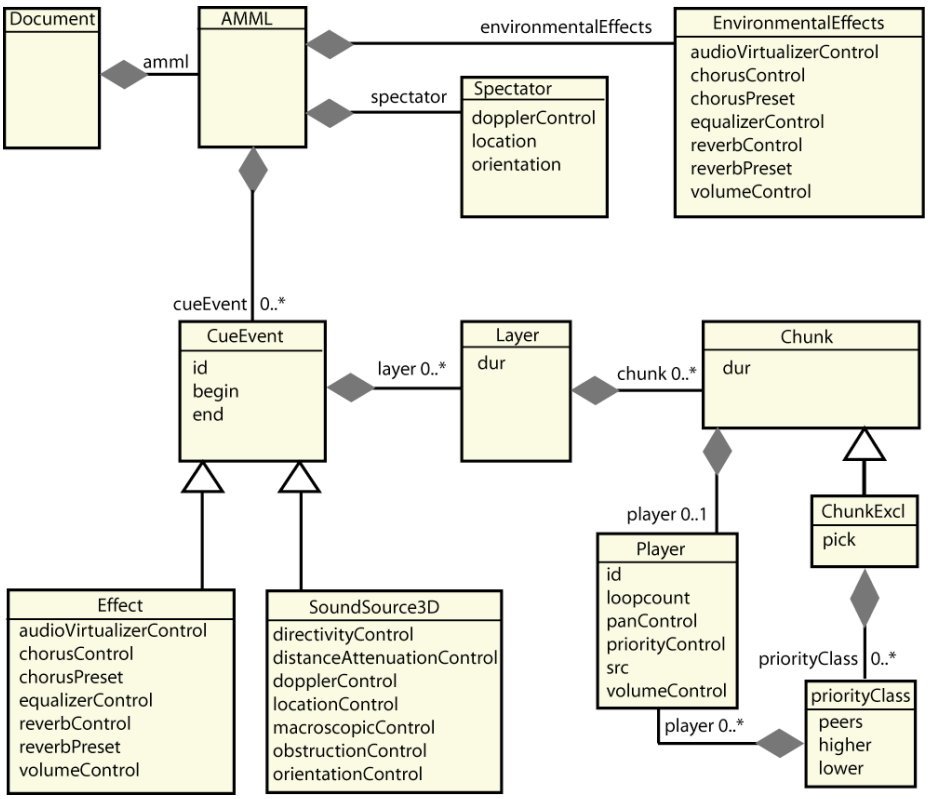

Fig. 5: AMML XML-Schema

6. Example

A detailed example will help to clarify the basic idea. We chose to develop a "Breakout" game for mobile phone with 3D interactive audio. A Breakout game is a well-known example for mobile gaming; see for example a Scalable 2D Vector Graphics (SVGT) implementation of this game on mobile by Vincent Hardy et al [6]. We have tested our implementation on a Nokia N95 phone with Scalable 2D Vector

Graphics for the visual part and with Java Advanced Multimedia Supplements for 3D interactive audio. It was found to run very satisfactorily both for the visual part and the audio part.

A layer of "bricks" lines the top third of the screen as shown on Figure 6. A ball travels across the screen, bouncing off the top and side walls of the screen. When the ball hits a brick, the ball bounces off and the brick disappears. The player loses a life when the ball touches the bottom of the screen, and to prevent this from happening, the player has a movable paddle to bounce the ball back into play.

