FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Gravação e Atuação Musical numa Rede Web de Instrumentos Acústicos

Tiago Carvalhido Morim Casanova

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO

Orientador: António Miguel Pontes Pimenta Monteiro (Prof. Auxiliar)

Julho de 2013

Gravação e Atuação Musical num Rede Web de Instrumentos Acústicos

Tiago Carvalhido Morim Casanova

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Ademar Manuel Teixeira de Aguiar (Prof. Auxiliar)

Vogal Externo: Helena Cristina Coutinho Duarte Rodrigues (Prof. Auxiliar)

Orientador: António Miguel Pontes Pimenta Monteiro (Prof. Auxiliar)

---

15 de Julho de 2013

Resumo

Ao disponibilizar uma plataforma *Web* para gravar ou executar remotamente composições MIDI em instrumentos acústicos de execução robotizada permite que compositores tirem vantagem daquilo que as duas tecnologias podem trazer para a expressão musical.

Estas vantagens refletem-se, por exemplo, na possibilidade de gravar a melodia sem estar presente no local, em operar instrumentos acústicos sem a necessidade e dependência de um intérprete, ou em apresentar melodias de forma simultânea em vários lugares e motivar a emergência de serviços que oferecem algumas destas valências.

Cientificamente falando, a implementação de um sistema similar iria explorar o estudo das consequências, do potencial e das limitações destes sistemas em expressão musical e as suas soluções tecnológicas, identificando ao mesmo tempo as direções futuras no campo de música em rede, concorrentemente num nível técnico e artístico.

A plataforma tecnológica para este sistema encontra-se continuamente em desenvolvimento. Não obstante, este trabalho foca-se essencialmente em adicionar suporte para a receção de composições MIDI via *Web*, agendando e gravando a reprodução de tais composições numa rede de instrumentos acústicos, nomeadamente num piano *Disklavier* e num instrumento de percussão *NotomotoN*, e publicando o ficheiro áudio gerado na *Web*.

Abstract

Providing a *Web* platform for recording or performing MIDI compositions on acoustic robotic instruments allows composers to take advantage of what the two technologies can bring to musical expression.

These advantages reflect, for example, on the possibility of recording without attending the venue, on operating acoustic instruments without the need for an interpreter, or on displaying simultaneously in several places and motivate the emergence of services that offer some of these valences.

Scientifically speaking, the implementation of a similar system would exploit on the study of the consequences of the potential and the limitations of these systems for musical expression and its technological solutions, while also identifying the future directions in the area of network music, concurrently at the technical and artistic level.

The technological platform for this system is in continuous development. Notwithstanding, this work focus mainly on adding support for the reception of *MIDI* compositions via *Web*, scheduling and recording the reproduction of said compositions in an acoustic instruments' network, namely on a *Disklavier* piano and a *NotomotoN*, and publishing the arrangements' file on the *Web*.

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

API	Application Programming Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DIAMOUSES	Distributed Interactive Communication Environment for Live Music Perfomance
DJ	Disc Jockey
DRY	Don't Repeat Yourself
HAML	HTML Abstraction Markup Language
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IP	Internet Protocol
LAMP	Linux, Apache, MySQL, PHP
MIDI	Musical Instrument Digital Interface
MSP	Max Signal Processing
MVC	Model View Controller
Ninjam	Novel Intervallic Network Jamming Architecture for Music
ORM	Object-relational mapping
OSC	Open Sound Control
P2P	Peer-to-Peer
Pd	Pure Data
PHP	Personal Home Page / PHP: HyperText Preprocessor
UDP	User Datagram Protocol

Capítulo 1

Introdução

A música é a forma mais pura de expressão. Não há sociedade que passe sem um vestígio de cultura musical, desde os tempos antigos até aos dias de hoje. Nesta sociedade moderna, em que qualquer forma de tecnologia de caráter pessoal e o uso da Internet se estão a tornar cada vez mais indispensáveis, é inevitável a sinestesia entre o ambiente computacional em rede e o musical[1].

A investigação na área da música em rede tem respondido a muitas das necessidades da comunidade, continuamente contribuindo na forma como executamos, interpretamos e compomos obras musicais. Da mesma forma, a evolução da Internet e das suas formas de utilização[2][3][4] permitiu às pessoas realizar transações síncronas e assíncronas de ideias e músicas[5], facilitando a comunicação sonora[6] com um conjunto de características únicas e oferecendo novos suportes para a criação musical[7][8].

1.1 Enquadramento e Motivação

Neste contexto introduzem-se a especificação MIDI, uma tecnologia cada vez mais aperfeiçoada no que toca à precisão dos seus dados[9], e os *softwares* de sequenciação, nomeadamente aqueles já embutidos nativamente em instrumentos acústicos como no caso dos pianos *Disklavier*, representado na Figura 1, e os descontinuados *Bösendorfer*[10][11]. Estas tecnologias, quando combinadas para providenciar um serviço de gravação autónomo em instrumentos acústicos, podem fornecer ao utilizador comum ferramentas poderosas, inacessíveis de outra forma, para engrandecer as suas próprias composições musicais.

Figura 1 - *Yamaha Disklavier*

A presente dissertação procura explorar o melhor método de implementação de um serviço como o mencionado anteriormente, aliando um sistema autónomo de gravação em estúdio com uma aplicação *Web* capaz de gerir os ficheiros envolvidos e de agendar gravações automáticas, tirando proveito das mais recentes inovações e apostando na interligação entre o espaço musical e o tecnológico.

Mesmo de uma perspetiva empreendedora, em que o desenvolvimento de *software* e a respetiva comercialização deste género de aplicações não é do domínio geral face à dificuldade da sua implementação e à exclusividade do seu público-alvo, é possível deduzir o potencial de semelhante plataforma.

Sendo esta uma área de trabalho em ascensão como ferramenta de investigação de música em rede, torna-se necessário ter em conta o impacto que os resultados de uma análise ao uso de um sistema deste tipo poderão trazer para a comunidade artística e científica.

A plataforma a ser elaborada encontra-se integrada num projeto de investigação em desenvolvimento, estando prevista a implementação contínua do suporte tecnológico para permitir um conjunto de funcionalidades das quais este sistema carece.

1.2 Descrição do Problema

O desenvolvimento das tecnologias no campo da engenharia musical trouxe novidades na forma como gravamos e reproduzimos obras musicais, independentemente da sua complexidade. Complexidade é também uma caraterística do processo de desenvolvimento de uma aplicação *Web* de gravação acústica deste género. Muito advém do facto de ser necessário integrar vários módulos distintos num só sistema, mas também porque a necessidade de tratamento de sinais digitais e dados de áudio e o seu reencaminhamento para um conjunto de instrumentos não tem uma solução universal e imediata.

Existem já *softwares* embutidos em instrumentos acústicos que, para além de se encontrarem num ponto que não lhes permite obter resultados perfeitos em qualquer um dos métodos mencionados, com erros a rondar os milissegundos, também pecam no reconhecimento da componente humana acrescentada na reprodução de qualquer melodia[12]. Isto porque as tecnologias que permitem a reprodução autónoma de melodias MIDI em instrumentos reais não têm em conta as limitações físicas e mecânicas desses instrumentos acústicos.

Apesar dos progressos feitos nesta área são poucos os produtos popularizados dentro da própria comunidade e ainda mais escassos os que chegam ao conhecimento do grande público. A vasta maioria dos desenvolvimentos deste campo é feita sem nenhum objetivo concreto exceto a pura inovação tecnológica, negligenciando fatores como a usabilidade, praticabilidade e utilidade para os utilizadores finais, fatores que têm bastante peso também nos portais *Web* desenvolvidos atualmente.

1.3 Objectivos

Esta dissertação, que tem como objetivo a implementação de um serviço *Web* aliada a uma plataforma de gravação e atuação autónomas de composições MIDI em instrumentos acústicos reais, procura responder a algumas das necessidades qualitativas mencionadas anteriormente.

No âmbito deste objetivo principal, podemos também considerar os seguintes objetivos específicos:

- Elaborar um relatório do estado da arte ao nível da tecnologia;
- Explorar métodos de distribuição de faixas sonoras por diferentes instrumentos e gravação simultânea;
- Desenvolver uma aplicação *Web* com ligações a redes sociais e comunidades artísticas/musicais;
- Criação de um protótipo que interligue todos os componentes desenvolvidos;

- Realização de um estudo das consequências, potencialidades e limitações de sistemas semelhantes;
- Testar e avaliar o protótipo;
- Escrita de um artigo científico com a descrição do modelo, da arquitetura de *software,* e com os resultados do teste ao protótipo.

1.4 Metodologia

A metodologia seguida para a realização deste projeto comportou diversas fases distintas. A primeira iniciou-se com a pesquisa do estado da arte das diversas áreas abordadas pela dissertação. De seguida entrou-se na fase de especificação, com a análise dos módulos e requisitos do sistema e a definição das funcionalidades básicas do servidor. Posteriormente, foi elaborado o módulo em ambiente de síntese de áudio e foi criado o *patch* que estabeleceu a ligação entre o servidor e os instrumentos acústicos. Por fim, foi desenvolvido o *Website* e estabelecidas ligações a comunidades *online*, completando o processo de prototipagem da solução, tendo sido as suas funcionalidades otimizadas conforme o resultado dos testes e avaliações das mesmas.

De forma paralela, foi realizado um estudo às potencialidades e limitações de sistemas deste género, para além da escrita da documentação requerida, que inclui um artigo científico com a descrição do modelo, com a arquitetura do *software* desenvolvido, e com os resultados obtidos ao longo deste projeto.

1.5 Resultados esperados

No âmbito deste projeto pretende-se criar uma aplicação *Web* que permita aos utilizadores enviar as suas composições musicais para serem gravadas em instrumentos acústicos inacessíveis de outra maneira, retornando um ficheiro áudio de maior qualidade sonora do que obteriam eletronicamente. De uma forma mais pormenorizada, espera-se obter um *Website* que atraia as atenções da comunidade científica e artística, estabelecendo ligações com algumas das existentes e com redes sociais, que permita a submissão de ficheiros MIDI por parte dos utilizadores registados e a subsequente obtenção dos ficheiros áudio com as gravações, como podemos ver no esquema da Figura 2.

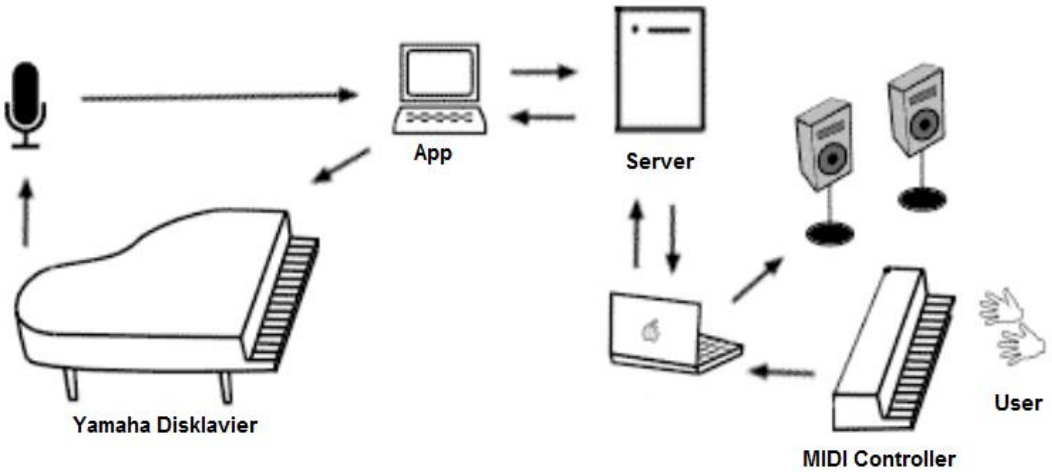


Figura 2 - Sistema Principal

Estes ficheiros poderão ser decompostos para separar as melodias por diferentes instrumentos, no caso do protótipo um piano *Disklavier* e um instrumento de percussão denominado *NotomotoN*, que as executarão sem qualquer necessidade de intervenção humana. As melodias serão gravadas, numa sala sonoramente isolada, por microfones de alta qualidade e os ficheiros resultantes serão disponibilizados com uma licença *Creative Commons* para toda a comunidade *online*. A Figura 3 mostra o instrumento de percussão a ser utilizado nas gravações.

Figura 3 - *NotomotoN*

Não obstante, para além da prototipagem de uma aplicação como a descrita, prevê-se a obtenção de resultados quanto à exequibilidade da utilização de sistemas autónomos de execução e gravação de melodias MIDI. Adicionalmente, no decorrer deste projeto, é esperado

realizar um estudo quanto às consequências, potencialidades e limitações de sistemas deste género.

1.6 Trabalho Relacionado

Visto que esta é uma dissertação com um âmbito bastante alargado, mas limitada no tempo, ao longo da pesquisa foi possível encontrar um grande número de artigos com trabalho relacionado de alguma forma com o que é aqui abordado. No entanto, em toda a literatura analisada, não foi encontrado um projeto que aborde o problema descrito de forma similar ao proposto por esta dissertação.

O projeto aqui proposto tem como caraterística uma multidisciplinariedade que permite separar os trabalhos relacionados em três grupos. Por um lado o protótipo irá permitir que uma obra musical seja criada de forma assíncrona e não presencial, no caso uma atuação musical em instrumentos acústicos gravada para um utilizador num local remoto. Esta caraterística é partilhada com sistemas de suporte composicional musical, como é o caso de projetos *Web* como o *Ninjam*[13], *eJamming*[14] ou *JackTrip*[15], entre outros.

É também importante considerar a semelhança de propriedades de áreas como a telepresença ou telerobótica, onde os utilizadores fazem uso de tecnologias que de alguma forma imitam a presença humana ou a sua habilidade de interpretação musical para realizar uma execução das suas melodias. Projetos como o quarteto robótico de *Sergi Jorda*[16], os vários robôs de percussão de *Ajay Kapur*, criador do *NotomotoN* [17], ou o próprio *software* "*Remote Lesson"* da *Yamaha*[18], criadora dos pianos *Disklavier*, compartilham destas funcionalidades, tal como o protótipo a desenvolver.

Finalmente, a acústica e os atributos únicos do local de gravação levam a ter em conta a reverberação do mesmo, e os espaços reverberantes disponíveis por rede, como no caso do *The Silophone*[19], que permite que artistas de som acedam remotamente a tipos de instalações diferentes das habituais salas de estúdio para gravar os seus sons.

1.7 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 5 capítulos. No capítulo 2, é descrito o estado da arte e são apresentados com mais detalhe alguns trabalhos relacionados. No capítulo 3 é detalhado o planeamento seguido ao longo desta dissertação, enquanto no capítulo 4 se explica de forma pormenorizada a implementação produzida. No penúltimo capítulo é oferecida uma apresentação de testes e resultados obtidos bem como alguns exemlos de

interface e modos de execução. No capítulo 6 são apresentadas as conclusões finais conjuntamente com algum do trabalho futuro para este projeto.

Capítulo 2

Revisão Bibliográfica

Ao longo do presente capítulo será abordado o estado da arte de cada uma das áreas relacionadas com o tema da presente dissertação. Durante toda esta análise serão apresentadas e discutidas diversas alternativas dando privilégio aquelas que serão, à partida, mais indicadas para uma utilização final de acordo com os requisitos da dissertação.

2.1 MIDI

O MIDI (*Musical Instrument Digital Interface*) é um *standard* protocolar que permite a comunicação entre vários dispositivos eletrónicos, computadores e outros dispositivos musicais relacionados que tenham a possibilidade de se ligarem e transmitirem dados de uns para os outros. Foi inicialmente desenvolvido a pensar na intercomunicação entre dispositivos eletrónicos de várias companhias diferentes, que devido a incompatibilidades técnicas não conseguiam sincronizar tempos entre si, mas rapidamente se expandiu para permitir que músicos por todo o mundo pudessem compor obras com grande grau de complexidade, sem a necessidade de possuírem grandes capacidades de notação musical ou *hardware* dispendioso, e sem terem de recorrer a estúdios de gravação.

Este protocolo é caraterizado por uma listagem de eventos que especificam notação musical, tom, frequência e velocidade da nota para além de transmitir sinais de controlo para parâmetros como o volume, *vibrato*, extensão do áudio e sinais do relógio que sincronizam o tempo entre os aparelhos envolvidos. Estes eventos podem estar divididos em no máximo 16 canais num só ficheiro, e cada canal pode ser transmitido separadamente para um dispositivo diferente. Quando estes sinais são enviados para instrumentos robotizados como um *Disklavier* ou um *NotomotoN*, podem controlar a geração de som e reproduzir melodias complexas.

As mensagens MIDI são enviadas apenas num sentido, de modo a que se for requerido uma ligação bidirecional, são necessários dois cabos. Estes cabos são limitados a uma extensão máxima de 15 metros de modo a limitar interferências, visto não existir nenhuma funcionalidade de deteção de erros inerente neste protocolo. No entanto, alguns processadores de dados MIDI podem conter filtros MIDI, que retiram dados supérfluos da sequência de eventos.

Uma das funções principais do processamento interno de um dispositivo MIDI é a de sincronizar os relógios e a sua velocidade entre os aparelhos interligados. Contudo, a transmissão de mensagens MIDI é feita sequencialmente, o que quer dizer que só pode enviar um evento de cada vez, sendo o segundo de dois eventos enviado apenas aquando do final do primeiro. Isto leva a problemas de cronometragem, visto que o segundo destes eventos MIDI é atrasado em 1 milissegundo, sendo a perceção humana capaz de detetar diferenças tão pequenas quanto 1/3 de um milissegundo.

A forma mais comum de corrigir este inconveniente foi o surgimento de dispositivos com vários canais de entrada e saída, visto que o *timing* melhora quando a sequenciação de eventos é distribuída por várias portas ao invés de vários canais por uma só porta.

2.2 Música em Rede

Avanços tecnológicos têm contribuído não só na forma como fazemos música em grupo mas também na estética da música em si ao longo dos anos, aprimorando métodos de como produzir e moldar sons e de como agregar sons em peças musicais[20]. Isto afeta a forma como compomos música, a sua atuação em instrumentos regulares e a interpretação da mesma do outro lado do processo, no utilizador ou nas audiências.

As redes de computadores, mais especificamente a Internet, facilitam a comunicação musical com um conjunto distinto de atributos, e oferecem novos suportes de criatividade na composição musical[21]. Estas caraterísticas podem ser vistas a emergir nos trabalhos de John Cage, um dos primeiros a ter consciência do potencial de explorar procedimentos e processos mais elaborados e interdependentes da criação de música nas suas composições, nomeadamente no seu trabalho de 1951, "*Imaginary Landscape No. 4"*. Esta melodia foi criada através da manipulação de 12 rádios AM por 24 intérpretes, com definições de volume e sintonização diferentes, soando de forma distinta dependendo do sinal AM que se apanhava. Explorando a evolução e a dinamização de novos contextos musicais através de *transístor radios*, J. Cage forneceu-nos o primeiro conceito básico de um cenário musical descentralizado dos instrumentos e centrado no processo em si[22].

Com o surgimento do computador pessoal, grupos como a *League of Automatic Music Composers*[23], começaram a experimentar projetos de criação de música ligando vários

computadores, instrumentos eletrónicos e circuitos analógicos. Mais tarde, com a evolução do grupo para *The Hub*[24] em 1997, surgiu uma das primeiras experiências com o envio de dados MIDI através da rede para localizações distribuídas. Este grupo é normalmente creditado como um dos pioneiros na criação conjunta de uma obra musical utilizando uma rede de computadores e explorando as suas possibilidades interativas[25].

Quando olhamos para a quantidade de projetos, ideias de pesquisas, investigações e publicações no campo de música em rede[26], complementadas com novas iniciativas por parte da indústria musical, é possível verificar o potencial desta área[27]. Existem ainda algumas barreiras culturais e técnicas, relacionadas com largura de banda, latência e sincronização do sinal de áudio no caso de sistemas em tempo real[5][17][21], que terão de ser ultrapassadas para trazer música em rede à comunidade de maneira a observar uma aceitação generalizada, requerendo uma abordagem que entenda as questões culturais, sociais, técnicas e musicais próprias a este tipo de sistemas[28].

2.3 Tecnologias Web

Desde o início da comercialização da *Web,* a indústria do desenvolvimento *Web* tem crescido exponencialmente, com pequenas e grandes companhias a aperceberem-se do potencial e das vantagens do mercado *online*.

O desenvolvimento *Web* é um campo que pode ir desde a mais simples das páginas estáticas até a aplicações ou serviços complexos, tendo em conta tarefas como *Web design*, segurança de rede, tratamento de bases de dados, *scripting* dos clientes e servidores, etc.

Não perdendo de vista o surgimento do *HTML5* e *CSS3* para desenvolvimento do lado do cliente, conjuntamente com *JavaScript* e a sua biblioteca *jQuery*, e as várias opções tecnológicas de base de dados, nesta seção pretende-se analisar com mais detalhe as linguagens e *frameworks* aplicáveis ao desenvolvimento do lado do servidor[29].

2.3.1 PHP

Esta é a linguagem mais utilizada atualmente para o desenvolvimento de um servidor na indústria, eclipsando qualquer outra em termos de popularidade, ofertas de trabalho e número atual de desenvolvedores, com exemplos como a *Wikipedia* ou *Facebook*. É uma linguagem interpretada, com uma 'tipagem dinâmica' e que na sua versão mais recente, *PHP 5*, permite a indução de tipos e a definição da visibilidade de objetos[30].

Comparativamente, a linguagem *PHP* tem uma simplicidade inerente, uma curva de aprendizagem praticamente rasa e os critérios de entrada são extremamente baixos, permitindo que *designers* e programadores inexperientes possam ser produtivos quase imediatamente. É graças a esta simplicidade que o *PHP* é tão popular, sendo completamente orientado para a

Web, ao contrário do *Ruby* ou do *Python*, e com uma imensidão de recursos disponíveis, quer em termos de *frameworks,* bibliotecas ou documentação[31].

Tem também um modo de implementação simples, permitindo aos programadores preocuparem-se apenas com a localização dos ficheiros, já que a maioria dos serviços anfitriões usa um ambiente *LAMP – Linux, Apache, MySQL, PHP*.

Por outro lado, é capaz de bater alguns dos seus concorrentes quando comparado em termos de tempo de execução nos *benchmarks* mais recentes. A simplicidade desta linguagem e o rumo que o seu desenvolvimento teve ao longo dos anos levou a que se perdessem de vista muitos *standards* de codificação, o que por sua vez leva a grandes porções de código desleixado e mal escrito, tornado penoso um processo de *upgrade* de versões anteriores da linguagem e sendo cada vez menos fluído, quiçá divertido, programar em *PHP*[31].

2.3.2 Python (django *framework*)

Python é uma linguagem de programação de alto nível com suporte para múltiplos paradigmas, sendo uma linguagem cuja filosofia de implementação sempre se focou na legibilidade e simplicidade do seu código, com uma sintaxe clara e expressiva. No repertório de aplicações desenvolvidas sobre esta linguagem encontra-se nada mais nada menos que o *Google* e o *YouTube*[32].

As suas funcionalidades incluem um sistema de tipagem dinâmico e gestão automática de memória tendo uma grande flexibilidade em termos de aplicações, com uma razoável variedade de bibliotecas disponíveis para complementar a já bem-equipada biblioteca *standard*. Os tempos de execução dos *benchmarks* mais recentes mostram que o *Python* deixa invariavelmente a concorrência para trás, o que também acontece quanto à usabilidade desta, sendo bastante recomendada para iniciantes devido à sua simplicidade[31].

Django é uma *framework* gratuita para aplicações *Web* que segue o padrão de arquitetura *Model View Controller (MVC)*, e tem como objetivo facilitar a criação de *Websites* complexos orientados a bases de dados. A filosofia por detrás do *Django* é enfatizar a reutilização e simplificar a ligação de novos componentes, permitir o desenvolvimento acelerado e seguindo o princípio *Don't Repeat Yourself (DRY),* disponibilizando ainda ferramentas administrativas de forma opcional[33].

2.3.3 Ruby (Rails *framework)*

Ruby é uma linguagem de programação dinâmica, reflexiva e orientada a objetos com suporte para vários paradigmas, partilhando também a funcionalidade de gestão automática de memória do *Python*. A filosofia por detrás do desenvolvimento desta linguagem foi a necessidade de criar um balanço entre os paradigmas funcional e imperativo, criando uma linguagem de *script* que fosse mais poderosa que *Perl* e mais orientada a objetos que *Python*.

Como propósito principal, o seu criador afirmou que o *Ruby* foi desenhado para tornar o processo de programação produtivo, flexível e divertido[34].

Para conseguir adaptar as propriedades da linguagem a um contexto *Web*, foi criada uma *framework* denominada *Ruby On Rails* ou simplesmente *Rails*, que segue muitos dos princípios da engenharia de *software* como *DRY,* uma arquitetura MVC, convenções ao invés de configurações, entre outras. Esta *framework* vem preparada para criar páginas/aplicações que adquiram informação do servidor, consultar uma base de dados e apresentar *templates* a partir do momento de instalação.

De modo a conseguir isso, a *framework* tem um sistema de encaminhamento independente do servidor. O *Rails* segue muitos dos princípios da linguagem em que assenta, disponibilizando geradores automáticos, uma imensidão de *plugins* denominados *gems* como contributo da sua comunidade, ferramentas de teste integradas e *Active Record ORM*, que expõe a relação de chaves estrangeiras numa base de dados na forma de uma instância de objeto[35].

A contrapartida de uma linguagem que ultrapassa o *PHP* em termos de *standards* é a dificuldade de aprendizagem, quer para a *framework Rails* quer para a linguagem *Ruby*. No *Rails*, muitos dos processos foram automatizados o que cria a ilusão de que o desenvolvimento foi facilitado, mas na verdade apenas o torna mais complicado de entender. Falando em entender, ao desenvolver uma aplicação em *Rails* é necessário entender a *stack* toda da *Web*, desde o servidor às páginas.

Caso exista algum erro, é necessário consultar *logs* e procurar o erro manualmente pois uma aplicação em *Rails* não tem as mesmas caraterísticas de *debug* que o *PHP*, ou seja, não existem as típicas mensagens de erro. Mesmo algumas das funcionalidades do *Ruby* não são imediatamente percetíveis, e aprender *Rails* implicaria também aprender novas linguagens como *CoffeeScript, Slim, HAML*, indicando que um programador teria de ser o mais poliglota possível pois os critérios de entrada não são benevolentes[31].

2.4 Espaços de Reverberação em Rede

Existem sistemas que permitem que artistas musicais possam aceder a instalações de reprodução e gravação incomparáveis, de forma remota através da rede *Web*, de modo a tirar proveito das caraterísticas únicas do local nas gravações das suas melodias, tal como no exemplo do *Silophone,* explicado mais adiante. Como o protótipo permite que os seus utilizadores acedam e explorem as caraterísticas acústicas de um *grand* piano e tirem proveito da reverberação do espaço onde ele se encontra, tornando-se naturalmente uma parte integrante da gravação, a reverberação é uma componente de qualidade inconfundível na peça musical gravada. A reverberação singular e incomparável da acústica do local remoto é uma das vantagens de ter uma gravação feita através deste sistema, permitindo analisar de forma mais detalhada este fenómeno.

