Examensarbete
LITH-ITN-MT-EX--05/059--SE

Adaptive Music System for DirectSound

Sebastian Aav

2005-12-01

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Science and Technology
Linköpings Universitet
SE-601 74 Norrköping, Sweden

Institutionen för teknik och naturvetenskap
Linköpings Universitet
601 74 Norrköping

LITH-ITN-MT-EX--05/059--SE

Adaptive Music System for DirectSound

Examensarbete utfört i Medieteknik
vid Linköpings Tekniska Högskola, Campus Norrköping

Sebastian Aav

Examinator Björn Gudmundsson

Norrköping 2005-12-01

Avdelning, Institution
Division, Department

Institutionen för teknik och naturvetenskap

Department of Science and Technology

Datum
Date

2005-12-01

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English

☐ _____

Rapporttyp
Report category

Examensarbete
☐ B-uppsats
☐ C-uppsats
☒ D-uppsats

☐ _____

ISBN

ISRN **LITH-ITN-MT-EX--05/059--SE**

Serietitel och serienummer
Title of series, numbering

ISSN

URL för elektronisk version

Titel
Title

Adaptive Music System for DirectSound

Författare
Author

Sebastian Aav

Sammanfattning
Abstract

With the intention of surveying the field of research in adaptive audio systems for interactive media, a suggested audio system design for adaptive music control is described, and a prototype implementation of key parts of the system is presented and evaluated.

Foregoing midi-triggered sound banks, the proposed design uses layered segmented audio files, defined and controlled by XML-scripts. The results demonstrate an inclination of a flexible system, capable of adequate adaptive behaviour of high quality sound.

The implemented system will serve as an extensive basis for future work contributing to the research of adaptive behaviour to both music and sound effects for interactive media, and also as a preliminary foray into the more experimental field of stand-alone non-linear music playback.

Nyckelord
Keyword

Adaptive Music, Adaptive Audio, DirectSound, DirectX

Master Thesis Report in Media Technology
at Linköping Institute of Technology, University of Linköping - Campus Norrköping

Adaptive Music System for DirectSound

Sebastian Aav

Norrköping 2005-12-07

sebaa241@student.liu.se

Abstract

This thesis has been done at the Linköping Institute of Technology, University of Linköping – Campus Norrköping, with the intention of surveying the field of research in adaptive audio systems for interactive media. A suggested audio system design for adaptive music control is described, and a prototype implementation of key parts of the system is presented and evaluated.

Foregoing midi-triggered sound banks, the proposed design uses layered segmented audio files, defined and controlled by XML-scripts. The results demonstrate an inclination of a flexible system, capable of adequate adaptive behaviour of high quality sound.

The implemented system will serve as an extensive basis for future work contributing to the research of adaptive behaviour to both music and sound effects for interactive media, and also as a preliminary foray into the more experimental field of stand-alone non-linear music playback.

Preface

This report is a part of the master thesis in Media Technology at the Linköping Institute of Technology, University of Linköping – Campus Norrköping.

The prototype implementation has been done with Microsoft Visual C# 2005 Express Edition Beta (.NET 2 Beta). The XML Serialization support that is part of the .NET 2 Beta Framework is used to parse the XML-scripts. Microsoft DirectX 9.0c SDK has been used for DirectSound functionality.

Note: Throughout, references to "He" is aimed at both "He/She". When referenced to computer games as a target application for adaptive audio, other media are also targeted. Computer games are at the forefront of adaptive audio, but that doesn't deter the fact that adaptive audio can be used in a wide variety of circumstances (art installations, non-linear music etc.)

Contents

1 Background

Adaptive Music (also known by the questionable term Interactive Music) is today primarily used in computer games. The difference between computer games and movies is that in a game - the events are under the players influence, and can be different each time, while a movie doesn't change no matter how many times it is played.

Music in games can take many forms and scales, and it is up to the developers as well as the composer to establish how and where the music should adapt to the events of the game. The music can respond to simple scenarios such as changing entire music cues when the player character confronts an enemy, or more complicated ones such as subtly introducing or changing instrumentation when the player character runs or walks through changing environments. The complexity of situations requires a music system that can perform an ample number of tasks, while providing a comprehensive design for the composer to work with.

As the game is changing or adapting to the players decisions and actions, so does the music. This is important knowledge to the composer. Instead of creating a linear piece of music for any given situation, the composer creates several different cues of music and then must figure out how to seamlessly transition musically between them. This is hard, and few people have done it well. The notion of composing music that can play indefinitely in one state and then transition seamlessly to another is far removed from traditional linear composing.

Pioneering work in game audio was done in the 90's when LucasArts graphical adventure games used a proprietary system for adaptive music control. The *Interactive MUsic Streaming Engine* or *iMuse*, used General MIDI (and later streamed audio files) – and was a way to manage all the different music cues and ascertain a relation between cues via a graphical layout as well as specifying transitions. *iMuse* was developed specifically for graphical adventure games, where a character roam through locations that took up a whole screen at a time – which meant that the game as well as the music was designed like a flowchart. *iMuse* was used in about eight LucasArts titles and was at the forefront of adaptive soundtracks of its time. (*iMuse Island*, 2005-11-14)

However the approach to adaptive control of the music in games today is often implemented poorly if at all and this does not have to be a purely technological problem; since the actual composing of an adaptive soundtrack is very hard to do successfully so many game developers does not even consider adaptive audio control worthwhile to implement. Thus, a modern general idea of non-proprietary tools for the sound designer/composer is yet to be established, and the first steps towards such are marred by difficulties. The main problem from the beginning days of the industry remains the prime problem to date; how to make the music not sound repetitive. But there is also the problem of how to create tools which a composer can use to design the music for adaptive control.

Recently, two important techniques have handled music in games: DirectMusic (using MIDI+DLS) and Linear Audio.

1.1 DirectMusic

The most popular and accessible platform available today is the PC/XBOX that uses the Microsoft DirectX API, and thus the audio part of DirectX – namely DirectSound and DirectMusic – is the most used today. There are of course other platforms and API´s and sound systems available – such as Miles Sound System (which is one of the oldest and quite popular) and OpenAL (a cross-platform audio solution). While OpenAL (*OpenAL, 2005-11-12)* is free, Miles Sound System (*The Miles Sound System*, *2005-11-20)* has to be licensed for use. OpenAL is more focused upon 3D-only sound rendering – which makes its use in terms of interactive music fairly limited.

DLS, or DownLoadable Sounds, are the primary reason for the success of DirectMusic, and the term MIDI+DLS is a fairly straightforward idea: Unlike the sound bank system used with General MIDI (originally the sound bank was a very small set of instruments on the sound card ROM), the DLS instruments are located on the computers hard drive – making it possible to use higher quality and, more importantly, custom made samples for the MIDI to trigger.

Since DirectX is a popular API, many game producers build their own audio systems and/or adaptive music systems upon its low-level DirectSound. Of course, there is the possibility of using the built in functions of DirectMusic, however a broad audience of users have found it extremely hard to work with (especially its authoring tool DirectMusic Producer) – and it doesn't always suit the needs of the music of their specific game.

