ADAPTIVE MUSIC ON THE WEB

Wayne Siegel, professor
DIEM
Royal Academy of Music
Skovgaardsgade 2C
Aarhus, Denmark

Ole Caprani, associate professor
Department of Computer Science
University of Aarhus
Aabogade 34
Aarhus, Denmark

ABSTRACT

The project *Adaptive Music on the Web* was initiated by DIEM in 2004 and realized in collaboration with the Department of Computer Science at the University of Aarhus with the support of a generous grant from *The Heritage Agency of Denmark* under auspices of the Danish Ministry of Culture. The goal of the project was to develop new musical paradigms that allow the listener to influence and alter a piece of music while listening within the context of a common web browser.

A set of artistic and technical definitions and guidelines for creating adaptive music for web publication were developed. The technical possibilities and limitations of presenting adaptive music on the web were studied. A programming environment was selected for the implementation of adaptive musical works. A group of electronic music composers was established and each composer was commissioned to create an adaptive musical work for web publication in accordance with the established guidelines. A prototype was created for each work. Specialized programming tools were developed for implementing these prototypes in the form of adaptive musical works capable of running within a standard web browser. A web site was set up for presenting these works.

1. INTRODUCTION

The presentation of art on the web today is largely limited to the distribution and presentation of works of art created for other media. Typically these include digitized photographs of paintings, digital text documents of literary works, digitized films and digital audio recordings. But for composers and musicians, the artistic potential of the web as a media in its own right is not limited to the distribution of digital recordings. *Adaptive music* is music that can be influenced by the listener. *Adaptive music* is constructed using flexible structures, which allow the music to unfold and develop in various ways under varying degrees of control on the part of the individual listener. The distribution of audio by means of radio or CD distribution is a one-way street. The listener's role is that of a passive "receiver" of a musical work. The web is an interactive media. There are numerous technical limitations and challenges in regard to bandwidth, compatibility, response time and dependability. Yet an increasing number of artists are attempting to utilize and incorporate the web's potential for interaction in their art. The goal of this project was to further develop musical forms specifically created for web publication.

2. DEFINITION AND DOGMA

2.1. Definition

Adaptive music can be defined as music that can be controlled and influenced by a non-expert end user or listener, either consciously or subconsciously. *Adaptive Music* might be considered a subcategory of *interactive music* [1]. But while *interactive music* generally refers to a musical situation involving a trained musician interacting with a computer program, *adaptive music* involves a non-expert user interacting with a computer program. The term *adaptive music* is sometimes used in the context of computer games, referring to music that adapts as a result of game play. In this context, controlling the music is generally not a primary concern for the player (not "part of the game", so to speak) [2].

For this project the following definition of *adaptive music* was adopted:

> *"Adaptive music is music that can be influenced and affected by the listener. The listener can control one or more significant musical parameters in the music. The end user is not only the "audience," but an active participant as well."*

2.2. Dogma

One important motivation for initiating this project was to develop new musical forms for the web as a media in its own right. Often sound is of secondary importance in the world of multi-media. Here the intention was to make sound the primary concern, suppressing the tendency of human perception to focus on vision. A list of requirements, or an artistic and technical *dogma*, was presented to the composers from the onset.

2.2.1. Criteria

For inclusion in the project works were expected to meet the following criteria:

- Works should have open form.
- Works should be rule-based and flexible.
- Significant musical parameters should be controlled by input.

- Software must be downloaded to the user's computer.
- Software must run locally on the user's computer without streaming data during use.
- Functionality should be independent of the user's internet bandwidth.
- Musical works should be experienced by the user as if he/she is using a browser-based service.
- Response between action and reaction must be experienced as fast and agile.
- Any sound files used must have a duration of maximum 2 seconds each and 20 seconds total.
- Graphics can be used as a user interface but must not become an important artistic element.

2.2.2. The Ten Commandments

For the sake of illustration, a somewhat tongue-in-cheek, but not entirely ironical, list of "ten commandments" was created:

1. *Thou shalt not covet long sound files*
2. *Thou shalt love thy listener as a participant*
3. *Thou shalt see possibilities in limitations*
4. *Thou shalt not mock modem connections*
5. *Thou shalt not reproduce (music)*
6. *Thou shalt not give thy programmer ambiguous messages*
7. *Thou shalt respect thy programmers limitations as thine own*
8. *Thou shalt not kill thy programmer*
9. *Thou shalt observe all deadlines*
10. *Thou shalt not create multimedia*

3. PLATFORM

The next step was to adopt a standardized software platform for the implementation of the adaptive compositions. The choice was by no means an easy one. At the time the project was initiated, there were no consistent, flexible and compatible industry standards for interactive web audio.

Our priority was to employ a platform that met the following criteria: 1) widespread, free and simple distribution, 2) functional on most commonly used operating systems and browsers, 3) real-time control of sample playback, sound synthesis and signal processing, 4) precise control of timing.

Several options were available, each with it's own advantages and drawbacks. In general the latter included limitation to specific browser support, incompatible architectures, lack of widespread distribution and complexity in installation and use. As stated in a report by the Interactive Audio Special Interest Group [3]:

> *"There is no higher audio-centric consciousness at work to facilitate a consistent set of audio capabilities. It will always be up to manufacturers to distinguish themselves through their own technologies, so standardized implementation will never be the answer."*

The following programming platforms were considered:

- Max/MSP
- Java
- JSyn
- Flash

3.1. MAX/MSP

Max/MSP is a graphical programming environment designed for music, audio, and multimedia. Over the past 15 years Max/MSP has developed into a leading software environment for composers creating interactive music. Marketed by Cycling74, Max/MSP is compatible with both Mac OS X and Windows XP [4].