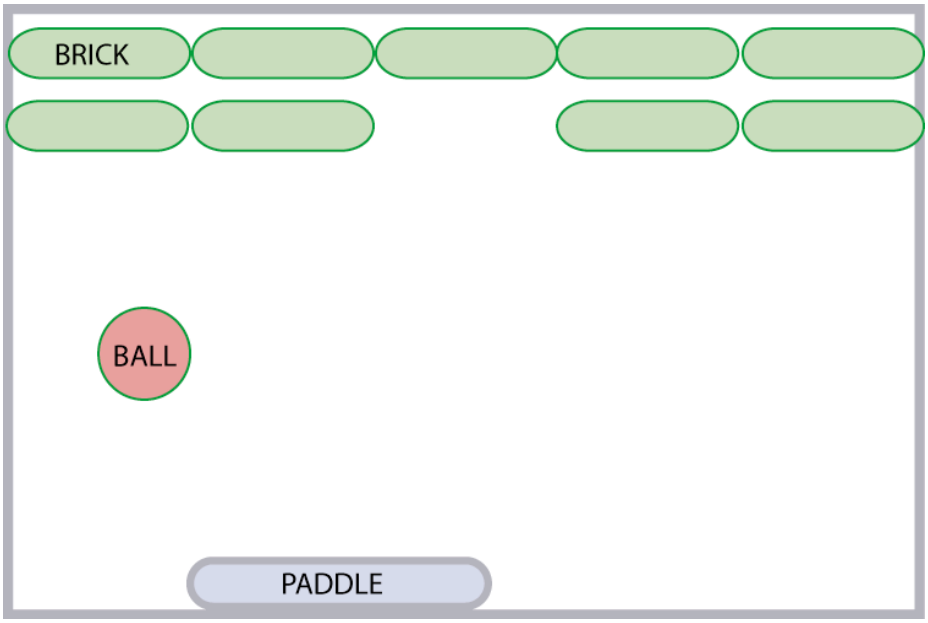

Fig. 6: Breakout Game Screenshot.

SVGT & JSR-226

SVG is a language for describing a two-dimensional graphics in XML. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), multimedia (such as raster images and video) and text. SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively (i.e., by embedding SVG animation elements in SVG content) or via scripting. Industry demand and requests from the SVG developer community has established the need for some form of SVG suited to displaying vector graphics on small devices. In order to meet these demands the SVG Working Group created a profile specification that was suitable for use on mobile devices as well as on desktops. The SVG Mobile 1.1 specification defined SVG Tiny (SVGT) 1.1, suitable for mobile devices, but the absence of scripting in SVG Tiny prompted a

second look at the use of programming languages and scripting. In conjunction with the Java JSR 226 group, a lightweight interface called the microDOM, or uDOM, was developed. With this advance, lightweight programmatic control of SVGT (in Java in case of JSR-226) became feasible on the whole range of platforms from cell phones through to desktops.

6.1 Visual description of the game

The following SVGT document, whose rendering is shown on Figure 6, illustrates the visual part of the game. It contains the game layout, the ball, the brick lines (only one to keep the document simpler to read) and a visual special effect whose id is *collisionFX*.

Example: breakout.svg document

```
<svg width="176" height="208" xmlns="http://www.w3.org/2000/svg"
version="1.2" baseProfile="tiny">
 <defs>
  <rect id="brick" x="0" y="0" rx="7" ry="7" width="33" height="16" fill="rgb(0,255,0)"/>
</defs>
<!--FX-->
<circle id="collisionFX" r="1" cx="80" cy="80" fill="rgb(140,140,0)">
   <animate attributeName="r" begin="collisionFX.focusin" dur="0.8s"
     from="1" to="3"/>
</circle>
<!--background-->
  <rect id="clear" x="0" y="0" width="176" height="144" fill="white"/>
<!-- Game Layout -->
<rect id="top" x="0" y="0" width="176" height="5" fill="rgb(200,200,200)"/>
<rect id="bottom" x="0" y="138" width="176" height="5" fill="rgb(200,200,200)"/>
<rect id="left" x="0" y="5" width="5" height="133" fill="rgb(200,200,200)"/>
<rect id="right" x="171" y="5" width="5" height="133" fill="rgb(200,200,200)"/>
<rect id="perimeter" x="5" y="5" width="166" height="133" fill="none" stroke="red"/>
<!-- lines of bricks -->
  <use xlink:href="#brick" x="5" y="6"  id="brick1"/>
<use xlink:href="#brick" x="38" y="6" id="brick2"/>
<use xlink:href="#brick" x="71" y="6" id="brick3"/>
<!-- paddle -->
<rect id="paddle" x="30" y="126" rx="5" ry="5" width="40" height="12"
  fill="rgb(0, 0, 255)"/>
<!-- ball -->
 <circle id="ball" cx="80" cy="80" r="10" fill="rgb(255, 0, 0)"/>
</svg>
```

6.2 Audio description of the game

The following AMML document illustrates the audio part of the breakout game. We have one link with the *breakout.svg* document, contained in the *LocationControl* element: *breakout.svg#ball*. This link instructs the system to use the location of the ball for the cueEvent whose id is *source3D0*. The user position is specified in the spectator element.