A reverberação natural é feita com a reflexão dos sons nas superfícies dos objetos ou paredes. Estas superfícies dispersam o som, enriquecendo-o num processo que estratifica o som com as suas próprias reflexões. Este mesmo processo permite colorir o som até um certo ponto, induzindo uma pequena variação no timbre, e qualquer intérprete que tenha atuado uma mesma peça em mais do que uma sala de concerto poderá atestar exatamente isso. As diferentes caraterísticas da reverberação de um local podem afetar uma melodia de várias formas, podendo levar a mudanças na dinâmica de instrumentos específicos ou na posição dos artistas, ou a um pequeno ajuste no ritmo da mesma para evitar efeitos indesejáveis. Podemos ver um exemplo de reverberação de som numa sala na Figura 4, onde *S* representa a origem do som, por exemplo um instrumento, e *L* o destino, ou seja, um ouvinte ou um microfone.

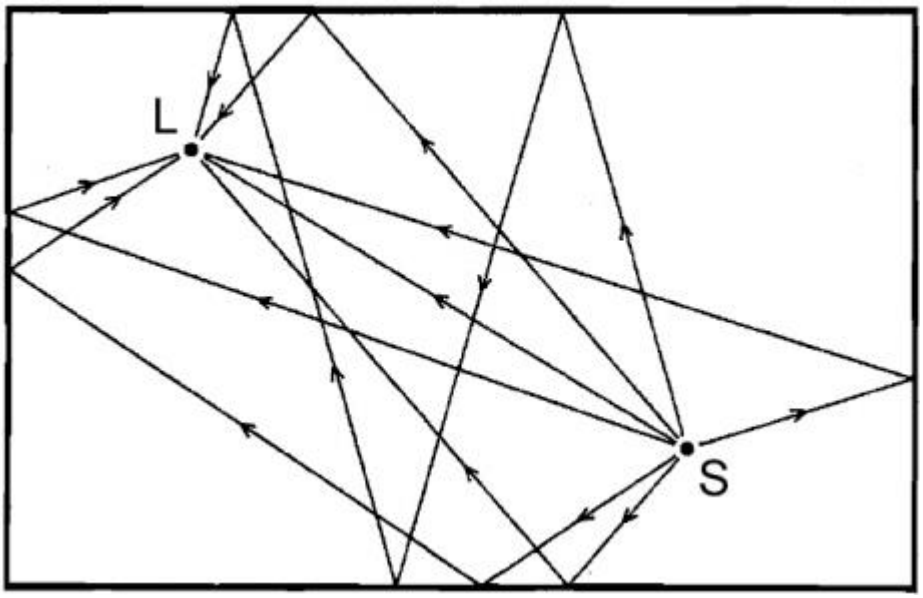


Figura 4 – Reverberação

A quantidade e a qualidade da reverberação que ocorre num determinado ambiente são influenciadas pela dimensão do local, pelo tipo, forma e número de superfícies que o som encontra. Um ouvinte apenas recebe parte do som diretamente, sendo que o resto é recebido com atraso pelas reflexões do som nas paredes, teto e chão do espaço. Estas reflexões prolongam o som percetível pelo utilizador, sendo que as ondas sonoras carregam menor amplitude quanto maior for a distância que percorrem[36].

Um projeto porta-estandarte para o estudo da reverberação de um local é o *Silophone*[37]. Este é um projeto musical em rede iniciado pelo grupo *The User* em *Montreal*, tirando proveito de uma instalação de armazenamento de cereais abandonada no porto da cidade. Os sons são transmitidos para o interior do silo pelo telefone, ou via *upload* pela *Internet,* ou ainda produzidos por um microfone num observatório sonoro perto do silo, e ouvidos pelo *stream,* em tempo real, da página do *Silophone* ou no observatório pelas colunas instaladas para o efeito. Podemos ver o interior do *Silophone* na Figura 5.

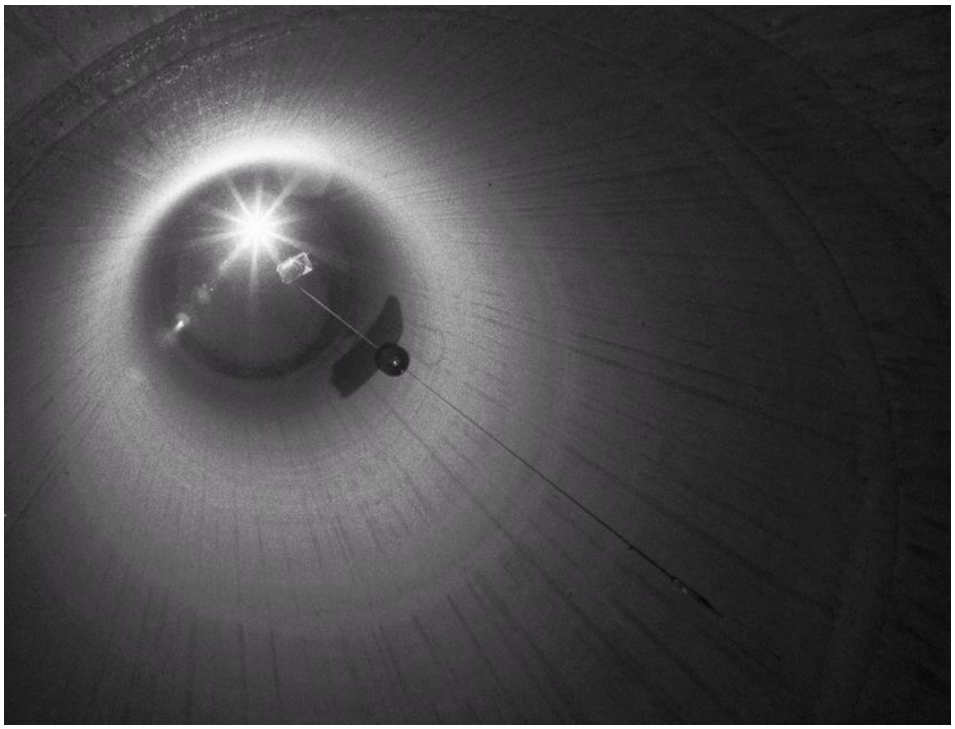


Figura 5 - *The Silophone (Silo #5)*

O som que entra no silo tira proveito das caraterísticas excecionais do local, com uma altura de dez andares e tempos de reverberação de mais de vinte segundos em alguns casos, transformando tudo o que entra naquele local em algo sonoramente único[19].

O projeto resultante desta dissertação não pode deixar de ter em conta este fator inevitavelmente crucial devido ao papel que terá no resultado final de cada processo de gravação. Numa situação de interpretação musical como é o caso, as ondas sonoras produzidas irão ser influenciadas não só pelas paredes da sala mas também pela mobília, tapetes, pessoas e roupas, alterando as caraterísticas da gravação e deixando uma marca indistinguível na obra musical.

2.5 Telepresença & Telerobótica

Tal como os nomes deixam implícito, a caraterística intrínseca a estas áreas que o sistema irá emular aquando da sua execução, será a emulação da presença humana ao executar a melodia no *Disklavier*, execução feita por meios quase robóticos. A fascinação por produzir autómatos de caráter musical reverte até ao século II[38], aos tempos do Renascimento de Leonardo Da Vinci, ou mesmo de Beethoven e Mozart, que compuseram músicas para instrumentos automáticos[39]. Abordagens mais recentes incluem artistas que dedicaram as suas vidas a explorar e a compor para instrumentos autónomos[40][41], ou para instrumentos controlados por MIDI[42], ou a criar novos instrumentos acústicos robóticos[43][44].

Sergi Jorda, de Barcelona, tem tido um grande contributo no desenvolvimento de tecnologias de atuação musical, incluindo em Robótica, com projetos como o *Afasia*. Criado em 1998, *Afasia* consiste num quarteto de robôs, música eletrónica, vídeo e animações controladas por um intérprete num exosqueleto. Os robôs são controlados por MIDI, correspondendo a cada um deles uma porta virtual MIDI e um driver no computador principal[16]. O quarteto é mostrado na Figura 6.

Figura 6 - *The Robot Quartet (Afasia)*

Estes robôs foram desenhados de modo a parecerem instrumentos e não intérpretes humanos como o violinista criado pela *Toyota,* e fogem à regra de antropomorfismo aplicada geralmente a estes sistemas[43].

Ajay Kapur também tem sido parte envolvente no desenvolvimento de trabalhos robóticos musicais, mais notoriamente em instrumentos de percussão, incluindo o *MahaDeviBot*, *Edholak and Etabla* e o *NotomotoN*[45]. Quando falamos de robôs em rede controlados por MIDI, conseguimos obter alguns exemplos com a *framework* GIGAPOPR[17].

Desenhado para suportar uma ligação bidirecional através da rede, este sistema transmite dados MIDI áudio e vídeo não compactados com o intuito de possibilitar que intérpretes dispersos geograficamente possam interagir uns com os outros. Desta forma, e como já visto anteriormente, a latência unidirecional ou de ida e volta afeta significativamente a qualidade da interação[17].

Existem também vários projetos envolvendo um *Disklavier* que tentaram controlar um instrumento acústico através da rede, como o "*Radio – Drum Driven Disklavier"* em 1998, que ligava uma bateria para controlar um *Disklavier* através de um computador[46].

Foi também lançado o *software* "*Remote Lesson"* pela *Yamaha* com o intuito de suportar ensino em tempo real para a execução de certas melodias, ligando dois ou mais instrumentos através da rede[18]. No entanto, a reprodução e transmissão de dados utilizando este *software* sofre de atrasos superiores a um segundo.

Entre estes projetos, não foram encontrados sistemas gratuitos e do conhecimento do grande público que permitam a colaboração de compositores utilizando como meio de troca de dados um MIDI gravado previamente à execução. Os sistemas encontrados preferem uma abordagem que tentam aperfeiçoar a interação em tempo real[21] do que a renderização e gravação de obras musicais cuidadosamente elaboradas, excluindo assim artistas que prefiram não atuar ou improvisar ao vivo mas que gostassem de tirar proveito das vantagens tecnológicas dos seus computadores ligados em rede à Internet para criar gravações com som 'puro' e exibir os seus trabalhos.

2.6 Sistemas de Suporte Composicional

O protótipo resultante deste projeto encapsula algumas semelhanças com algo que podemos chamar de "sistemas de apoio composicionais", partilhando a natureza remota assim como síncrona e assíncrona de criar uma peça musical, interpretada e gravada numa localização remota.

Visto que podemos considerar o resultado da gravação dos MIDIs como uma obra de arte criada conjuntamente, pode também ser considerada uma composição gerada por um utilizador, tal como feito por um *Disc Jockey* (DJ) na sua mesa de mistura ou numa *mixtape* onde os compositores adicionam as suas contribuições remota e sequencialmente.

Apesar das contribuições individuais não serem agrupadas umas sobre as outras, como acontece na maioria dos projetos a rever nos pontos seguintes, o sistema final compartilha das caraterísticas assíncronas e remotas de criar música em grupo. Os próximos tópicos analisam alguns dos muitos exemplos disponíveis[47][48].

2.6.1 eJamming

O *eJamming* foi lançado em 2008 e pode ser considerada uma plataforma de áudio multi-utilizador onde músicos podem procurar outros colaboradores de acordo com as suas preferências, seja instrumento, destreza, proficiência ou estilo musical, através das suas bases de dados. Também permite a criação de grupos abertos *online*, onde cada utilizador se pode juntar a uma sessão a decorrer[49].

Este *software* não é a primeira plataforma *online* a disponibilizar uma comunidade focada nos seus utilizadores, utilizando uma arquitetura *peer-to-peer* e um protocolo de pacotes *UDP*, mas fá-lo através da transmissão de dados MIDI o que reduz a largura de banda requerida a um mínimo.

A vantagem adicional de usar este tipo de dados prende-se com o facto de estes apenas serem transmitidos aquando do desencadeamento de um evento, o que significa que o sistema não necessita de manter uma corrente constante de dados, tornando-o menos vulnerável a *jitter*. Mesmo quando ocorre este atraso, o pacote MIDI é simplesmente tocado de forma sequencial à chegada.

A aplicação de *desktop* fornece um sistema de gravação para múltiplas faixas. Ao utilizar este modo, tirando proveito de uma interface com informação relativa a instrumentos e a outros colaboradores, os músicos podem monitorizar a sua atuação de forma completamente síncrona com faixas previamente gravadas, disponibilizadas também para edição instantânea[14].

2.6.2 SoundJack

Esta é uma aplicação de *software* independente desenvolvida desde 2005 como o resultado das experiências musicais e da investigação em música distribuída na Internet de Alexander Carot, e é inspirado no projeto *JackTrip* do grupo de investigação *SoundWIRE*[50] da Universidade de Stanford[51][3], que por sua vez inspirou a mais recente *framework DIAMOUSES*[52].

O ponto fulcral de diferenciação do *SoundJack* é o facto de permitir acesso direto ao *buffer* da placa de som de maneira a permitir o envio do mesmo para o destino especificado via

protocolos *UDP* e *IP* sem requerer nenhum tipo de servidor central, enquanto utilizando uma arquitetura *P2P*[53].

A contrapartida prende-se com a maior dependência da qualidade do *stream* de áudio com as definições da placa de som, das suas especificações e da qualidade geral da rede em termos de velocidade e largura de banda. De maneira a reduzir as falhas de *jitter* o *buffer* de áudio pode ser aumentado manualmente. A interface da aplicação fornece toda a informação necessária e os valores significativos como atrasos, tamanho dos *buffers* e as derrapagens e défices do áudio[54].

2.6.3 Ninjam

Ao contrário das aplicações mencionadas anteriormente, que se tentam focar numa abordagem que permita interação musical em tempo real, o *Novel Intervallic Jamming Architecture (Ninjam)* coloca o enfâse na experimentação e expressão musical criando um ambiente de improvisação assumindo que a latência da rede torna impossível uma sincronização em tempo real perfeita para os colaboradores participantes[13].

Para contornar esta hipótese, foi introduzido um princípio na comunicação de dados em que a latência é incrementada de tal maneira que os intérpretes na sessão recebem os dados uns dos outros com o atraso de pelo menos uma unidade. Tecnicamente, os músicos tocam de forma assíncrona em relação à música que os restantes participantes tocaram, há pelo menos uma unidade de tempo de atraso atrás, ao que os criadores do *Ninjam* chamaram de *faketime*[55].

Comparativamente às aplicações anteriores, esta usa um sistema de servidores central para controlo de sessões e para aplicar os mecanismos de atraso. O princípio de atraso básico para o *stream* de áudio embutido no *Ninjam* por medidas pré-definidas requer que este necessite de um metrónomo de maneira a ter uma instância de controlo do tempo central[47].

2.6.4 JackTrip

O *JackTrip* encontra-se em desenvolvimento pelo grupo *SoundWIRE* da Universidade de Stanford. Este grupo explora várias áreas no uso de redes de computadores para edição e composição de música, tais como o *stream* de áudio por múltiplos canais[56], novos modelos físicos de instrumentação acústica bem como novos meios virtuais, a sonorização de atuações via *Internet*, psicoacústica[57] e práticas de novos meios de composição musical pela rede[15][58].

O *software* em si assenta nos sistemas operativos *Linux* e *Mac OS X*, sendo utilizado para colaboração musical pela *Internet* através de vários canais, tantos quanto os computadores e a qualidade da rede permitir. Comparativamente a outras aplicações, o *stream* é feito de forma

bidirecional entre os utilizadores em sessão, utilizando sinais de áudio de alta qualidade e não compactados, eliminando a latência introduzida por algoritmos de compressão aquando da transmissão de dados utilizando *UDP*. Esta aplicação usa o *JACK Audio Connection Kit*[59] como o seu servidor de áudio, sendo de salientar que os parâmetros latência, largura de banda, tamanho de *buffers*, têm de coincidir em todos os computadores da sessão[47].

Este tipo de *design* foi implementado de modo a servir de base para aplicações futuras já que se encontra bastante dependente da qualidade dos recursos utilizados e da distância dos utilizadores em cada sessão, mas tem por objetivos conseguir maximizar o fluxo de dados colocando pacotes de áudio na rede assim que são tratados na placa de som, trabalhar com qualquer número de canais e ser flexível no roteamento e na mistura dos canais de áudio de e para novos anfitriões[60].

2.7 Ambientes de Síntese de Áudio

Softwares de ambientes de síntese de áudio consistem tipicamente numa linguagem de programação de áudio, que pode ser textual ou gráfica, e numa interface de utilizador onde realizar o *design* e a execução da mesma. Apesar de muitos dos ambientes existentes atualmente serem equiparáveis no que diz respeito à produção de áudio de grande qualidade, as suas diferenças e especialidades são o que atraem os utilizadores a cada plataforma específica.

Tomando em conta os aspetos tecnológicos mais relevantes, pretende-se nesta secção realizar uma comparação que saliente os problemas e vantagens da utilização de cada uma delas[61]. Em termos de possibilidades de escolha, podemos ver algumas comparações técnicas disponíveis na tabela incluída em anexo[62].

2.7.1 MAX/MSP

Max é uma linguagem de programação visual orientada para música e conteúdos multimédia, providenciando os elementos necessários para a criação de sons originais ou *media* interativa[63]. Originalmente escrito nos anos 80 por Miller Puckette, na forma de um editor denominado *Patcher*[64], tem sido frequentemente usado por compositores, investigadores, desenvolvedores de *software* e artistas para a criação dos seus projetos, enquadrando-se no paradigma de linguagens de fluxo de dados.

O programa em si é altamente modular, com a maioria das rotinas na forma de bibliotecas partilhadas, e com um suporte nativo para uma grande variedade de controladores e dispositivos, especificamente no suporte a MIDIs. Dispõe ainda de uma API que permite o desenvolvimento de novas rotinas na forma de objetos externos, podendo escolher entre várias linguagens de programação como *Java* ou *Ruby*, conferindo-lhe uma grande base de

utilizadores, também eles programadores, fidelizados a melhorar o *software* e as suas funcionalidades.

Graças ao seu *design* extensível e a uma interface gráfica *user-friendly*, onde podemos escolher entre conectar objetos representativos dos sensores ou rotinas que queremos utilizar ou codificar manualmente o que pretendemos que o programa execute, é hoje em dia considerada a *língua franca* para o desenvolvimento de *software* musical interativo[65].

2.7.2 Pure Data

Pure Data, ou denominado simplesmente *Pd*, é um ambiente de programação gráfico para a criação de *software* musical ou trabalhos multimédia interativos[66]. Desenvolvido pelo mesmo criador do *Max/MSP*, este é um projeto *open source* mais utilizado como ambiente de composição interativo[67], como estação de síntese e processamento de áudio em tempo real[68] e como ferramenta de pesquisa na área de sonoridade acústica[69].

Ao contrário do seu antecessor espiritual, o *Pure Data* trabalha com o que podemos considerar "dados puros", ou seja tem funcionalidades que lhe permitem aceder aos *bytes* dos dados que analisa, e foi desenhado desde o início para controlar e processar os elementos áudio na CPU da unidade anfitrião da aplicação, ao invés de delegar essa tarefa a uma placa de processamento de sinal digital. Curiosamente, o código da extensão *MSP* responsável pela parte de processamento de áudio no *Max* é feito em *Pure Data*[70].

O *Pure Data* é modular, como o *Max,* estando as suas funcionalidades dispersas por objetos ou módulos, *externals* ou *patches* respetivamente, que são as bases para os programas deste *software*. No entanto, o *Pd* é também uma linguagem de programação, extensível por uma API pública para um *range* variado de linguagens[71]. Permite ainda, de forma nativa, colaboração em tempo real pela rede ou Internet entre músicos, através de um protocolo de rede denominado *FUDI*[72], e pode ser utilizado como interface gráfica para outro ambiente de síntese de áudio, o *SuperCollider*.

2.7.3 SuperCollider

SuperCollider[73] é um ambiente e linguagem de programação originalmente lançado com o intuito de sintetizar áudio[74] em tempo real e para composições algorítmicas[75]. Desde o seu lançamento, o *software* foi desenvolvido por programadores, cientistas e artistas de igual maneira, evoluindo para uma linguagem de programação eficiente e dinâmica, fornecendo uma *framework* para programação interativa, pesquisa em sonoridade acústica e composição algorítmica[76].

Este ambiente tem a particularidade de ser estruturado com uma arquitetura cliente[77] e servidor[78], *scsynth* e *sclang*, que podem ser usados individualmente e que comunicam utilizando *Open Sound Control*, tornando possível o controlo dos mesmos através de outra aplicação ou linguagem de programação[79].

A linguagem de programação combina a estrutura orientada a objetos do *Smalltalk* e algumas funcionalidades do paradigma de linguagens funcionais com uma sintaxe em tudo semelhante a *C* ou *Ruby*[80]. As caraterísticas dinâmicas desta linguagem permitem que o *SuperCollider* possa ser usado para *live coding*[25], permitindo partilhar e modificar instantaneamente código e objetos durante uma execução.

2.7.4 Csound

Csound[81] é um projeto de síntese de áudio e um sistema de processamento de sinal que providencia funcionalidades que facilitam a composição e atuação musical em todos os sistemas operativos e plataformas atuais. O seu uso não é restrito a nenhum estilo de música, tendo sido aplicado em ambientes musicais experimentais para filmes, televisão e composições algorítmicas[82].

Este ambiente diferencia-se dos restantes em vários níveis. Em primeiro lugar, o seu *input* são dois ficheiros de texto, um com a descrição da natureza dos instrumentos a utilizar, e outro com as notas e outros parâmetros do cronograma. A linguagem possui vários IDEs disponíveis, sendo possível personalizar um próprio, uma interface *multitrack* e várias ferramentas que facilitam a análise e a ressíntese do áudio gerado[83].

Fornece também uma API para linguagens como *Lisp, Haskell, Lua* e *Python*, existindo também uma biblioteca de composição algorítmica nesta linguagem. O *Csound* permite ainda a computação de áudio com dupla precisão: apesar de isto não significar melhorias em termos da qualidade do áudio, torna-se necessário para reproduzir com precisão a excentricidade de algumas melodias existentes[84].

2.7.5 openFrameworks

openFrameworks[85] é um *kit* de ferramentas *open source* em C++ desenhado para auxiliar o processo criativo providenciando uma *framework* simples e intuitiva orientada para experimentação. O *kit* foi criado de forma a fornecer as ferramentas necessárias para fins artísticos gerais, juntando várias bibliotecas vulgarmente usadas como *OpenGL* e *GLUT* para gráficos, *Quicktime* para vídeo, *OpenCV* para visão por computador, etc.

É suportado em todos os sistemas operativos principais quer para computadores quer para dispositivos móveis, com uma API também ela desenhada para ser mínima e fácil de

compreender, dispondo ainda de uma comunidade ativa, em crescimento, que possibilita o desenvolvimento constante das ferramentas.

O *kit* consiste num conjunto de *addons*, alguns integrados outros feitos pela comunidade, que permitem aos seus utilizadores criarem os seus próprios projetos, denominados aplicações, que podem ser extensíveis acrescentando tantos *addons* quanto sejam necessários, ou alterando o código fonte de alguma biblioteca incluída, recompilando-a, ou da ferramenta em si.

2.8 Resumo e Conclusões

Ao longo da análise realizada durante este capítulo procedeu-se à caraterização das várias tecnologias disponíveis e aplicáveis aos componentes a desenvolver, tentando salientar as suas vantagens e desvantagens comparativamente umas às outras.

Tornou-se evidente a larga quantidade de soluções existentes nomeadamente no caso de ambientes de síntese de áudio, não se traduzindo no entanto na descoberta de uma solução que seja perfeitamente adequada ao contexto deste projeto, com nenhuma das alternativas a evidenciar-se como absolutamente melhor do que todas as outras em todos os casos de utilização possíveis, sendo cada uma detentora de caraterísticas próprias que podem torná-la adequada para um determinado tipo de situações e não para outros.

De um ponto de vista mais técnico, a utilização de tecnologias que sigam *standards* adequados de engenharia de *software* é mais apelativa para o desenvolvimento de aplicações que possam singrar no futuro, apesar de por vezes serem tecnologias mais 'frescas' e em desenvolvimento e de não seguirem a norma dentro da indústria.

O risco de tais tecnologias prende-se com as limitações que estas possam revelar ao longo do desenvolvimento de um projeto, mas facilitam o processo de continuidade de desenvolvimento e exploração que se pretende integrar nesta aplicação, inserida numa investigação ainda a decorrer atualmente.

Mencionando agora tópicos para lá do âmbito técnico, podemos verificar que existem algumas abordagens para este tipo de projeto musicais, apesar de se diferenciarem do que é pretendido alcançar com o protótipo final aqui mencionado.

A investigação no campo da música está a tender para a procura de novas interfaces para a sua composição e, mais especificamente, a área de música em rede procura eliminar erros de latência e melhorar a qualidade de sinais de áudio para sistemas de colaboração em tempo real, descartando a possibilidade de serviços assíncronos de gravação via *Web* como o explorado nesta dissertação.

Capítulo 3

Gravação e Atuação Musical numa Rede Web de Instrumentos Acústicos

No sentido da realização dos objetivos desta dissertação, avançou-se para o seu projeto aqui concretizado numa apresentação detalhada do planeamento da solução para o problema, a criação de um serviço *web* autónomo de gravação e atuação de peças musicais em instrumentos acústicos em estúdio, e a estruturação dos seus componentes.

3.1 Arquitetura do Sistema

O sistema encontra-se primariamente dividido em dois grandes componentes, um servidor alojando um *website* responsável pela interação com o utilizador, e uma aplicação localmente instalada no estúdio, conetada aos instrumentos acústicos e aos microfones. Esta organização esquematiza-se na Figura 7.