Max/MSP was considered to be an ideal platform for this project, since many of the composers involved were familiar with it and had previously used it to create interactive music. A run-time version allows end users to run applications created in Max/MSP without purchasing the software. Karlheinz Essl uses this approach to distribute Max/MSP patches [5].

There were, however two crucial drawbacks: 1) the end user is required to download a runtime version of the software, a process that can be somewhat complicated, especially for Windows users, and 2) it is not possible to create Max/MSP applications that run within a web browser.

3.2. JAVA

Java is an open source and platform independent programming language compatible with Windows, Mac OS X and Unix. Java applets are applications or programs that are embedded in other applications [6]. The use of Java applets in web pages is widespread. The Java Sound API specification provides low-level support for audio operations including audio playback and recording, mixing, synthesis and signal processing within a flexible framework. The Java applets of J-DAFX [7] and Jass [8], show that advanced digital audio applications can indeed be written in pure Java based on low-level operations of the Java Sound API.

3.3. JSYN

JSyn is a freeware plug-in and object library for Java, which provides real-time unit-generator based synthesis for stand-alone Java applets in a web page [9]. We considered JSyn to be a powerful and flexible platform in reference to this project. Our main concern was that JSyn was created and supported by a sympathetic but very small company [10]. The JSyn website indicated that the programming environment had not been maintained on a regular basis. We were not entirely convinced that JSyn would be supported or even available in the distant or near future.

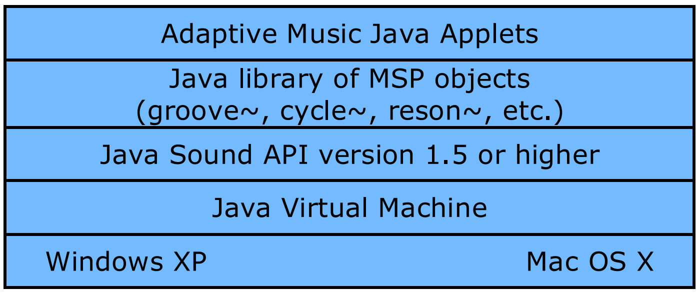
3.4. FLASH

Flash is a commercial product, marketed by Adobe Systems, which includes a professional authoring program and a player provided free for end users. Since its introduction by Macromedia in 1996, Flash has become a popular platform for adding animation and interactivity to web pages. Flash is commonly used to create animation, advertisements, web art and to integrate video into web pages [11]. Originally designed for animation, Flash has a number of limitations and drawbacks in relation to audio in general, and regarding interactive audio in particular. Synchronization is limited, and in certain situations unreliable. Audio playback is timeline based. Synthesis and signal processing, such as filtering and modulation, are not possible [12]. These limitations were considered to be serious hindrances in relation to the project goals.

3.5. PLATFORM SELECTION

In addition to the platforms mentioned, a combination of Max/MSP and Flash was also considered. The *flashserver* external for Max/MSP [13] allows for bidirectional communication between Max/MSP and Flash. Essentially, this allows Flash to be used as an interface to Max/MSP, making the Max/MSP application invisible to the user. This platform was tested and rejected for three reasons: 1) clumping of data and resulting delays were found to occur in the TCP/IP link used between Flash and the *flashserver*, 2) Max/MSP was found to be somewhat unstable on the Windows platform and 3) this configuration would require that the user download and install additional Max/MSP runtime software.

A decision was made to use Java as the main platform for the project. It was also decided that composers preferring to work directly with Flash as a compositional medium could do so. Most of the composers associated with the project preferred to create prototypes in the form of Max/MSP patches. The strategy chosen for implementing the Max/MSP prototypes in Java was to create a toolbox or object library in Java that mimicked the functions of the Max/MSP objects used in the prototypes, Fig. 1. A similar approach has been used in Jsyn [9] and Jass [8].


Adaptive Music Java Applets
Java library of MSP objects (groove~, cycle~, reson~, etc.)
Java Sound API version 1.5 or higher
Java Virtual Machine
Windows XP
Mac OS X
Fig. 1. Java implementation of Max/MSP patches.

4. COMPOSITIONS

Six composers were involved in the project:

- Jens Hørsving
- Mikko Jensen
- Lasse Laursen
- Morten Riis
- Wayne Siegel
- Kristian Vester

The composers were first asked to create a proposal or description of their composition. The various proposals were presented and discussed in a group forum. The composers were then asked to create prototypes of their compositions in Max/MSP. The project programmers worked with the composers porting their compositions to Java. A description of the works created by the composers follows:

4.1. Jens Hørsving

Mass Is The Opposite Of Space is constructed over the spoken sentence "Mass is the opposite of space." The sentence is broken into six parts:

- mmmm: whispered
- mass: short, whispered
- is: short, whispered
- the op-op-site of: whispered rhythmically
- space: normal voice (more substantial)
- ssss: whispered

Each of these six sentence fragments corresponds to a letter on the computer keyboard (A-Z), but the user does not know which fragments and which letters correspond. The sounds are processed using the following effects: pitch shifting (transposition), panning, looping and fading.