However, despite its inherent complexity, DirectMusic can be considered as the first and important attempt of an adaptive music standard tool using the technique of MIDI+DLS. (*DirectX 9 Audio Exposed*, 2004)

1.1.1 Pros

- Broad interactive possibilities. DirectMusic is very feature rich. You can do amazing things with interactivity in games if you have the tenacity to figure it all out.
- Extensive possibilities for variable music. With a combination of variations, scripting, and chord maps, you can produce scores that surprise you even when you have heard the piece countless times and thought you had heard all the possible variations.
- DirectMusic content is flexible and can be independent of the game engine, allowing you to tweak many things without having to rebuild the game.

(*Game Audio Pro*, 2004-08-09)

1.1.2 Cons

- DirectMusic development has stopped at Microsoft. This means DirectMusic is quickly becoming dated, and things that should have been fixed or improved are still broken, badly implemented, or counter intuitive.
- DirectMusic is not easy to learn. There are many concepts that can be difficult to grasp.
- DLS instruments are hard to make sound good. This is because all the DLS samples must be loaded into RAM and are limited to whatever amount of RAM/CPU the development team decides to use for music. If you try to counter this by using a lot of effects you are going to use too much CPU power. It is very difficult to match

composers using state of the art samplers or live orchestras with only 5-12 MB of DLS.

- DLS is not a popular format. To acquire good sounds, a lot of time is probably going to have to be spent creating all the instruments from scratch or converting from other sample libraries.
- DLS licensing/encryption is a great difficulty. Many major sample companies do not want their samples in DLS collections to ship in a game. If you can work something out with the sample companies, making the content secure is a time consuming task for the programmers.
- Wave tracks are buggy and badly implemented. Lately DirectMusic composers have switched to using wave tracks so they can avoid DLS issues and get higher quality sound. But there is much to improve here.
- The authoring tool DirectMusic Producer is highly counter-intuitive and difficult to comprehend.

(*Game Audio Pro*, 2004-08-09)

1.2 Linear Audio

The simplest possible way of presenting music in a game is actually what is often most resorted to today. Simply playing CD- or MP3 audio tracks and changing to another track of music when encountering an action sequence. This change is often a jarring experience since there is no inherent system for transitional cues to smooth out the transitions between two music cues. It's mostly like flipping through the tracks on a CD player. (*Game Audio Programming*, 2003)

The reason for it being a popular choice for music presentation today is that linear audio is of high sound quality – there are no restrictions to available amount of memory such as when using DLS-instruments.

1.3 Comparison

Linear audio gives the highest possible quality of sound – but with apparent limited adaptability/variation. DirectMusic (with its MIDI+DLS) gives a substantial control of the music – even on note by note level – however it can lack in sound quality.

Reviewing the different techniques and tools for creating and playing back interactive music in games, the only real "off the shelf" tool publicly available would be DirectMusic. But its drawbacks are beginning to make composers and sound designers to steer away from it. This leads to the question if something can be done in the field of interactive music; music that has the degree of interactivity found in MIDI+DLS, and the sound quality on the same level of linear audio.

One solution is a sort of combination of the concepts of MIDI+DLS and linear audio. As an example, a parallel is drawn to the workings of a symphony orchestra:

The hardest sounds to emulate on computers are those of acoustical instruments or ensembles, for example a symphony orchestra. A high degree of realism and *real* sound quality of the orchestra can be achieved by recording the orchestra playing chunks of music – and then transitioning from chunk to chunk. This is a simple, yet useful way of handling the music – however it can be improved upon. Break down the music of a

symphony orchestra and one quickly ascertains the building blocks of the orchestra – the different instrument groups or *sections*.

The symphony orchestra normally consists of strings, woodwinds, brass and percussion - each of these is a section of the orchestra (these sections can of course be broken down in more detailed groupings if one would like; violins, violas, cellos, double basses, trombones, trumpets, french horns, etc). Instead of having a single audio chunk playing and then transition to the next, each section of the orchestra is playing in separate audio channels. The reason for this is *variation*; consider that the music is embedded in a single channel, and the whole orchestra performs the music – the amount of variation is none, the next time the music plays it is exactly the same. However, if we separate the orchestra into four sections; and record each section playing the same music as before – the possibility of playing back these sections in any combinations arise. It is still the same amount of music material, but since it is broken up into five sections – the increased number of musical combinations is substantial. The opportunity to create enough variation to keep the material fresh is now prevalent, which is of help against repetitiveness.

In the case of a symphony orchestra broken up into our five sections – the maximum polyphony of five channels (ten if stereo tracks are used) is playing at the same time. Compare this to a strictly MIDI+DLS approach (with one or two channels playing *one note* at any time) where the polyphony must be tightly restricted to not overwhelm the CPU. The drawback is that while MIDI+DLS offer an almost unlimited adaptiveness to the game (on note per note level), the approach of four sections playing has not the same kind of immediate adaptiveness. However the variations of these five sections are adequate for a skilled composer to render the music varied.

1.4 Other techniques

The only off-the-shelf product that exists today in terms of game audio is DirectMusic and DirectMusic Producer. However game developer studios almost always use proprietary software and tools made specific for their uses. And until the next attempt at a game audio standardization is developed by the game industry – the situation today is a substantial lack of competitive game audio management techniques. This thesis project intends to contribute to that particular field of research.

1.5 Next Generation Platforms

As of this writing, Microsoft has released a Beta of XACT – *Cross-Platform Audio Creation Tool* which is an audio authoring tool aimed at the XBOX360 and PC platforms. This is what is most likely to replace Microsoft's previous DirectMusic; however a complete overview of the capabilities of the XACT is currently unknown to the author of this thesis report. (*MSDN Beta*, 2005-11-20)

1.6 Glossary

Cue	Music accompanying a scene, environment or setting. In the audio engine, a cue is music for a specific intensity level.
Transition cue	In the audio engine, a transition cue is a cue specifically for transition music.
MIDI	Musical Instrument Digital Interface. Commonly used as an interface that enables a musical instrument, such as an electronic keyboard, to communicate with sequencer and/or samplers.

Metronome	A device designed to mark exact time by a regularly repeated tick. In the audio engine, a metronome beat is for instance used to correctly synchronize segments to the tempo.

2 Purpose

The purpose of this thesis project is to propose an adaptive music system for interactive applications. This report will describe the workings of the design requirements of the system, as well as describe the implemented prototype application that is intended as an assessment of the method of the proposed design.

Foregoing midi-triggered sound banks, the audio engine uses layered segmented audio files, defined and controlled by XML-scripts. The system attains a level of flexible and adequate adaptive behaviour using a responsive transition system and high quality audio playback.

In regards of this report; firstly the design of the proposed system will be described. Secondly the actual implemented prototype application will be discussed and evaluated. Finally, assessments of future work will be made.

3 System Design

This is a design description of the audio system.

3.1 Overview

The adaptive music system consists of several parts: scripts, audio engine, audio player and API connectivity for external application.