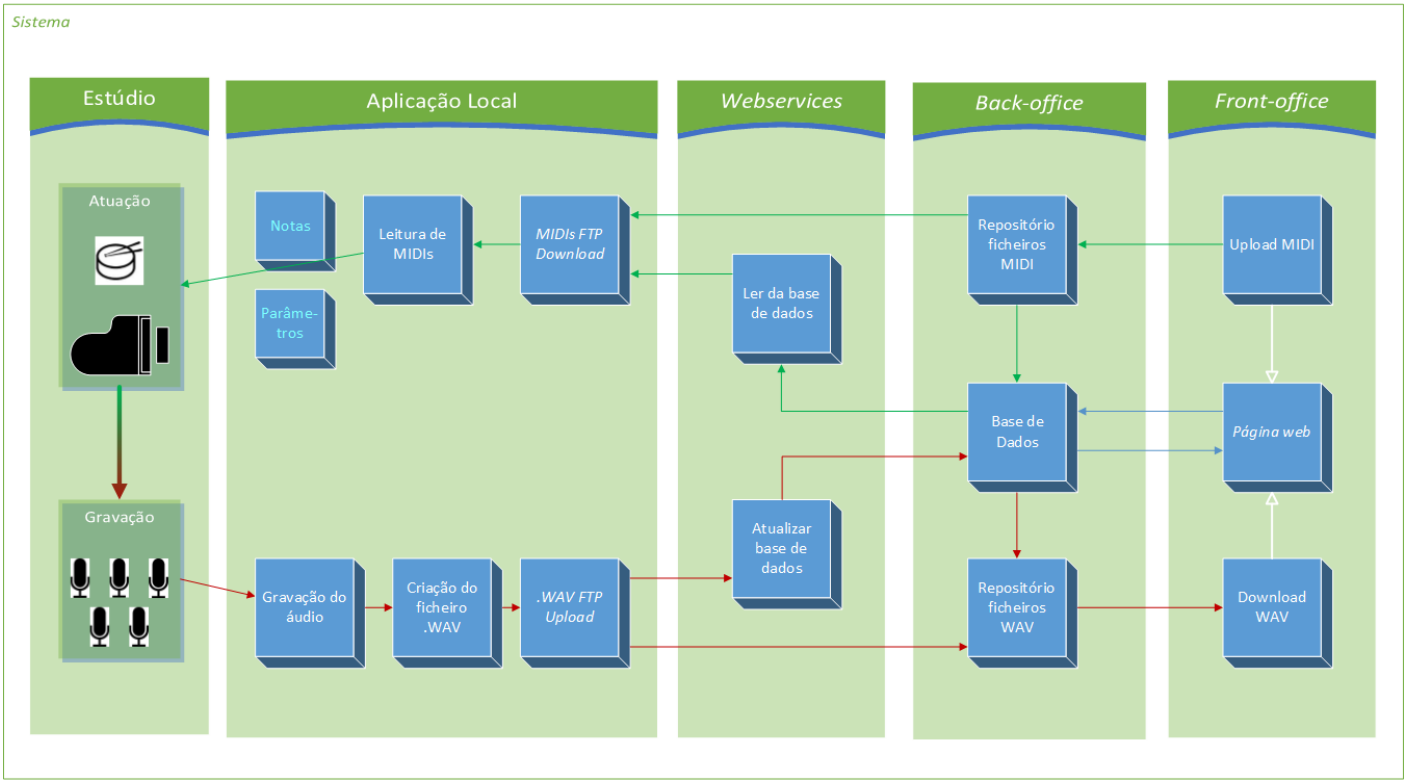


Figura 7 - Arquitetura do Sistema (Diagrama de Blocos)

Algo que não se encontra explicitado no diagrama anterior é a divisão da aplicação local em dois módulos com diferentes responsabilidades. Um deles trata de operar toda a interação com os instrumentos acústicos, sendo de referir as suas funções mais importantes: o *parsing* dos MIDIs e a gravação do ficheiro áudio. O segundo tem a seu cargo as operações em rede, utilizando os *webservices*, lendo e atualizando os registos da base de dados antes e após as gravações e de transferir os ficheiros conforme for necessário.

No esquema podemos não só ver uma representação do fluxo de dados desde o *upload* de um ficheiro *MIDI* por parte do utilizador até à gravação no sentido das setas verdes, mas também desde a captação do áudio até ao *download* do ficheiro *WAV* por parte do utilizador com as setas vermelhas. Também todas as interações entre os vários componentes deste projeto estão presentes, sendo as setas brancas meramente representativas de herança, indicando páginas *web* singulares onde é possível utilizar as funcionalidades de *upload* e de *download*, enquanto as setas azuis indicam as interações entre o *front-office* e a base de dados.

De referir que o ficheiro MIDI, criado pelo utilizador e contendo a melodia que este quer reproduzir e gravar em estúdio, terá de ser criada pelo mesmo através de um controlador MIDI ou por *software* de suporte composicional musical externo, como alguns dos mencionados no capítulo anterior. No final, obtém um ficheiro de áudio *WAV* com a melodia gravada nos instrumentos acústicos em estúdio.

3.2 Requisitos funcionais

Formalizando as funcionalidades e necessidades identificadas, chegou-se à lista de requisitos funcionais discriminada de seguida.

3.2.1 Aplicação Local

Rede

- Sempre que a aplicação for executada ou quando esta terminar de gravar o *set* de melodias atual, a aplicação deverá ler da base de dados quais as melodias por gravar, se existirem
- Confirmando a existência de melodias por gravar e após as identificar, a aplicação deverá fazer *download* dos ficheiros MIDI, do servidor para o sistema local
- A aplicação deverá fazer *upload* dos ficheiros com as melodias gravadas, do sistema local para o servidor
- Após terminar o *set* de gravações, a aplicação deverá atualizar a base de dados registando quais as melodias que foram gravadas
- Até ao final de uma execução, os dados locais deverão ser apagados após o término dos *uploads*

Instrumental

- Terminado o *download* do *set* de ficheiros para gravação, a aplicação deverá ler os dados dos ficheiros MIDIs, notas e parâmetros, antes de iniciar a reprodução
- Quando um ficheiro MIDI é aberto com sucesso, a aplicação deverá segmentar as notas fazendo-as corresponder ao instrumento a que se destinam, enviando-as pelo canal apropriado
- Ao iniciar a reprodução de um MIDI, a aplicação deverá criar um novo ficheiro de áudio com o mesmo nome que o ficheiro MIDI original, com um número de canais de áudio igual ao número de microfones atualmente ligados
- A aplicação deverá assegurar a reprodução sequencial e ritmada das notas nos instrumentos acústicos aquando da sua gravação
- Paralelamente à reprodução das notas, a aplicação deverá tratar os dados recebidos pelos microfones e armazená-los no ficheiro áudio previamente aberto

- No final da gravação, a aplicação deverá colocar os metadados e fechar o ficheiro de áudio

3.2.2 Website

- Este componente terá de permitir o registo de novos utilizadores no sistema
- Possuindo uma conta registada, o sistema permitirá que um utilizador possa usar os seus dados de entrada para efetuar o *login* no sistema, e consequente *logoff*
- O *website* deverá permitir o *upload* de novos MIDIs para utilizadores registados
- Um utilizador registado poderá substituir um MIDI existente se fizer *upload* de um ficheiro com o mesmo nome
- Caso ocorra algum problema no *upload*, o sistema deverá reportar o sucedido ao utilizador
- O sistema deverá permitir o *download* dos ficheiros MIDIs próprios, para utilizadores com *login* efetuado
- O sistema terá de permitir, após a correspondente gravação, o *download* de qualquer ficheiro de áudio, para utilizadores com *login* efetuado
- Deverá ser possível informar os utilizadores dos momentos em que o sistema se encontra *online*
- Os administradores deverão poder informar a comunidade de alterações no sistema ou dar-lhes outras informações relacionadas com o projeto através deste componente
- Em cada notícia deverá existir a possibilidade de cada utilizador registado comentar o conteúdo da mesma
- Deverá ser possível, para utilizadores registados, alterar a sua *password* e o seu *email*
- O sistema deverá assegurar a singularidade dos *usernames* de todos os utilizadores registados
- Cada utilizador registado deverá ter a opção de apagar o seu perfil da base de dados do sistema
- Acedendo à sua lista pessoal de melodias, um utilizador registado deverá ser capaz de requisitar uma nova gravação de um MIDI seu ou apagá-lo

- Um utilizador registado deverá poder adicionar um vídeo do *YouTube* à sua lista pessoal fornecendo o *link* do mesmo
- Acedendo à sua lista pessoal de vídeos, um utilizador registado deverá poder apagar cada um dos vídeos listados individualmente
- O sistema deverá permitir que qualquer utilizador possa navegar pelas listas de melodias e de vídeos
- Deverá ser possível a cada utilizador navegar por uma listagem dos utilizadores na comunidade, sendo que cada página pessoal deverá conter as melodias e vídeos correspondentes

3.3 Casos de Uso

Os utilizadores deste sistema estarão divididos em grupos sendo atribuído a cada um conjunto de permissões. O mais alto pertence ao administrador, chamado *Admin* daqui em diante, e o mais baixo a um utilizador autenticado que denominaremos *User*. O nível de permissões por omissão para novos utilizadores no *website*, e não autenticados, pertencerá a um utilizador denominado *Guest*.

3.3.1 Guest

Um *Guest* poderá navegar por todo o *website*, mas até que este se registe no sistema e inicie uma sessão autenticando-se no *website*, apenas poderá visualizar a informação e as listas expostas, sem nenhum nível de interação sobre o conteúdo *online*. Na Figura 8 mostram-se as funcionalidades a que um utilizador deste grupo tem acesso.

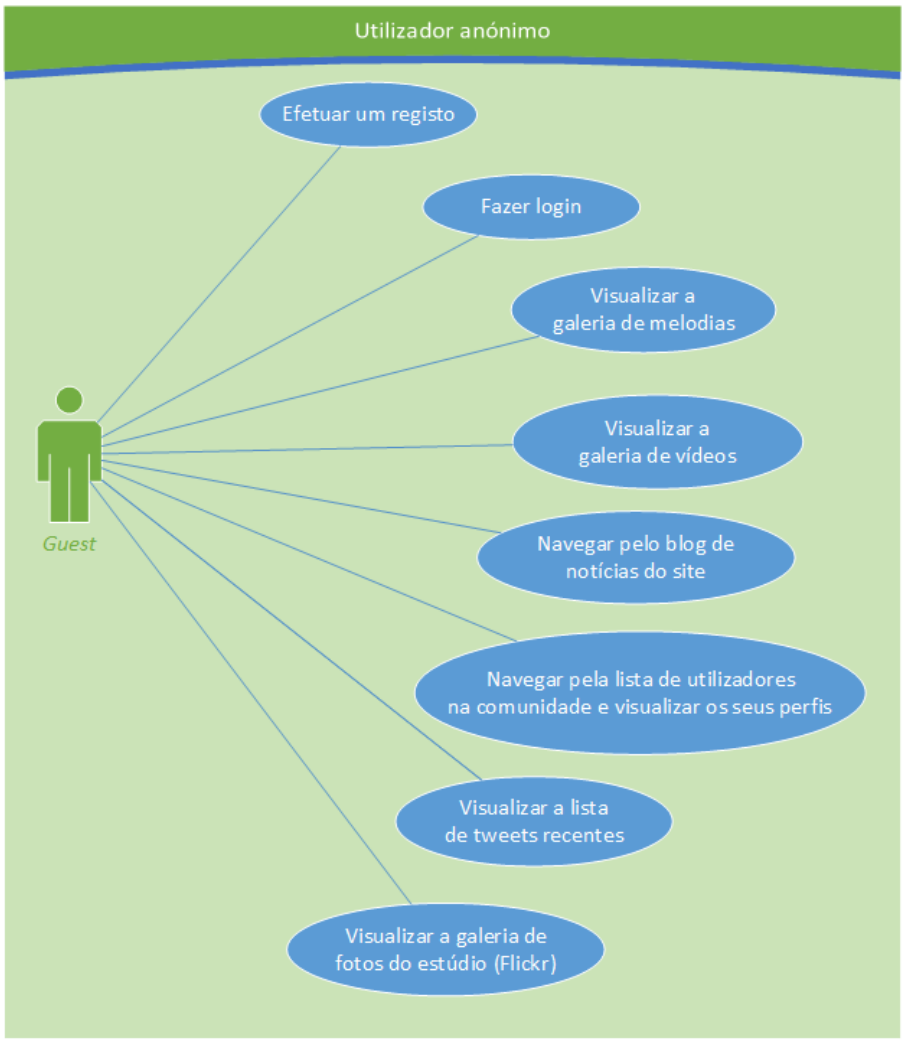


Figura 8 - Casos de Uso (*Guest*)

3.3.2 User

Por outro lado, um *User*, um utilizador registado e autenticado no *website*, poderá interagir com a comunidade bem como ter o privilégio de utilizar as funcionalidades de *upload* e *download* disponíveis, para além de poder gerir o seu próprio perfil. O detalhe das suas permissões e acesso às funcionalidades mostra-se na Figura 9.

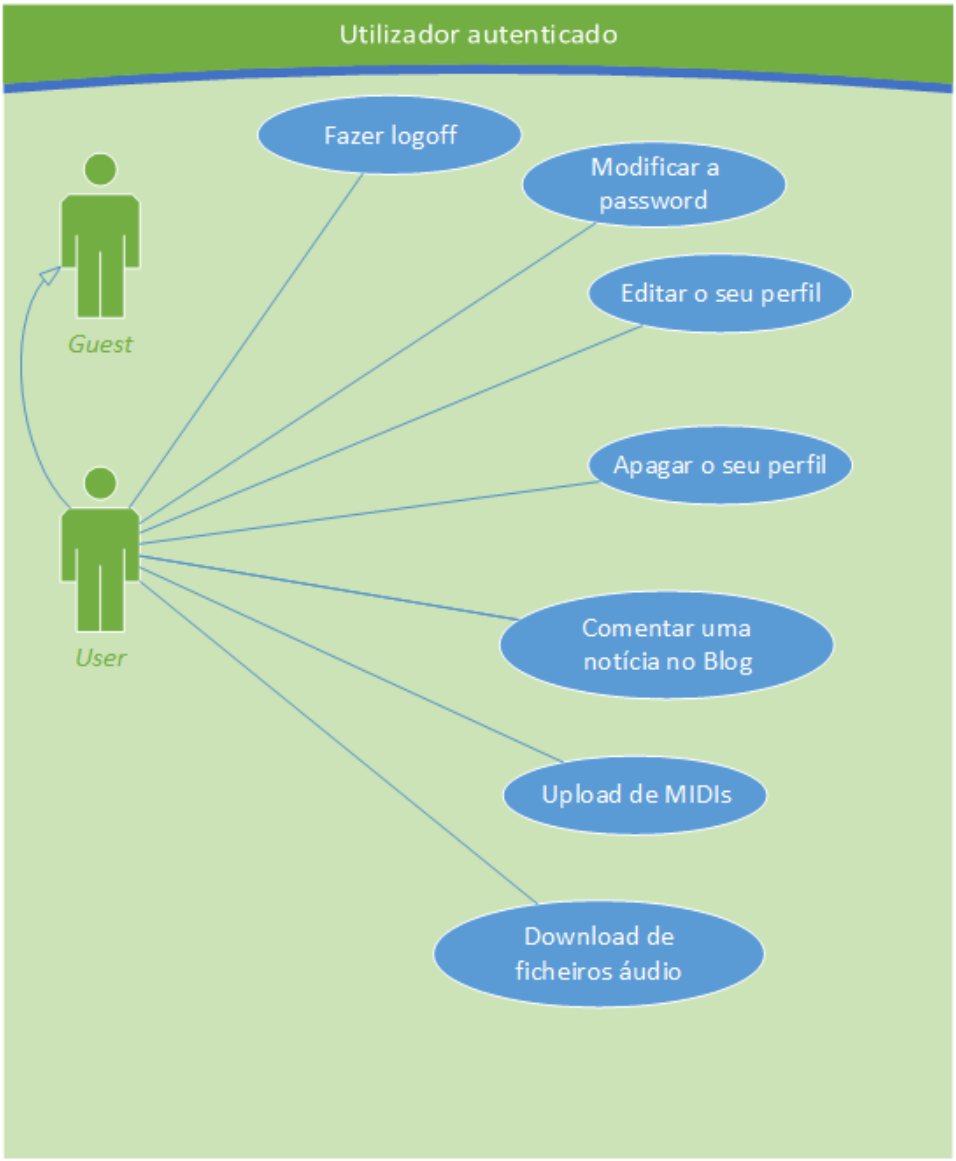


Figura 9 - Casos de Uso (*User*)

3.3.3 Admin

Finalmente um *Admin* terá todo o controlo sobre os conteúdos do *website*, estando encarregado da gestão do mesmo, resumidamente ilustrado na Figura 10.

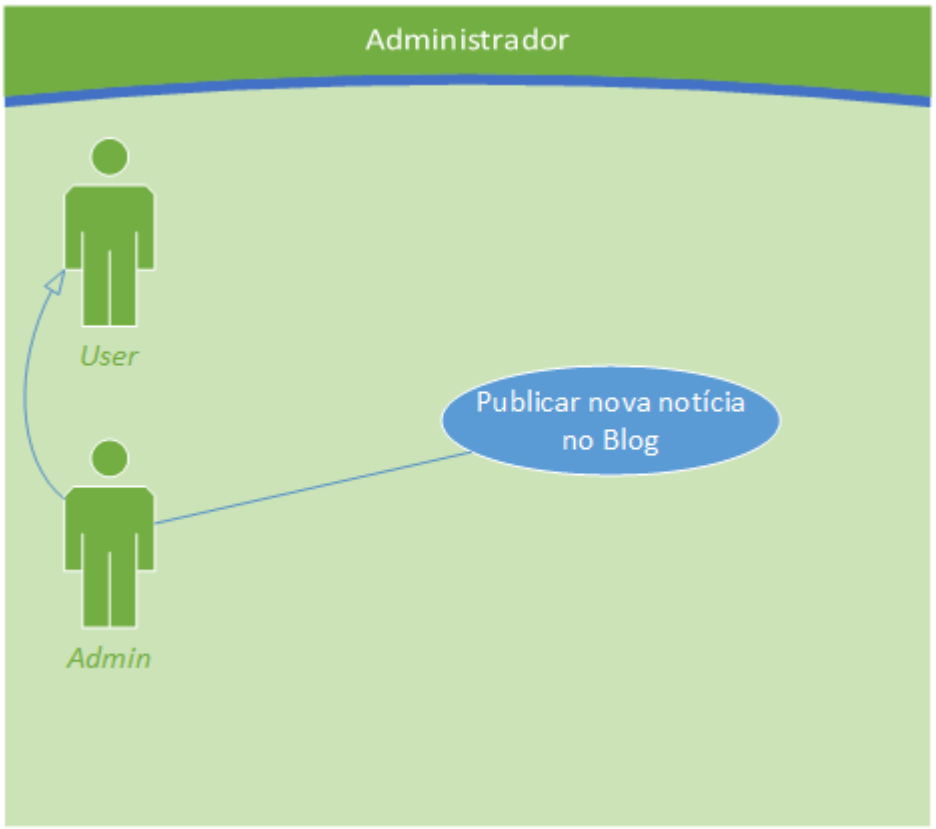


Figura 10 - Casos de Uso (*Admin*)

A administração deste sistema terá também a responsabilidade de preparar o estúdio de gravação da aplicação e de a disponibilizar: as instalações e os instrumentos onde serão feitas as gravações não estarão sempre disponíveis, nem os microfones estarão montados a tempo inteiro nem o computador com a aplicação estará constantemente ligado. Para este efeito foi incluída uma fila de espera para os ficheiros MIDI por gravar, mas cabe aos administradores montarem todo o *setup* de gravação e de executarem a aplicação manualmente quando tiverem disponibilidade para tal.

3.4 Sequência de Ações do Sistema

Tendo em conta Figura 7, pretende-se dar uma breve explicação das mecânicas do projeto, das interações do utilizador e do fluxo de dados ao longo dos componentes do sistema.

Um utilizador começa por criar um ficheiro MIDI com um *software* à sua escolha, ou através de um controlador MIDI como o da Figura 11. Este tipo de dispositivos não produz som, tendo apenas o propósito de criarem uma gravação digital da melodia pretendida.

Figura 11 - Controlador MIDI

Posteriormente este terá de se registar no *website* e fazer *login* no mesmo para poder utilizar as funcionalidades de *upload* e enviar o seu ficheiro MIDI para o repositório *online* e criar um novo registo na base de dados, indicando também que ainda não foi gravado. Esta sequência é ilustrada na Figura 12.

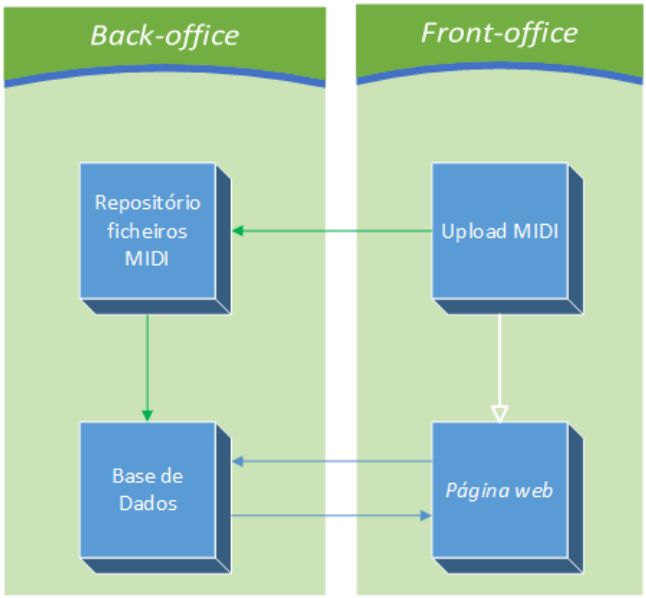


Figura 12 - *Upload* de MIDIs

A próxima fase corresponde a um período de espera para o utilizador, estando dependente da disponibilidade do estúdio de gravação remoto, dos instrumentos e da montagem de todo o *setup* - microfones, ligações MIDI, placa de som e computador - que terá de ser feita pelos administradores do sistema. Opcionalmente, poderão anunciar na sua conta de *Twitter* que o sistema se encontra montado, informando a comunidade. Estes terão também de executar manualmente a aplicação quando todas as condições anteriores se verificarem.

O programa começará por analisar os registos da base de dados através de um *webservice* e verificar quais os MIDIs que ainda não dispõem de gravações acústicas ou que foram marcados para serem regravados. Depois de obter os nomes dos ficheiros a aplicação acede ao repositório de MIDIs no servidor e transfere-os para uma pasta local para serem posteriormente processados. Este conjunto de ações é apresentado na Figura 13.

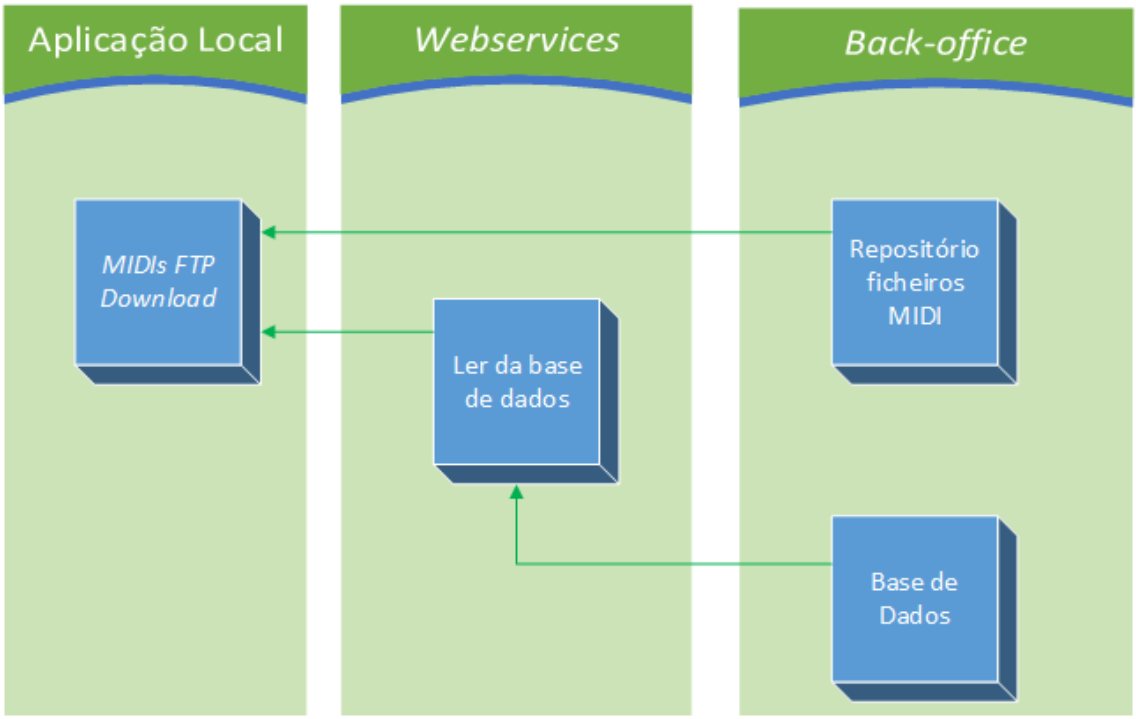


Figura 13 - *App FTP Download* de MIDIs

Com todos os MIDIs armazenados localmente, a aplicação começará por processá-los um a um, enviando os dados necessários para os instrumentos acústicos robotizados e reproduzindo a melodia criada pelo utilizador, enquanto paralelamente guarda os dados analógicos dos microfones num ficheiro áudio *WAV*, como ilustrado pela Figura 14.

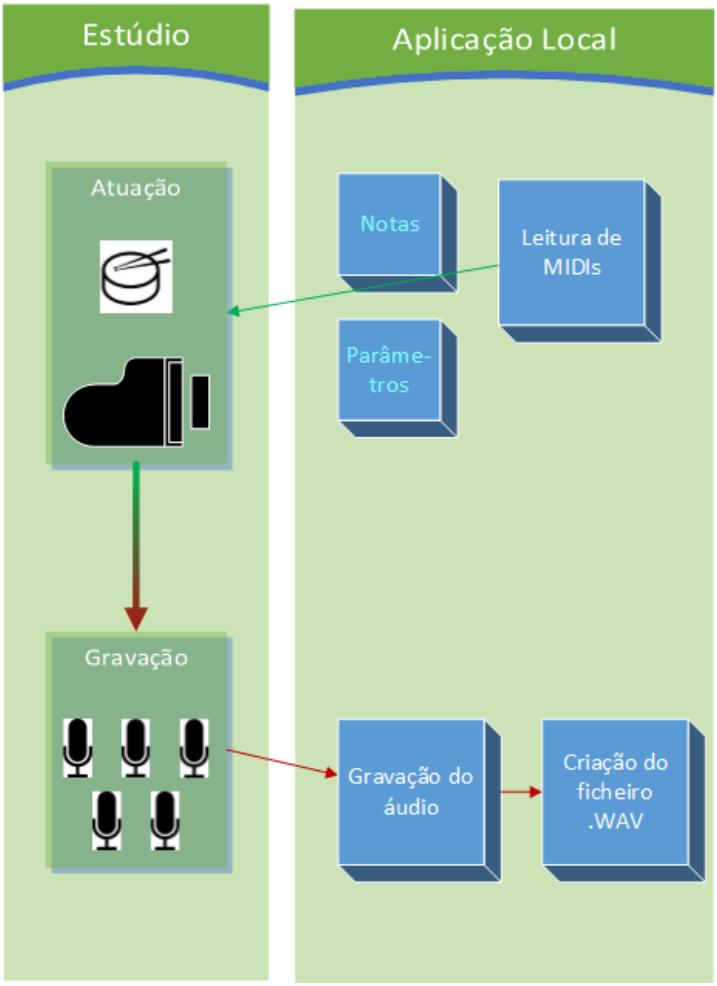


Figura 14 - Reprodução e Gravação de MIDIs

Como representado na Figura 15, assim que todas as gravações estiverem terminadas, os ficheiros de áudio são transferidos para o repositório *online*, atualizando a base de dados a cada *upload* concluído através do *webservice* criado para tal.