The user can play with this sentence and create a three-voice texture. Typing a letter will start the first voice, shift-typing a letter will start a second voice and control/shift-typing a letter will start the third voice. In this manner, the user can create polyphonic stereo voice textures.

The sounds are supplemented by graphics in a window whose background color changes constantly, gradually and almost unnoticeably through the entire color spectrum. The three polyphonic layers are visualized in black circles The louder the sound, the larger the circle; the higher the circle is placed on the screen the higher the sound is transposed. When a sound fades out its corresponding black circle gradually decreases in size. The horizontal placement of the circle shows the panning position of the corresponding sound. The movement of the circles visualizes the sounds of the music and the placement of these sounds within the stereo image.

It was the composer's intention to create an intimate mood reflecting on the concepts of sonic mass and silence, combined with a simple but beautiful interface.

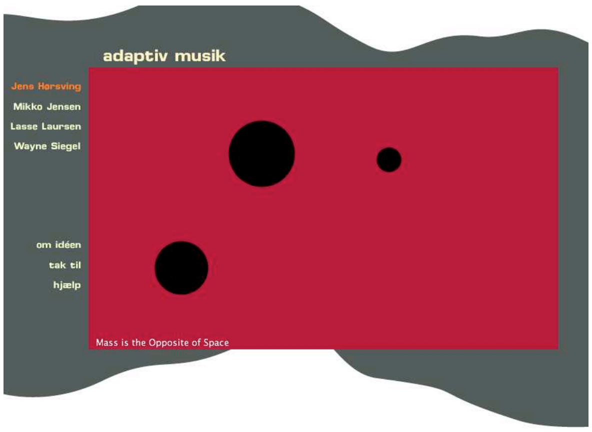


Fig. 2. Screen shot of *Mass is the Opposite of Space.*

4.2. Mikko Jensen

Mikko Jensen's work is based on related sound progressions divided into segments. The user can manipulate the segments by clicking on the images that represents the sound segments. The size of each image reflects the duration of the sound segment (larger images associated with longer sounds) while the placement of the image reflects the pitch (higher vertical placement represents higher pitch). When the user has clicked on various images a few times, the basic sounds and the images will change and the user can begin exploring once more.

There are three different basic sounds. The amount of time (number of mouse clicks) that one can play with a particular set of basic sounds is random. Relatively quick mouse clicks are most fruitful, since these allow the user to interrupt the segments before they have finished playing, giving the user more control over the sounds. But this is up to the user's own taste.

The work is a sort of picture bingo in sound. After playing for a while, the user will begin to remember which sounds are represented by which images and how they are transformed. The intention was to create a work that did not use pitch and rhythm in a standard way. The idea is related to Mikko Jensen's project *Pol Mod Pol (Pole Against Pole)*, which to a great extent uses rhythm as a means of expression, but which would be extremely difficult to notate or describe, since it is edited by hand "with feeling". This performance option is offered the user in Mikko Jensen's adaptive work.

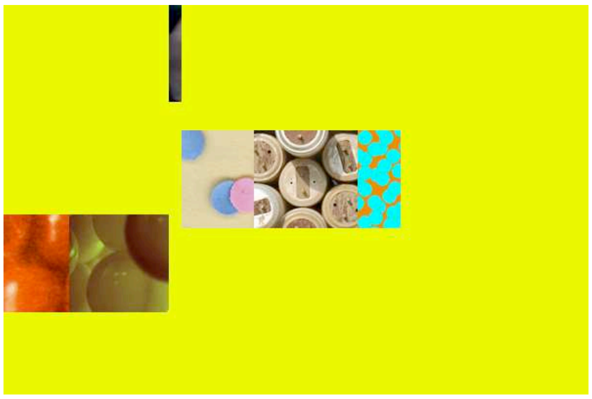


Fig. 3. Screen shot of adaptive work by Mikko Jensen.

4.3. Lasse Laursen

Og (the Danish word for: "and") is a kind of audio-visual text sound composition, which is played and created using the computer keyboard and mouse. There are two versions, an interactive version which the listener can generate and edit and a playback version, which can be activated via the playback interface.

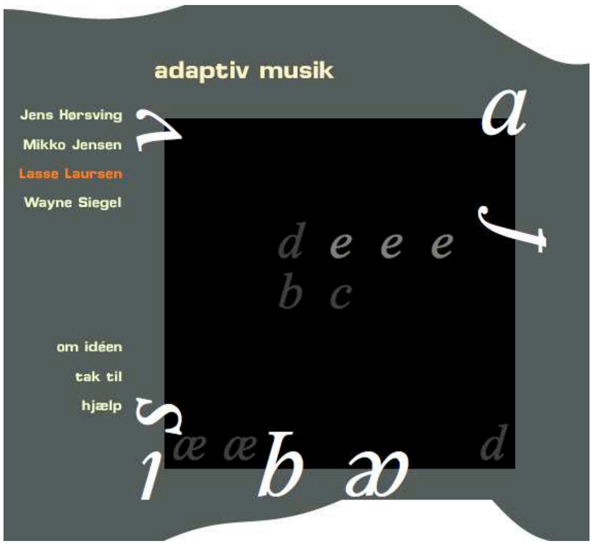


Fig. 4. Screen shot of *Og* by Lasse Laursen.

