Monografia

MAC0499 – Trabalho de formatura supervisionado

Endeavour
Um sistema colaborativo para compartilhamento de metadados musicais

Aluno: Nilo César Teixeira
Orientador: Prof. Dr. Marco Aurélio Gerosa
Novembro / 2009

Sumário

Prefácio

Ao longo desta disciplina, a viabilidade de implementação da idéia que baseia este projeto, surgida em 2005, foi objeto de maturação. Desde aquele momento, havia o intuito de produzir o tema em questão como um trabalho de graduação, e a motivação de conseguir apresentar um protótipo da ferramenta ao final do mesmo.

Visamos descrever nesta monografia o conceito como foi concebido em sua origem, com as decisões de projeto que foram tomadas e suas justificativas, a fim de documentar o produto final elaborado para a apresentação e também de registrar todas as funcionalidades originalmente previstas.

Esperamos continuar o desenvolvimento do mesmo ao término desta disciplina, e agradecemos as críticas construtivas e sugestões que foram apresentadas neste período.

Em tempo, agradeço à minha família pelo suporte durante este período de trabalho intenso, e aos amigos Wesley Seidel e Ricardo Bittencourt pela leitura do documento e sugestões. E, significativamente, a Deus por me permitir concluir este trabalho.

Agora, ao tema.

Nilo César Teixeira

Parte objetiva

Introdução

O compartilhamento de letras de música pela Internet é algo intrínseco à atualidade. Com a disseminação da distribuição digital de músicas (seja ela por meios legais ou não), foi quebrado o modelo antigo de comercialização musical (onde necessariamente tais letras eram disponibilizadas no encarte do meio físico), e assim uma nova necessidade de informação surgiu. Para supri-la, foram criados sites que, com o auxílio do Google, simplificaram encontrar a letra de sua canção preferida na Internet.

Porém, tal simplicidade não se mostrou tão prática quanto ter o encarte em suas mãos, ou, equivalentemente, ter a letra disponível exatamente no momento em que seu programa de execução de músicas comece a tocá-la.

Tal funcionalidade é relevante, pois geralmente temos uma lista de músicas carregadas no media player, e, ao ouvi-las em seqüência, caso desejemos ter a letra via este método, é necessário interromper a execução se antes não tivermos obtido a letra.

Assim, desenvolvedores identificaram este nicho, e surgiram os plugins para os media players mais populares, cumprindo exatamente essa função. Ao utilizá-los, deixou de ser necessário buscar manualmente a letra: O plugin encarrega-se de analisar os tags ID3 do arquivo mp3 e buscar em um ambiente online a letra correspondente registrada em seu banco de dados.

Porém, com esta solução, surgiram dois problemas:

> Primeiro: A eficácia da busca pelas letras estava limitada pela abrangência do banco de dados e pelo algoritmo de reconhecimento dos tags ID3. Caso o banco de dados não fosse abrangente o suficiente, buscas por novos artistas ou artistas menos conhecidos não retornariam nada para o usuário. Da mesma maneira, se os tags ID3 estivessem corrompidos ou mal-formulados não haveria resultados.
>
> Segundo: Ao utilizar tal solução, o usuário seria forçado a escutar sua biblioteca de música sempre no media player para o qual o plugin foi codificado.

Mesmo com estas restrições, este método de compartilhamento tornou-se extremamente popular.

Mas, e se fosse possível basear sua busca de artistas em algum repositório externo e atualizado?

E ainda, seria possível conceber um ambiente semelhante a um karaokê, em que fosse possível a qualquer usuário preparar qualquer música para exibição rítmica da letra da mesma no momento de sua execução?

Com estas motivações iniciamos o empreendimento deste projeto.

Objetivo do trabalho

Grosso modo, o trabalho realizado pôde ser dividido em duas etapas complementares: A implementação do ambiente servidor, responsável pela persistência dos artistas, álbuns, e letra em banco de dados, e pela apresentação da interface visual; e do componente no ambiente local do usuário, que seria responsável pela gravação e exibição rítmica dos dados.