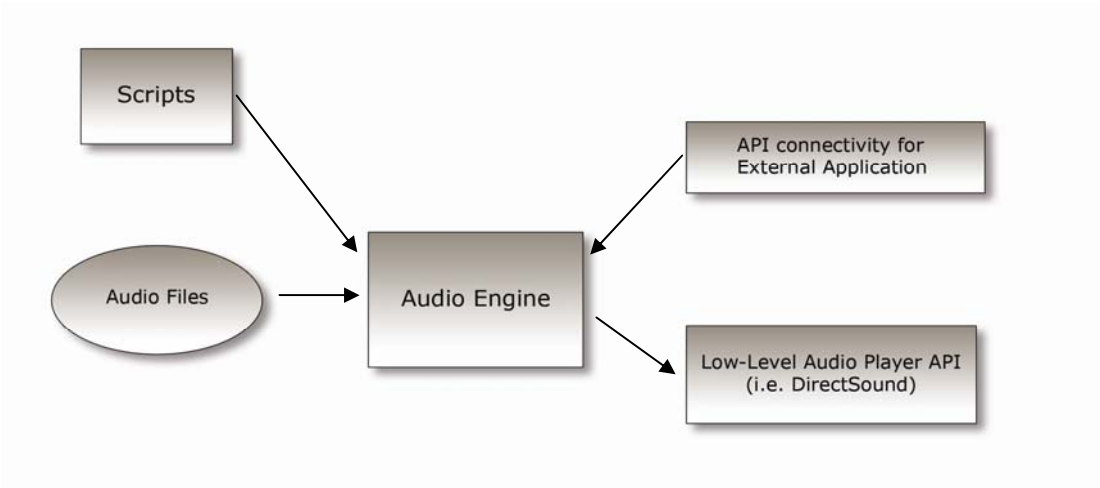

Fig 1. System overview.

Furthermore, two GUI applications provide scripting functionality and audition playback capabilities.

3.1.1 Audio Engine

In the system, the audio engine, programmed in Visual C#, is the centrepiece. It contains the functionalities for parsing the XML scripts and decides which audio files to play according to these scripts and events from the external application (such as a game) using the system. The Audio Engine receives events from the game, as well as parses the scripts and issues proper calls to the audio player API.

3.1.2 Scripts

The audio engine is driven by the XML-scripts, which contains all rules, definitions and resources for all the music in the game. This design is data-driven, meaning that audio resources and rules are never hard-coded into the audio engine; they are completely defined by separate data. This enables the composer to change the behaviour of the music and also replace audio files without having to rebuild the whole game application.

3.1.3 Integration and exterior application connectivity

While the audio engine uses C#, the application (i.e. the game) itself might be written in a different language, commonly C++. Thus calls to the audio engine are handled through a wrapper API which provides proper connectivity.

3.1.4 Low-Level Audio Player

The audio engine issues calls through an API to the correct low-level audio provider used (i.e. DirectSound). Through this separation, an API to another audio playback (for instance OpenAL) can eventually be implemented without having to rebuild the audio engine.

Volume for mono/stereo audio channels can be controlled in real-time. The system supports initial panning of each audio channel.

3.1.5 Audio Authoring and Audition Tools

It would be possible to script everything manually with an XML-editor; however this is somewhat daunting and prone to human errors. Therefore, a tool provides the user (sound designer/composer) with the ability to automatically assemble the scripts through the GUI.

Testing the scripting is made through an additional audition tool which simulates the events from the game engine with a GUI – enabling the audio designer to try out if their scripted transitions, cues etc. are working correctly without having to run it through the game engine.

3.2 Design concepts

3.2.1 Intended user

The design concepts of the system are of importance not only for its actual implementation structurally, but also to the intended user of the system. Whether or not the composer or sound designer has programmer-knowledge – it is important that the tools and concepts are reasonable and usable. However, to successfully compose an adaptive score, even with the help of the tools and the design notion of the script logics – is hard from a musical point of view. A certain amount of skill is required to compose an adaptive music score, and that is a skill that is quite different from linear composing.

3.2.2 Groups and Cues

Since a computer game features all kinds of music for numerous occasions, the music that underscores an environment in one part of the game might be different than in another part of the game. As the player moves through game environments, the music follows. Perhaps the composer wants the music to change when the player enters a particular room. There are numerous of situations of where the music can change. The two most important concepts in the audio system that is used when the music needs to change are *Groups* and *Intensity Levels*.

A *group* is a set of *cues* (each with corresponding *intensity level*) that is used as a compilation of music for a particular scene or environment or state in the game. Different Intensity Levels are used in groups where the music still belongs to the group but have different musical intensity.

As a hypothetical example, adaptive music shall be added to the Google Earth system (*Google Earth*, 2005-11-13). Google Earth is an application which lets the user zoom in on earth and view geographical locations of interest by the help of combined satellite imagery and maps. Supposed adaptive music functionality would for example be in the form of appropriate music for countries and locations – as well as zoom-dependant intensity shifts. So music for a certain part of Asia has specific music which has different intensities depending on how far in the user has zoomed. For this case, each geographical-based music has specific groups (such as Asia, Africa, Ireland etc.) and consists of, for instance two or three intensity levels for the zoom-levels – first intensity level for when zoomed out and then higher levels for when zoomed in.

Suppose the user starts out with a zoomed out view of Ireland. The composer wants a change in intensity of the music when the user zooms in. So he creates two cues in the group *Ireland* called *Ireland1* and *Ireland2*. Each of these cues are designated an intensity level – *Ireland1* has intensity level 1 and *Ireland2* intensity level 2.

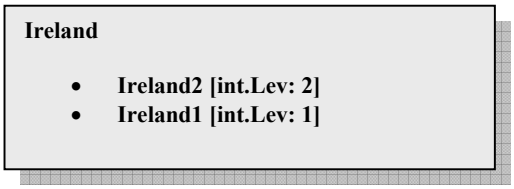

Fig 2. A group named "Ireland" with two cues.

The composer can for instance use fairly sparse and ambient music of low intensity playing in *Ireland1* while music for *Ireland2* is much more up tempo.

3.2.3 Start- and Transition group

There are two reserved groups that are available: *startGroup* and *transitionsGroup*. A start group is the first group that the audio player plays upon activation of the music engine. The start group consists of a cue that only plays once, and then transitions to a designated cue of another group. So to continue our example – the user hypothetically always finds himself at a zoomed out view over Ireland. So the start group with a *startCue* is added and

at the end of *startCue* the composer sets it to transition directly to *Ireland1* in the group *Ireland (see fig 3).*

The *startCue* is part of its own group but uses a transition event in its pattern to change cue (and implicitly - its group) to the stated cue. Therefore it does not use a group transition matrix (*see 3.2.4 Group Transitions*), it transitions to the new cue without additional transition cues.

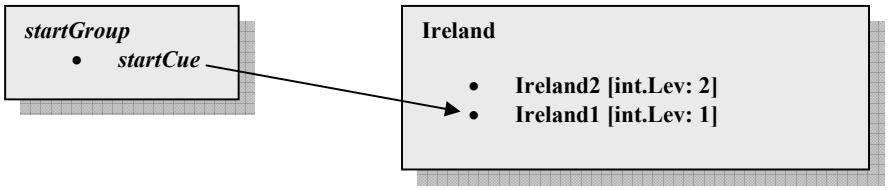

Fig 3. Transition from start cue.