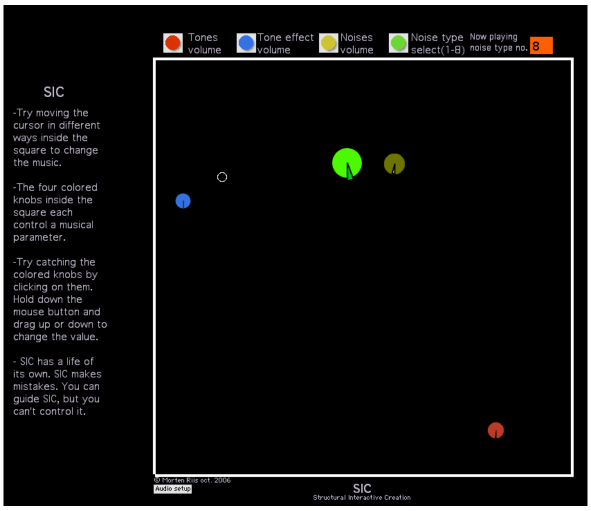
Og uses 40 sounds consisting of sampled spoken words (in Danish) that are controlled by typing the first letter of the word. Each word can be automated (looped), adjusted, deactivated and reactivated by typing the letter that triggers sample playback. The word associated with a letter is looped when the letter is typed more than once within a five second interval. A sound is deactivated if its letter is struck again in the rhythm of the looped sound. A sound can be adjusted if more than five loops are activated. When more than five loops are active some of the letters begin to float around the screen. A sound associated with a floating letter can be adjusted by clicking on the letter. The arrow keys can now be used to change the playback speed. Only floating letters can be adjusted. A sound can be reactivated by deactivating two loops in a row and tying the letter displayed in green in the pulse of the loop (as in a game of tennis).

Og is a work that monitors the actions of the listener and adapts the music to the listener. *Og* is predictable. The listener is encouraged to imagine specific results and then attempt to realize the desired version of the work.

4.4. Morten Riis

SIC (Structural Interactive Creation) is an adaptive musical environment designed to present the composer's musical aesthetic. The intention is to have the listener "compose" music within an artistic framework created by Morten Riis, who was both composer and programmer. The interface uses mouse movement to change the sound, but the relationship between user control and sonic results are relatively complex. Few control parameters control many different internal compositional processes. Gradually, the user learns how the controls affect musical processes of the work. But an element of uncertainty and surprise is built into the system, and the user interface itself is constantly changing.

SIC was designed and programmed in Max/MSP by Morten Riis. For technical reasons, *SIC* has at this writing not yet been implemented in Java and is not presented on the *Adaptive Music* web site. The Max/MSP patch can be downloaded from the composer's web site [14].



-Try moving the cursor in different ways inside the square to change the music.

-The four colored knobs inside the square each control a musical parameter.

-Try catching the colored knobs by clicking on them. Hold down the mouse button and drag up or down to change the value.

- SIC has a life of its own. SIC makes mistakes. You can guide SIC, but you can't control it.

Fig. 5. Screen shot of *SIC* by Morten Riis.

Below is a description of the interface:

4.4.1. Red knob (sine wave synthesizer volume)

This knob controls the volume of the synthesizer module, which consists of several sine wave tone oscillators. The sine wave tones become louder when the knob is "turned up". This knob also controls cross fade between pure sine waves and sine waves modulated by noise. Frequency and envelope are controlled by cursor position. Vertical changes affect pitch, while horizontal changes generate random envelopes. Clicking and holding down the mouse anywhere in the large square will activate random gates that mute and un-mute the synthesizer.

4.4.2. Yellow knob (noise sample player volume)

This knob controls the volume and playback of sampled sounds. The vertical position of the cursor controls the playback pointer position within a "noise" sound file, while the horizontal position controls cutoff and resonance of the built-in filter. Clicking on the mouse switches randomly between filter types (low pass, high pass, band pass and notch).

4.4.3. Green knob (noise type select)

This knob is used to select between 8 noise sound files that are used by the noise sample player.

4.4.4. Blue knob (effects)

This knob controls two effects that are applied to both the sine wave synthesizer and the noise generator. The effects are 1) a granular effect, controlled by the vertical position of the cursor, that changes pitch and 2) a sample degrade effect, controlled by the horizontal position of the cursor, that reduces the sample frequency.

4.4.5. Overall control

The functions described above work together simultaneously. The makes *SIC* complicated and, due to the many random functions incorporated, unpredictable. The expression is constantly changing.

4.5. Wayne Siegel

Synchopath is an adaptive musical toy designed for a younger audience (ages 6-12). The work consists of three elements, the controls of which are straightforward and easy to learn.

4.5.1. Accelomouse

Accelomouse registers mouse motion, calculates the current speed of the mouse, and uses this data to control volume and filter parameters in real time applied to a noisy sampled sound source and a white noise oscillator. Mouse movement data is also used to control the frequency of a sine wave oscillator within a range of 80 and 200 Hz. The result is an "instrument" that creates a noisy sound when the mouse is moved. The faster the mouse is moved, the louder and brighter the sound. When the mouse is not moved, there is no sound. *Accelomouse* is always active when the program is running and provides a sonic representation of mouse movement.

4.5.2. Synchopatch

Synchopatch consists of several elements. The musical idea was to create a work that would generate rhythms according to a set of rhythm algorithms and melodies within a set of four different scales. The activity level and tempo of the rhythm generator is controlled by mouse speed. The faster the mouse moves the more percussive hits occur and the faster the tempo.