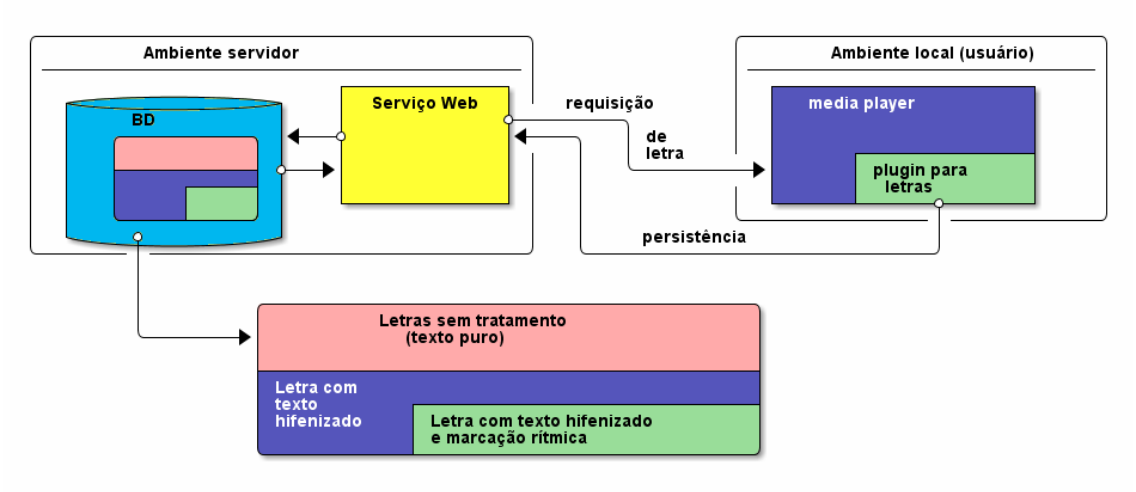

Figura 1 - Diagrama geral

Inicialmente, havia uma premissa básica: deveria ser mais simples apenas criar um plugin para marcação rítmica, aproveitando a API de um media player popular, do que codificar o ambiente local inteiro, mesmo que isso implicasse em uma instalação do plugin em cada máquina do usuário**.**

Acreditando nessa premissa, escolhemos um media player hospedeiro (Winamp), e então listamos os seguintes problemas iniciais para o ambiente local:

- Fazer a comunicação do plugin com o ambiente servidor, para a requisição de letra e para persistência;
- Renderizar a letra no media player local;
- Permitir a demarcação de sílabas de maneira simples pelo usuário;
- Exibir marcação rítmica pré-gravada no sistema.

E os problemas para o ambiente servidor:

- Coordenar a edição e persistência das letras;
- Hifenizar as letras, preparando-as para a marcação silábica;
- Disponibilizar letras com metadados rítmicos para o plugin.

Conceitos e tecnologias estudadas

Listados os problemas, o trabalho no ambiente local foi iniciado pela tentativa de renderização da letra no media player. Todos os plugins concorrentes tratavam a renderização da letra como em um bloco de notas, em texto sem formatação. Para que a exibição do karaokê fosse interessante (no mínimo marcando sílabas com cor diferente), seria necessário estudar alguma tecnologia que permitisse o desenho do texto em negrito com "outline", por exemplo (como a maioria das legendas).

Para isto, foram analisadas três diferentes tecnologias: **DirectX**, **GDI+** e **GDI** (Graphics Device Interface).

Embora potencialmente versátil, a implementação com DirectX seria desnecessariamente complexa, além da necessidade adicional de ter que garantir a instalação deste em cada máquina, possivelmente embutindo o arquivo DLL do mesmo com o plugin, o que encareceria o arquivo de instalação.

Assim, durante duas semanas foi feito um estudo comparativo das duas tecnologias restantes, assim como foi lida a documentação online disponível (Microsoft, 2009).

GDI é uma tecnologia legada, que foi atualizada na versão GDI+. Resumidamente, enquanto em GDI o elementro principal é um handler de um **device context**, ao qual são vinculados objetos de desenho, em GDI+ foi criado o objeto Graphics, reforçando o paradigma de orientação a objetos, e permitindo reutilização de elementos de desenho.

Para ilustração, o trecho importante de código GDI+ que escreve uma estrofe na tela, com "outline" (faltou a inicialização do GDI e o tratamento de mensagens):

```
/* GDI+ */

using namespace Gdiplus;

VOID OnPaint(HDC hdc) {
      Graphics graphics(hdc);
      FontFamily fontFamily(L"Arial");
      PointF      pointF(0.0f, 0.0f);
      SolidBrush  solidBrush(Color(255, 0, 0, 0));
      Pen pen(&solidBrush, 1);
      GraphicsPath path(FillModeWinding);

      graphics.SetSmoothingMode(SmoothingModeHighQuality);
      path.AddString(L"In and around the lake\nMountains come out of
the sky\nand they stand there",-1,&fontFamily, FontStyleBold, 30,
pointF, StringFormat::GenericTypographic());
      graphics.DrawPath(&pen, &path);
      SolidBrush solidWBrush(Color(255, 192, 192, 192));
      graphics.FillPath(&solidWBrush, &path);
}
```

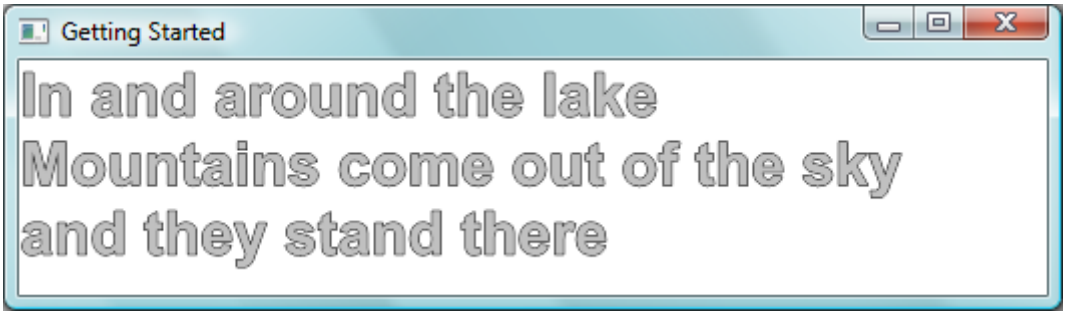

Figura 2 - Texto com outline renderizado com GDI+

O problema parecia resolvido, certo? Errado. Tínhamos que controlar também o container no Winamp em que o output do plugin seria renderizado. Em testes, ao tentarmos criar uma janela independente (Figura 2) para exibição da letra, notamos uma degradação de performance no momento de inicialização do media player: sempre alguns segundos de espera até o plugin ser reconhecido e ativado pelo Winamp.