In this game, the audio design is built from two cues, one corresponding to the sound of the ball and the other corresponding to the sound effect associated with the collision between the ball and the paddle. The audio associated to the ball is composed of two layers, one playing music in a shuffle (random) mode. The sound is spatialized according to the position of the ball. The audio associated to the collision has only one layer composed of two chunks which are played sequentially and to enhance the effect, the DSP parameters are animated using an animate element inside a reverbControl element (animation of graphics parameters is done the same way in SVGT).

Example: Breakout.aml document

```
<p:amml xmlns:p="amml">
 <spectator location="-1 0.5 0.5" />
 <environmentalEffects reverbPreset="arena" chorusPreset="flanger">
   <equalizerControl wetLevel="13" modulationDepth="20"/>
 </environmentalEffects>

 <cueEvent xsi:type="SoundSource3D" id="source3D0" begin="0">
   <locationControl location="breakout.svg#ball"/>
   <layer>
      <chunk>
       <player src="ball0.aac" loopCount="-1"/>
      </chunk>
   </layer>
   <layer>
      <chunk xsi:type="excl" pick="random">
       <priorityclass peers="stop">
           <player src="ball1.aac" loopCount="-1"/>
           <player src="ball2.aac" loopCount="-1"/>
           <player src="ball3.aac" loopCount="-1"/>
       </priorityclass>
      </chunk>
   </layer>
 </cueEvent>

<cueEvent xsi:type="Effect" id="effect0">
    <volumeControl level="24"/>
    <reverbControl>
      <animate attributeName="reverbTime" begin="1" end="3" from="120" to="412"/>
   </reverbControl>
   <layer>
       <chunk dur="0.4s">
           <player src="collision0.aac"/>
      </chunk>
      <chunk dur="0.6s">
```

```
            <player src="collision1.aac"/>
        </chunk>
      </layer>
    </cueEvent>
</p:amml>
```

6.3 Programmatic Control

The application follows the Model-View-Controller paradigm as shown in Figure 7 where we have the presentation layer composed of four modules (3DSound Source, Audio effects, Visual Objects and Visual FX), the model composed of two modules (Game and Effects objects) and the control composed of three modules (Game Canvas, Input and Collision Detection).

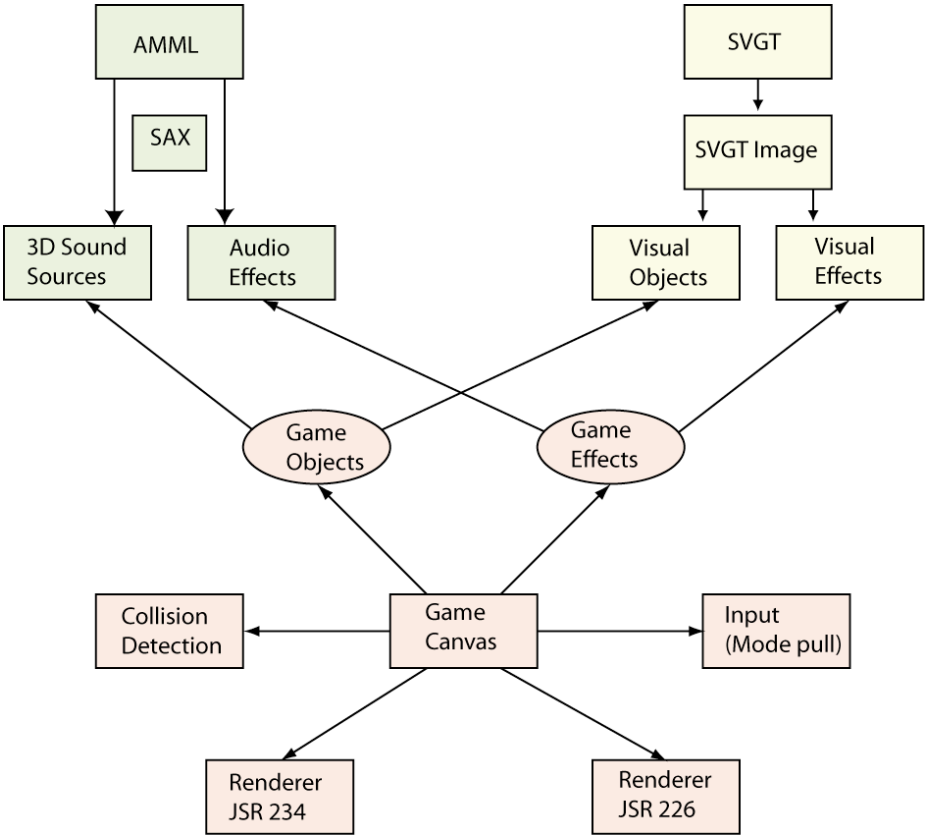

Fig. 7: Programmatic control.