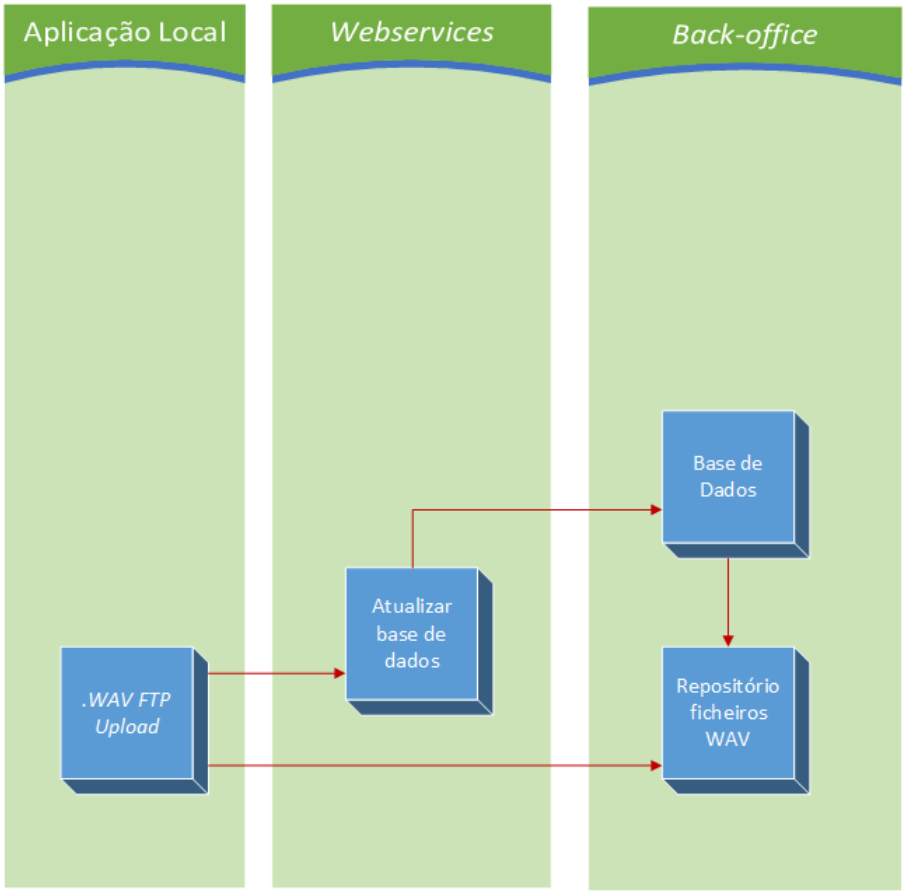


Figura 15 – *App FTP Upload* de ficheiros de áudio

Entretanto, com os ficheiros de áudio no repositório do servidor e os registos da base de dados atualizados, o *website* passa a ter disponível o *link* para *download* das gravações executadas. Neste ponto, estas gravações estarão disponíveis para qualquer utilizador com sessão iniciada no sistema, como é ilustrado na Figura 16.

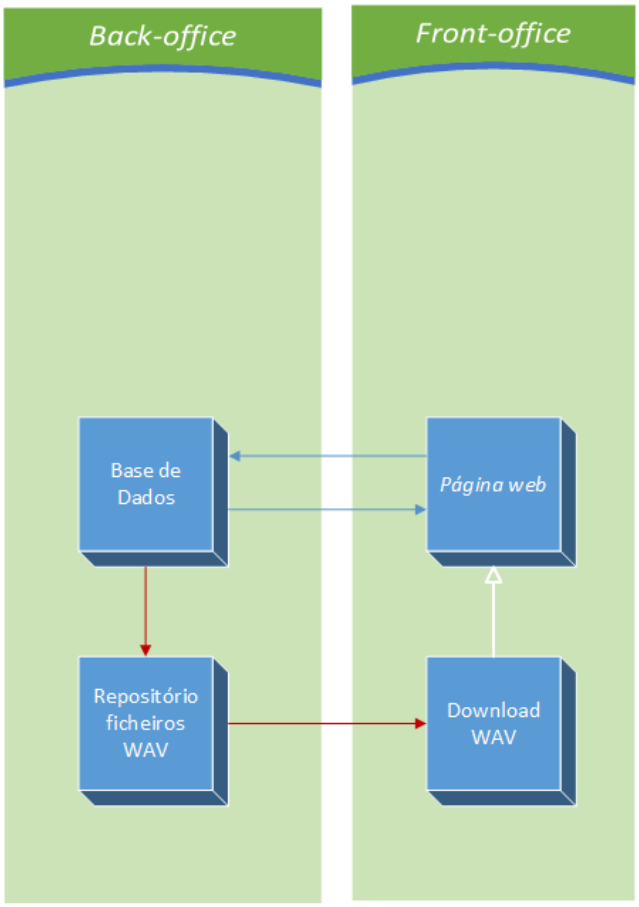


Figura 16 - *Download* de ficheiros de áudio

A aplicação local apaga todos os dados locais e utiliza um *webservice* para voltar a verificar se existem novos ficheiros MIDI para serem reproduzidos e gravados. Caso existam, todo o processo repete-se automaticamente, mas se não existiram novas adições ou alterações é apresentada uma mensagem de finalização no ecrã e o programa termina. Neste momento, os administradores poderão voltar a desmontar todo o sistema.

3.5 Resumo e Conclusões

Como podemos deduzir pelas particularidades de cada módulo do sistema a partir dos requisitos apresentados, o planeamento e a implementação deste projeto tornam-se complexos como consequência do conjunto de tarefas singulares que um processo de execução tem de completar.

A partir dos casos de uso inferimos também que os utilizadores do sistema terão disponíveis interações simplistas, retendo a complexidade do sistema e expondo apenas a acessibilidade de uma interface limpa e compreensível.

Capítulo 4

Realização do Projeto

Após a especificação e planeamento do projeto, satisfazendo os objetivos iniciais, passou-se à sua realização, analisando aqui de forma mais detalhada cada componente desenvolvido e quais as particularidades de cada um destes módulos.

4.1 Metodologias e Tecnologias Utilizadas

O método de desenvolvimento de *software* utilizado neste projeto foi o de um processo iterativo e incremental algo modificado, planeando, desenvolvendo, testando e avaliando cada um dos componentes antes de se prosseguir para o seguinte. Iniciou-se o desenvolvimento após a fase inicial de planeamento de todo o projeto em que se definiram as funcionalidades principais pretendidas e os casos de uso a ter em conta.

O planeamento inicial do desenvolvimento resultou na separação fulcral de componentes após uma análise cuidada das funcionalidades pretendidas. De seguida, o processo de desenvolvimento baseou-se na construção dos módulos mencionados, divididos da seguinte forma:

- Aplicação de reprodução e gravação de MIDIs desenvolvida em *openFrameworks*, erguida na linguagem C++
- Aplicação responsável por operações em rede como transferências FTP e execução de *webservices* projetada em C++ utilizando a biblioteca *libcurl*
- Aplicação em rede desenvolvida em *PHP*, juntamente com *HTML5, CSS3, JavaScript, jQuery*.

- Servidor em rede utilizando o pacote de *software LAMP*, com *phpmyadmin* para acesso à base de dados do *back-office*.

A escolha da utilização do *openFrameworks*, e consequentemente da linguagem C++ e suas bibliotecas externas, relacionou-se com a propensão desta ferramenta para tornar todo o processo autómato e desprovido da necessidade de interações de manuais com o utilizador ao contrário doutros *software* como Max/MSP, que pode ser analisado com mais detalhe nos próximos tópicos.

O pacote *LAMP* associado à linguagem *PHP* é um dos mais utilizados, praticamente um *standard* de desenvolvimento, o que facilitará o continuado desenvolvimento da ferramenta e sua eventual expansão de funcionalidades remotas.

4.2 Reprodução e Gravação de MIDIs (App local)

O primeiro foco do desenvolvimento deste projeto passou pela aplicação que estaria responsável por reproduzir os MIDIs nos instrumentos acústicos, resultando num executável separado daquele responsável pelas operações em rede do sistema.

4.2.1 Análise do *patch* em Max/MSP

Esta fase consistiu numa análise do sistema existente e das possibilidades das tecnologias em que este estava desenvolvido. Um projeto de uma tese de doutoramento anterior resultou na especificação de um *patch* desenvolvido em Max/MSP, que tratava do processamento do áudio e apenas funcionava para o piano *Disklavier*, e de um *shell script* responsável pelas funcionalidades de rede necessárias.

A execução do processo envolvia a presença constante de um indivíduo no computador, de modo a correr os *scripts* e executar os passos manuais necessários no *patch* Max, no caso a escolha da pasta com os MIDIs a reproduzir, a seleção do ficheiro e o simples ato de carregar em *play*.

Considerando a possibilidade de automatizar o processo com a execução dos *patches* criados em Max através de um *shell script*, graças à sua funcionalidade de invocação por linha de comandos, procedeu-se a uma análise mais aprofundada das tecnologias Max, Pd e SuperCollider através de tutoriais e da leitura da documentação.

Verificou-se a existência de uma API nativa disponível para a criação de novas funcionalidades, novos objetos de Max, novas rotinas de execução para expandir as opções do que poderia ser executado num único *patch*. No entanto, a invocação das funcionalidades criadas, quer das nativas quer daquelas que poderiam vir a ser desenvolvidas, não tinham forma de ser controladas por linha de comandos com novos parâmetros. Este modo de execução do

programa estava limitado às opções disponibilizadas pela API existente, sem possibilidade de expansão ou desenvolvimento.

Findada a procura de alternativas que permitissem o uso destas tecnologias para realizar o projeto, concluiu-se que seria impossível usar uma tecnologia como Max ou Pd para automatizar todo o processo de *download*, reprodução, gravação e *upload* das melodias.

4.2.2 Módulo *openFrameworks*

Tendo em conta as conclusões previamente obtidas, e aquando da procura ativa de alternativas à tecnologia Max, foi feita uma análise da tecnologia *openFrameworks*. Dado o seu caráter mais generalizado, uma *framework* numa linguagem abrangente como o C++, a multiplicidade de componentes e a diversidade de projetos desenvolvidos nesta ferramenta, esta parecia a solução ideal.

Não obstante, a multiplicidade e diversidade da solução eram também caraterísticas dos problemas que trazia consigo. Para além dos múltiplos problemas "comuns" de compilação e execução encontrados com o último *release* da *framework* quando executada nas duas últimas versões do *Visual Studio* (2010 e 2012), verificou-se também que esta era incompatível com outros IDEs comuns como *Eclipse*.

Apesar de se promover como uma ferramenta generalizada, a faceta pública do *openFrameworks* revela uma comunidade muito, se não mesmo exclusivamente, orientada para o universo *Apple* e extremamente propensa para o seu sistema operativo, sendo rara a ocasião em que é possível encontrar suporte e exemplos generalizados.

Um outro empecilho no uso desta *framework* durante a fase inicial de desenvolvimento foi a inutilidade da sua ferramenta de criação de novos projetos, também limitada noutros sistemas operativos que não o da *Apple* mas igualmente improfícuo neste. O processo de cópia e adaptação de exemplos funcionais reduzidos aos comandos básicos e indispensáveis, que neste ponto eram inexistentes, manifestava-se como a única forma de dar início a uma nova solução.

Apesar dos obstáculos mencionados, a utilização do *openFrameworks* continuava a mostrar-se como a melhor solução para desenvolver todas as funcionalidades necessárias. A partir de um exemplo reduzido aos seus comandos fundamentais, procedeu-se à correção de erros, nomeadamente de compilação.

Para além das funcionalidades inerentes ao *openFrameworks* este é distribuído com um leque de bibliotecas gratuitas. Estas são indispensáveis caso um desenvolvedor queira desfrutar de todo o potencial da ferramenta. Por conseguinte, algumas bibliotecas como a *POCO* e a *rtAudio* tiveram de ser recompiladas e incluídas posteriormente, tendo ainda sido necessário modificar campos no *Linker* e nas definições de pré-processamento das propriedades do projeto.

Finalmente, também foi imprescindível alterar *Includes*, *Namespaces* e algumas linhas de código essenciais, de modo a ser possível compilar com sucesso a ferramenta.

4.2.2.1 *Addons* utilizados

A inclusão de *addons* poderia ser feita através da ferramenta de criação de novos projetos ou através da utilização de um *setup* exemplar quando disponível, surgindo no entanto dificuldades semelhantes às referidas anteriormente, independentemente de qual o *addon* utilizado.

Todavia, a solução passou por incluir estes novos módulos através de *filters* criados manualmente e da alteração de vários campos nas propriedades do projeto de modo a ser possível utilizar as suas funcionalidades no desenvolvimento da aplicação.

ofxLibsndfileRecorder

O *addon ofxLibsndfileRecorder* permite gravar amostras de áudio em vários formatos diferentes e com variadas taxas de amostragem utilizando a biblioteca *libsndfile*.

libsndfile

Esta é uma biblioteca escrita em C para ler e escrever ficheiros com amostras ou dados de áudio. Foi projetada de modo a conseguir lidar com dados em *little-endian* e em *big-endian*, como os ficheiros *Wav* ou *Aiff* respetivamente, e para ser compilada e correr corretamente em processadores *little-endian* e *big-endian*, como *Intel* ou *MIPS* respetivamente.

ofxMidi

O *addon ofxMidi* fornece capacidades de *input* e *output* de MIDIs a uma aplicação em *openFrameworks*, e foi também incluído.

RtMidi

Conjunto de classes em C++ que fornece uma API em comum para *input (RtMidiIn)* e *output (RtMidiOut)* de MIDI em tempo real transversalmente em todos os sistemas operativos disponíveis. Cada classe, *RtMidiIn* e *RtMidiOut*, suporta apenas uma única ligação MIDI. O *RtMidi* não disponibiliza funcionalidades de tempos, sendo as mensagens de *output* enviadas imediatamente, enquanto as de *input* têm um *timestamp* com a resolução do segundo. Os dados MIDI são passados como *bytes*, utilizando um vetor de carateres *unsigned*.

ofxThreadedMidiPlayer

O *ofxThreadedMidiPlayer* é um *addon* para reproduzir ficheiros MIDI a partir do disco, imprimindo os comandos para a consola e enviando os dados por uma única porta MIDI.

jdksmidi

Biblioteca de funções utilitárias para trabalhar com MIDIs, originalmente escrita em C mas evoluída entretanto para uma biblioteca em C++ com funcionalidades desde o *parsing*, a escrita e leitura de todos os tipos de ficheiros MIDI, e processamento de cadeias de mensagens até ao cálculo de tempos, cálculo de códigos temporais SMPTE, e sequenciação e edição de eventos.

ofxRuiThread

O addon ofxRuiThread oferece uma extensão do *ofxThread* com mais métodos de controlo e *query* de *threads*.

4.2.2.2 Alterações importantes

Neste tópico descrevem-se as alterações mais importantes efetuadas ao *core* das ferramentas utilizadas, quer no *openFrameworks* em si quer nos *addons* encontrados, sem os quais não teria sido possível obter uma execução como a pretendida.

Inicialmente, foram criados vários projetos base capazes de executar cada tarefa requerida individualmente, facilitando o processo de identificação de erros que se revelassem necessários corrigir. Exisita então um projeto por cada tarefa, não necessariamente por cada *addon*, separando assim leitura de MIDIs, *parsing* de MIDIs, o envio de mensagens MIDIs, a reprodução da melodia, a gravação do som e a criação e escrita do ficheiro de áudio.

A listagem e leitura dos ficheiros começou por ser feita recorrendo a funções auxiliares que utilizam *wide characters*, *wchar_t*, e uma estrutura *WIN32_FIND_DATA*, que entre outros atributos guarda o tamanho do ficheiro e o seu *filename*. Depois de concluídas estas funções, unir esta tarefa com o *parsing* das mensagens MIDI revelou-se um processo fácil.

Contudo, este não foi o caso com o envio das mensagens. O *addon* responsável pela reprodução dos MIDIs nos instrumentos apenas operava com uma *thread* em cada instante, bloqueando o sistema para execução exclusiva das operações dessa *thread*, sem mais nenhum processo em paralelo. Adicionalmente, o envio de dados dessa *thread* apenas se destinava a um dos instrumentos, e a porta MIDI utilizada necessitava de ser aberta e fechada a cada novo MIDI carregado.

De modo a corrigir os problemas identificados alterou-se o *addon* de forma a criar *threads* que não executassem com bloqueios exclusivos, alterando também a forma como estas eram atualizadas. Uma *app* em *openFrameworks* está intrinsecamente ligada à biblioteca *glut*, um kit de utilidades para *OpenGL*, recorrendo ao *glutMainLoop* para correr as funções de gestão de eventos e de atualização do programa. Uma primeira solução passou por substituir o *glut* por inteiro do *core* do programa por uma biblioteca menos datada e que não bloqueasse totalmente a aplicação e tomasse o controlo desta, como era o caso da *GLFW* ou da *freeglut*. Porém, como esta mudança iria alterar a forma como a aplicação em *openFrameworks* seria executada, dificultando a compreensão do novo código no processo, optou-se então por modificar a forma de atualização das *threads* com um ciclo próprio enredado no *glutMainLoop*.

A criação deste novo ciclo permitiu também resolver um novo problema, desta feita relacionado com a gravação das melodias aquando da sua reprodução, sem inclusão de atrasos ou 'tempos mortos' na gravação. Este problema ocorria mesmo quando a *thread* de reprodução de MIDIs não tinha permissões exclusivas pra bloquear os outros processos, existindo um *delay* notório entre a gravação de cada *set* de notas musicais. Incluindo o processo de gravação embutido no novo ciclo criado, em oposição a depender somente e unicamente do *glutMainLoop*, não ocorriam atrasos na transferência de dados entre os *buffers* de gravação e o ficheiro de áudio, melhorando significativamente a eficácia da aplicação e mostrando também algumas melhorias na sua eficiência de execução. Este ciclo facilitou também o controlo da reprodução sequencial de MIDIs.

O *addon* responsável pela gravação dos MIDIs apresentava também algumas adversidades, desta feita relacionadas com o tamanho e o tempo dos ficheiros de áudio, sendo este último o dobro dos MIDIs originais. Para MIDIs de maior duração, ocorria também um erro de acesso de memória de forma esporádica, associado a um fecho precoce do ficheiro de áudio para o qual os dados eram transferidos. O tempo excessivo das gravações levou a que fossem feitas correções nos tempos do *clock* interno do *addon* de reprodução de MIDIs, alterando também o manuseamento de eventos e mensagens MIDI na sua reprodução. Paralelamente, foi otimizada a manipulação de eventos da *app*, realizando algumas alterações no número e ordem de chamadas do *glutMainLoop*.

Por fim, no *addon* de reprodução de MIDIs, foi preciso complementar o envio de dados e das mensagens através da criação de uma nova porta MIDI com um novo objeto *RtMidiOut*, para o segundo instrumento, e de refinar a análise e o *parsing* das notas provenientes do MIDI, mais precisamente nas *MIDITimedBigMessage*. Os primeiros testes foram executados com portas virtuais, com ajuda do programa *loopMIDI*, tendo posteriormente sido realizados novos testes bem-sucedidos e novas verificações no estúdio com os instrumentos acústicos fornecidos.

4.3 Servidor / *back-office*

Findado o desenvolvimento da aplicação, as atenções foram voltadas para como iria ser estruturado o servidor da aplicação, quais seriam os *webservices* necessários para existir um fluxo de informação suficiente entre os componentes do sistema e quais seriam os dados que iriam requerer entrada numa base de dados.

4.3.1 Base de dados

Partindo do ponto fulcral do sistema, os utilizadores e as suas melodias, foi realizado um breve esquema de *brainstorming*, disponível no Anexo A – Diagrama de *Brainstorming*, de modo a ser possível perceber que funcionalidades poderiam ser acrescentadas ao projeto de modo a não só acrescentar-lhe mais valor do ponto de vista do utilizador, mas também para facilitar a gestão do *website* por parte dos administradores. Ao analisar esse esquema, idealizou-se uma base de dados como a mostrada no seguinte diagrama.

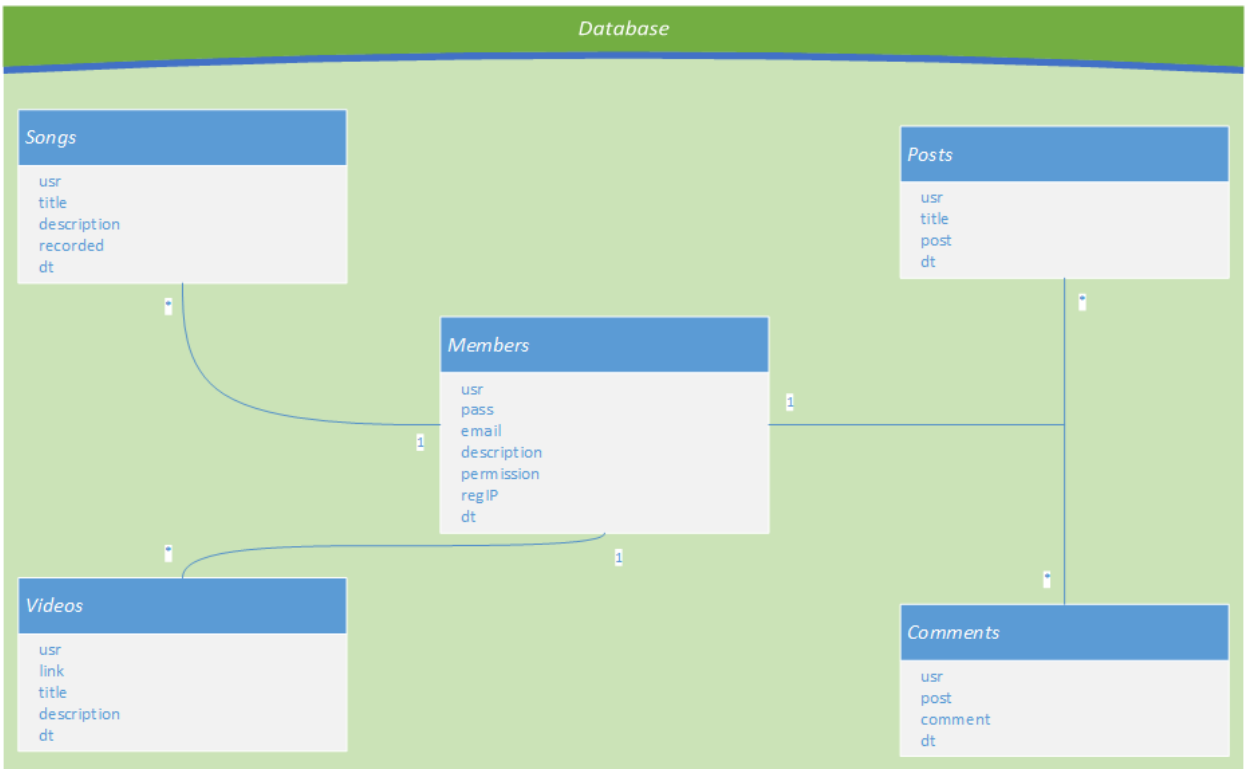


Figura 17 - Base de Dados

Para além de guardarmos os dados mais importantes de cada utilizador, bem como o seu nível de permissão no sistema para diferenciar os administradores, iríamos também armazenar os conteúdos multimédia que estes poderiam oferecer à comunidade do *website*, nomeadamente as suas melodias originais e vídeos onde estas mesmas fossem destacadas.

Visualizando ainda um *blog* intrínseco no *website* como plataforma de anúncios por parte da administração, tornou-se lógico a adição de tabelas para guardar os *posts* e os comentários relacionados com cada um deles.

Detalhando um pouco mais alguns dos campos presentes no diagrama, nomeadamente na tabela *Songs*, podemos ver que o campo *recorded* se destinará a controlar se a gravação do MIDI foi executada ou não, servindo também como *trigger* para um certo ficheiro ser descarregado para ser gravado ou não. Na tabela *Members*, temos ainda o campo *permission* que, como dito anteriormente, servirá apenas para destacar os *admins* do sistema dos seus utilizadores normais, abrindo-lhes o leque de funcionalidades disponíveis no *website*. Finalmente, o campo *dt*, diminutivo para *datetime*, tem como função controlar a sincronização dos dados ao serem inseridos, editados ou apagados da base de dados, estando por isso presente em todas as tabelas.

4.3.2 *Webservices*

A ligação entre o servidor e a aplicação local onde iriam ser gravadas as melodias teria de estar intrinsecamente embutida na operação, eliminando qualquer interação manual por parte de um administrador. Para isso, foram criados um par de *webservices* para facilitar a execução das funções necessárias aquando de um ou de vários conjuntos de gravações de MIDIs.

O primeiro, *get_unrecorded_songs*, permite ao programa obter facilmente a lista de ficheiros que se encontram preparados para gravação, verificando quais das entradas na tabela *Songs* da base de dados têm o campo *recorded* como *false*, e devolvendo a listagem dos nomes dos ficheiros para serem posteriormente obtidos por FTP.

Já o *webservice update_recordings*, garante a sincronização dos dados na base de dados após as gravações das melodias e o envio dos ficheiros de áudio para o servidor, atualizando os campos necessários para disponibilizar esses ficheiros no *website*.

Os principais componentes dos módulos *online* do projeto - o *back-office* e os *webservices* bem como o *front-office* a ser descrito no próximo tópico – são apresentados esquematicamente na Figura 18.

Figura 18 - Módulos *Online*

4.4 *Website / front-office*

Consecutivamente, iniciou-se o desenvolvimento do *front-office*, dando especial relevância aos módulos de *login*, *upload* e de *download* que viriam a ser o núcleo da interação do sistema com os utilizadores finais. Houve também algum cuidado em tentar desenhar uma interface limpa e de fácil interação para qualquer utilizador.

Aproveitando algumas funcionalidades das últimas tecnologias de desenvolvimento *web*, o sistema de registo e *login* foi projetado de modo a estar presente a partir de todas as páginas do projeto, e assim a implementação de um *slider* no topo permitiu dar-lhe a versatilidade necessária. Da mesma forma, a página de *upload* de MIDIs deveria estar presente e acessível a

partir de qualquer ponto, tendo sido implementada uma ligação nesse mesmo *slider* estando visível apenas para utilizadores registados e autenticados.

Outras implementações que serviram para complementar a usabilidade do *website* e facilitar as trocas de comunicação entre a administração e a comunidade foram a inclusão de um *feed* de *tweets* na barra de navegação lateral e a de um *blog* cujas mensagens são passíveis de ser comentadas. Outro complemento acrescentado foi uma *stream* de imagens de uma conta *Flickr* onde os administradores poderiam colocar imagens dos locais onde as gravações são feitas, dando a possibilidade dos utilizadores conhecerem o estúdio integralmente.

4.5 Comunicação com *back-office* (App local)

O último componente desenvolvido tratou de interligar os módulos *web* com a aplicação local, utilizando para isso os *webservices* desenvolvidos anteriormente e mantendo o *website* como o único ponto de comunicação com o *back-office* de modo a evitar problemas de sincronização ou de conflito de dados.

Numa primeira instância, considerou-se integrar as comunicações na aplicação *openFrameworks*, recorrendo para isso a *addons* adicionais que executassem as transferências de ficheiros por FTP requeridas e que alterassem os dados necessários na base de dados. Tais soluções poderiam ser encontradas com *ofxFTP* e *ofxMySQL*.

Contudo, como acrescentar as operações de rede no ciclo de execução de uma aplicação *openFrameworks* iria acrescentar uma complexidade desnecessária ao código e ao programa, isto sem mencionar todos os problemas que tal integração poderia vir a ter, optou-se por tentar desenvolver uma ligação de *sockets* entre o cliente, a aplicação local, e o servidor, adicionando uma nova camada entre a aplicação e o servidor.

Por sua vez, a ligação por *sockets* acabou por ser igualmente descartada pois iria requerer uma execução sempre ativa de cliente e servidor nas máquinas do sistema, o que se desviava do cenário pretendido não só porque não iria existir uma máquina a operar a tempo inteiro no estúdio, mas também porque haveria pouco controlo sobre o programa a ser executado no lado do servidor do serviço.