Além disso, teríamos que fazer uma interface com a API do Winamp, cuja documentação era (e continua) bastante escassa: tínhamos apenas um arquivo compactado com os cabeçalhos das funções e definições das principais constantes disponibilizadas pela API.

Não obstante, ainda havia o problema de acoplar ao plugin a fonte tipográfica utilizada (para permitir que usuários conseguissem visualizar o texto caso não tivessem a fonte instalada – o que não estava claro se funcionaria), e, principalmente, o vínculo obrigatório com o Winamp (e com Windows, por conseqüência).

Em resumo, perdeu-se muito tempo tentando fazer código para aproveitar o fato de que o componente no ambiente local deveria se utilizar de um media player existente.

Foi então que uma idéia que parecia inocente começou a ganhar força: **Implementar todo o ambiente local em Flash**.

Com isto, quebrava-se o vínculo obrigatório com um determinado sistema operacional ou media player: garantiríamos que a partir de qualquer browser o ambiente local pudesse ser executado.

Assim, a tempo de desenvolver o protótipo para a apresentação, comecei a estudar a documentação da Adobe que acompanha o IDE.

ActionScript 3.0 (Flash 10)

Baseado no padrão internacional ECMAScript (ECMA-262), ActionScript é a linguagem de programação que permite controle sobre os elementos de uma animação Flash (um arquivo SWF). Porém, em sua versão 3.0, ela apresenta elementos sintáticos e características de geração de código binário que a distanciam de suas "irmãs" (como Javascript) e de suas próprias versões anteriores. Doravante, denominaremos essa versão como AS3.

No AS3, o código é compilado em bytecode para a versão 2.0 da AVM (ActionScript Virtual Machine), e foram adicionadas construções sintáticas de maneira a permitir orientação a objetos e melhor reutilização de código. (Grossman & Huang, 2009)

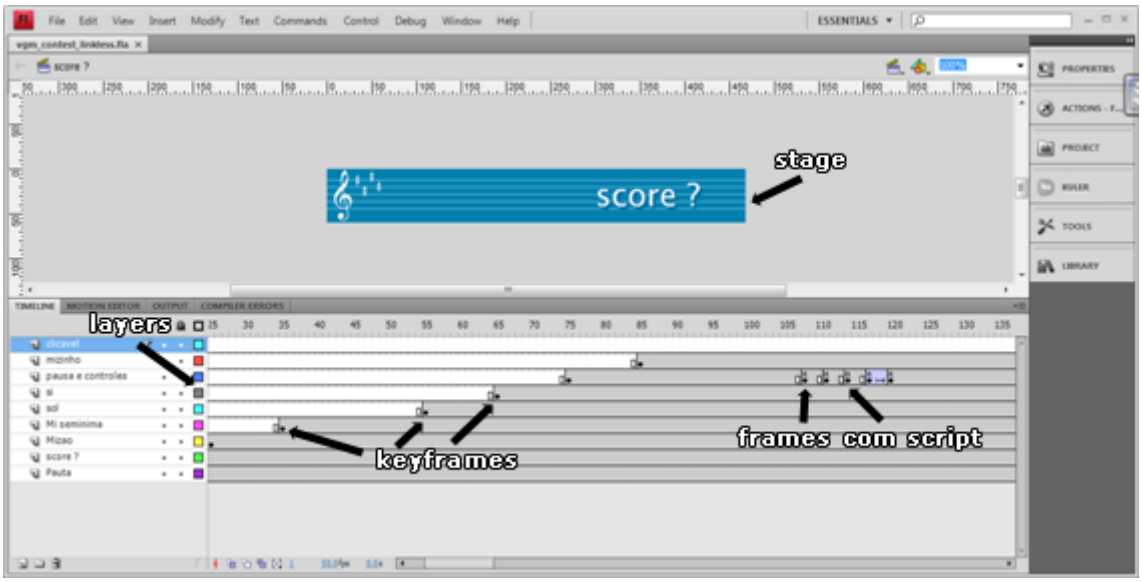

Figura 3 - Organização comum em projetos antigos

Como o modelo anterior não reforçava esses aspectos, era comum instanciar os elementos gráficos diretamente no stage, e "diluir" código ao longo de frames conforme necessário (Figura 3). Ainda, **todas** as animações gerenciadas pelo Flash ("tweens") necessariamente eram definidas na IDE e embutidas no timeline, sem possibilidade de reutilização. Isto mudou radicalmente.

No Flash 10, não é obrigatório instanciar sequer um único elemento gráfico pela IDE, tudo pode ser resolvido com arquivos AS3 externos (*.as). Ainda, alia-se o potencial da IDE para projetar layouts e "tweens" à versatilidade de exportar tais definições automaticamente para um script externo.