A transition between *Ireland1* and *Ireland2* is needed, so as the user zooms in on Ireland – the music will play a *transition cue* that the composer can use to smooth the transition to the target cue. All transition cues are located in the reserved group *transitionsGroup.*

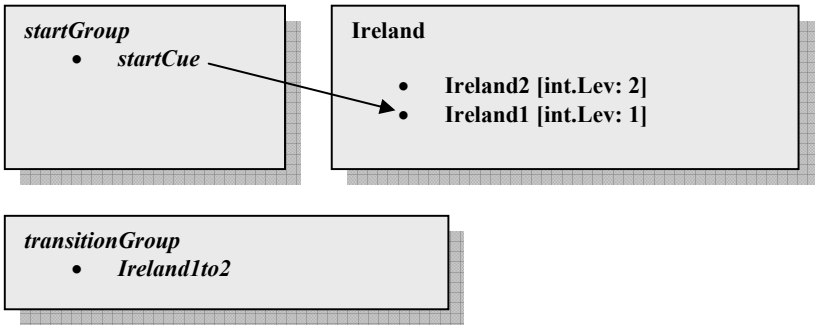

Fig 4. The transition group contains all transition cues.

So when the user zooms in, the application triggers an event that the music should increase its intensity level. With the help of the groups own transition matrix (*see Appendix B - Transition Matrix, and Appendix C – Transition Matrix Script*), the audio engine locates the correct transition cue in consideration of its current group and target intensity level. In this example, the *Ireland1to2* plays. Transition cues do not loop, and will play the target cue immediately after it has finished - so after *Ireland1to2* has played once, it immediately starts playing *Ireland2.*

Transition Matrix		Ireland1	Ireland2
	Ireland1	-	***Ireland1to2***
	Ireland2	***Ireland2to1***	-

Fig 5. The Transition matrix for Ireland (see fig 4) with an added transition from Ireland2 to Ireland1. When the intensity level is raised, the audio engine uses this matrix to find which transition cue corresponds to a transition from start- and target cue.

3.2.4 Group Transitions

Any number of groups can be added. For example, a group *Africa* can be added which has music for the continent of Africa. Instead of ending *Ireland* with an end cue and starting the music engine over again but with *Africa* instead – a *group transition* can be used in its place. A group transition is just like any other transition cue and is also located in the *transitionsGroup*. However the group transition cue is set to transition to a cue in another group.

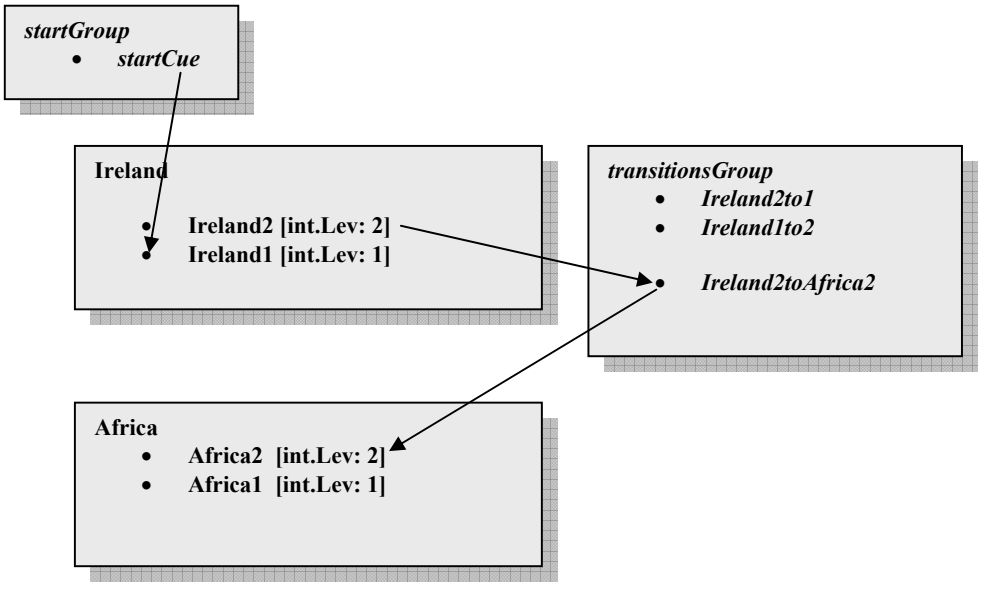

Fig 6. Transition between two cues in different groups goes via a corresponding transition cue in transitionsGroup.

A group transition cue can also be designated to transition to a cue which has a different intensity level. For example: *Ireland2toAfrica1* is a group transition from *Ireland2* to *Africa1*. The concept of group transitions enables the music to theoretically never actually stop playing in a game – the music can just keep transitioning to new groups.

3.2.5 Cue variation with Patterns and Tracks

Cues contain information such as intensity level, tempo, length, patterns, tracks and more. Variations of a cue are achieved by the concept of patterns, tracks and tags. When a cue plays, a pattern is randomly chosen using non-standard randomizer (*see 3.2.7 Non-standard randomization*). The principle of track-based variations is established with *Tracks* and *Tags*. The principle can be best described as "patterns within patterns". The basic idea is that each track holds a specific set of segments, each with a designated *Tag* Id (*see 3.3.2 Cues*), which are related. Any number of these sets of segments can exist, but in each set – only one *tagId* (with corresponding segments) can be played.

For example, the composer has designated four tracks – *brass*, *strings*, *woodwinds* and *percussion*. In the *brass* track there exists a number of segments – each of these has a designated *tagId* (note that several segments can lay under one tag, giving them all an identical *tagId*). During playback, the audio engine chooses a pattern and then randomly selects one tag to play *for each track*.
This could mean that the following simultaneous playback occurs:

Track: Brass	**Play Tag:** 1
Track: Strings	**Play Tag:** 5
Track: Woodwinds	**Play Tag:** 3
Track: Percussion	**Play Tag:** 2

The resulting playback is a pattern that has a variation to itself – without this feature; every pattern would play back the same way every time.

The concept of tracks is actually an example of how composers often construct their music as each instrumental groupings of, for example, an orchestra can be given a specific part. Of course the tracks designation is up to the composer and any other examples of tracks are possible:

Percussion, *Bass*, *Guitars*, *Backing Vocals* and *Lead Vocals*
Or
Synthesizers and *Drums*
Or
Background, *Middleground* and *Foreground*
Etc.

The composer can also forego track-based variation if he for some reason want the music to play back the same way each time the pattern plays – this will then be scripted as just one single track with only one tag.

3.2.6 End Cues

Another type of cue is the *End Cue*. Any cue in a group can have a corresponding (shared intensity level) end cue which plays if the game engine triggers the audio engine to end the music. The end cue is played once and then the music engine stops the metronome beat, discontinuing the audio playback.

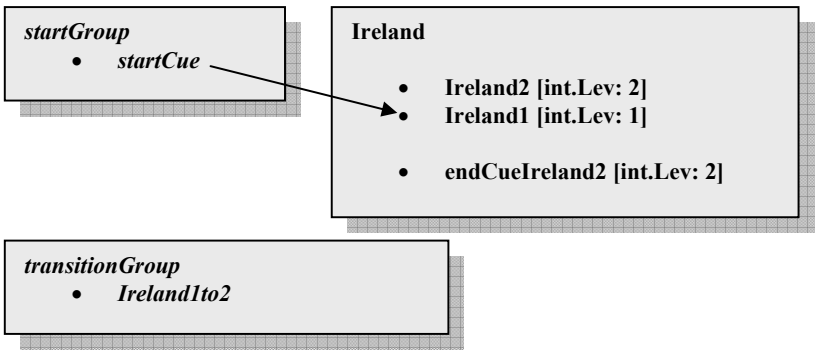

Fig 7. An added end cue in "Ireland". Note that it shares intensity level with its corresponding cue.