Four musical scales, each with its own mood, are associated with the four extreme corners of the computer screen. When the cursor is located in the upper left corner of the screen, melody and accompaniment is generated within scale 1, when the cursor is located in the lower left corner of the screen, scale 2 is used, when the cursor is located in the lower right corner of the screen, scale 3 is used and when the cursor is located in the upper right corner of the screen, scale 4 is used.

Moving the cursor between the four corners of the screen transforms the melodic and harmonic content. Transitions are gradual. If the cursor is positioned at the top center of the screen, half of the notes produced will lie within scale 1 and half within scale 4, using a weighted random selection. The further the cursor moves to the right, the greater the chance that notes from scale 4 will be used. When the curser is moved all the way into the upper right corner, only notes from scale 4 will be heard. The idea was to allow the listener to control gradual transformations between different musical moods. When the cursor is located in the center of the screen notes from all four scales will be used and the result is harmonically chaotic.

Sounds used for melody and accompaniment are produced using a simple FM synthesis algorithm. Each mood employs particular sounds, but continuous transformation between these sounds is achieved by gradually changing carrier and modulator envelopes and the modulation index. The rhythm section uses sample playback of four short samples at various speeds and amplitudes.

4.5.3. Drones

Three drones can be controlled by the listener. If the listener presses the letters *A, S* or *D* on the computer keyboard a drone tone will be heard. A note from the appropriate scale is selected according to the selection process used in *synchopatch.* The pitches will correspond to the current scale using a weighted random function. *A* produces low pitches, *S* produces middle pitches and *D* produces high pitches.

4.5.4. Graphics

The screen image represents the four moods of the four corners.

Fig. 6. *Synchopath* screen image.

4.6. Kristian Vester

The Adaptive Sara-Black DUMP is an alternative interpretation of the idea of adaptive music. It consists of an e-bay auction, where the composer sells music that does not yet exist at the time of sale. The buyer is required to send an item to the seller that the seller and buyer have agreed upon. The seller then somehow incorporates this item into a piece of music, which he then ships to the buyer. Kristian Vester (aka *Goodiepal)* has chosen to defy the dogma and disobey the 10 commandments (they say there's always one in the bunch). In his own view, Kristian Vester has only "bent" the rules:

Thou shalt not covet long sound files
"I don't necessarily, if they are long they are at least generative."
Thou shalt love thy listener as a participant
"I do indeed."
Thou shalt see possibilities in limitations
"I do that, too."
Thou shalt not mock modem connections
"Ebay works fine with a modem connection, and receiving items by snail mail makes it even easier."
Thou shalt not reproduce (music)
"I am generative and adaptive."
Thou shalt not give thy programmer ambiguous messages
"I am my own HTML programmer."
Thou shalt respect thy programmers limitations as thine own
"I am my own HTML programmer."

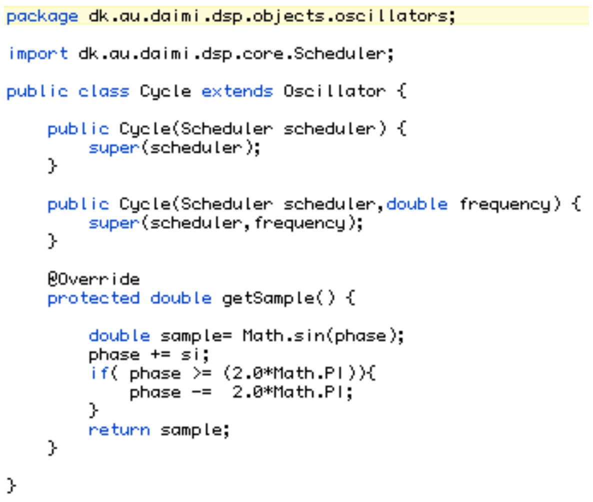
Thou shalt not kill thy programmer
"I plan to continue to the bitter end."
Thou shalt observe all deadlines
"I'm trying, how about if we say May 1st?"
Thou shalt not create multimedia
"I create adaptive music."

5. IMPLEMENTATION

5.1. MSP objects

Most of the composers created prototypes in Max/MSP. Our strategy was to create an object library in Java that mimicked the functions of the Max/MSP objects used in the prototypes. The object library created is by no means complete, since it is basically limited to a collection of audio objects that were employed in the prototypes.

Below is an example of the Java implementation of an MSP object. The MSP object *cycle~* is a general table based oscillator but only a sin wave oscillator was needed in this case.



```java
package dk.au.daimi.dsp.objects.oscillators;

import dk.au.daimi.dsp.core.Scheduler;

public class Cycle extends Oscillator {

    public Cycle(Scheduler scheduler) {
        super(scheduler);
    }

    public Cycle(Scheduler scheduler,double frequency) {
        super(scheduler,frequency);
    }

    @Override
    protected double getSample() {

        double sample= Math.sin(phase);
        phase += si;
        if( phase >= (2.0*Math.PI)){
            phase -=  2.0*Math.PI;
        }
        return sample;
    }

}
```

Fig. 7. Java implementation of the Max/MSP object *cycle~*

Each Max/MSP (sub) patch was implemented by 1) creating object instances of Java classes corresponding to the Max/MSP objects and 2) connecting the objects corresponding to the connections in the patch. This is similar to the unit generator approach of Jsyn,[9]. The approach is illustrated with the *simpleFM* patch from the MSP tutorial 11, Fig. 8 and Fig 9. This subpatch has been used in the *FMvoice* patch of *Synchopath.*