Outras características do AS3 utilizadas no projeto:

- Standard mode vs. Strict mode:

  O modo standard se assemelha à programação funcional (Javascript), onde podem ser definidas funções anônimas e não há checagem de tipos. Já o strict mode garante checagem de tipos em tempo de compilação e execução. Optamos pelo strict mode (padrão no AS3). Mesmo no strict mode, é possível definir variáveis não tipadas e delegar a checagem para tempo de execução;

```
//Exemplo de função anônima
var funcao:Function = function (nome:*) {
      //sintaxe parametro:tipo (Asterisco denota variável não
tipada)
      trace(nome);
};
```

- Packages:

  Packages em ActionScript são semelhantes aos de outras linguagens, como Java. Porém, em AS3, apenas uma classe dentro de um package pode ter o modificador **public**. Todas as demais devem ser declaradas sem modificador ou com o modificador **internal** (que é o modificador padrão), de modo que elas não são visíveis a código fora do arquivo AS3 em que são definidas. O código foi completamente estruturado em packages;

- Eventos:

  No AS3, além de escrever código que trata o acontecimento de eventos, é possível definir o seu próprio tipo de evento. Cada evento é representado por um **objeto-evento**, que é criado e "disparado" (dispatched) para objeto-alvo do evento (event target). É possível "escutar" objetos-eventos utilizando **event listeners**. Event listeners são os métodos ou funções que são escritos para responder a eventos específicos. Para garantir que o aplicativo responda a eventos, é necessário adicionar event listeners no objeto-alvo.

```
function eventResponse(eventObject:EventType):void {
    //Código que trata o evento
}

eventTarget.addEventListener(EventType.EVENT_NAME,
eventResponse);
```

  Este código cumpre dois papéis: Primeiramente, define uma função, que é o modo de **especificar as ações** que serão executadas **em resposta ao evento**. Segundo, **registra a função no objeto-alvo** de modo que, quando o evento acontece, as ações da função são executadas.

  Para definir eventos próprios, o procedimento é semelhante: apenas difere no "disparo" do evento, que deve ser explicitamente invocado no código;

```
package Endeavour.events {
      public class CarregaLetraEvent {
            public static const CONCLUIDO:String = "concluido";
      }
}

/* fase de registro do event listener */
doc.addEventListener(CarregaLetraEvent.CONCLUIDO,carregaLetra);

/* doc é o objeto-alvo, e o disparo explícito sinaliza
ocorrência do evento */
doc.dispatchEvent(new Event(CarregaLetraEvent.CONCLUIDO));
```

- Exceptions:

  Apesar de não definir a cláusula **throws**, forçando a captura manual de erros com blocos **try/catch**, a hierarquia de classes de exceções do AS3 permite subclassing, permitindo tratamento de exceções no projeto, descrito na próxima seção.

E alguns features interessantes:

- **Métodos get e set (açúcar sintático):** Utilizando os tokens get e set na assinatura de um método, podemos **simular propriedades.** Assim, torna-se possível atribuir e ler valores de uma propriedade com este nome em um objeto!

```
/*
A classe exemplo abaixo, GetSet, inclue métodos com
modificadores "get" e "set", que provêem acesso à variável
privada chamada privateProperty
*/

class GetSet {
    private var privateProperty:String;
    public function get publicAccess():String {
        return privateProperty;
    }
    public function set publicAccess(setValue:String):void {
        privateProperty = setValue;
    }
}

/*
Se tentarmos acessar a propriedade diretamente, um erro irá
ocorrer
*/

var myGetSet:GetSet = new GetSet();
trace(myGetSet.privateProperty); //erro

/*
Ao invés disto, os usuários da classe GetSet irão utilizar algo
que "aparenta" ser uma propriedade chamada publicAccess, mas que
realmente é um par de funções _de mesmo nome_ com modificadores
"get" e "set" que operam na propriedade privada chamada
privateProperty.
*/

var myGetSet:GetSet = new GetSet();
trace(myGetSet.publicAccess); // output: null
myGetSet.publicAccess = "hello";
trace(myGetSet.publicAccess); // output: hello
```

  Tais modificadores também tornam possível **sobrescrever (to override)** propriedades desta natureza que são herdadas de uma superclasse, algo que não é possível ao utilizar propriedades convencionais.

- **Hoisting**: Em algumas linguagens de programação, tais como C++ e Java, variáveis declaradas dentro de um bloco de código (delimitado por chaves) não estão disponíveis fora do mesmo. Esta restrição de escopo é chamada block-level scope, e não existe no ActionScript (Adobe, 2009).

Assim, torna-se possível escrever ou ler de uma variável **antes** de sua declaração, contanto que ela seja definida até o término da função. Isto devido a uma técnica chamada **hoisting**, que significa que o compilador move todas as declarações de variáveis para o topo da função. O compilador, no entanto, não move nenhuma declaração de atribuição. Ainda, hoisting não funciona para funções anônimas, que estão disponíveis apenas após a sua definição.

```
/*
A variável num está disponível antes de sua declaração, porém
neste momento não contém o valor 10, pois foi feito o hoisting
da declaração e não da atribuição.

O valor padrão para variáveis do tipo Number é a constante
"NaN", e não "undefined" (padrão em Javascript).
*/

trace(num); // NaN
var num:Number = 10;
trace(num); // 10
```

Atividades realizadas

Durante o primeiro semestre de 2009, ao longo da disciplina MAC0416/5855 - Tópicos Especiais em Desenvolvimento para Web, foi implementado o ambiente servidor com o repositório compartilhado de letras a hifenizar. Ainda não se tinha tomado a decisão de implementar um media player próprio, nem qual algoritmo seria utilizado para a hifenização. Descreveremos o ambiente servidor na seção **Resultados e produtos obtidos**.

Implementação do media player

Assim, nos focamos na criação do ambiente local.