3.2.7 Non-standard randomization

Ordinary randomization when selecting patterns (as well as selecting tags; *see 3.3.2 Cues*) would sometimes result in the same pattern being played more than once in sequence. To remedy this, the selection procedure uses a non-standard randomization.

This means that the audio engine keeps track of which patterns have already been played, resulting in that before a pattern is repeated all patterns in a cue must have played once. (*Audio For Games – Planning, Process, and Production*, 2005)

3.3 Scripting

Having established the concept of the music system, the scripts describe these structures of groups, cues, patterns and transitions etc.

A number of XML-files are used to contain the scripts*: resources.xml, cues.xml, transitions.xml, groups.xml, grouptransitions.xml*

3.3.1 Resources

The resources script is a listing of all the audio file resources used in the cues. Audio files are called segments in the cues. The *resource.xml* can contain any number of segments, and each of these descriptions holds information about filename, tempo, initial volume setting etc. Also of importance are the properties of the file itself, such as what type of compression is used and at what sample rate and bit rate etc, which are used by the audio engine to set up the streaming buffers correctly. The segment is referred with a segmentId, which is used in the pattern events of the cues in *cues.xml*

```
<Resources>
        <Segments>
            <Segment>
                <segmentId>drums00</segmentId>
                <filename>drums00.wav</filename>
                <timesignature>4/4</timesignature>
                <segmentTempo>84</segmentTempo>
                <segmentLength>1</segmentLength>
                <volume>-6.0</volume>
```

```
                <pan>0</pan>
                <type>wav</type>
                <bitrate>16</bitrate>
                <samplerate>44100</samplerate>
                <channels>2</channels>
            </Segment>
            ...
        </Segments>
</Resources>
```

Fig 8 Part of the resources.xml script, showing the definition of a segment.

The definition of a segment includes:

segmentId	A referenced string Id for the segment.
filename	Path to the audio file.
timesignature	Describes which time signature the music is in.
segmentTempo	Tempo of the segment.
segmentLength	This describes how long the segment is in measures.
volume	Volume attenuation in dB (decibel).
pan	Panning of the segment (-100 to +100 ; 0 being in center and -100 and +100 being far left and right respectively).
type	If and what codec is used (MP3, Ogg-Vorbis, uncompressed WAV etc.)
bitrate	File bit rate.
samplerate	File sample rate in kHz.
channels	Number of channels. (Mono, Stereo)

3.3.2 Cues

The *cues.xml* is the script that designates how the patterns play, which segments should be triggered when etc. Properties such as *cueId*, *intensity level* and *tempo* etc. are designated. Secondly, the *play list* is defined: it consists of any number of *patterns*, each referred with an Id. When a cue plays, a pattern is randomly chosen and is played back. Each pattern can have two types of events, *play* events and *transition* events. Play events is straight-forward: it refers to a *segmentId* that shall play at a specific beat in the cue. Transition events refer to another cue which shall start playing the beat *after* the referred beat. Transition events are used in transition-cues as well as the start cue.

```
<Cues>
        <Cue>
            <cueId>Ireland1</cueId>
            <intensityLevel>1</intensityLevel>
            <tempo>80</tempo>
            <timesignature>4</timesignature>
            <cueLength>1</cueLength>
            <transitiongrid>2</transitiongrid>
            <playlist>
                <pattern>
                    <patternId>1</patternId>
                    <track>
                        <trackId>"RythmSection"</trackId>
                        <tag>
                            <tagId>1</tagId>
                            <play>
```

```
                    <playSegmentId>bodhran00</playSegmentId>
                    <playatBeatNumber>1</playatBeatNumber>
                  </play>
                  <play>
                    <playSegmentId>cymbals0</playSegmentId>
                    <playatBeatNumber>1</playatBeatNumber>
                  </play>
                  <play>
                    <playSegmentId>claps0</playSegmentId>
                    <playatBeatNumber>1</playatBeatNumber>
                  </play>
                </tag>
                ...
              </track>
            ...
          </pattern>
        </playlist>
      </Cue>
      ...
</Cues>
```

Fig 9 Part of the cues.xml script, showing the definition of a single cue.

The definition of a cue includes:

cueId	A referenced string Id for the cue.
intensityLevel	The designated intensity level of the cue.
tempo	The tempo of the cue.
timesignature	number of beats per measure in the cue.
cueLength	Cue length in measures.
transitionGrid	Beat division for transitions. *Ex: "1" designates that the cue can transition on every beat. A "4" designates that the cue can only transition on every fourth beat etc.*
patternId	A referenced Id for the pattern.
trackId	The track Id.
tagId	The tag Id for a group of segments.

Script commands for describing the pattern events:

play Play a segment (referenced by its segment Id) at a certain beat number.

```
...
    <play>
        <playSegmentId>guitar0</playSegmentId>
        <playatBeatNumber>1</playatBeatNumber>
    </play>
...
```

transition Transition to a cue (referenced by its cue Id) at the beat following the calculateTransitionAtBeatNumber.

Note that transition events are located in a specific track with a reserved name "transitionTrack", which always looks like this:

```
...
<track>
```

```
<trackId>transitionTrack</trackId>
  <tag>
  <tagId>1</tagId>
      <transition>
        <playCueId>cueB</playCueId>
        <calculateTransitionAtBeatNumber>6</calculateTransitionAtBeatNumber>
      </transition>
  </tag>
</track>
...
```

Fig 10. Part of the cues.xml script, showing the definition of the reserved transition track.

3.3.3 Transitions

The intra-transitions in a group are located in *transitions.xml*. This is the aforementioned transition matrix depicted in XML (*See Appendix B - Transition Matrix, and Appendix C – Transition Matrix Script).*

3.3.4 Groups

The Groups script (*groups.xml*) is a listing of all the groups as well as all the corresponding cues as well as the number of intensity levels contained in a group. It is used for the audio engine to keep track of which cue belongs to which group. Reserved groups for use for the starting cue as well as the transition cues are labelled *startGroup* and *transitionsGroup* respectively. Of note is that the reserved groups (*startGroup* and *transitionsGroup)* do not use intensity levels.

```
<groups>
    <group>
          <groupId>Ireland</groupId>
          <intensityLevels>2</intensityLevels>
          <cues>
              <cue>
                <cueId>Ireland1</cueId>
              </cue>
              <cue>
                <cueId>Ireland2</cueId>
              </cue>
              <cue>
                <cueId>endCueIreland1</cueId>
              </cue>
              <cue>
                <cueId>endCueIreland2</cueId>
              </cue>
            </cues>
      </group>
    ...
</groups>
```

Fig 11. Part of the groups.xml script, showing the definition of group.

The definition of a group has two important designations:

groupId A string reference to the group Id.

intensityLevels The number of intensity levels in this group (*start group* and *transition groups* do not use intensity levels).