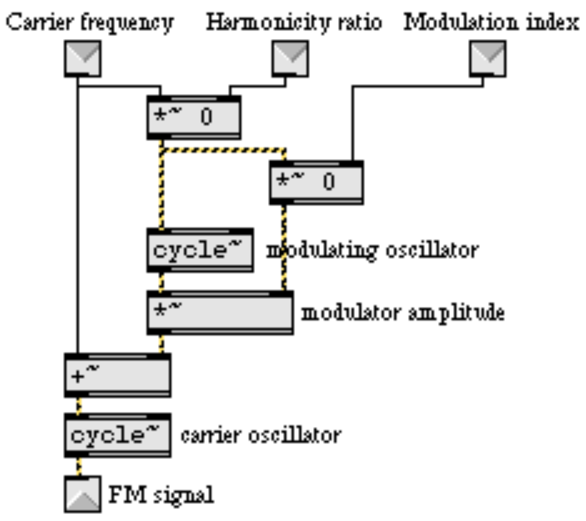
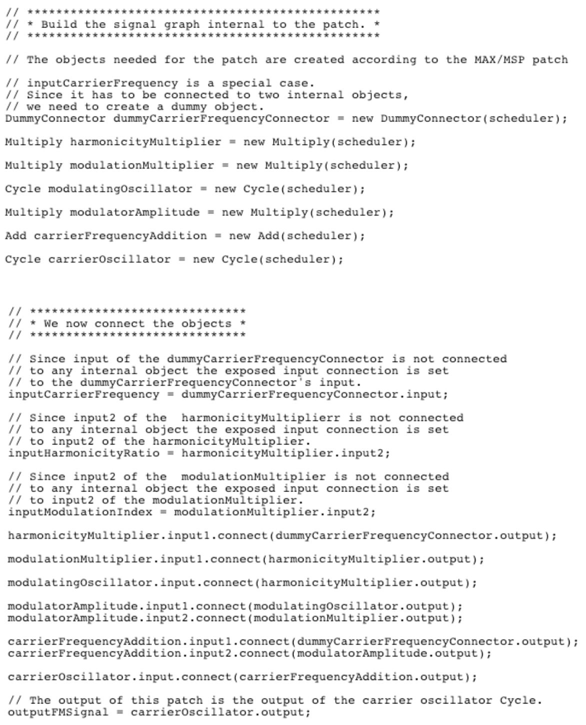


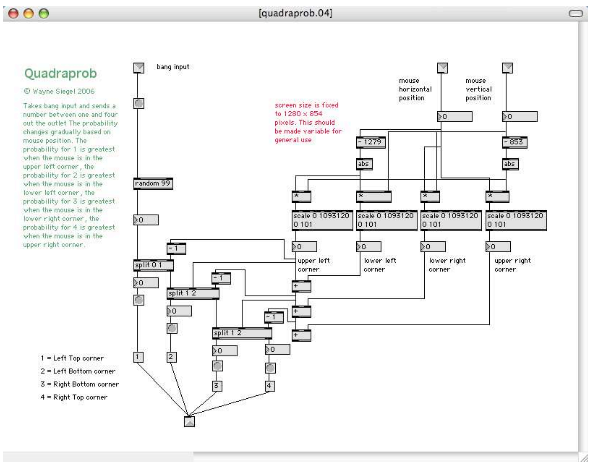
Fig. 8. *simpleFM* a Max/MSP subpatch used in *FMvoice* of *Synchopath.*



```java
// **************************************************
// * Build the signal graph internal to the patch. *
// **************************************************

// The objects needed for the patch are created according to the MAX/MSP patch

// inputCarrierFrequency is a special case.
// Since it has to be connected to two internal objects,
// we need to create a dummy object.
DummyConnector dummyCarrierFrequencyConnector = new DummyConnector(scheduler);

Multiply harmonicityMultiplier = new Multiply(scheduler);

Multiply modulationMultiplier = new Multiply(scheduler);

Cycle modulatingOscillator = new Cycle(scheduler);

Multiply modulatorAmplitude = new Multiply(scheduler);

Add carrierFrequencyAddition = new Add(scheduler);

Cycle carrierOscillator = new Cycle(scheduler);


// ******************************
// * We now connect the objects *
// ******************************

// Since input of the dummyCarrierFrequencyConnector is not connected
// to any internal object the exposed input connection is set
// to the dummyCarrierFrequencyConnector's input.
inputCarrierFrequency = dummyCarrierFrequencyConnector.input;

// Since input2 of the  harmonicityMultiplierr is not connected
// to any internal object the exposed input connection is set
// to input2 of the harmonicityMultiplier.
inputHarmonicityRatio = harmonicityMultiplier.input2;

// Since input2 of the  modulationMultiplier is not connected
// to any internal object the exposed input connection is set
// to input2 of the modulationMultiplier.
inputModulationIndex = modulationMultiplier.input2;

harmonicityMultiplier.input1.connect(dummyCarrierFrequencyConnector.output);

modulationMultiplier.input1.connect(harmonicityMultiplier.output);

modulatingOscillator.input.connect(harmonicityMultiplier.output);

modulatorAmplitude.input1.connect(modulatingOscillator.output);
modulatorAmplitude.input2.connect(modulationMultiplier.output);

carrierFrequencyAddition.input1.connect(dummyCarrierFrequencyConnector.output);
carrierFrequencyAddition.input2.connect(modulatorAmplitude.output);

carrierOscillator.input.connect(carrierFrequencyAddition.output);

// The output of this patch is the output of the carrier oscillator Cycle.
outputFMSignal = carrierOscillator.output;
```

Fig. 9. Java implementation of the Max/MSP subpatch *simpleFM.*

Max/MSP patches created by the composers were not always reproduced directly in Java. Often the musical ideas behind a large patch needed to be analyzed, disassembled and reconstructed in a new way more native to Java. An example is the subpatch *quadraprob* in *Synchopatch*, Fig. 10, and the implementation in Java as shown in Fig 11.