Optou-se então sobre uma solução que iria resultar num novo executável, desta feita um programa desenvolvido com bibliotecas que pudessem interagir com o servidor FTP e realizar as transferências necessárias. Também neste momento se tinha pensado em utilizar bibliotecas para lidar diretamente com a base de dados tal como *MySQL++* ou outras nativas, mas como explicado anteriormente, foram criados *webservices* para esse efeito, de modo a também eliminar qualquer possibilidade de falhas de sincronização ou consistência dos dados. Assim, a aplicação passou a ser focada nas transferências FTP e na interação com os *webservices*.

Preliminarmente, testaram-se algumas bibliotecas para verificar a sua eficiência e facilidade de utilização, mas rapidamente se verificou que todas as soluções, *libcurl*, *curl*++, *ftplib*++, apresentavam problemas de integração num novo projeto. Recorrendo então à *libcurl*, amplamente considerada como a solução mais generalizada, iniciou-se um processo de análise e de tentativa de correção dos erros que surgiram, obtendo uma solução apenas com a compilação própria da biblioteca com várias mudanças nas propriedades do projeto e no pré-processamento deste.

Com a inclusão bem-sucedida da *libcurl* num projeto, foi possível desenvolver todas as funcionalidades requeridas para que o programa pudesse operar autonomamente aquando da sua execução, que apenas iria ocorrer assim que o sistema de gravação estivesse montado, o que não será a tempo inteiro como já foi referido. De notar que também se inclui a exclusão dos dados locais depois de efetuadas todas as operações para não sobrelotar a capacidade do disco na máquina a ser utilizada, e que a chamada para a aplicação de reprodução e gravação dos MIDIs também é feita de forma automática. Na Figura 19 podemos ver um esquema elucidativo dos componentes principais da aplicação local desenvolvida.

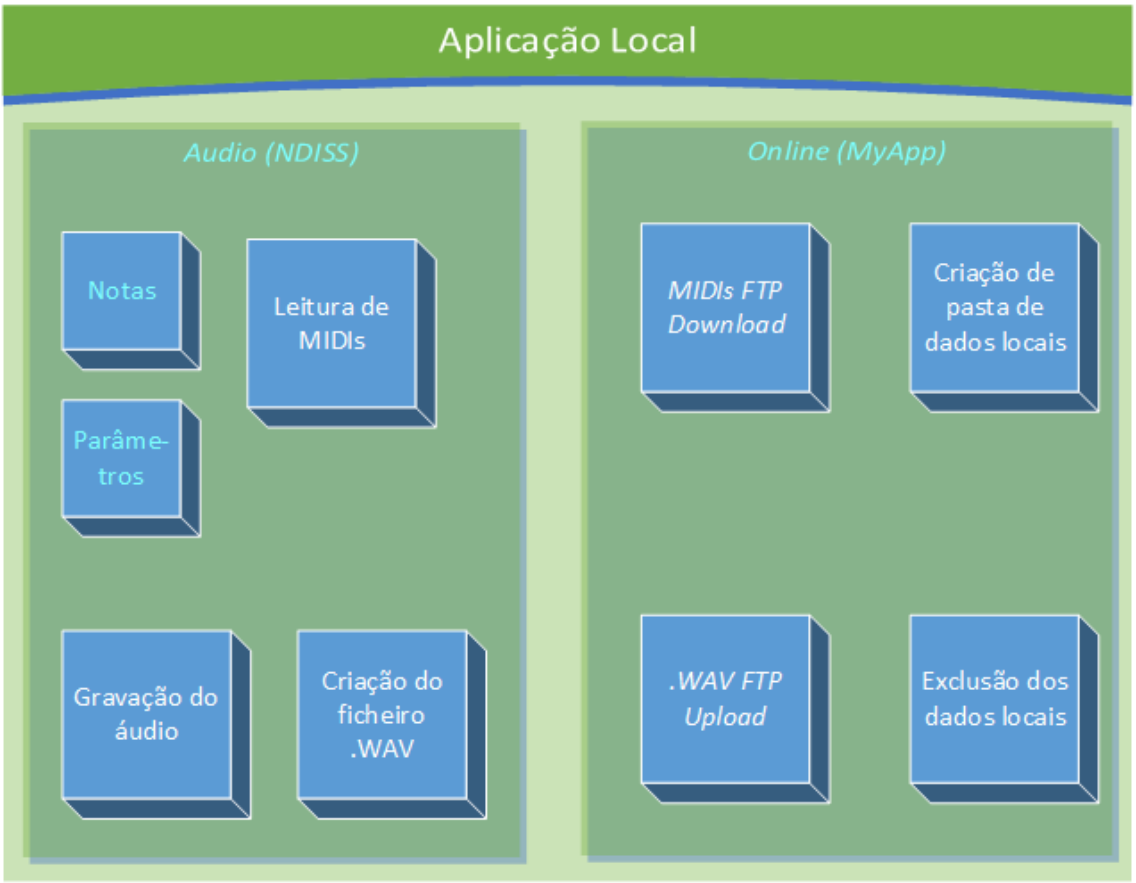


Figura 19 - *App* Local

4.6 Concerto

Dada a separação da aplicação local em dois executáveis distintos, sendo um deles apenas dedicado a tratar das operações em rede necessárias, é ainda possível utilizar o segundo executável independentemente de forma a providenciar um concerto acústico a uma audiência que se encontre nas instalações desses mesmos instrumentos. De referir que as aplicações poderão ser usadas com qualquer instrumento acústico robotizado.

No início do projeto, um dos possíveis cenários em que se idealizou poder utilizar esta aplicação, seria a de poder disponibilizar a um público um concerto ao vivo utilizando para isso os mesmos instrumentos robotizados que executam as gravações. Seguindo as instruções de execução descritas no capítulo seguinte, é possível correr um dos programas de forma independente e automatizada para conseguir obter este cenário.

4.7 Resumo e Conclusões

Todos os componentes da aplicação local tiveram alguns contratempos no seu desenvolvimento e integração devido a compatibilidades com bibliotecas e modos de compilação das mesmas, enquanto o desenvolvimento da aplicação em rede foi mais linear e menos atribulado. Também no módulo local do sistema, foi necessário alterar alguns pormenores relativos à *framework* utilizada o que resultou num maior controlo do ciclo de execução da *app*.

Com a pluridade de problemas resolvidos durante o desenvolvimento da aplicação, foi possível obter um produto finalizado que vai de encontro aos requisitos estabelecidos no capítulo anterior.

Capítulo 5

Testes e Resultados

Abordam-se de seguida alguns testes levados a cabo durante o desenvolvimento do projeto, fornecendo-se também exemplos da interface do projeto e como executá-lo.

5.1 Reprodução de MIDIs

Foi necessário efetuar testes a ficheiros MIDI, mais precisamente aos parâmetros possíveis na especificação de notas musicais, tentando delimitar as possibilidades da sua reprodução nos instrumentos acústicos disponibilizados. Dos resultados desses testes foram retirados algumas conclusões.

5.1.1 Escala

No início da fase de análise da dissertação, foram realizados testes preliminares para averiguar quais seriam as limitações físicas e mecânicas dos instrumentos ao qual o sistema teria acesso. Nesta altura, quer o *Disklavier* quer o *NotomotoN* padeciam de falhas mecânicas que os impedia de reproduzir todas as suas notas musicais. É importante referir que o *NotomotoN* é apenas capaz de reproduzir notas MIDIs desde o número 0 até ao 17, sendo que as notas MIDIs variam entre o valor 0 e 127, e o valor 21 corresponde à 1ª nota do *Disklavier* utilizado.

Segue uma tabela com os testes efetuados às notas MIDI possíveis em cada instrumento, sendo que de cada vez que a nota não se aplica ao instrumento, por não existir maneira de a reproduzir neste, é indicado com um ‘-‘.

Notas MIDI	*NotomotoN*	*Disklavier*
0	✓	-
1	✓	-
2	✓	-
3	✓	-
4	-	-
5	×	-
6	×	-
7	✓	-
8	✓	-
9	-	-
10	✓	-
11	✓	-
12	-	-
13	-	-
14	-	-
15	-	-
16	-	-
17	-	-
…	-	-
21	-	✓
…	-	✓
108	-	✓

Tabela 1 - Testes de Escala Musical / MIDI

Durante o semestre, foi realizada uma manutenção no *Disklavier*, tendo sido afinado e calibrado. Depois de novos testes, verificou-se que este executava sem nenhum erro ou falha toda a escala musical presente no piano de 88 teclas.

O *NotomotoN*, tendo apenas 10 baquetas instaladas das 18 possíveis, também não era capaz de reproduzir as notas 5 e 6 da escala, continuando a necessitar de alguma calibração e reparação nos seus motores.

Na tabela 1, podemos verificar que todas as notas no *Disklavier* eram reproduzidas sem problemas, do 21 ao 108. Por outro lado, no *NotomotoN*, nem todas as baquetas instaladas foram capazes de reproduzir as notas correspondentes.

5.1.2 Velocidade

Num novo conjunto de testes, pretendia-se testar a velocidade de impulsão das teclas ou baquetas ao reproduzir uma nota musical. De referir novamente que os testes realizados inicialmente revelaram falhas mecânicas, corrigidas após uma fase de manutenção dos instrumentos.

Velocidade	*NotomotoN*	*Disklavier*
0	×	×
1	×	✓
5	×	✓
10	×	✓
15	×	✓
20	×	✓
25	×	✓
30	×	✓
33	×	✓
34	✓	✓
35	✓	✓
100	✓	✓
110	✓	✓
120	✓	✓
127	✓	✓

Tabela 2 - Testes de Velocidade da nota MIDI

As velocidades das notas MIDIs encontram-se entre os valores 1 e 127, sendo o valor mais alto o mais rápido para simular a reprodução veloz de uma tecla, e o valor mínimo o de uma tecla com um som abafado e silencioso. O valor nulo também é possível de incluir um MIDI, servindo apenas como *placeholder* para uma eventual nota, nunca sendo reproduzida com este valor.

A teoria confirmou-se nos testes efetuados com o *Disklavier*, sendo apenas o valor nulo o único que não reproduzia qualquer som. No *NotomotoN*, a realidade era outra. Depois de confirmar que a velocidade 0 de uma nota MIDI não reproduzia som algum, o aumento gradual de valores revelou que mesmo nas baquetas mais próximas o som só surgia quando a velocidade da nota tinha no mínimo 34 unidades.

Na tabela 2 verifica-se que o *Disklavier* é capaz de reproduzir qualquer velocidade de nota graças ao seu mecanismo de reprodução. Já as baquetas no *NotomotoN* necessitam de uma nota com um mínimo de velocidade de valor 34 para reproduzir uma nota MIDI corretamente.

5.1.3 Duração

Executaram-se ainda testes com o intuito verificar a extensão de reprodução das notas, o prolongamento da sua duração, utilizando para isso os tempos e figuras musicais mostradas na Figura 20.

Som	Pausa	Nome
		Semibreve
		Mínima
		Semínima
		Colcheia
		Semicolcheia
		Fusa
		Semifusa

Figura 20 - Figuras e Tempos Musicais

Em nenhum dos instrumentos foi possível reproduzir com precisão notas no âmbito das quartifusas, uma das não listadas na tabela e a mais rara de todas as figuras. O *Disklavier* não as consegue executar, mas o *NotomotoN* reproduzia uma nota a cada unidade do compasso, por mais notas que lhe passasse consecutivamente num curto período de tempo. O *NotomotoN*, ao contrário do *Disklavier,* consegue reproduzir as suas notas recebendo um impulso da nota MIDI correspondente ao número das baquetas, daí ser capaz de executar uma nota independetemente da extensão desta.

Figura Musical	Tempo	*Disklavier*	*NotomotoN (Notas 1 & 2)*	*NotomotoN (Notas 0, 3, 7, 8, 10, 11)*
Semibreve	1/1	✓	✓	✓
Mínima	1/2	✓	✓	✓
Semínima	1/4	✓	✓	✓
Colcheia	1/8	✓	✓	✓
Semicolcheia	1/16	✓	✓	✓
Fusa	1/32	✓	×	✓
Semifusa	1/64	✓	×	×
Quartifusa	1/128	×	×	×

Tabela 3 - Testes de Tempos MIDI

O *NotomotoN* também não consegue reproduzir com precisão notas na ordem da semifusa, ocorrendo o mesmo fenómeno de incapacidade de reprodução de notas consecutivas. No caso das notas 1 e 2 deste instrumento, o mesmo ocorre para tempos a 1/32 do compasso.

Na tabela 3 resumem-se os resultados descritos anteriormente.

5.2 Gravação de áudio

Executaram-se também testes a parâmetros relacionados com a gravação dos ficheiros de áudio e a sua audibilidade correta.

5.2.1 Tipo de ficheiro

Existem vários tipos e formatos de ficheiros disponíveis para fazer, tendo-se tentado averiguar qual a melhor solução com um novo conjunto de testes, procurando por diferenças de qualidade no áudio das gravações finais.

Os testes realizados serviram para corroborar as possibilidades listadas na documentação da *libsndfile*, existindo pouca ou nenhuma diferença percetível na sonoridade do áudio final obtido. A tabela resultante poderá ser consultada nos anexos deste documento.

5.2.2 Sample Rate

Sample Rate ou taxa de amostragem é a quantidade de amostras do sinal de áudio analógico avaliadas numa unidade de tempo para serem convertidos num sinal digital na gravação. Para este parâmetro foi tido em conta o facto de que, de acordo com o Teorema de Nyquist, uma taxa de amostragem de no mínimo duas vezes o valor da frequência máxima alcançada pelo sinal analógico é necessária para possibilitar o registo digital de todas as frequências analisadas, sem perda de qualidade ou distorsão.

Na Tabela 4 temos o resultado dos testes a vários valores de *sample rate*, juntamente com uma pequena descrição de que para tipo de sinais ou dispositivos esse valor é normalmente usado. Apenas alguns dos valores resultaram numa gravação com todas as notas completamente audíveis, sendo que certos *sample rates* resultaram numa gravação mais longa e em tons semelhantes, mas não iguais, ao som natural no estúdio.

Sample Rate	Descrição	Gravação áudio	Comentários
8 000	Telefone & *Walkie-Talkie* Encriptado	×	-
11 025	1/4 CD áudio & PCM/MPEG de baixa qualidade	×	-
16 000	VoIP & VVoIP	×	-
22 050	1/2 CD áudio & digitalização de formatos antigos	×	-
32 000	miniDV / gravador de vídeo & digitalização de rádio FM	×	-

44 056	Áudio digital de NTSC a cores	✓	Som natural
44 100	CD áudio	✓	Som natural
47 250	1º gravador de som PCM	✓	Tom gravado difere do ouvido no estúdio (+2s de gravação)
48 000	*Standard*	✓	Tom gravado difere do ouvido no estúdio (+2s de gravação)
50 000	1º gravador de som digital	✓	Tom gravado difere do ouvido no estúdio (+2s de gravação)
50 400	Gravador de som digital Mitsubishi X-80	✓	Tom gravado difere do ouvido no estúdio (+5s de gravação)
88 200	Algum equipamento profissional de gravação	×	-
96 000	DVD-audio, Blu-Ray & HD DVD audio	×	-
176 400	Gravadores HDCD	×	-
192 000	DVD-audio, Blu-Ray & HD DVD audio	×	-
352 800	Digital Xtreme & Super Audio CD	×	-
2 822 400	SACD & Direct Stream Digital (DSD)	×	-
5 644 800	Gravadores DSD Pro & *Double-Rate DSD*	×	-

Tabela 4 - Testes de *Sample Rate*

Os resultados obtidos foram bastante variáveis, principalmente quando foram usados valores que logo à partida seriam teoricamente descartáveis. As variações mais notórias

prenderam-se com a duração da gravação final, obviamente diferenciadora conforme o valor do *sample rate* utilizado correspondendo os valores mais baixos às gravações mais curtas, e com o tom das notas das gravações, diferente mesmo em casos em que a gravação deveria ter um som natural como ouvido no estúdio.

5.2.3 Número de microfones

O número de canais no ficheiro de áudio irá variar conforme o número de microfones disponibilizados durante a gravação. Os testes realizados foram limitados ao uso de dois microfones, um exterior e o integrado no computador, devido à falta de interfaces de áudio capazes de se ligarem ao computador da mesma forma que aquela utilizada para testes, uma *M Box 2 Mini*, tendo esta apenas uma ligação MIDI disponível.

O uso quer de um canal, quer de dois simultâneos foram bem-sucedidos e resultaram em gravações distintas entre si, limitando diferentes sons a um dos altifalantes conforme o microfone de onde este era obtido.

5.2.4 *Buffers* de dados

Os dados da gravação são transmitidos para o ficheiro de áudio através de um protocolo de *sliding window*, pelo que se torna importante verificar a correlação entre a qualidade do áudio e a existência de erros nos dados guardados com o tamanho da janela e o número de *buffers* utilizados aquando do processo de gravação.

5.2.4.1 Número de *buffers*

Os *buffers* aqui mencionados são utilizados de forma bidirecional, sendo a quantidade de dados MIDI *IN* e *OUT* utlizados de uma só vez dependente do número de *buffers* disponíveis para transmissão.

Na Tabela 5 estão representados os testes feitos com diferente número de *buffers*, tendo os resultados variado pouco ou nada uns em relação aos outros.

Nº *Buffers*	Gravação áudio
1	✓
2	✓
3	✓
4	✓

5	✓
6	✓
7	✓

Tabela 5 - Testes de Número de *Buffers*

De referir que não existiram diferenças notórias nas gravações finais com as alterações efetuadas durante estes testes. No entanto, sistemas mais limitados que o utilizado no desenvolvimento desta ferramenta e na execução dos testes poderão necessitar de ver estes parâmetros alterados na chamada da execução.

5.2.4.2 Tamanho dos *buffers*

No protocolo de *sliding window*, o tamanho dos *buffers* definido poderá limitar a eficaz utilização da largura de banda da ligação. Estes testes pretendiam averiguar se isso também ocorria nas gravações de áudio, e se existiam erros associados à mudança de tamanho dos contentores de transmissão de dados. Nenhum dos casos se verificou, mas volta-se a mencionar a possibilidade de isto ocorrer em sistemas mais limitados.

Como se mostra na Tabela 6, efetuaram-se testes para averiguar a existência de erros nas gravações ao utilizar diferentes tamanhos de *buffers*.

Tamanho *Buffers (bytes)*	Gravação áudio
64	✓
128	✓
256	✓
512	✓
1024	✓
2056	✓
4112	✓

Tabela 6 - Testes de Tamanho de *Buffers*

5.3 Execução da *app*

Neste tópico pretende-se explorar e explicitar as intruções de execução para as aplicações locais desenvolvidas, nos contextos em que poderão vir a ser aplicadas, nas gravações das melodias e num concerto para uma audiência ao vivo.

5.3.1 Gravação

De modo a executar as gravações dos MIDIs presentes no servidor nos instrumentos acústicos, deve-se ter os executáveis da aplicação 'lado a lado', e simplesmente executar *MyApp*, ficando todos os procedimentos a cargo do programa.

Quando executada neste modo *default*, a aplicação utiliza os parâmetros com os valores aplicados por definição, tendo estes sido obtidos conforme o resultado dos testes feitos anteriormente. No entanto, é possível alterar alguns destes parâmetros de forma prévia à execução dos programas, através da linha de comandos. Os parâmetros existentes são mostrados na Tabela 7.

Parâmetros	*Default*	Alterável?
Channels	1	✓
Sample Rate	44100	✓
Buffers	4	✓
Buffer Size	1024	✓
File Type	WAV	×
Data Type	PCM 16 bit	×

Tabela 7 - Parâmetros de Execução

Os dois últimos parâmetros permaneceram inalteráveis devido a complicações de execução quando se realizaram tentativas de passar estas opções pela linha de comandos. Visto que todas as execuções e testes foram efetuados no sistema operativo Windows e num processador *Intel*, a utilização de ficheiros *Wav* e de uma modulação *PCM* revelou-se como a mais apropriada.

Os restantes parâmetros são alteráveis a partir de uma consola, com o comando e ordem de chamada ilustrados na Figura 21.

```
> MyApp (CHANNELS) (SAMPLE_RATE) (BUFFERS) (BUFFER_SIZE)
```

Figura 21 - Execução *MyApp* (linha de comandos)

Channels irá definir o número de microfones a ser utilizado, enquanto que *Sample_Rate* variará a taxa de amostragem pretendida para as gravações. A inclusão do número de *buffers* e do seu tamanho nos parâmetros alteráveis prendeu-se com a possibilidade do projeto ter de vir a ser executado em sistemas mais limitados, existindo a necessidade de baixar a quantidade de memória utilizada durante as execuções.

5.3.2 Concerto

Graças à divisão modular da aplicação, a existência de dois executáveis separados é justificada de modo a existir a possibilidade de utilizar o segundo de forma manual para possivelmente realizar um concerto para uma audiência ao vivo, um cenário discutido com os responsáveis pelo projeto no início da fase de planeamento deste.

A utilização solitária desta aplicação continua a implicar a montagem de pelo menos um microfone e de todos os instrumentos acústicos no sistema, acrescentando também a necessidade de criar manualmente uma pasta *data* ao mesmo nível do segundo executável, *NDISS*. Nesta pasta deverão ser colocados todos os MIDIs que pretendem ser reproduzidos durante o concerto, bastando depois correr a aplicação *NDISS* para desfrutar das melodias.

Tal como a anterior, quando esta é executada de forma normal, os parâmetros são definidos por *default*. Todavia, podem ser alterados pela linha de comandos, como se mostra na Figura 22.

```
> NDISS (CHANNELS) (SAMPLE_RATE) (BUFFERS) (BUFFER_SIZE)
```

Figura 22 - Execução *NDISS* (linha de comandos)

5.4 Interfaces

Com o projeto completo, todos os testes efetuados e a forma de execução das aplicações locais definida, pretende-se agora mostrar alguns exemplos das interfaces apresentadas aos utilizadores e administradores do sistema.

5.4.1 Interfaces *app*

Na Figura 23 mostra-se uma execução de *MyApp* com parâmetros definidos manualmente.

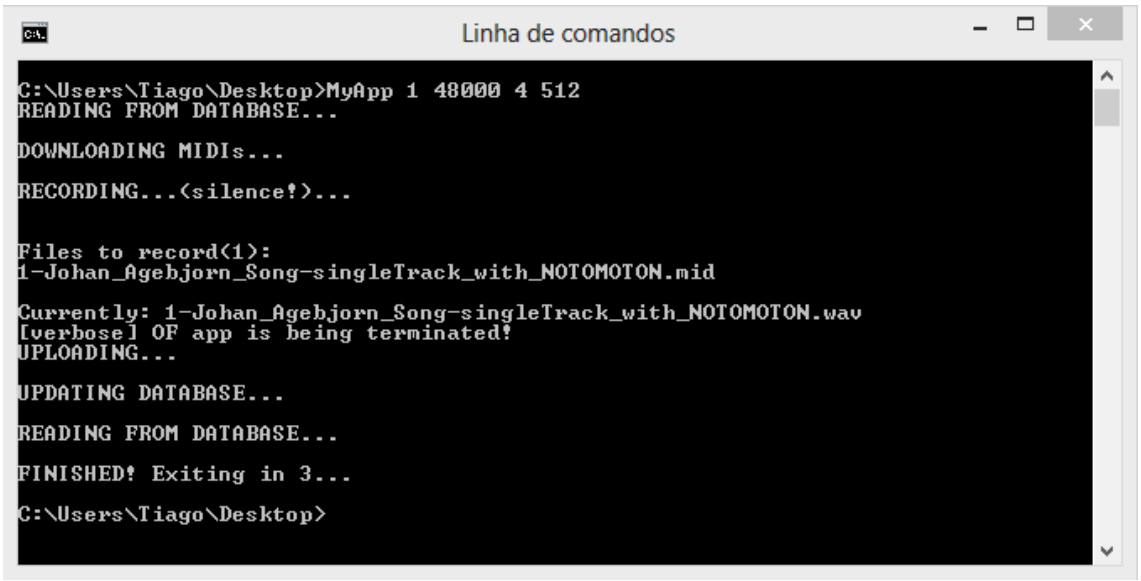


Figura 23 - *MyApp* com parâmetros manuais

Na Figura 24 é representada a janela do *openFrameworks* aquando da execução das gravações.

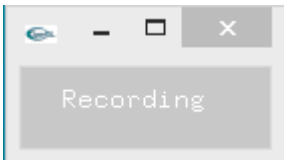


Figura 24 - Janela de gravação *openFrameworks*

Na Figura 25 apresenta-se uma execução de *NDISS*, aplicação para um concerto, com parâmetros definidos manualmente.

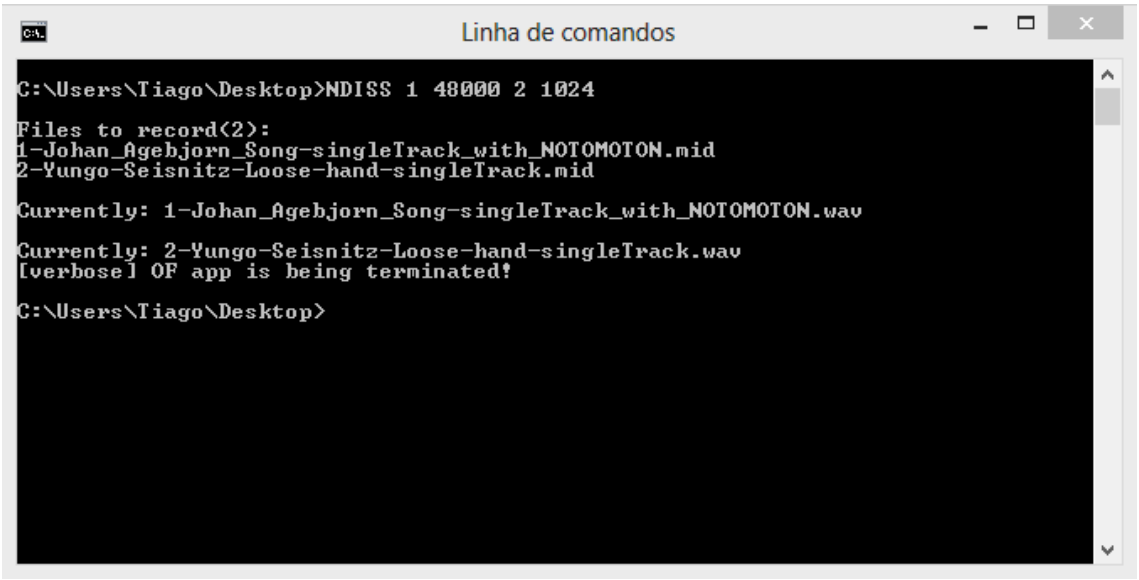


Figura 25 - *NDISS* com parâmetros manuais

As mensagens apresentadas não diferem quando as aplicações são executadas de forma *standard*, sem definir manualmente os parâmetros da gravação.

5.4.2 Interfaces *web*

Com as interfaces da aplicação apresentadas, mostram-se agora as interfaces mais relevantes do *website* desenvolvido. Na Figura 26 é apresentada a página inicial.