3.3.5 Group Transitions

The group transitions script (*groupTransitions.xml*) is constructed in the exact same way as the transition matrix for intra group transitions (See *Appendix B – Transition Matrix*), but where the initial and target states are different groups respectively.

```
<GroupTransitionMatrix>
<groupTransitionID>IrelandtoAfrica</groupTransitionID>
        <!-- GroupId is the Group to which this transitionmatrix
        belongs-->
        <start_groupId>Ireland</start_groupId>
        <target_groupId>Africa</target_groupId>
        …
        <!—Group Transition Matrix-->
…
</GroupTransitionMatrix>
```

Fig 12. Part of the grouptransitions.xml script.

groupTransitionID A string reference to the group transition matrix.
start_groupId The Id of the current group.
target_groupId The Id of the target group that will be transitioned to (via the transition cue).

3.3.6 Script inter-connectivity

All the scripts are inter-connected, that is for example – references to *segmentId* (in *resources.xml*) are made in *cues.xml*. This inter-connectivity is what renders the scripting somewhat cumbersome if done by hand: common human errors for example such as simple misspellings of referenced keywords in a transition matrix will cause the audio engine to lose sync as it tries to operate with cueId's that do not match.

3.4 Audio Engine data-loop

The audio engine issues data calls and trigger pattern events on every beat. On start-up, the audio engine performs these tasks:

A. The audio engine parses all of the scripts and builds corresponding data structures.
B. The reserved *startGroup* is selected as start group, and the corresponding startCue. The metronome beat is paused. Run-time information from the cue (tempo, intensity level) is selected.
C. When a "Start Music"-event from an external application is received, the metronome is started at correct tempo. A pattern is selected (through non-standard randomization) and the subsequent pattern events are triggered as the metronome progresses.

The following constitutes the real-time data-loop of the audio engine:

1. Each beat, the audio engine updates its state and notifies the external application for example whether or not the audio engine is locked.
2.
    a. If no transition event is triggered from the pattern event list, a new pattern is selected (to play at the next beat) at the last beat of the current pattern.
    b. If a transition event is triggered from the pattern event list, the corresponding cue is selected and plays at next beat.
3. Intensity level change / Group change events issued by external application immediately triggers the audio engine to select a transition cue via a corresponding transition matrix.
4. If the currently playing cue is a transition cue (residing in the group *transitionGroup*), start cue or end cue the audio engine locks and cannot receive external events. These events will instead be queued and trigger the audio engine when it is unlocked.
5. If an "End Music"-event is triggered from an external application; the audio engine selects the cue in the current group, which has a shared intensity level as the currently playing cue – and transitions on the next beat. At the last beat of the end cue, the metronome is stopped.

3.5 Tools

3.5.1 Authoring Tool

As stated before; if scripting by hand, the probability of human error is substantial. These scripts can be automatically generated by the authoring tool.

The audio authoring tool is for composers/sound designers aimed at providing music for the game, and serve as a content provider and constructor of scripts and events. While it is completely possible to manually script the complete musical behaviour, a tool is pivotal to simplify and automatically write the scripts. The authoring tool provides a way of setting transition points, transition matrices, transition cue logics, group supervision and management of audio file resources.
See Appendix A for Authoring Tool GUI prototype.

3.5.2 Audition Tool

The audition tool is of help to the composer/sound designer when testing the music and the scripts with the audio engine, without having the game engine triggering events. The same events and group listings that are available to the game engine are available in the authoring tool.

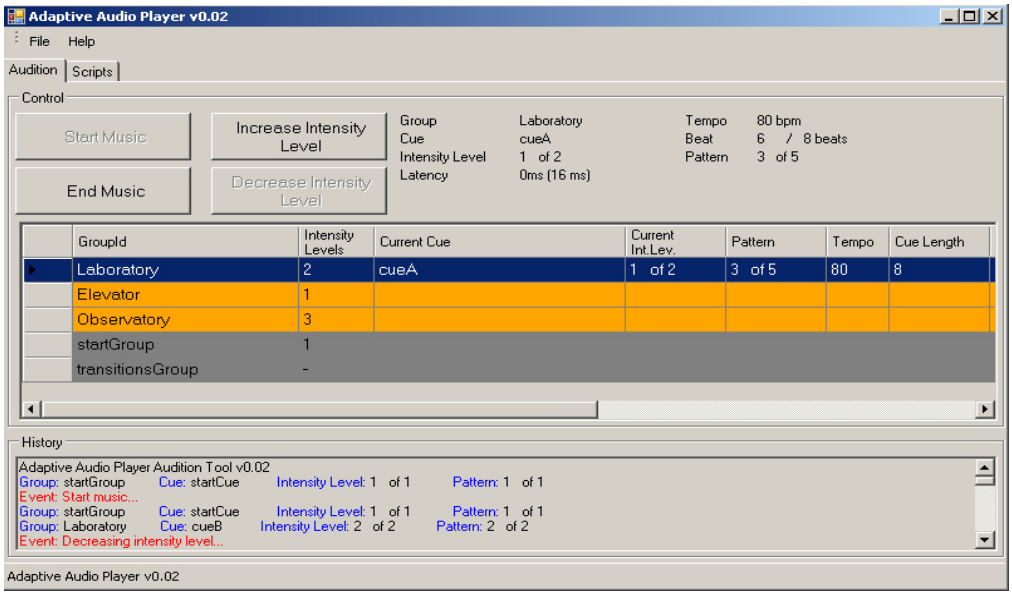

Fig 13. The Implemented Prototype Audition Tool GUI. The highlighted blue row shows the current group.

From this GUI, events can be sent to the audio engine:

Start Music	Starts the metronome beat, and starts playing *startCue* in *startGroup*.
End Music	Transitions to the current playing cue's corresponding end cue (if existing).
Increase Intensity Level	Increases the intensity level by one level.
Decrease Intensity Level	Decreases the intensity level by one level.

Group changes are done by clicking on the target group in the data grid. The groups are color-coded whereas only orange-colored groups are possible to transition to (have a corresponding group transition matrix) and non-colored groups are not. Dark grey groups are reserved for the *startGroup* and *transitionsGroup* and cannot be selected.

The data grid also presents information of relevancy such as *groupId*, *intensity levels* (currently playing intensity level, and also the total number of intensity levels), currently play *cue* as well as its *tempo*, *pattern* information and *length*.

During playback, information is updated *per beat*, meaning that all information is updated from the metronome notification from the audio engine. Updates to the audio engine are also issued through user input (changing groups etc) – which simulates updates from the game engine to the audio engine.

Certain states of the audio engine are also depicted to the user – for example if the current intensity level is the highest possible in a certain group, it is not possible to raise the intensity level further and feedback of this is presented as disabled buttons.

4 Prototype Implementation

The prototype implementation consists of the audio engine with XML parser as well as an auditioning tool. At the writing of this report, an API for external connectivity with game engine, as well as the authoring tool - is not implemented.