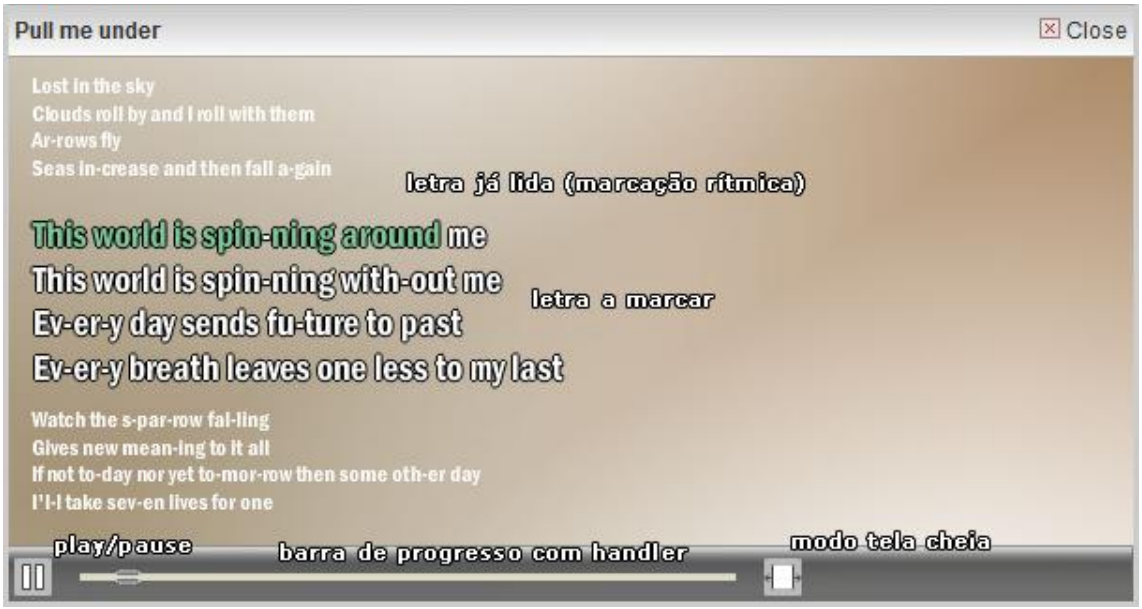

Figura 4 - Media player (hints são ilustrativos)

Conforme dito anteriormente, todo o código foi distribuído em arquivos AS3 externos, após termos definido o layout acima na IDE. Assim, no arquivo Flash final, foi necessário utilizar apenas o **primeiro frame**, inicializado com o seguinte código:

```
import flash.display.*;
import Endeavour.ui.*;

/* Definição widescreen para modo tela cheia */
stage.scaleMode = StageScaleMode.SHOW_ALL;
stage.align = StageAlign.RIGHT;

/* Cerne do projeto, aqui passamos o display object container (this)*/
var barraControle:BarraControle = new BarraControle(this);

/* Paramos a exibição do arquivo .swf */
this.stop();
```

O conceito de Display Object Container (presente no código acima) é importante: resumidamente, todo elemento visual que é desenhado na tela denomina-se **display object**, e para que seja renderizado necessariamente deve estar em um container que foi anexado ao **stage** ou a um container já anexado à árvore de exibição (**display list**). Assim, é necessário passar a referência do container para o objeto responsável pela criação da interface, que irá chamar o método **addChild()** para inserir elementos visuais na tela. Todos os componentes visuais do media player foram inseridos no stage desta maneira.

Um feature **essencial** foi a possibilidade de manter um arquivo externo AS3 para classes correspondentes a símbolos exportados da biblioteca do projeto. Isto permitiu complementar o comportamento dos mesmos em código (arquivo .as com o nome da classe exportada).