Quadraprob

© Wayne Siegel 2006

Takes bang input and sends a number between one and four out the outlet The probability changes gradually based on mouse position. The probability for 1 is greatest when the mouse is in the upper left corner, the probability for 2 is greatest when the mouse is in the lower left corner, the probability for 3 is greatest when the mouse is in the lower right corner, the probability for 4 is greatest when the mouse is in the upper right corner.

screen size is fixed to 1280 x 854 pixels. This should be made variable for general use

1 = Left Top corner
2 = Left Bottom corner
3 = Right Bottom corner
4 = Right Top corner

Fig. 10. *Quadraprob*, Max/MSP patch from *Synchopath*.

```java
@Override
public void init() {
    setNumberOfInlets(1);
    inputBang = getInlet(0,Connection.Type.CONTROL);
    setNumberOfOutlets(1);
    output = getOutlet(0,Connection.Type.CONTROL);
    width = Toolkit.getDefaultToolkit().getScreenSize().width;
    height = Toolkit.getDefaultToolkit().getScreenSize().height;
    System.out.println(width+":"+height);
    myMessage = new Message(scheduler);
}

@Override
protected void handleMessage(ControlMessage message, Connection source) {
    if (source == inputBang) {
        handlePositions();
        currentQ = calculateOutput();
        output.sendMessage(myMessage.setMessage(message,currentQ));
    }
}

private void handlePositions() {
    horizontalPosition = MouseInfo.getPointerInfo().getLocation().x;
    verticalPosition = MouseInfo.getPointerInfo().getLocation().y;
    upperLeft = Math.abs(horizontalPosition - width) * Math.abs(verticalPosition-height);
    lowerLeft = Math.abs(horizontalPosition - width) * verticalPosition;
    lowerRight = horizontalPosition * verticalPosition;
    upperRight = horizontalPosition * Math.abs(verticalPosition-height);

    upperLeft /=  (width*height) / 100.0;
    lowerLeft /= (width*height) / 100.0;
    lowerRight /= (width*height) / 100.0;
    upperRight /=  (width*height) / 100.0;
}

private int calculateOutput() {

    double random = Math.random()*99;
    int q = 0;
    if (random < upperLeft) {
        q  = 0;
    } else
    if (random < upperLeft + lowerLeft) {
        q = 1;
    } else
    if (random < lowerRight + lowerLeft + upperLeft) {
        q = 2;
    } else
        q = 3;

    return q;
```

Fig. 11. *Quadraprob* implemented in Java.

5.2. Java

The interfaces of the Java platform to the underlying audio and graphics hardware made it possible to implement the Max/MSP prototypes of the composers in pure Java with integration of the user interaction, audio processing and graphics generation in a single framework. As of Java 1.5 the Java Sound API now provides the audio quality needed to implement the Max/MSP prototypes.

5.3. Compatibility

The Java applet framework allows Java programs to run within a Java enabled browser without special download actions from the user. This made it possible to design and implement browser based adaptive musical works with the works of the composers embedded within a single web page.

6. CONCLUSION

6.1. Dogma vs. practice

One important project goal was to explore the artistic potential of the web as a media in its own right and develop musical forms specifically created for web publication. In order to do this, it was deemed necessary to encourage the artists involved to break with some very strong cultural norms embedded in our musical traditions and media conventions. A dogma, or set of rules, was created to this end. But complying by these rules was clearly no easy task. Composing adaptive music is quite a different thing than composing music for distribution as an audio production or composing music intended for performance by trained musicians. The role of the composer must be redefined in relation to more tradition musical forms [15]. The composer must relegate a certain degree of artistic control to the listener. But the composer cannot expect or demand that the listener possesses any musical skills. The paradigm of a musical game might be more useful than that of a musical composition. The quality of a game in terms of aesthetics and entertainment does not depend on the outcome of a particular instance of playing the game, but rather on the design, architecture and rules of the game itself. Interaction must go hand in hand with artistic content. However, the success of an adaptive musical work will not depend on the sonic result during a particular sitting alone, but on the level of engagement of the listener as well [16]:

"But interaction is not an end in itself, it is necessary to stipulate that the quality of the interactivity depends upon the extent to which the work of art can encourage both critical reflection and creative engagement."

6.2. Visualization

In the foreword to Chion's book *Audiovision,* the well-known sound designer Walter Murch writes [17]:

"For as far back in human history as you would care to go, sound had seemed to be the inevitable and 'accidental' (and therefore mostly ignored) accompaniment of the visual — stuck like a shadow to the object that caused them"

Our intention was to bring sound to the foreground, to suppress the physiological and cultural tendencies to focus on vision. But the web is largely used and accepted as a visual medium. We could not simply ignore the visual aspects of interface design. Visual artist and graphic designer Iben West was associated with the project in the final stages. Her tasks were to 1) design an index web page that would serve as a visual identity to the project, 2) coordinate existing user interfaces for the various works and 3) design new graphic user interfaces for works without integrated graphic elements.