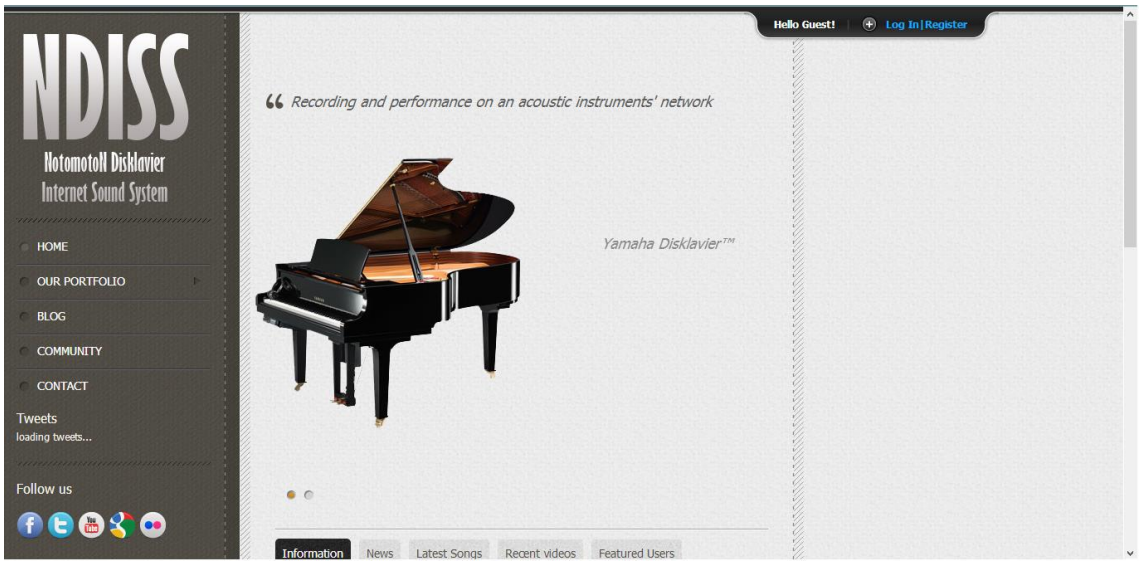


Figura 26 - Página Inicial

Na Figura 27 mostra-se a página de *upload* de MIDIs antes de qualquer interação.

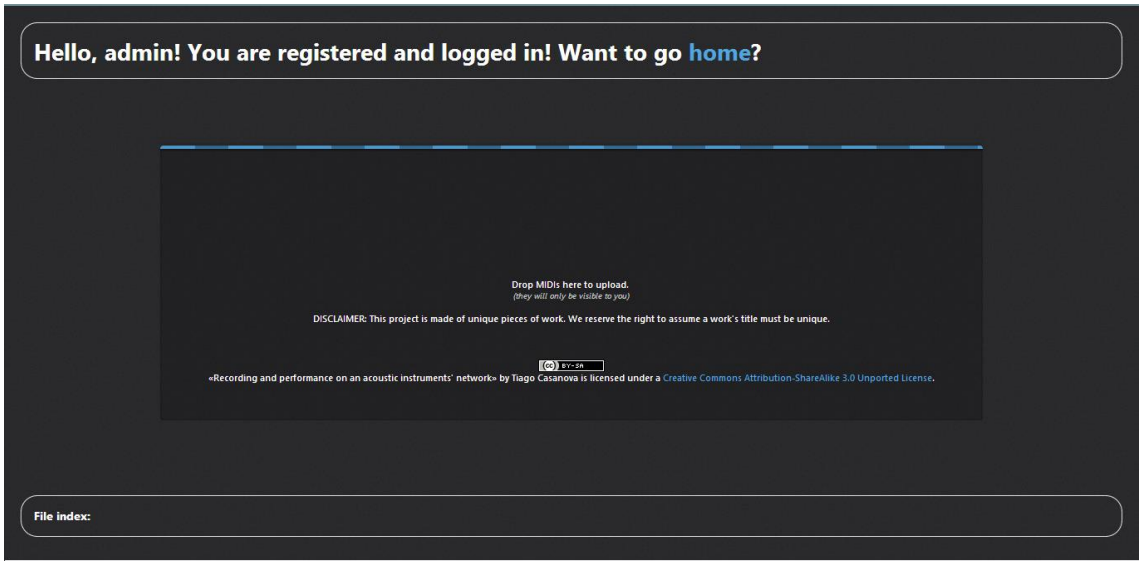


Figura 27 - Página de *upload* de MIDIs

Na Figura 28 é dado um exemplo de respostas possíveis do *website* aquando do *upload* de diversos ficheiros MIDI.

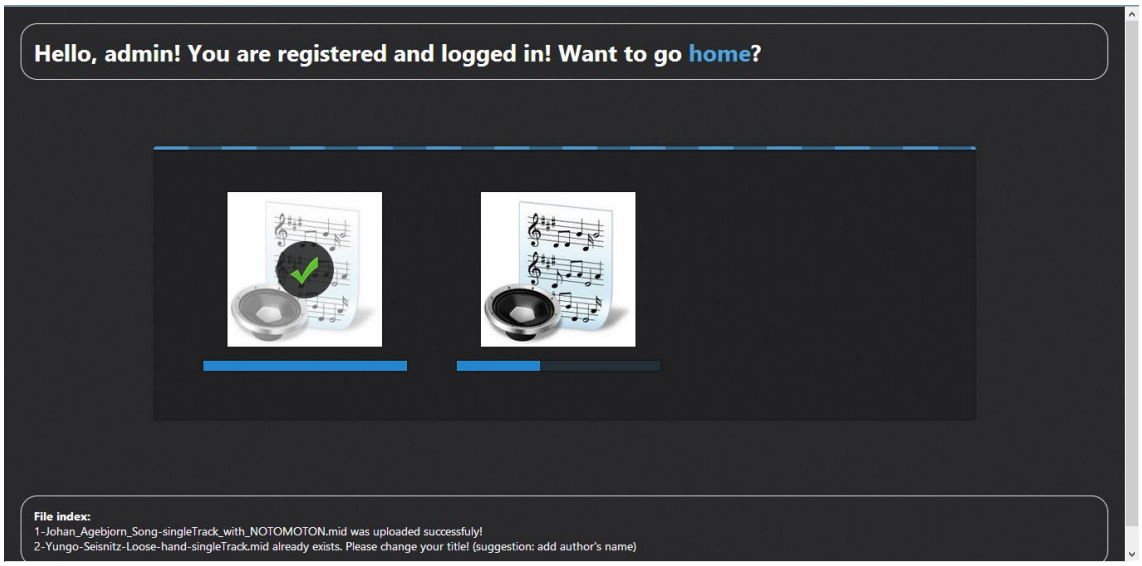


Figura 28 - *Upload* de vários MIDIs

Na Figura 29 apresenta-se a galeria de música onde é possível fazer *download* das melodias gravadas com este sistema, caso seja um utilizador autenticado com sessão iniciada.

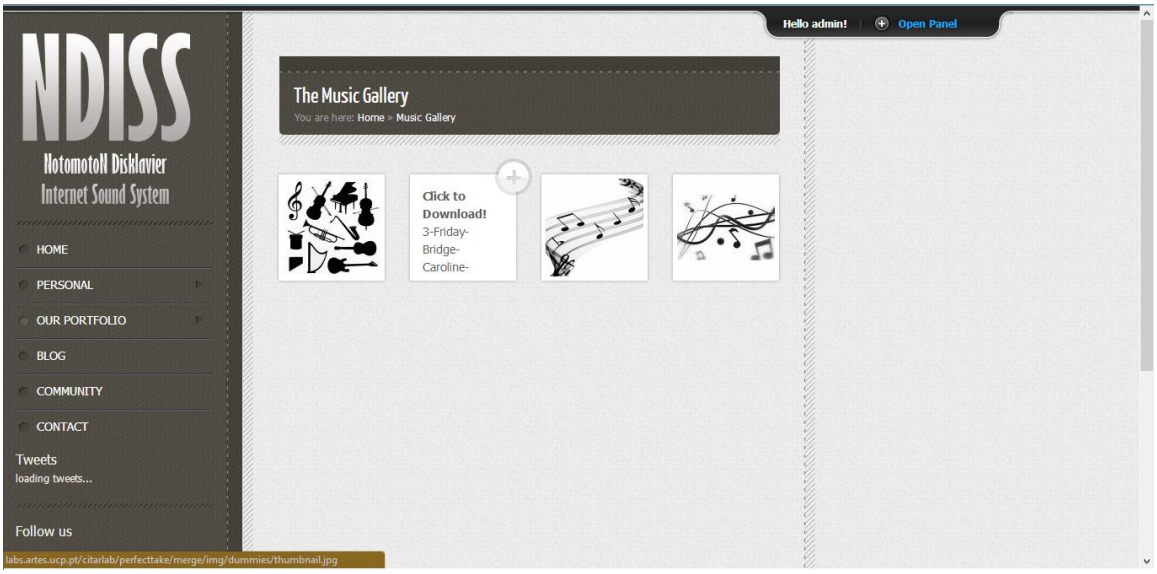


Figura 29 - Galeria de Música (*Download*)

5.5 Resumo e Conclusões

Analisando os testes anteriores e os seus resultados, podemos concluir que o sistema não padece de muitas restrições, nenhuma delas crucial, estando principalmente limitado por questões mecânicas dos instrumentos acústicos robotizados.

O método de execução das aplicações, mais propriamente a definição manual de parâmetros de gravação, permaneceu igual em ambos os casos de modo a simplificar a sua utilização para os administradores do sistema. Também as mensagens das *apps* e a interface do *website* permaneceram o mais simples possível, como podemos ver pelas imagens anteriores, com o propósito de esconder toda a complexidade inerente ao sistema dos seus gestores e utilizadores.

Capítulo 6

Conclusões Finais e Trabalho Futuro

Este projeto de dissertação incidiu sobre várias temáticas de diversas áreas, salientando-se pela multidisciplinaridade dos tópicos abordados e pela complexidade do sistema final pretendido, um serviço *web* de gravação remota de MIDIs em instrumentos acústicos robotizados.

A criação de um sistema com funcionalidades de gravação acústica permite a qualquer compositor de MIDIs utilizar instalações remotas únicas para gravar as suas melodias em instrumentos acústicos robotizados, ao invés de utilizarem *samples* pré-gravados separadamente como nos *softwares* de composição tradicionais, tirando proveito da reverberação do local, obtendo uma gravação com um som mais natural.

Simplesmente analisando a arquitetura do sistema tem-se uma ideia dos vários módulos desenvolvidos de modo a obter um produto funcional e que corresponde às expetativas, objetivos e requisitos delineados previamente.

6.1 Satisfação dos Objectivos e Benefícios

Após alguns contratempos naturalmente associados ao desenvolvimento de qualquer *software*, o projeto final cumpre com todos os requisitos definidos, atingindo os objetivos propostos no início desta dissertação, juntamente com a documentação necessária e um artigo científico com a descrição do modelo, da arquitetura de *software,* e com os resultados do teste ao protótipo.

O estudo realizado ao longo desta dissertação revelou também as várias potencialidades deste projeto que, para além das já mencionadas, poderá servir como um produto rentável aplicando-lhe o modelo de negócio certo, seja cobrando por gravação, por inscrição ou outra opção possível. Por outro lado, também se salientaram as limitações deste sistema, muitas delas associadas aos instrumentos em si e ao leque de opções disponíveis, no caso apenas instrumentos acústicos robotizados. Por serem instrumentos robotizados, também se pode argumentar que as gravações perdem alguma expressividade musical normalmente associada ao intérprete da melodia, mas o risco de haver erros na reprodução nesses casos não é nulo.

6.2 Trabalho Futuro

Ainda existem algumas possibilidades de desenvolvimento para a ferramenta como a expansão suas funcionalidades, oferecendo um produto com maior valor para os eventuais utilizadores deste sistema.

Após vários contatos com músicos, engenheiros de áudio, entre outros na área musical, foi possível obter uma perceção melhorada sobre quais seriam as mais-valias do projeto para além daquelas obviamente destacadas anteriormente. A possibilidade de gravação acústica de instrumentos reais foi amplamente recebida com boas críticas, no entanto foi também aconselhada a expansão do leque disponível, principalmente se o alvo fossem instrumentos únicos como o caso do Gamelão Robótico na Casa da Música, que se mostraram disponíveis para colaborar no projeto mas o tempo limitado de desenvolvimento assim não o permitiu.

Foi também ponderado utilizar um serviço *online* como o *SoundCloud* para armazenar as gravações de áudio resultantes do sistema, mantendo-se os MIDIs privativamente guardados no servidor local, de modo a não sobrecarregar a memória de armazenamento disponível. Visto que as melodias acústicas virão a estar disponíveis publicamente no *website* criado, não haveria problemas em ter as permissões de acesso às gravações com o nível 'público', e poderíamos tirar proveito de um serviço de armazenamento na *cloud* sem contratempos. Apenas seria necessário redirecionar o utilizador para o *link* de *download* deste serviço e não para o repositório de ficheiros de áudio, e o *upload* da aplicação local teria de ser feito por serviços *RESTful web services*, neste caso POST, para o *SoundCloud* ao invés de por *FTP* para o servidor.

Referências

1. Taub RD, Cabanilla JA, Tourtellot G. Collaborative music creation [Internet]. 8035020, 2011 [citado 2 de Fevereiro de 2013]. Obtido de: http://www.google.pt/patents?id=3uH3AQAAEBAJ

2. Sawchuk AA, Chew E, Zimmermann R, Papadopoulos C, Kyriakakis C. From remote media immersion to Distributed Immersive Performance. Proc. 2003 Acm Sigmm Work. Exp. Telepresence [Internet]. New York, NY, USA: ACM; 2003 [citado 2 de Fevereiro de 2013]. p 110–20. Obtido de: http://doi.acm.org/10.1145/982484.982506

3. Carôt A. Musical Telepresence: A Comprehensive Analysis Towards New Cognitive and Technical Approaches. 2009.

4. Gu X, Dick M, Kurtisi Z, Noyer U, Wolf L. Network-centric music performance: practice and experiments. Ieee Commun. Mag. Junho de 2005;43(6):86 – 93.

5. Mills R. Dislocated Sound: A Survey of Improvisation in Networked Audio Platforms. Anon. Conf. New Interfaces Music. Expr. Nime 2010. 2010.

6. Chafe C, Wilson S, Leistikow R, Chisholm D, Scavone G. A simplified approach to high quality music and sound over IP. Cost-G6 Conf. Digit. Audio Eff. 2000. p 159–64.

7. Keller D, Flores LV, Pimenta MS, Capasso A, Tinajero P. Convergent Trends Toward Ubiquitous Music. J. New Music Res. 2011;40(3):265–76.

8. McPherson A. The Magnetic Resonator Piano: Electronic Augmentation of an Acoustic Grand Piano. J. New Music Res. Setembro de 2010;39(3):189–202.

9. Gingras B, McAdams S. Improved Score-performance Matching Using Both Structural and Temporal Information from MIDI Recordings. J. New Music Res. Março de 2011;40(1):43–57.

10. Goebl W, Bresin R. Measurement and reproduction accuracy of computer-controlled grand pianos. J. Acoust. Soc. Am. 2003;114(4 I):2273–83.

11. Flossmann S, Goebl W, Grachten M, Niedermayer B, Widmer G. The Magaloff Project: An Interim Report. J. New Music Res. Dezembro de 2010;39(4):363–77.

12. Bhatara A, Tirovolas AK, Duan LM, Levy B, Levitin DJ. Perception of emotional expression in musical performance. J. Exp. Psychol. Hum. Percept. Perform. Junho de 2011;37(3):921–34.

13. Ninjam [Internet]. [citado 1 de Fevereiro de 2013]. Obtido de: http://www.cockos.com/ninjam/

14. Blythe M, Sewell T. Ejamming All over the World. 2008;

15. jacktrip [Internet]. Jacktrip - Syst. High-Qual. Audio Netw. Perform. Internet. [citado 14 de Dezembro de 2011]. Obtido de: http://code.google.com/p/jacktrip/

16. Jordà S. Afasia the Ultimate Homeric One-man-multimedia-band. 2002. Obtido de: files/publications/Nime2002-jorda.pdf

17. Kapur A, Wang G, Davidson P, Cook PR. Interactive Network Performance: a dream worth dreaming? Organised Sound. 2005;10(03):209–19.

18. Tohgi Y, Iyatomi A, Hara M, Hirose T. Music lesson system with local training terminal and remote supervisory station [Internet]. 6211451, 2001 [citado 5 de Fevereiro de 2013]. Obtido de: http://www.google.pt/patents?id=WDoGAAAAEBAJ

19. The Silophone [Internet]. Silo 5 Grain Elev. [citado 2 de Fevereiro de 2013]. Obtido de: http://www.silophone.net/eng/about/desc.html

20. Faria RRA. Profiling New Paradigms in Sound and Music Technologies. J. New Music Res. 2011;40(3):191–204.

21. Carôt A, Werner C. Network music performance-problems, approaches and perspectives. Proc. «music Glob. Village»-Conf. Bp. Hung. 2007.

22. Pritchett J. The Music of John Cage. Cambridge University Press; 1996.

23. Brown C, Bischoff J. Computer Network Music Bands: A history of the League of Automatic Music Composers and the Hub. Distance Precursors Art Act. Internet. 2005;372–91.

24. Gresham-Lancaster S. The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music. Leonardo Music J. 1998;8:39.

25. Collins N, McLEAN A, Rohrhuber J, Ward A. Live coding in laptop performance. Organised Sound. 2003;8(03):321–30.

26. Weinberg G. Interconnected Musical Networks: Toward a Theoretical Framework. Comput. Music J. 1 de Junho de 2005;29(2):23–39.

27. Gabrielsson A. Music Performance Research at the Millennium. Psychol. Music. 1 de Julho de 2003;31(3):221–72.

28. Weinberg G. Interconnected Musical Networks – Bringing Expression and Thoughtfulness to Collaborative Music Making. Massachusetts Institute of Technology Media Laboratory; 2003.

29. Web development [Internet]. Wikipedia Free Encycl. 2013 [citado 3 de Fevereiro de 2013]. Obtido de: http://en.wikipedia.org/w/index.php?title=Web_development&oldid=535809641

30. PHP [Internet]. php. [citado 3 de Fevereiro de 2013]. Obtido de: http://www.php.net/

31. Udemy Blog » Code Wars: Ruby vs Python vs PHP [Infographic] [Internet]. Udemy Blog. [citado 3 de Fevereiro de 2013]. Obtido de: http://www.udemy.com/blog/modern-language-wars/

32. Python [Internet]. python. [citado 4 de Fevereiro de 2013]. Obtido de: http://www.python.org/

33. django [Internet]. [citado 4 de Fevereiro de 2013]. Obtido de: https://www.djangoproject.com/

34. Ruby [Internet]. [citado 4 de Fevereiro de 2013]. Obtido de: www.ruby-lang.org

35. Ruby On Rails [Internet]. Ruby Rails. [citado 4 de Fevereiro de 2013]. Obtido de: http://rubyonrails.org/

36. Schroeder MR. Digital Simulation of Sound Transmission in Reverberant Spaces. J. Acoust. Soc. Am. 1970;47(2A):424–31.

37. Madan E, McIntosh T. Silophone. Espace Sculpt. 2002 de 2001;(58):23–5.

38. Buchner A. Mechanical musical instruments. Greenwood Press; 1959.

39. Roads C. Research in music and artificial intelligence. Acm Comput Surv. Junho de 1985;17(2):163–90.

40. Gann K. The Music of Conlon Nancarrow. Cambridge University Press; 2006.

41. Rowe R. Interactive music systems: machine listening and composing. Cambridge, MA, USA: MIT Press; 1992.

42. Singer E, Feddersen J, Redmon C, Bowen B. LEMUR's Musical Robots. 2004.

43. Kapur A, Eigenfeldt A, Bahn C, Schloss WA. Collaborative Composition For Musical Robots. J. Sci. Technol. Arts. 5 de Maio de 2009;1(1):48–52.

44. Weinberg G, Driscoll S, Parry M. HAILE–AN INTERACTIVE ROBOTIC PERCUSSIONIST. 2005;

45. Kapur A, Hochenbaum J, Darling M, Diakopoulos D, Murphy J. The KarmetiK NotomotoN: A New Breed of Musical Robot for Teaching and Performance. Proc. Int. Conf. New Interfaces Music. Expr. p 228–31.

46. Jaffe DA, Schloss A. A Virtual Piano Concerto-Coupling of the Mathews/Boie Radio Drum and the Yamaha Disklavier Grand Piano in« The Seven Wonders of the Ancient World». Proc. Int. Comput. Music Conf. [Internet]. 1994. p 192–192. Obtido de: http://people.finearts.uvic.ca/~aschloss/publications/ICMC94_Aarhus_Virtual%20Piano.htm

47. Renaud A, Carôt A, Rebelo P. Networked music performance: State of the art. Proc. Aes 30th Int. Conf. 2007.

48. Schiavoni FL, Bianchi AJ, Queiroz M. Ferramentas livres para distribui\ccao de áudio em rede.

49. eJamming [Internet]. [citado 1 de Fevereiro de 2013]. Obtido de: http://ejamming.com/

50. CCRMA. SoundWIRE [Internet]. soundwire. [citado 13 de Dezembro de 2012]. Obtido de: https://ccrma.stanford.edu/groups/soundwire/

51. Alexander Carot. SoundJack [Internet]. Soundjack Prof Dr Alexander Carot. 2013 [citado 3 de Fevereiro de 2013]. Obtido de: http://www.carot.de/soundjack/

52. Alexandraki C, Akoumanakis D. Exploring New Perspectives in Network Music Performance: The DIAMOUSES Framework. Comput. Music J. Summer de 2010;34(2):66–83.

53. Carôt A, Werner C. External latency-optimized soundcard synchronization for applications in wide-area networks. Aes 14th Reg. Conv. Tokio Jpn. 2009. p 7.

54. Carôt A, Werner C. Distributed network music workshop with soundjack. Proc. 25th Tonmeistertagung Leipz. Ger. 2008;

55. Anderson M. Resources: Virtual Jamming. Ieee Spectr. Julho de 2007;44(7):53 –56.

56. Chafe C, Wilson S, Walling D. Physical model synthesis with application to Internet acoustics. 2002 Ieee Int. Conf. Acoust. Speech Signal Process. Icassp. 2002. p IV–4056 – IV–4059.

57. Fastl H, Zwicker E. Psychoacoustics: Facts and Models. Springer; 2006.

58. Cáceres J-P, Chafe C. JackTrip: Under the Hood of an Engine for Network Audio. J. New Music Res. 2010;39(3):183–7.

59. JACK Audio Connection Kit [Internet]. jack. [citado 3 de Fevereiro de 2013]. Obtido de: http://jackaudio.org/documentation

60. Cáceres J-P, Chafe C. JackTrip/SoundWIRE Meets Server Farm. Comput. Music J. 1 de Setembro de 2010;34(3):29–34.

61. Llc B. Audio Programming Languages: Mathematica, Csound, Max, Pure Data, Supercollider, Comparison of Audio Synthesis Environments. General Books; 2010.

62. Comparison of audio synthesis environments [Internet]. Wikipedia Free Encycl. 2013 [citado 29 de Janeiro de 2013]. Obtido de: http://en.wikipedia.org/w/index.php?title=Comparison_of_audio_synthesis_environments&oldid=533863317

63. C'74. O que é Max? [Internet]. Cycl. 74. [citado 31 de Janeiro de 2013]. Obtido de: http://cycling74.com/whatismax/portuguese/

64. Puckette M. The Patcher. Em: Lischka C, Fritsch J, editores. 1988. p The International Computer Music Association–429.

65. Puckette MS, Ucsd MSP, Apel T, Edu U (tapel@ucsd, Zicarelli DD, Com C (www C. Real-time audio analysis tools for Pd and MSP. 1998.

66. Pure Data [Internet]. FLOSS Manuals; 2012 [citado 1 de Fevereiro de 2013]. Obtido de: http://en.flossmanuals.net/_booki/pure-data/pure-data.pdf

67. Puckette M. Using Pd as a score language. Proc Int Comput. Music Conf. 2002. p 184–7.

68. Puckette M. A divide between 'compositional'and 'performative'aspects of Pd. Proc First Int. Pd Conv. 2004.

69. Puckette M, others. Pure Data: another integrated computer music environment. Proc. Second Intercollege Comput. Music Concerts. 1996;37–41.

70. Barkl M. Composition : pure data as a meta-compositional instrument. Univ. Wollongong Thesis Collect. [Internet]. 1 de Janeiro de 2009; Obtido de: http://ro.uow.edu.au/theses/794

71. Gräf A. Signal processing in the Pure programming language. Proc. 7th Int. Linux Audio Conf. Parma Casa Della Musica. 2009.

72. Puckette M. The theory and technique of electronic music. World Scientific Publishing Company Incorporated; 2007.

73. SuperCollider [Internet]. Supercollider Real-Time Audio Synth. Algorithmic Compos. [citado 1 de Fevereiro de 2013]. Obtido de: http://supercollider.sourceforge.net/

74. Pope ST. Sound and Music Processing in SuperCollider. Cent. Res. Electron. Art Technol. Univ. Calif. St. Barbara. 1997;

75. Wilson S, Cottle D, Collins N. The SuperCollider Book. The MIT Press; 2011.

76. McCartney J. SuperCollider: a new real time synthesis language. 1996.

77. Rutz HH. Rethinking the Supercollider Client...

78. Blechmann T. supernova, a multiprocessor-aware synthesis server for SuperCollider. Proc. 8th Linux Audio Conf. 2005. p 32.

79. Orlarey Y, Gräf A, Kersten S. DSP programming with Faust, Q and SuperCollider. Proc. 4th Int. Linux Audio Conf. Lac06. 2006. p 39–47.

80. McCartney J. Rethinking the Computer Music Language: SuperCollider. Comput. Music J. Winter de 2002;26(4):61–8.

81. Csound [Internet]. [citado 1 de Fevereiro de 2013]. Obtido de: http://www.csounds.com/docs

82. Csound [Internet]. FLOSS Manuals; 2012 [citado 1 de Fevereiro de 2013]. Obtido de: http://en.flossmanuals.net/_booki/csound/csound.pdf

83. Vercoe B, Ellis D. Real-time CSOUND: Software synthesis with sensing and control. Proc. Int. Comput. Music Conf. 1990. p 209–11.