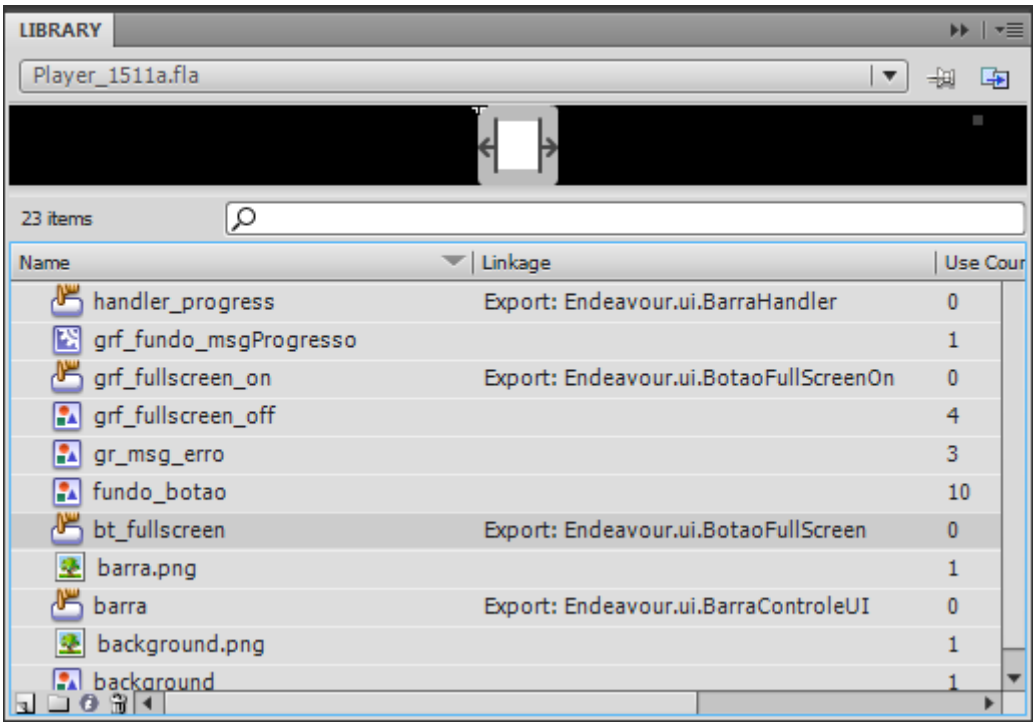

Figura 5 – Library do projeto com classes dos símbolos exportadas para AS3

Já a renderização da letra utilizou a possibilidade de atribuir **filtros** a um objeto do tipo **TextField**. A preocupação adicional de **embutir fontes tipográficas** no projeto foi relativamente fácil de resolver: o Flash 10 permite extrair os glifos da fonte e acoplá-los ao arquivo .swf. Ou seja, tornou-se desnecessário ter a fonte instalada!

A legibilidade do texto também pôde ser granularmente ajustada: A classe TextField permite "antialiasing" configurável (através das propriedades **sharpness** e **thickness**), que notavelmente no modo fullscreen conferiu excelente legibilidade ao texto.

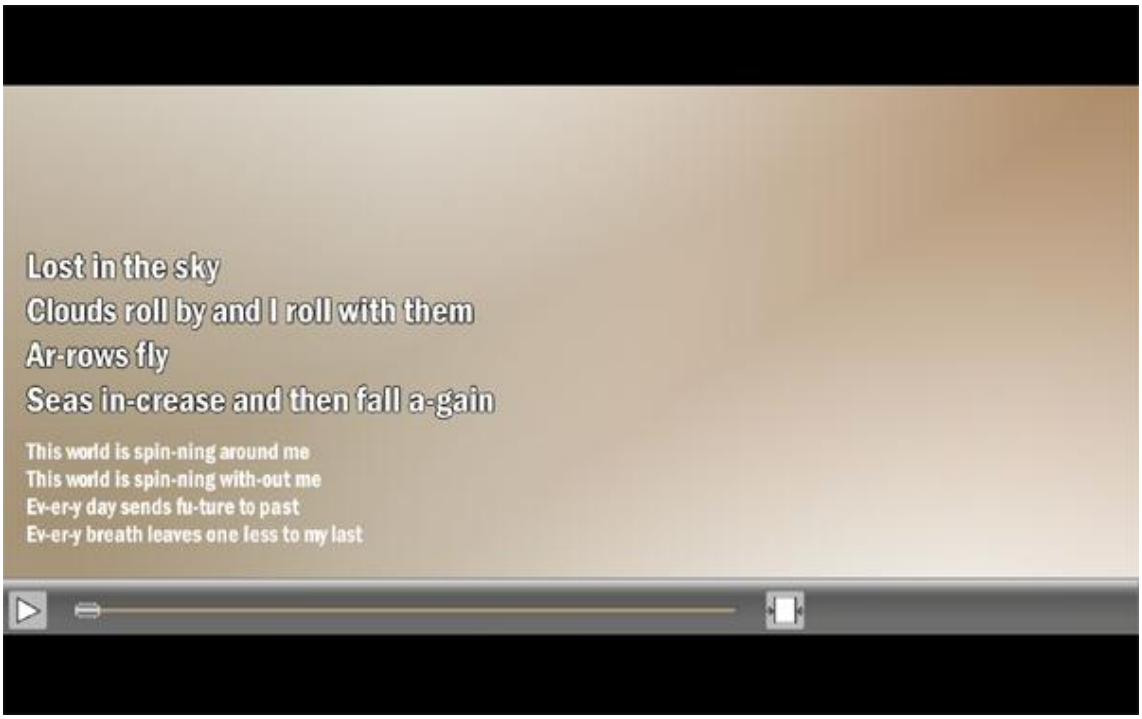

Figura 6 - Modo fullscreen (com barras widescreen)

Tratamento de exceções

Foi criada uma classe que estende **Error**, denominada **VisualException**. Esta classe define uma maneira centralizada de exibir todas as mensagens de erro no player, através de um banner na parte superior que surge e depois desliza para a direita, após um certo tempo da exibição da mensagem.

```
package Endeavour.exceptions {
      /* imports omitidos */
      public class VisualException extends Error {
            public function
VisualException(doc:DisplayObjectContainer,msg:String) {
                  var mc:MovieClip = new BannerMsgErro();
                  mc.x = 0;
                  mc.y = 0;
                  mc.msgErro.text = "ERRO: " + msg;
                  doc.addChild(mc);
                  mc.play();
            }
      }
}
```

Figura 7 - Banner de mensagem de erro

Carregamento de uma letra

Implementamos esta funcionalidade a partir do ambiente servidor estendendo a classe **URLLoader**.

```
package Endeavour {
/* Declarações de imports, exceptions e constantes omitidos */
public class PegaLetra extends URLLoader {
      public var dados:Object;
      public function PegaLetra(doc: DisplayObjectContainer,
codmusica:int = 0) {
            this.dataFormat = URLLoaderDataFormat.TEXT;
            this.addEventListener(Event.COMPLETE, fimCarregamento);
            this.load(new URLRequest(HOST + ":" + PORTA + QUERYSTRING +
codmusica));
      }
      private function fimCarregamento(e:Event) {
            dados = JSON.decode(this.data);
            doc.dispatchEvent(new Event(CarregaLetraEvent.CONCLUIDO));
      }
}
}
```

Note que a chamada retorna um objeto no formato JSON. Decodificamos os dados no ambiente Flash utilizando a biblioteca as3corelib (Cantrell, Chambers, Dura, & Schall, 2008).