The application accesses the SVGT document through the JSR-226 type called SVGImage. We use a simplified SAX parser to read the AAML document and map

the elements and attributes onto Effects and 3DSoundSources modules of the JSR-234. The Game Canvas is a central object, which has access to all the necessary information in order to control the game. It has a main rendering loop which access player input in a pull mode, detect collision, modify accordingly game and effects objects and activate the audio and graphic rendering at an appropriate frequency.

7 Conclusions

Currently, the history of audio for game is repeating itself on mobile platforms. Started as simple beeps, it progressed to high quality sounds. Indeed, while only a few years ago mobile games produced simple beeps, they are now catching up to the level of audio on video games. We have presented an architecture for games on mobile devices separating content creation (audio and graphics) from content manipulation. The development of several 2D and 3D games for mobiles showed us that a format for 3D interactive audio was missing and this evidence was the motivation for creating the AMML format. In order to describe sound sources, special effects and environments properties, the Advanced/Audio Multimedia Markup Language (AMML) format uses an object-oriented approach, which follows rather closely the advanced multimedia API for mobiles, described in the JSR 234. For describing interactive audio aspects, time and synchronization, animation of Digital Signal Processing (DSP) parameters, we have used the corresponding multimedia SMIL based modules and embedded them in our markup language. We have also show how to link the audio presentation layer with the visual presentation layer. Having a format using XML is a big advantage as it is easy to parse and generate code from it: it should be possible to use this format to reach other interesting devices like game consoles. It is also possible to use this format for building an authoring system for interactive and spatialized audio. This authoring system will allow us to put artistic control in the hand of a sound designer, same as SVGT (M2G) or Mobile 3D graphics (M3G) allows us to put artistic control in the hand of a visual designer using authoring system such as Adobe illustrator or 3ds Max.

References

1. Interactive 3D Audio Rendering Guidelines Level 2, Interactive Audio SIG, available at: http://www.iasig.org
2. JSRs : Java Specification Requests, JSR 234: Advanced Multimedia Supplements, available at : http://jcp.org/en/jsr/detail?id=234
3. Hayden Porter, interview, Sonaptic discusses JSR-234 3D Audio, available at: http://sonify.org/tutorials/interviews/jsr234
4. W3C, Synchronized Multimedia Integration Language (SMIL 2.0), available at: http://www.w3.org/TR/2005/REC-SMIL2-20050107/

5. Pihkala K. and Lokki T., “Extending SMIL with 3D audio”. In: Proceedings of the 2003 International Conference on Auditory Display. Boston, MA, USA, 6-9 July 2003, pages 95-98. 2003.
6. R. Deltour, C. Roisin, “The LimSee3 Multimedia Authoring Model”, DocEng 2006, ACM Symposium on Document Engineering, 10-13 October 2006, Amsterdam, The Netherlands, pp. 173-175.
7. W3C, Mobile SVG Profiles: SVG Tiny and SVG Basic, available at : http://www.w3.org/TR/SVGMobile/
8. Suresh Chitturi, Michael Ingrassia, Vincent Hardy, Building portable and scalable mobile applications using JSR226, TS-7105, JavaOne 2005.
9. Fmod, sound designer tools, available at: http://www.fmod.org/
10. Creative Technology Ltd., “Environmental Audio Extensions: EAX 2.0,” EAX 2.0 Extensions SDK, available at: http://developer.creative.com
11. Trip Hawkins (Founder & CEO, Digital Chocolate), Making Mobile Phones the Ultimate Game Platform, Games Developers Conference, 2007
12. Collins, Karen (2007). “An Introduction to the Participatory and Non-Linear Aspects of Video Games Audio.” Eds. Stan Hawkins and John Richardson .
13. Marc Weidenbaum, Musique for Shuffling, Kenneth Kirschner’s interview, available at : http://www.disquiet.com/kirschner.html