84. Boulanger RC. The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming. MIT Press; 2000.

85. openFrameworks [Internet]. openframeworks. [citado 12 de Dezembro de 2012]. Obtido de: http://www.openframeworks.cc/about/

Referências

Anexos

Anexo A – Diagrama de *Brainstorming*

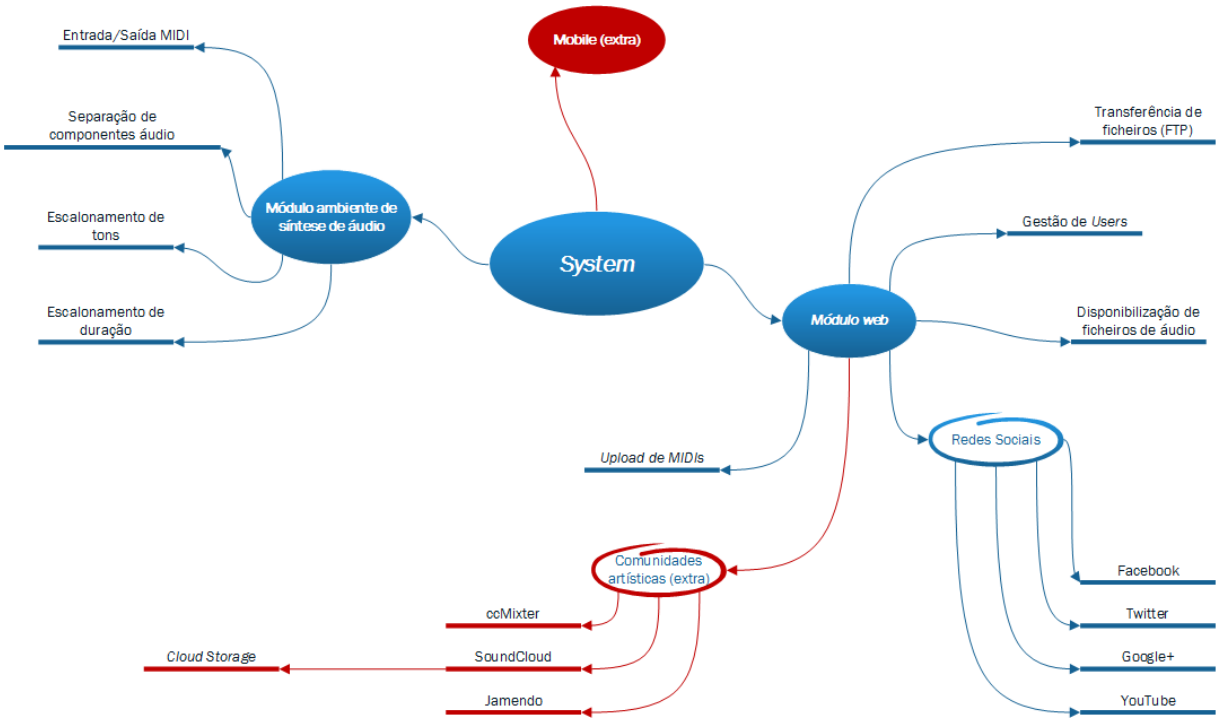


Figura 30 - Diagrama de *Brainstorming*

Anexo B – Diagrama de *Gantt*

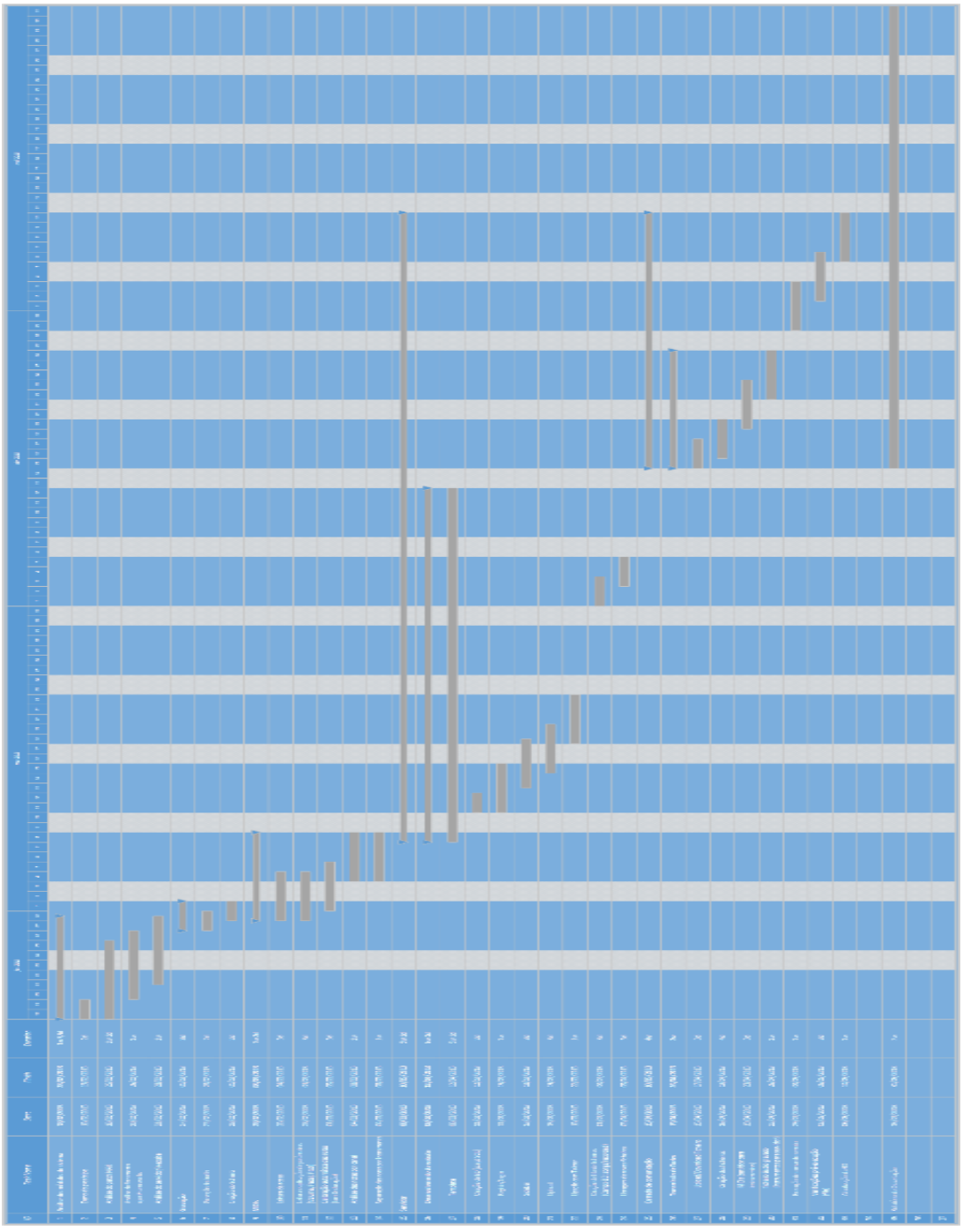


Figura 31 - Diagrama de *Gantt*

Anexo C – Imagens do *website*

C.1 - *Community*

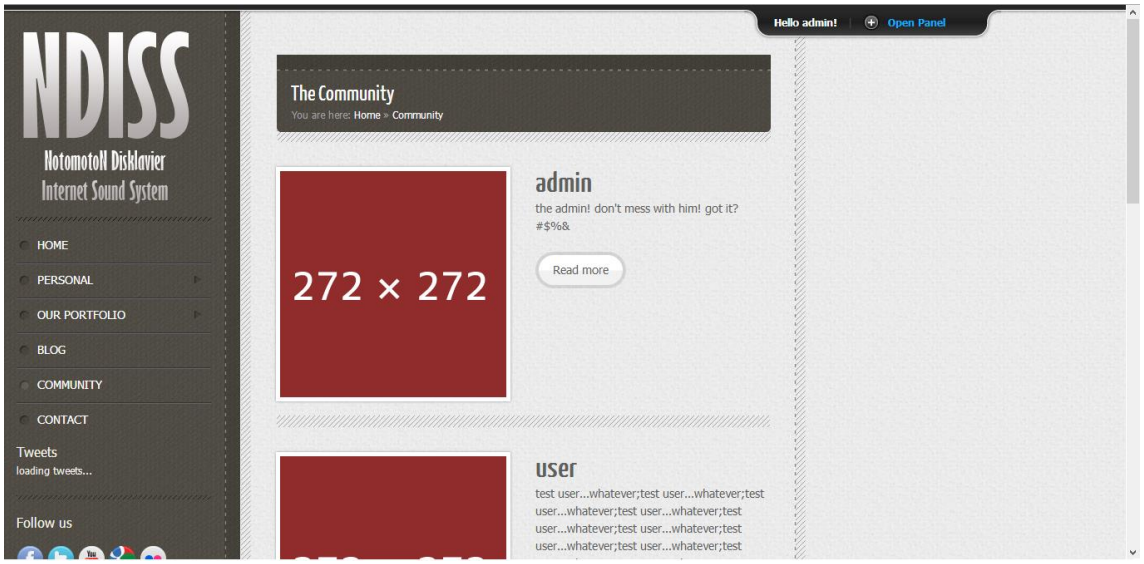
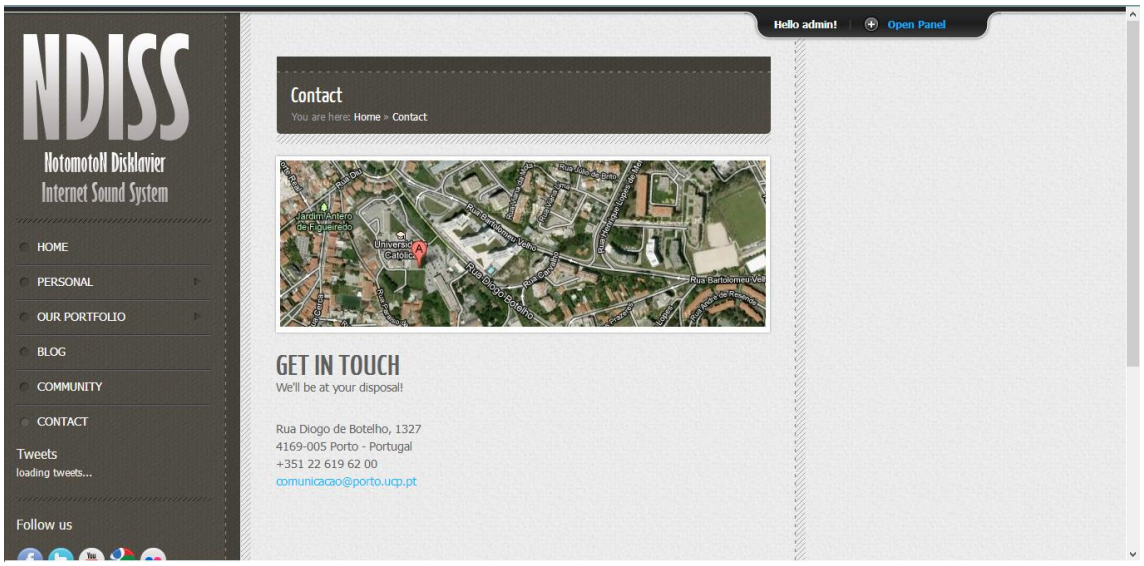
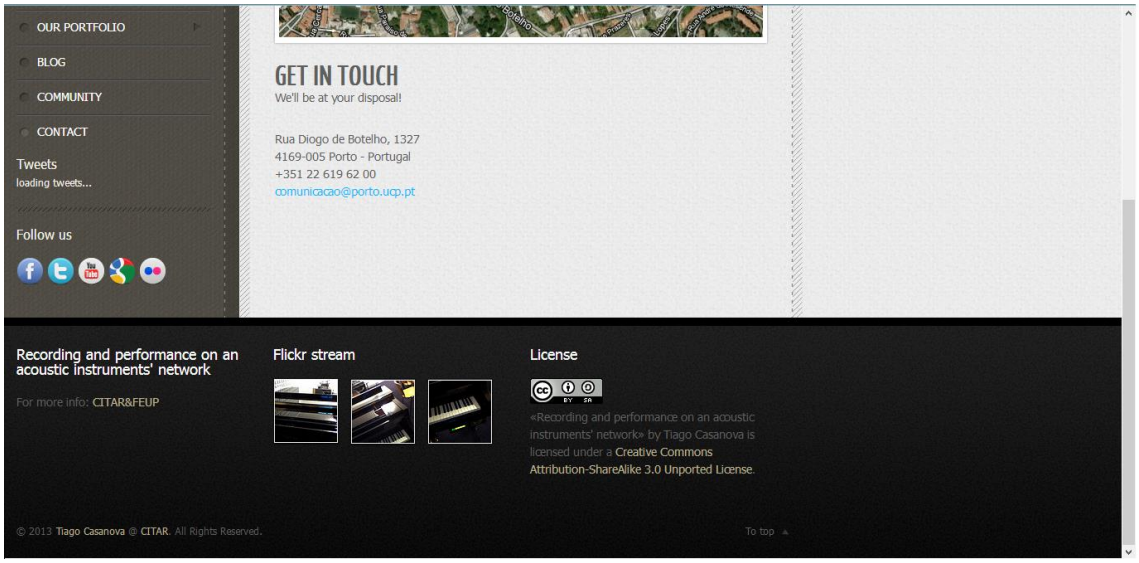