Carregamento do arquivo MP3

No Flash 10, duas classes foram criadas para lidar com arquivos de áudio: **Sound** e **SoundChannel**. A classe Sound gerencia a execução do arquivo, enquanto a SoundChannel oferece, entre outros dados, a posição atual de execução em milissegundos (**SoudChannel.position**).

Foi utilizada uma biblioteca de áudio (Martin-Sperry, 2008) para o carregamento do MP3 a partir da máquina do usuário. Tal carregamento é instantâneo, já que o arquivo SWF é executado no mesmo ambiente do arquivo MP3.

Sincronização da letra com a música

Partindo da premissa que tínhamos o arquivo texto com as sílabas delimitadas, e o media player funcionando, teríamos que arranjar um jeito conveniente de sincronização.

Aqui entra a idéia original:

"The syllable is a very important unit. Most people seem to believe that, even if they cannot define what a syllable is, they can count how many syllables there are in a given word or sentence. If they are asked to do this, ***they often tap their finger as they count****, which illustrates the syllable's importance in the rhythm of speech." (Roach, 2007, p. 70)*

Assim, por que não usar o teclado para sincronização? O usuário, ao ouvir a música sendo executada, "batuca" no teclado marcando o ritmo. A cada tecla pressionada (escolhemos as setas e a barra de espaço), o media player grava, além do texto na tela, os momentos do **key down (inicial)** e **key up (final)**, em milissegundos, e avança uma sílaba. As informações são armazenadas em um vetor.

Caso o usuário retorne a execução para um momento anterior no arquivo (pela barra de controle), no momento em que a nova posição atual é determinada, faz-se uma varredura no vetor de posições para verificar a partir de qual célula devemos retomar a execução. Tal verificação é eficiente (uma busca binária é suficiente, pois os tempos de início e fim são estritamente crescentes ao longo do vetor).

Quando o usuário ativa o evento **Key down** novamente (indicando portanto que deseja regravar), invalida-se o vetor da posição atual para a frente, e retoma-se a gravação como descrito acima.

Quanto à execução, a idéia teórica seria: a cada novo frame carregado (e portanto estamos falando de código num event listener para Event.ENTER_FRAME), verifica-se a posição atual do arquivo executado. Tal posição está entre os instantes inicial e final de exatamente uma posição do vetor. Novamente buscamos esta posição, e carregamos o texto armazenado nessa posição (que contém a formatação de parte das sílabas marcadas e parte a marcar).

Infelizemente não houve tempo de validar este algoritmo no protótipo apresentado.