The prototype implementation has been done with Microsoft Visual C# 2005 Express Edition Beta. The XML Serialization support that is part of the .NET Framework is used to parse the XML-scripts. Microsoft DirectX 9.0c SDK has been used for use of DirectSound functionality. (*MSDN DirectX*, 2005-11-20)

4.1 Currently Implemented System Overview

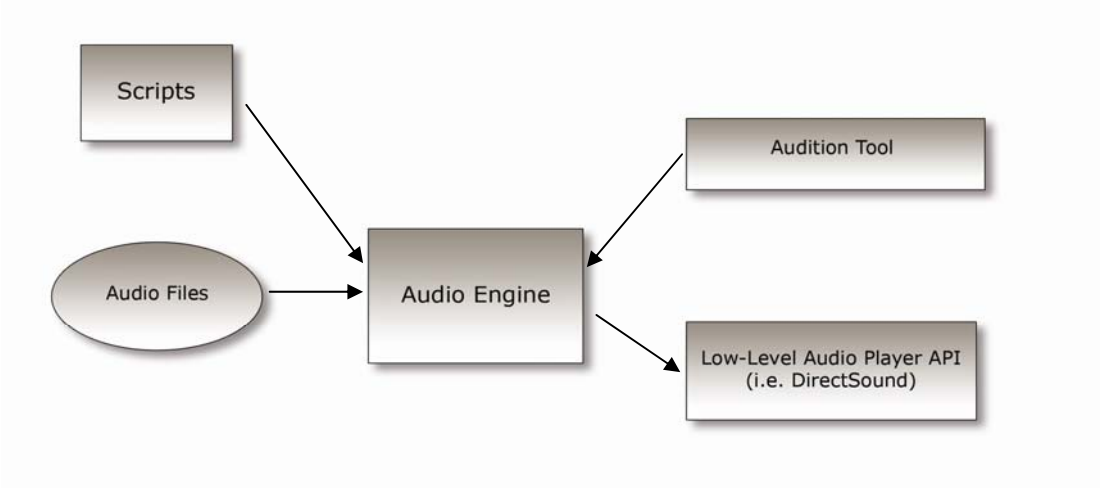

Fig 14. The Currently Implemented Prototype System

4.2 Limitations

- Panning
- Real-time effects
- Surround audio file support
- Streaming buffers
- Compressed audio files
- Track-based variation of patterns
- Intensity Level transitions

4.2.1 Panning

No panning settings are currently implemented, though supported in the scripts. Panning of audio is thoroughly supported in DirectSound and would be simple to implement.

4.2.2 Real time effects

Each wave file is recommended to be recorded with a reverb tail end. Support for additional reverb, chorus, phasing etc. effect controllers are currently not supported. These types of effects are however already supported in DirectSound itself and a proper implementation of effect controllers in the audio engine would therefore be fairly straightforward.

4.2.3 Surround audio file support

The system supports mono and stereo tracks and outputs. Multi channel (surround) audio management is currently not supported. Also this feature has support in DirectSound, which leads to a feasible implementation.

4.2.4 Streaming buffers

The current implementation has no support for streaming buffers, but uses static buffers instead. To be viable as a useable application with a game engine, the audio engine must use streaming data buffers – however as a prototype, the functionality of the script based logic can still be thoroughly examined with the current use of static buffers. DirectSound supports streaming sound buffers and proper implementation would, if not straightforward, be not entirely difficult.

4.2.5 Compressed audio files

The audio engine does not support compressed audio files, such as Ogg Vorbis (*Ogg Vorbis Audio Codec*, *2005-11-12)* etc. While the functionalities of the system can be demonstrated with uncompressed audio (WAV-file) – compressed audio files are a necessity if the system is to be used in conjunction with a game engine. Implementing a third-party audio compression algorithm could require ample development time.

4.2.6 Track-based variation of patterns

Current script support of track-based variations of patterns is incomplete. The segments in a pattern can only be designated a single track and has no tag id and as such, the patterns themselves play in the same manner every time they are triggered. Proper implementation of this feature is undemanding and feasible.
See - 3.2.5 Variation with Patterns and Tracks, and 3.3.2 Cues.

4.2.7 Intensity Level transitions

The intensity levels can only transition incrementally; larger leaps - i.e. raising the intensity level by more than one level - are currently unsupported in the audio engine however supported in the scripting. This feature of the audio engine would also be reasonably effortless to implement.

5 Results and Evaluation

The prototype implementation is an ample inclination of the workability of the adaptive audio system. While providing a framework for the composer, the relative simplicity of the script logics offers a powerful approach to adaptive audio for interactive media.

This type of adaptive audio system can be used in a variety of forms, adding adaptive behaviour to both music and sound effects for interactive media, but also as a basis to the more experimental field of non-linear playback.

5.1 Bugs and issues

- If a group transition matrix exists for two cues, but not all the intensity levels have a set target cue (i.e. the composer wishes a group transition to not take place specifically on the third intensity level of a particular group), the audio engine will lose sync as a group transition is deemed viable if a group transition matrix exists. A solution to this would be to have the audio engine check if the target cue is viable in the group transition matrix.
- When all patterns in a cue have played once, the audio engine refreshes its pattern history but does not keep track of what patterns had been playing prior to refreshing. This results in the possibility that the last pattern played before refreshing, and the first pattern to play after refreshing will be the same.
- The audition tool uses a Windows form data grid to display the group listings; however Windows has a substantial CPU overhead when updating data grids (15-16 ms per update). The result is a slight latency of the audio engine since the data grid is updated after every pattern change. This latency would not be present if the audio engine would be used in conjunction with a game engine. A solution would be to avoid the use of data grids altogether, and have another way of displaying the group information.
- A rare issue is the systems failure to keep track of segments that should fade-out. This has to do with an issue with synchronization of certain data structures using C#.
- Given that implementation has been done in Microsoft Visual Studio Express Edition 2005 Beta and Microsoft .NET 2 Framework Beta, performance can be marred by bugs that are due to issues with Beta software.

5.2 Future work

In addition to the stated system design, a number of functions are currently planned for later implementation.

5.2.1 Stingers

A stinger is a one-shot sound or musical "sound effect" which can be triggered in particular circumstances. They are often a loud and sudden crash - or "sting", hence the name.
Ex: If the composer wants to accentuate something in a game which happens once, but does not want to alter the intensity level – a stinger can be used instead. The game engine thus sends a triggered event to the audio engine which immediately plays the stinger (in disregard to the metronome beat). Examples of stingers used in movies are highly prevalent in horror scores or other situations where the visuals, sound effects and music try to frighten the audience with sudden scares.

An advantage of letting the audio engine handle musical stingers (instead of treating them as for instance ordinary sound effects) is the possibility of using a different stingers dependant on which *group* or *cue* currently plays. Therefore, the audio engine only needs to receive a single event to trigger a stinger and it will automatically play the cue's corresponding stinger.

Implementation-wise, this feature will require some work to enable control-features for the external application, as well as providing audio engine script support for the stingers.

5.2.2 Extended cue playback techniques

The cues will also have rules that dictate which segments play against each other. For instance, the composer will be able to designate that a particular segment can only play if another particular segment is also playing etc.

5.2.3 Extended transition techniques and cue-to-cue beat synchronization

The ability to cross fade directly to another intensity level (without the use of a transition cue) is a traditional way for the composer to handle transitions. Cue-to-cue beat synchronization would propose the possibility of synchronizing the playback of for instance different intensity levels – meaning the metronome beat would be synchronized and a cross fade transition between for example intensity level 1 and intensity level 2 would result in fading between two different cues but at the same beat position. An example: intensity level 1 and 2 features the same music but different instrumentation, a cross fade transition would fade out the former intensity level and into the new intensity level while never resetting the metronome beat – so the music cross fades to a new level but at the same beat position it were on in the original intensity level. This would contribute to the transitioning techniques available to the composer. An extensive expansion of the audio engine capabilities is required for this type of implementation.