Figura 32 - *Community*

C.2 - *Contact*



C.3 – *Footer*



C.4 – *Info* Página Inicial

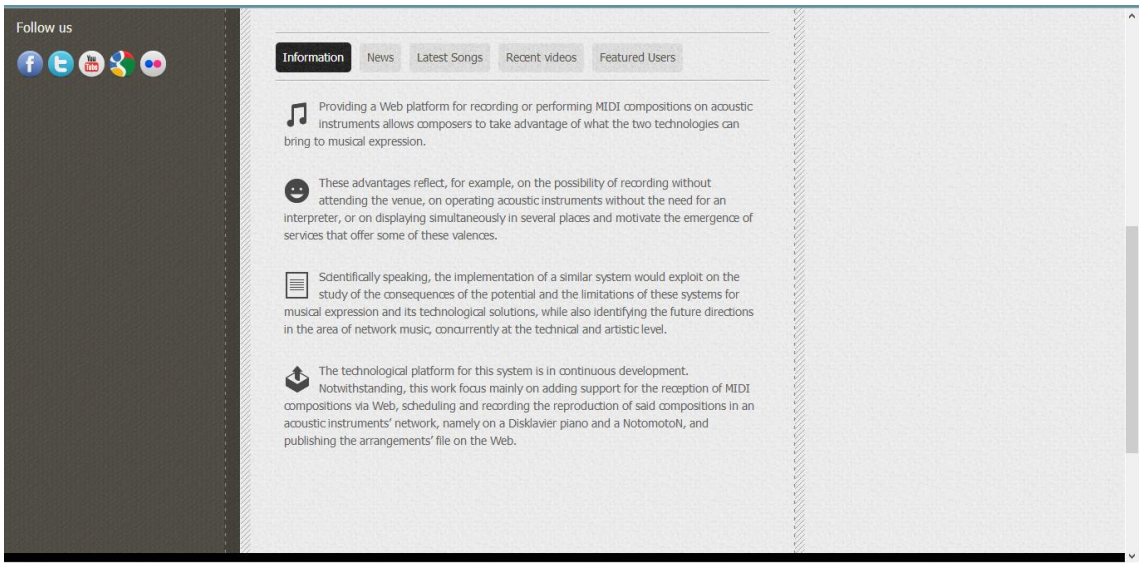


Figura 33 - *Info* Página Inicial

C.5 – *News* Página Inicial

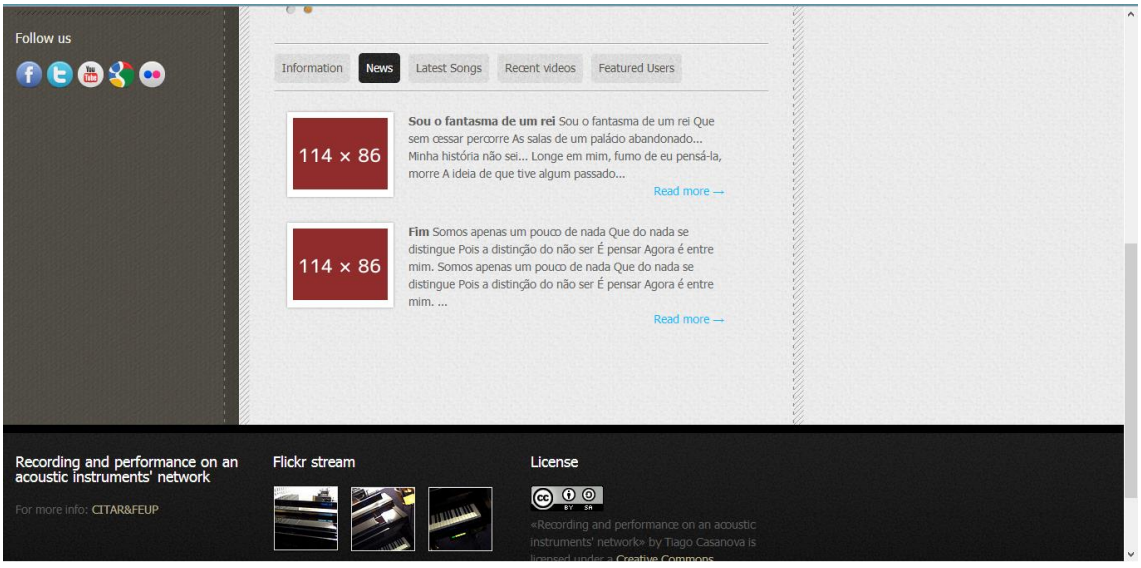


Figura 34 - *News* Página Inicial

C.6 – *Songs* Página Inicial

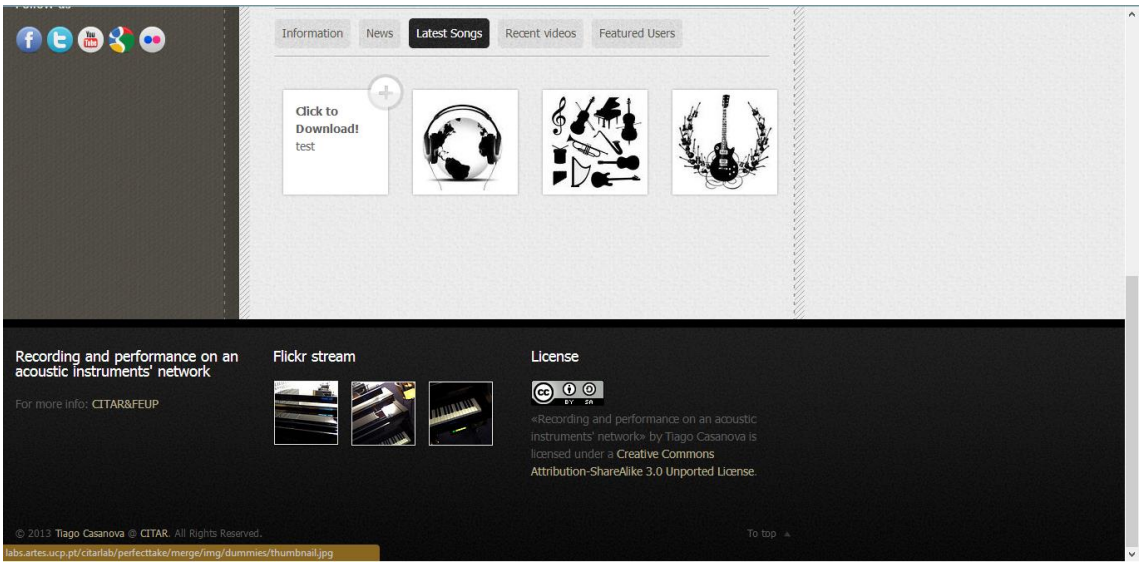


Figura 35 - *Songs* Página Inicial

C.7 – *Videos* Página Inicial

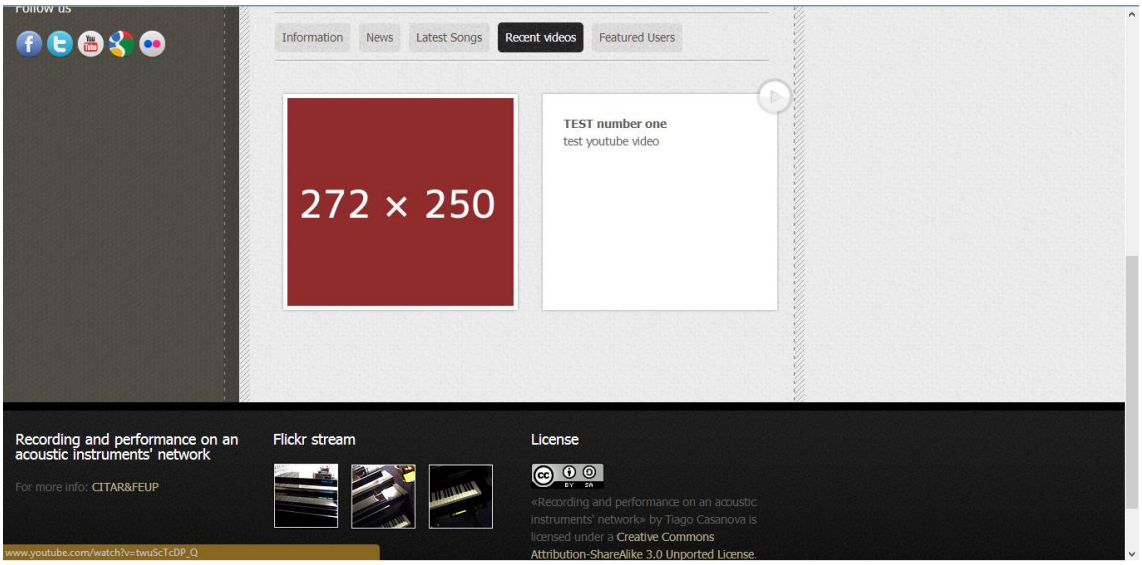


Figura 36 - *Videos* Página Inicial

C.8 – *Music Gallery*

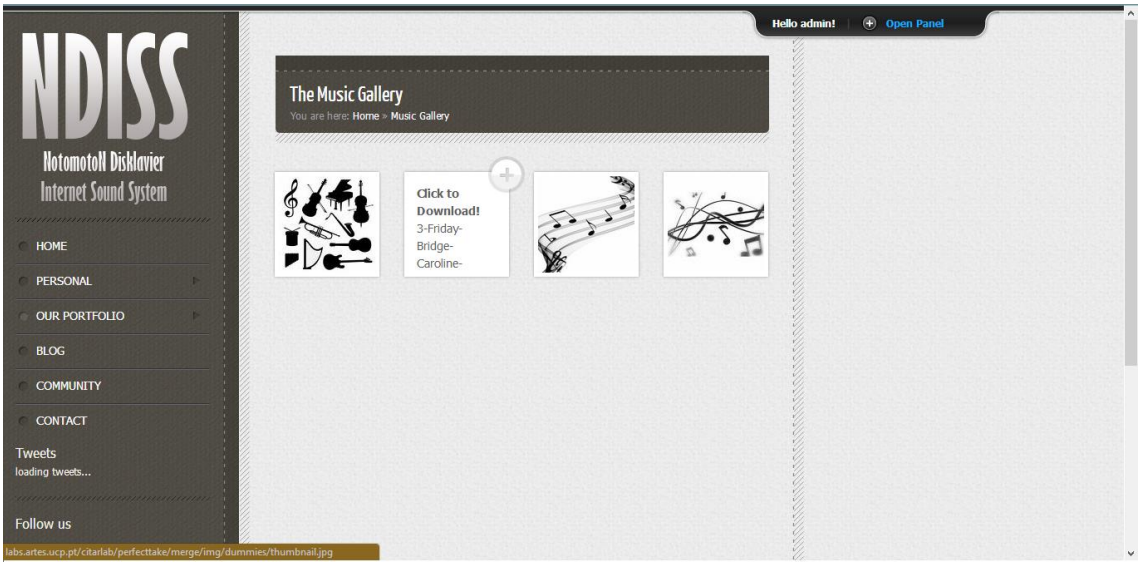


Figura 37 - *Music Gallery*

C.9 – *News Blog*

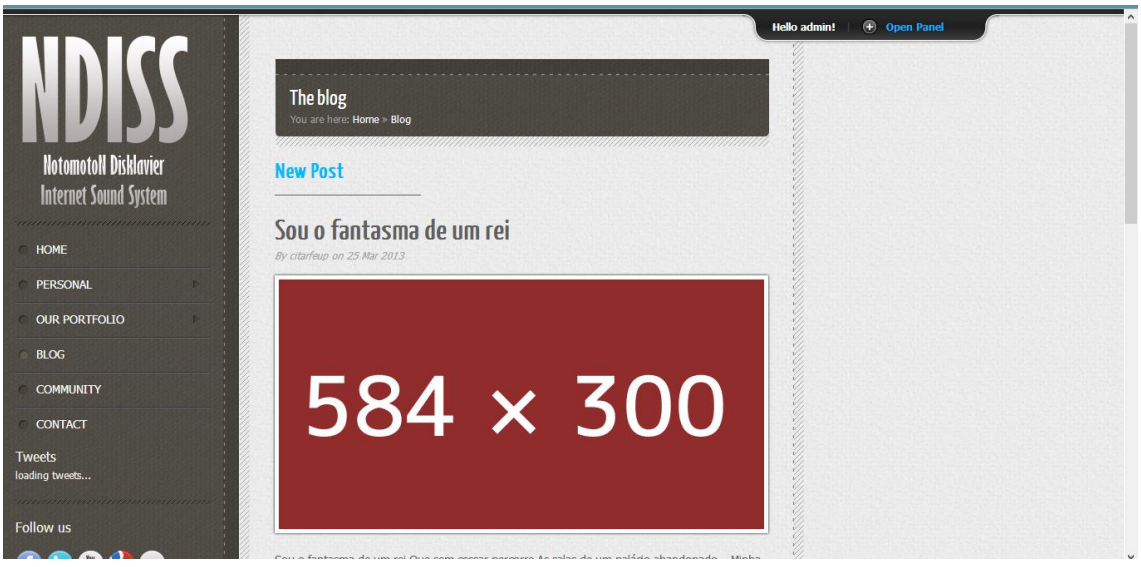


Figura 38 - *News Blog*

C.10 – *News Blog Comments*

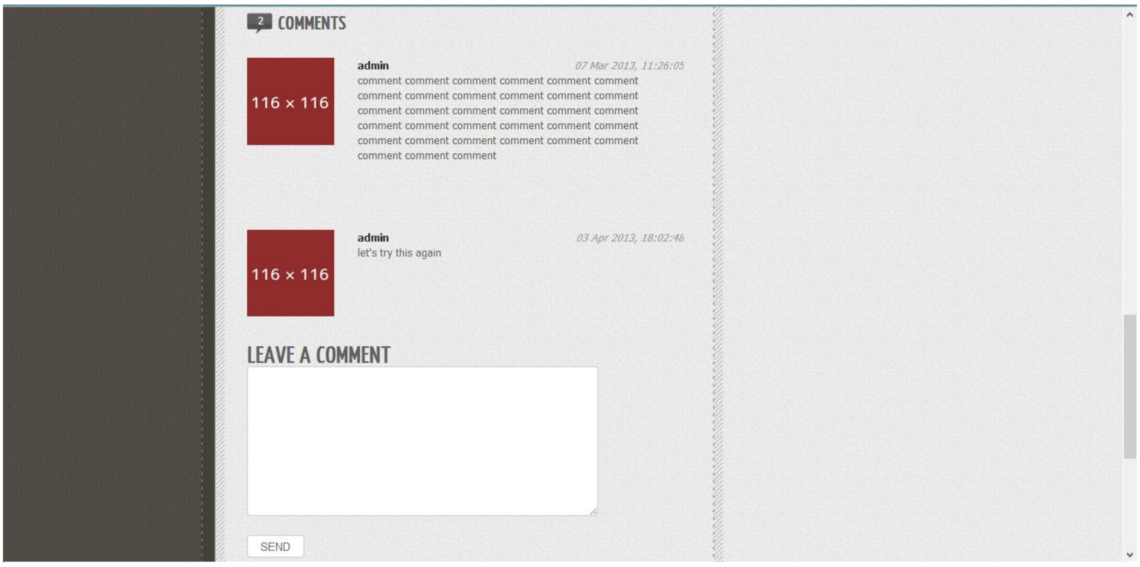


Figura 39 - *News Blog Comments*

C.11 – *News Blog New Post*

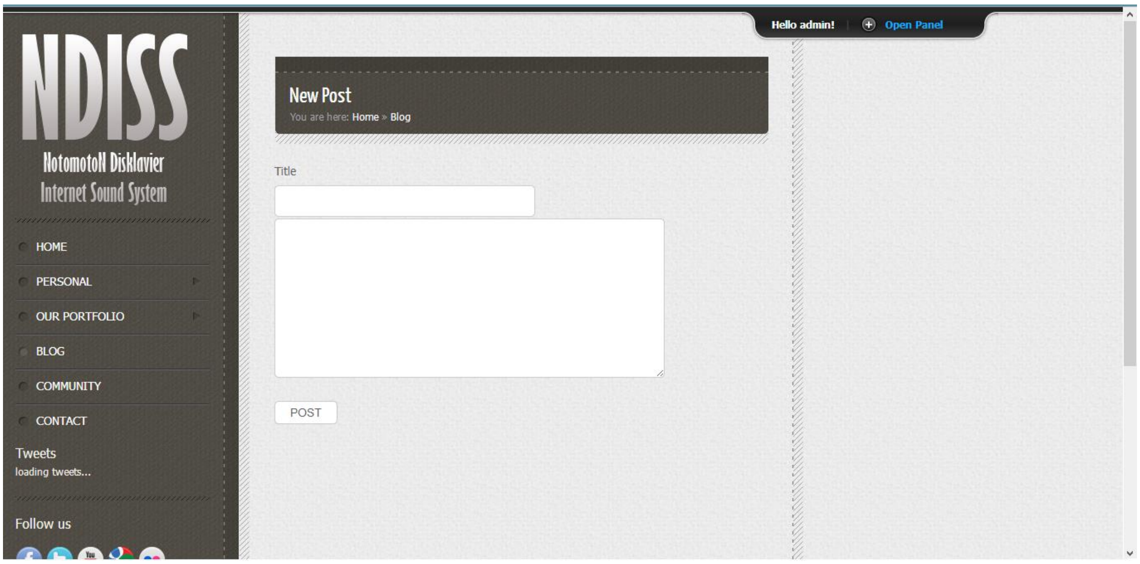


Figura 40 - *News Blog New Post*

C.12 – *Profile Details*

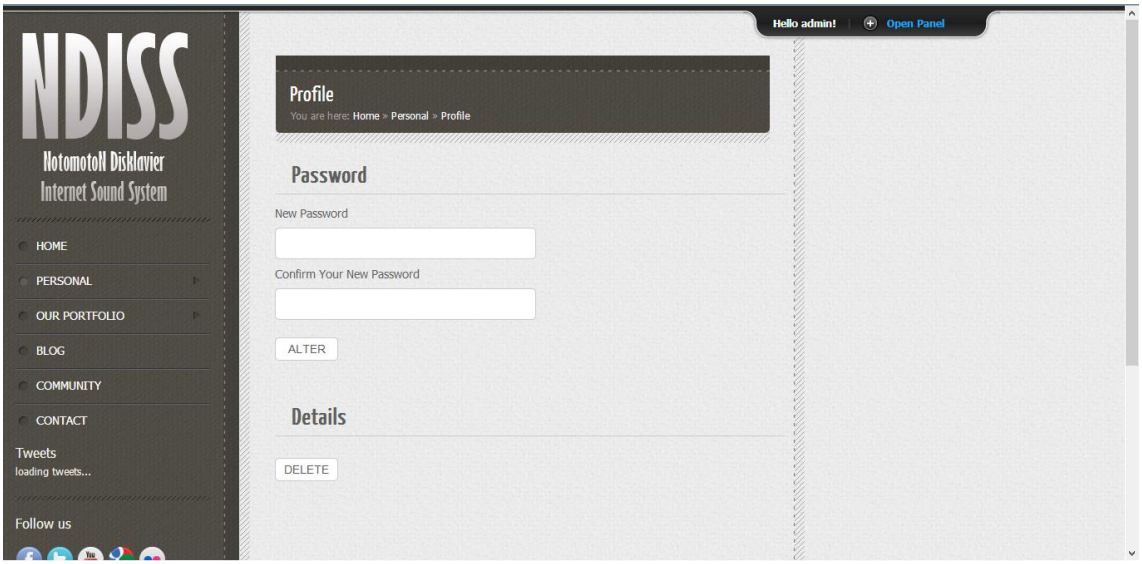


Figura 41 - *Profile Details*

C.13 – *Profile Music*

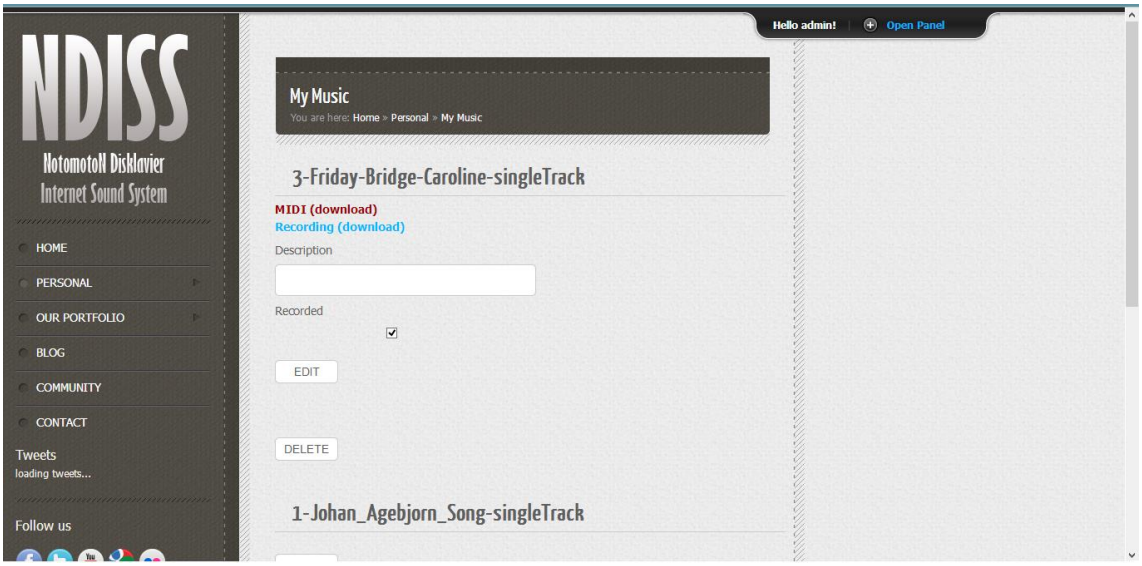


Figura 42 - *Profile Music*

C.14 – *Profile Videos*

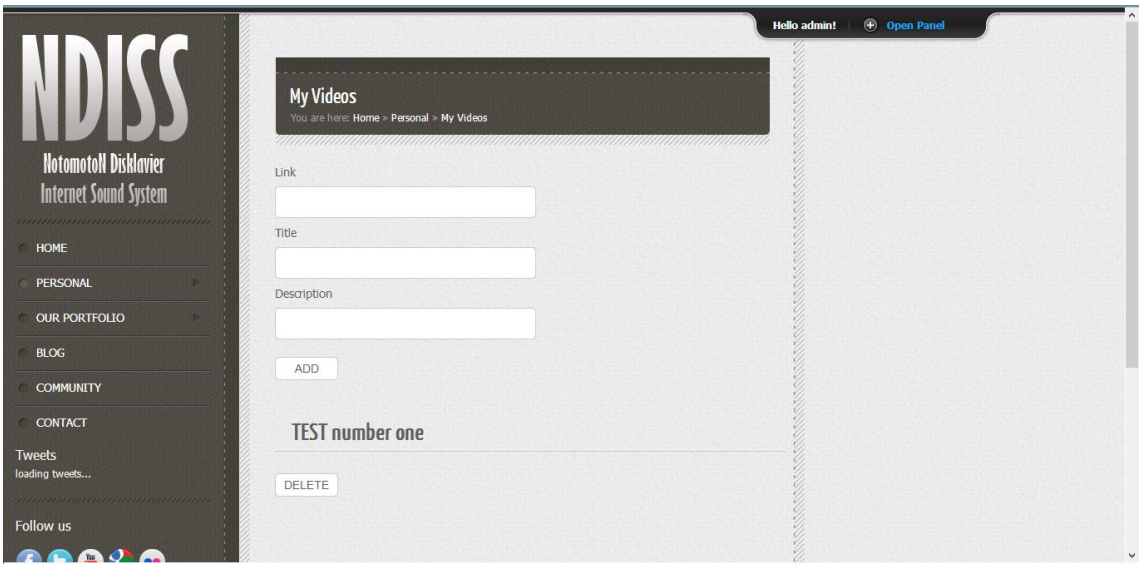


Figura 43 - *Profile Videos*

C.15 – *Register & Login*

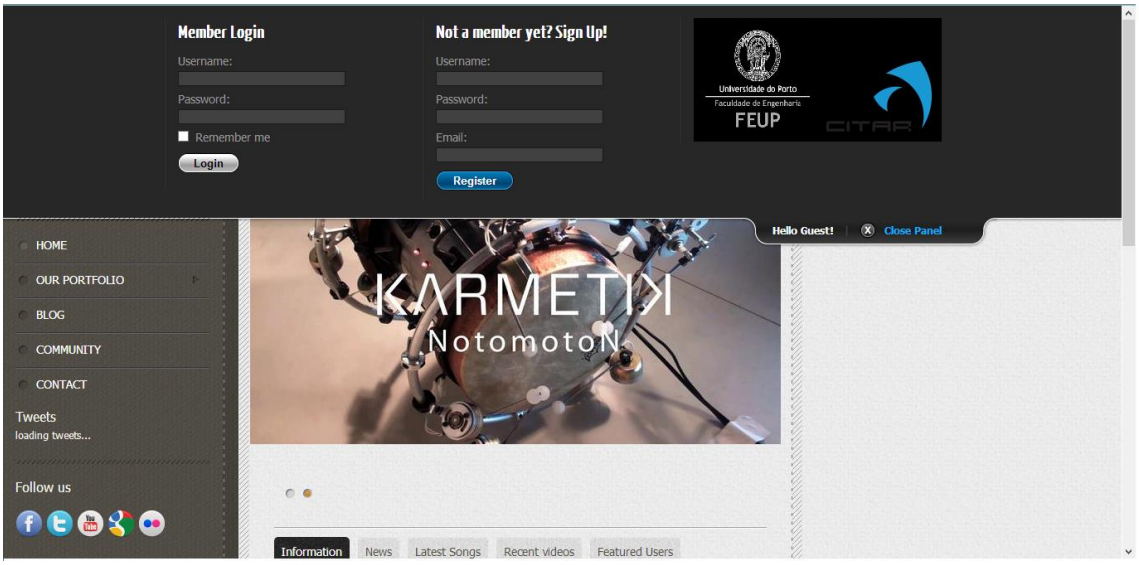


Figura 44 - *Register & Login*

C.16 – *Upload & Logoff*

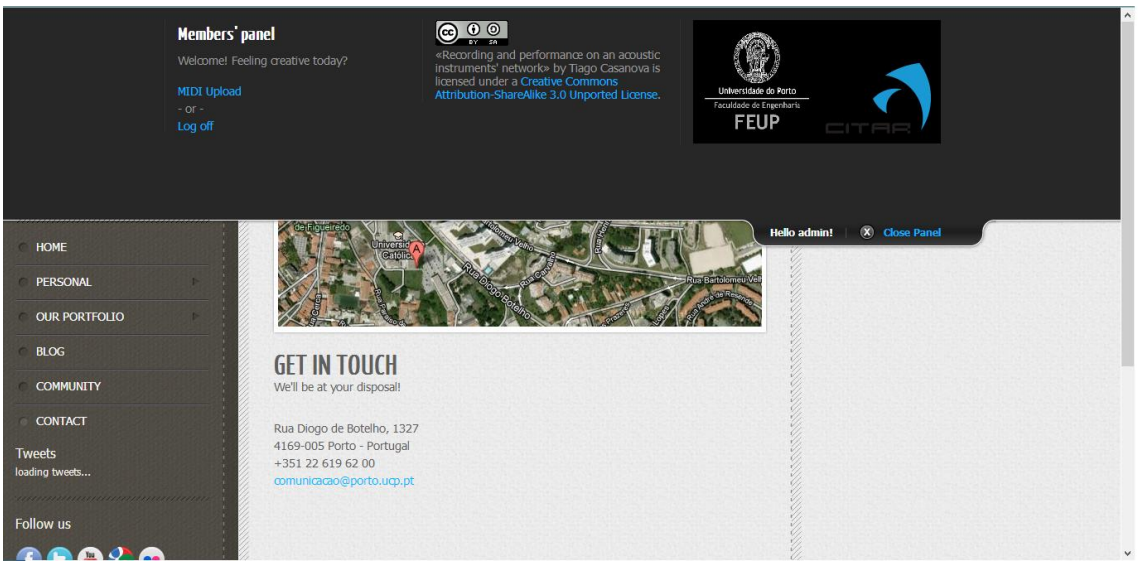


Figura 45 - *Upload & Logoff*

C.17 – *User Page*

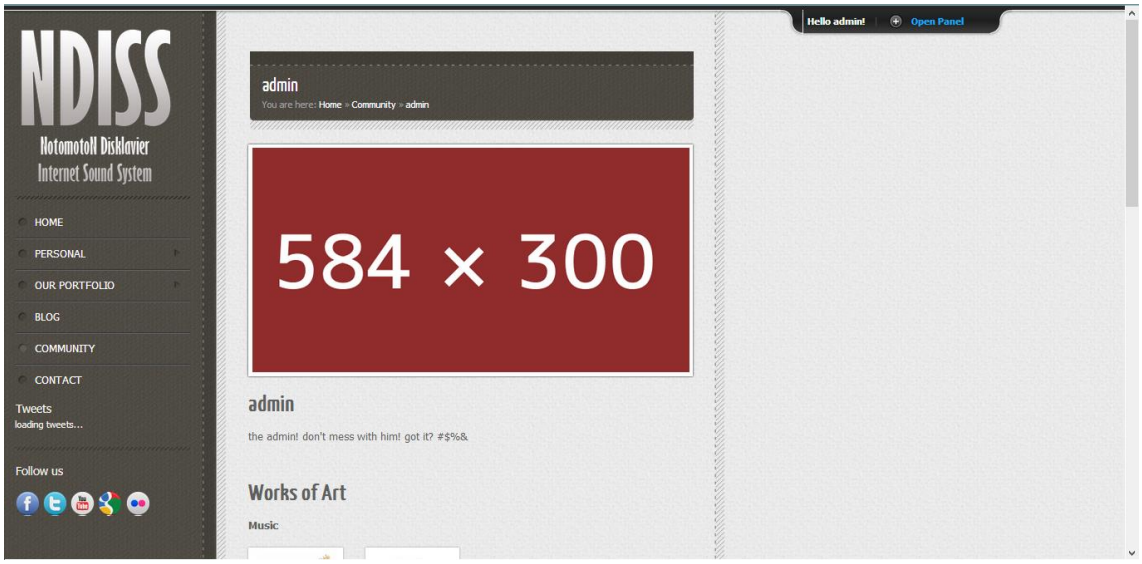


Figura 46- *User Page*

C.18 – *User Page (Music, Videos, Other Users)*

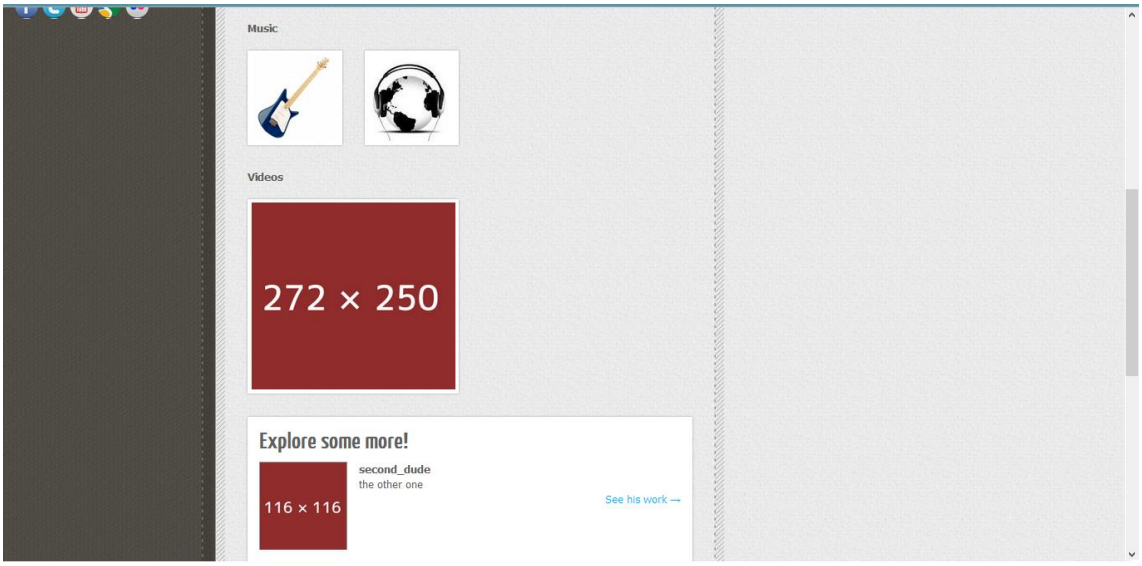


Figura 47 - *User Page (Music, Videos, Other Users)*

C.19 – *Users* Página Inicial

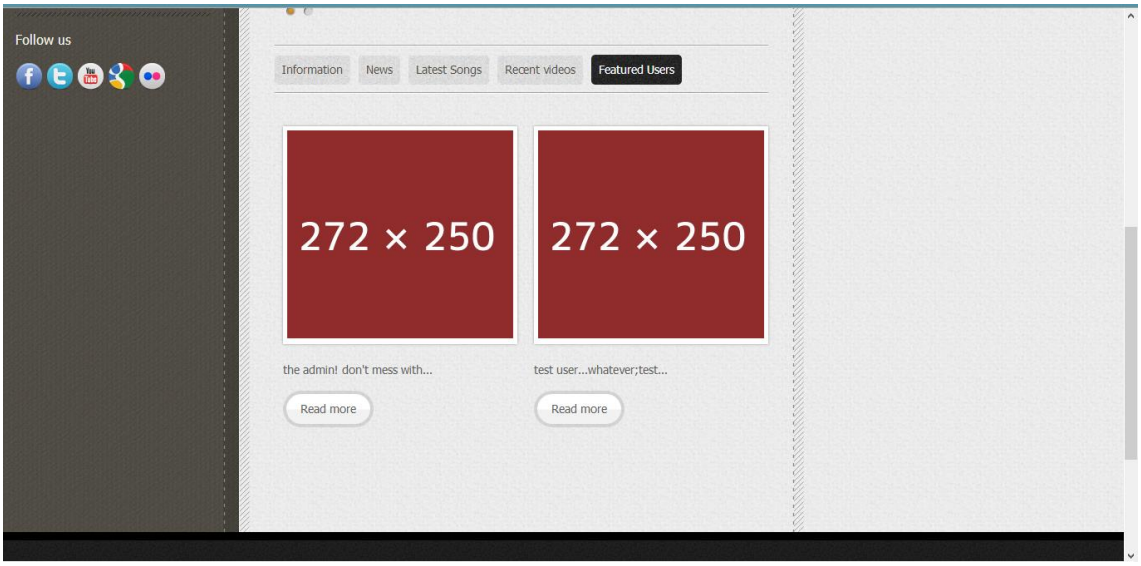


Figura 48 - *Users* Página Inicial

C.20 – *Video Player*

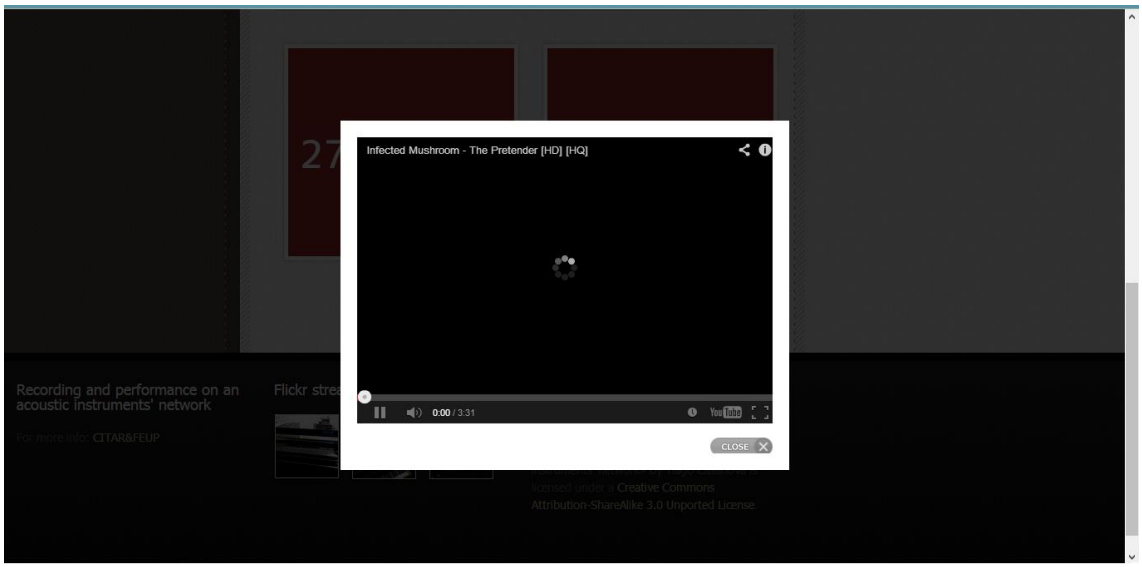


Figura 49 - *Video Player*

Anexo D - Testes de Tipos de Ficheiros e Dados

Tipos de dados	Microsoft WAV	SGI Apple AIFF AIFC	Sun DEC NeXT AU SND	Headerless RAW	Paris Audio File (PAF)	Commodore Amiga IFF SVX	Sphere Nist WAV	IRCAM SF	Creative VOC	Soundforge W64
Unsigned 8 bit PCM	✓	✓	×	✓	×	×	×	×	✓	✓
Signed 8 bit PCM	×	✓	✓	✓	✓	✓	✓	×	×	×
Signed 16 bit PCM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Signed 24 bit PCM	✓	✓	✓	✓	✓	×	✓	✓	×	✓
Signed 32 bit PCM	✓	✓	✓	✓	×	×	✓	✓	×	✓
32 bit float	✓	✓	✓	✓	×	×	×	✓	×	✓
64 bit double	✓	✓	✓	✓	×	×	×	×	×	✓
u-law encoding	✓	✓	✓	✓	×	×	✓	✓	✓	✓
A-law encoding	✓	✓	✓	✓	×	×	✓	✓	✓	✓

GNU Octave 2.0 MAT4	GNU Octave 2.1 MAT5	Portable Voice Format (PVF)	HMM Tool Kit (HTK)	Apple CAF	Sound Designer II (SD2)	Free Lossless Audio Codec (FLAC)	OGG Vorbis
×	✓	×	×	×	×	×	✓
×	×	✓	×	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓	✓	✓
×	×	×	×	✓	✓	✓	✓
✓	✓	✓	×	✓	×	×	✓
✓	✓	×	×	✓	×	×	✓
✓	✓	×	×	✓	×	×	✓
×	×	×	×	✓	×	×	×
×	×	×	×	✓	×	×	×

Anexo E - Comparação de Ambientes de Síntese de Áudio

Nome	Criador	Objetivos	Data Lançamento	Data de atualização	Licença	Interface	Orientado a Objetos	*Type system*	*Shell scripting*	MIDI	OSC	HID	VST	Anfitrião áudio	Unidade áudio	Outros	Sistemas Operativos	Código-Fonte	APIs	Outras funcionalidades
Audulus	Taylor Holliday	Síntese em tempo real, processamento de áudio, música num dispositivo móvel	2011	2013/01	proprietário	gráfica	não	-	-	sim	-	-	-	-	-	-	iOS, Mac OS X	C++, Objective-C, Lua	-	-
ChucK	Ge Wang and Perry Cook	Síntese em tempo real, codificação em tempo real, pedagogia, pesquisa acústica, composição algorítmica	2004	2012/09	GPL	documental	sim	estático	-	sim	sim	sim	-	sim	sim	-	Mac OS X, Linux,Windows	C++	-	mecanismo de tempo unificado (sem separação entre taxa de áudio e taxa de controlo), acesso por linha de comandos
SuperCollider	James McCartney	Síntese em tempo real, codificação em tempo real, composição algorítmica,pesquisa acústica, linguagem de programação generalista	1996	2012/03	GPL	documental	sim	dinâmico	sim	sim	sim	sim	-	não	sim	-	Mac OS X, Linux,Windows,FreeBSD	C, C++,Objective-C	C++	arquitetura cliente-servidor, cliente e servidor podem ser usados de forma independente, acesso por linha de comandos
Max/MSP	Miller Puckette	Síntese de vídeo + áudio em tempo real, controlo de *hardware*	1980s	15/02/2012	proprietário	gráfica	não	-	-	sim	sim	sim	sim	sim	não	ponte bidirecional Scheme para Objective-C	Mac OS X,Windows	C, Objective-C	C, Java, JavaScript, Python, Ruby	-
Csound	Barry Vercoe	atuação em tempo real, síntese de som, composição algorítmica, pesquisa acústica	1986	2011/12	LGPL	documental, gráfica para tempo real	não	-	sim	sim	sim	-	sim	-	-	-	Mac OS X, Linux,Windows	C, C++	C, Python, Java, Lisp, Lua,Tcl , C++	IDE (QuteCsound), interface multifaixas (blue), várias funcionalidades de análise e ressíntese, computação de áudio com dupla precisão, biblioteca de composição algorítmica em Python
Reaktor	Native Instruments	Síntese em tempo real, controlo de hardware, desenho de interfaces	1996	2011/09	proprietário	gráfica	não	-	-	sim	sim	-	sim	não	não	bindings for Python, Lua, Java, GStreamer	Mac OS X,Windows	-	-	-
nsound	Nick Hilton	Síntese em tempo real, renderização de áudio offline, composição algorítmica, pesquisa acústica	2003	26/06/2011	GPL	documental	sim	dinâmico	sim	não	não	não	não	sim	-	-	Mac OS X, Linux,Windows	C++	C++, Python	Filtros digitais dinâmicos em tempo real
Pure Data	Miller Puckette	Síntese em tempo real, controlo de hardware, pesquisa acústica	1990s	29/03/2011	BSD-like	gráfica	não	-	sim	sim	sim	sim	sim	não	não	C++ API, Python API	Mac OS X, Linux,Windows, iPod, Android	C	C, C++, FAUST, Haskell, Java,Lua, Python, Q, Ruby, Scheme, outros	-
Impromptu	Andrew Sorensen	codificação em tempo real, composição algorítmica, controlo de hardware, Síntese em tempo real, programação de gráficos 2D/3D	2006	2010/10	proprietário	documental	-	dinâmico & estático	-	sim	sim	-	-	-	-	Haskell (hCsound)	Mac OS X	Lisp, Objective-C, Scheme	C, C++, Objective-C, Scheme	Acesso nativo à maioria das APIs do OS X incluindo Core Image, Quartz, QuickTime e OpenGL; inclui também o seu próprio sistema de tipagem (de inferência) estático, linguagem para processamento numérico pesado - OpenGL, RT AudioDSP, etc
Usine	Olivier Sens	Audio manipulation, codificação em tempo real, composição algorítmica	2006	2010/06	proprietário	gráfica	sim (scripts)	dinâmico	sim	sim	sim	sim	sim	-	-	Multi-toque	Windows	Delphi	C++	-