Any graphic user interface, no matter how simple, has a visual aesthetic. It was important that the visual aesthetic of the user interfaces served to complement (or in the least not disrupt) user interaction with the audio content of the various adaptive works. In practice, most of the composers chose to incorporate graphic elements, making graphics an integrated part of the adaptive work. This was a natural approach, since interaction design in the adaptive audio works could hardly be imagined without an existing graphic interface with which to interact.

In some cases functionality and user-friendliness were sacrificed in favor of an artistic approach to graphics. Looking back, it would have been useful to involve a graphic designer and to agree upon the exact role of the graphic user interfaces at an earlier stage.

6.3. Technical considerations

6.3.1. Choice of platform

The Java applet platform turned out to be a suitable choice because it is possible to implement the kind of adaptive musical work that the composers came up with in Max/MSP. Unfortunately the download of the applets turned out to be rather slow and we did not have time to investigate if the download times could be shortened.

6.3.2. Java implementation of Max/MSP patches

As in the DAF-X [7] and Jass [8] systems we managed to implement real time interactive audio and graphics in pure Java. The methodology of using Max/MSP prototypes turned out to be a productive way of conveying the musical ideas to the programmers, at least in the beginning of the implementation process. Later, the composers and the programmers worked together to get a more efficient Java implementation out of the Max/MSP prototype. It might have been an advantage to involve the programmers in the programming of the Max/MSP patches.

6.3.3. Compatibility issues and compromises

Embedding the Java applets into different browsers and different versions of browsers turned out to be a tremendous challenge. A browser environment is not a well-defined environment. We experienced numerous unpleasant surprises when we tested the applets in different environments. The present web site offers no solution to this problem. The applets may work as intended, or they may not. This seems to be the nature of the browser as a media: the amount of processing time available to the browser cannot be defined by an applet and this amount varies according to system configuration. The graphics look different on different browsers, the interaction may not work because the applet does not receive the keyboard and mouse inputs, and the underlying Java implementations on different platforms are not equally efficient. As a consequence the adaptive musical works on the web site do unfortunately behave very differently on different platforms.

6.4. Future work

User interaction in computer games is becoming increasingly sophisticated. In most cases emphasis is placed on visual parameters. The potential of user interaction with sound is great, and here we have only scratched the surface.

7. ACKNOWLEDGEMENTS

Ole Caprani and Jacob Langdal Jensen performed the initial investigations of platforms and designed the Max/MSP to Java methodology. Jacob Langdal Jensen started the implementation of the Java Max/MSP object library, the graphics library, scheduling of Max/MSP java objects during real time playback and he implemented works by Jens Hørsving, Lasse Laursen and initiated the implementation of the work by Wayne Siegel. Later, Timo Dinnesen, Mikkel Gravgaard, Kristian Klüver, Sigurd Melgaard, and Ole Caprani completed the Java implementation of the library and the works by Mikko Jensen and Wayne Siegel. The graphics for the index web page were designed by Iben West. The graphics for the work by Jens Hørsving were designed by the composer. The graphics for the works by Mikko Jensen and Wayne Siegel were designed by Iben West. The graphics for the work by Lasse Laursen were designed by Jacob Langdal Jensen and the composer.

8. REFERENCES

[1] Rowe, R. *Interactive Music Systems. MIT Press,* Cambridge, 1993, p.1

[2] Clark, A. *Defining adaptive music.* Gamasutra, http://www.gamasutra.com/features/20070417/clark_01.shtml

[3] Bergman, L., et al. *Interactive Audio on the Web.* Interactive Audio Special Interest Group, 2002, http://www.iasig.org, p. 8

[4] http://www.cycling74.com

[5] Essl, Karlheinz, *Music Software*, http://www.essl.at/software.html

[6] http://en.wikipedia.org/wiki/Java_%28programming_language%29

[7] Guillemard, M. et al. *J-DAFX –Digital Audio Effects in Java*, Proc. of 8’th Int. Conference on Digital Audio Effects, Madrid, Spain, 2005

[8] Van den Doel, K. *Jass: A Java Audio Synthesis System for Programmers*, Proceedings of the 2001 International Conference of Auditory Display, Espoo, Finland, 2001

[9] Burk, P. *JSyn – A Real-time Synthesis API for Java*, Proceedings of the 1998 International Computer Music Conference, *MIT Press*, Cambridge, 1998.

[10] http://www.softsynth.com/

[11] http://en.wikipedia.org/wiki/Adobe_Flash

[12] Bergman, L., et al. *Interactive Audio on the Web*. Interactive Audio Special Interest Group, 2002, http://www.iasig.org, p. 13

[13] Matthes, Olaf, *flashserver:: communication between Max/MSP and Flash*, http://www.nullmedium.de/dev/flashserver/

[14] http://www.mortenriis.dk/

[15] Siegel, W. *The Challenges of Interactive Dance, An Overview and Case Study*, Computer Music Journal, vol. 22, no. 4, MIT Press, Cambride, 1998

[16] Coulter-Smith, G. *Deconstructing Installation Art: Fine Art and Media Art, 1986-2006,* On-line book, http://www.installationart.net/

[17] Chion, M. *Audiovision: Sound on Screen.* Columbia University Press, New York 1990, p. xvi