5.2.4 Surround support

Panning of segments in surround is supported in DirectSound, and would be fairly uncomplicated to implement.

5.2.5 Real-time effects

Effects such as reverb, chorus, real-time panning, phasing and other effect controllers is feasible however not entirely easy (although supported by DirectSound) to implement in the future.

5.2.6 Non-linear music format

Games are currently the most common application to use non-linear music; however the fairly new prospect of non-linear playback of commercial music is in its initial stages. The technique used in the audio engine as well as other techniques involving adaptive audio (such as DirectMusic etc.) can be used without games or interactive media in mind – but for a non-linear playback of commercial or popular music.

The workings of a non-linear music playback system would require a quite different approach to controlling the audio engine than from a game engine type point of view, and thusly it is safe to say that implementation of a non-linear music format (although using

the same fundamental functions as in a game engine controlled adaptive music player) would be a serious endeavour. (*Beyond Games: Bringing DirectMusic into the Living Room*, 2004)

5.2.7 Application updates from Audio Engine

Since the audio engine already updates the audition tool every beat – it is reasonable to implement a similar update call to the exterior game application. In this way for example, elements in the game can be synchronized to events from the audio engine – this gives way to a number of possibilities, such as having game characters synchronize their footsteps to the beat of the music, or waiting for specific points in the music before game events occur.

6 References

James Boer, *Game Audio Programming*, Charles River Media, Inc. ISBN: 1-58450-245-2, 2003

Todd M.Fay et. al., *DirectX 9 Audio Exposed – Interactive Audio Development,* Wordware *Publishing Inc.* ISBN: 1-55622-288-2, 2004

T. Buttram, *Beyond Games: Bringing DirectMusic into the Living Room,* - in *DirectX 9 Audio Exposed: Interactive Audio Development*, T.M. Fay (ed.),WordwarePublishing Inc. ISBN: 1-55622-288-2, 2004

Alexander Brandon, *Audio For Games – Planning, Process, and Production*, New Riders, ISBN: 0-7357-1413-4, 2005

http://groups.yahoo.com/group/gameaudiopro/, Mail group *Game Audio Pro (Professional Game Audio Designers discussion group), 2005-11-12*

http://msdn.microsoft.com/directx/sdk/, *Microsoft Developers Network, 2005-11-12*

http://msdn.microsoft.com/directx/beta/, *MSDN DirectX Beta Program, 2005-11-20*

http://www.openal.org/, *OpenAL - Cross-platform 3D Audio, 2005-11-12*

http://www.radgametools.com/miles.htm, *The Miles Sound System, 2005-11-20*

http://www.vorbis.com/, *Ogg Vorbis Audio Codec, 2005-11-12*

http://earth.google.com/, *Google Earth*, 2005-11-13

http://imuse.mixnmojo.com/what.shtml, *iMuse Island*, 2005-11-14

Appendices

Appendix A – Audio Authoring Tool GUI Prototype

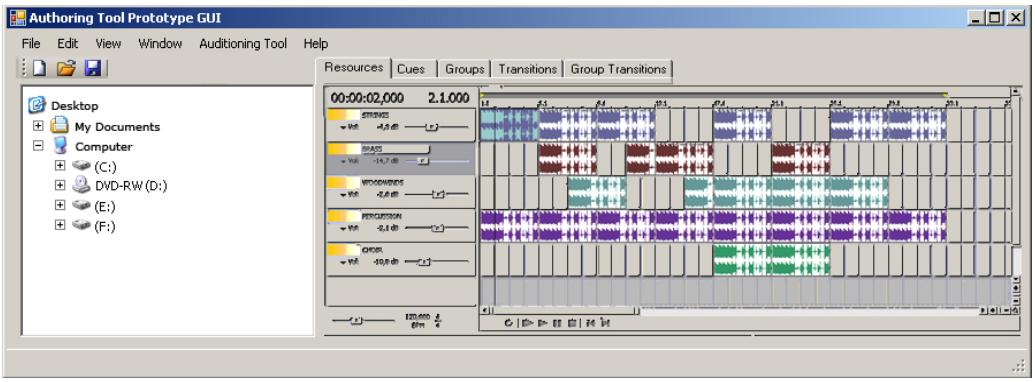

Fig 15. Prototype of the Audio Authoring Tool GUI.

Appendix B – Transition Matrix

Initial state \ Target state	S1	S2	S3
S1	-	T1-2	T1-3
S2	T2-1	-	T2-3
S3	T3-1	T3-2	-

Fig 17. The concept of the Transition matrix

For 3 music states (that is; intensity levels) 6 transition cues are needed.

Appendix C - Transition Matrix Script (transitions.xml)

```
<Transitions>
  <TransitionMatrix>
  <!-- GroupId is the Group to which this transitionmatrix belongs. -->
    <groupId>Template</groupId>
    <!-- ******************Example of TransitionMatrix definition -->
    <!-- ..... |cueId1 |cueId2 |cueId3 -->
    <!-- cueId1 |cell 1 (empty) |cell 2 (tr.Cue)|cell 3 (tr.Cue) ... -->
    <!-- cueId2 |cell 4 (tr.Cue) |cell 5 (empty)|cell 6 (tr.Cue) ... -->
    <!-- cueId3 |cell 7 (tr.Cue) |cell 8 (tr.Cue)|cell 9 (empty) ... -->
    <!--********************************************************** -->
    <!-- When a transition from a specific cue to another is triggered, the cue id in the
    corresponding cell is set as transition cue - and the target cue will play after that. --
    >
      <rows>
        <row>
            <!-- This is the name of the row. -->
            <cueId>cueA</cueId>
            <cell>
                        <!-- cell 1 -->
                        <transitionCueId>emptyCell1</transitionCueId>
            </cell>
            <cell>
                        <!-- cell 2 -->
                        <transitionCueId>trCell2</transitionCueId>
            </cell>
            <cell>
                        <!-- cell 3 -->
                        <transitionCueId>trCell3</transitionCueId>
            </cell>
        </row>
        <row>
            <cueId>cueB</cueId>
            <cell>
                        <!-- cell 4 -->
                    <transitionCueId>trCell4</transitionCueId>
            </cell>
            <cell>
                        <!-- cell 5 -->
                        <transitionCueId>emptyCell5</transitionCueId>
            </cell>
            <cell>
                        <!-- cell 6 -->
                        <transitionCueId>trCell6</transitionCueId>
                        </cell>
        </row>
        <row>
            <!-- This is the name of the row. -->
             <cueId>cueC</cueId>
            <cell>
                        <!-- cell 7 -->
                        <transitionCueId>trCell7</transitionCueId>
```

```
            </cell>
            <cell>
                        <!-- cell 8 -->
                        <transitionCueId>trCell8</transitionCueId>
            </cell>
            <cell>
                        <!-- cell 9 -->
                        <transitionCueId>emptyCell9</transitionCueId>
            </cell>
        </row>

        <!-- ...and so on -->

      </rows>
  </TransitionMatrix>
  ...
<Transitions>
```

Part of the transitions.xml script, showing the scripting template of a transition matrix.