Resultados e produtos obtidos

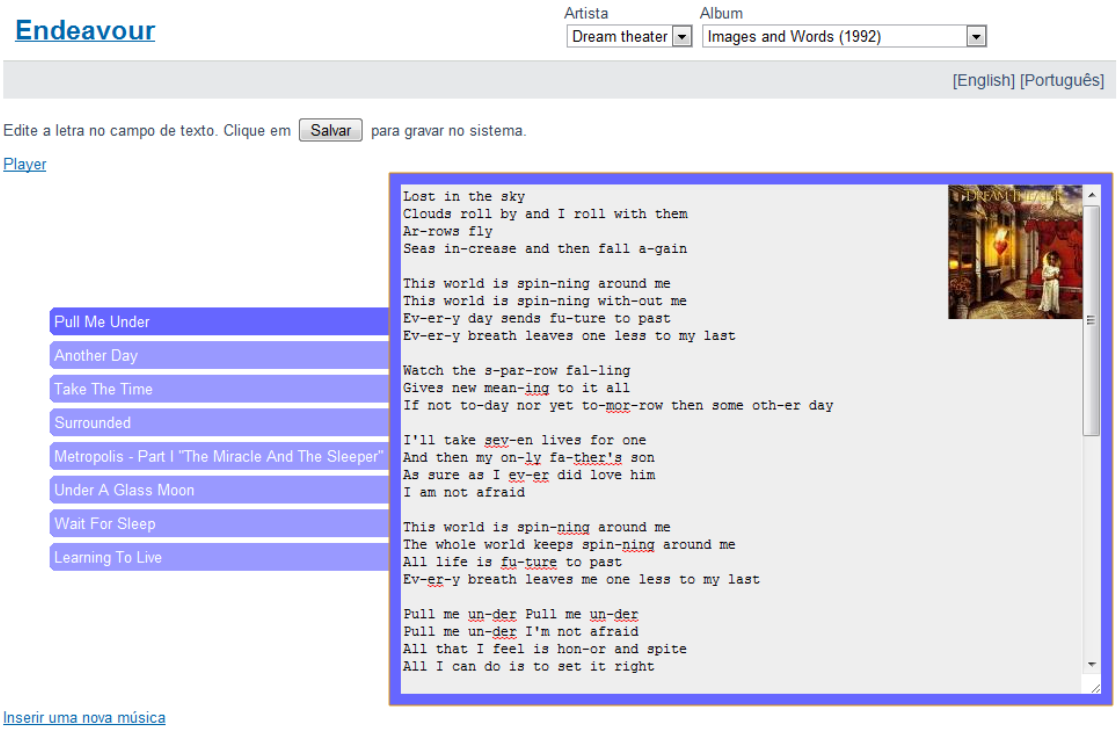

Figura 8 - Ambiente servidor

O ambiente servidor (finalizado no 1º sem/2009) foi implementado utilizando Java para Web. A interface web utilizou JQuery para algumas animações, e, **principalmente**, para simplificar a utilização de AJAX e Json através de solicitações assíncronas ao servidor em busca dos álbuns, músicas e letras. Isto permitiu que a tela de edição de letras não obrigasse o usuário a recarregar toda a página para atualizar as informações.

A camada de persistência foi implementada utilizando Hibernate. Assim, não foi escrito código SQL sequer para a criação das tabelas.

Foi utilizado o padrão MVC (Model View Controller), através do arcabouço Struts 1.3.

No protótipo que acompanha este trabalho, também constam os testes unitários e de aceitação que foram feitos para este ambiente.

Já o ambiente local foi descrito nas seções anteriores.

Conclusões

O esforço empregado neste empreendimento (e portanto daí o “working title” do projeto) visou suprir uma necessidade identificada ao longo de muito tempo ao trabalhar e ao mesmo tempo escutar músicas no computador. É uma tentativa de renovar este paradigma e ao mesmo tempo uma implementação “proof-of-concept”. Embora nem todas as funcionalidades tenham sido implementadas a tempo, acredito que o trabalho tenha sido satisfatório.

Finalmente, uma versão funcional deste projeto pode ser encontrada no endereço onde foi disponibilizada a monografia.

Parte subjetiva

Desafios e frustrações encontrados

Certamente, o tempo despendido na tentativa de codificar o ambiente local em C++ (para o plugin do Winamp) poderia ter sido melhor aproveitado. Ao conversar com o prof. Marco no início do projeto, havia sido sugerida por ele a implementação em Java caso o desenvolvimento "emperrasse". Mas havia dificuldade para fazer a ponte entre um programa escrito em Java e os cabeçalhos da API em C++. Portanto, um certo tempo de maturação foi necessário até que surgiu a idéia do media player em Flash.

Outra frustração foi não conseguir um algoritmo eficaz de hifenização. Foi adotado um algoritmo em Java que baseia-se nas definições de hifenização do LaTeX, porém na implementação utilizada houve inúmeros erros crassos (como a separação silábica incorreta de **you**, entre outras palavras). Isto atrapalhou, entre outras coisas, a apresentação do protótipo.

Disciplinas cursadas mais relevantes para o trabalho

As disciplinas de laboratório certamente foram as que contribuíram mais diretamente neste projeto (Laboratório de programação I e II, e Tópicos de desenvolvimento para a web).

As disciplinas introdutórias, como MAC-110 e MAC-122, ajudaram nos conceitos básicos de programação em Java, e também diretamente no algoritmo de busca binária.

Trabalhos futuros neste tema

Os seguintes temas faziam parte da idéia original e serão possivelmente implementados numa versão futura:

- **Pesquisa ao cloud da Amazon por artista/album/musica:** A consulta a este repositório externo, conforme dito no início deste documento, permitiria ao ambiente servidor estar sempre atualizado com os lançamentos musicais. Isto não implicaria no carregamento automático de novas letras, mas tornaria a interface mais convidativa ao usuário para que este pudesse informar tais letras;
- **Playback rítmico:** Como dito na seção de atividades realizadas, o algoritmo de execução (playback) não foi implementado por falta de tempo hábil;
- **Persistência dos metadados rítmicos:** Os dados gravados hoje não estão sendo persistidos no banco de dados, mas considerando que o diálogo via Json foi estabelecido, isto é relativamente simples de conseguir;
- **Implementar playlist de músicas:** Isto assemelharia o media player aos atuais, onde se pode escolher um diretório de músicas e montar uma lista de execução.

Bibliografia

Adobe. (2009). *Understanding variable scope*. Retrieved from http://help.adobe.com/en_US/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d4fbf351e63e3d118a9b90204-7f9d.html#WS5b3ccc516d4fbf351e63e3d118a9b90204-7f8c

Cantrell, C., Chambers, M., Dura, D., & Schall, D. (2008, Novembro). *ActionScript 3.0 library for several basic utilities.* Retrieved from http://code.google.com/p/as3corelib/.

Grossman, G., & Huang, E. (2009). *ActionScript 3.0 overview*. Retrieved from http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html

Martin-Sperry, C. (2008). *How to load MP3 files from a FileReference*. Retrieved from http://www.flexiblefactory.co.uk/flexible/?p=46

Microsoft. (2009). *Changes in the Programming Model (GDI vs. GDI+)*. Retrieved from http://msdn.microsoft.com/en-us/library/ms536339(VS.85).aspx

Roach, P. (2007). *English Phonetics and Phonology: A practical course.* Cambridge: Cambridge University Press.

Sideris, S. (2009). *Ditaa - DIagrams Through Ascii Art*. Retrieved from http://ditaa.sourceforge.net/