UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

"MHEG-5/SMIL: Um ambiente de integração entre os padrões"

MARCOS ROBERTO MACEDO

São Carlos – SP
Agosto de 1999.

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

Departamento de Computação

Programa de Pós-Graduação em Ciência da Computação

"MHEG-5 / SMIL: Um Ambiente de integração entre os padrões"

Marcos Roberto Macedo

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Profa. Dra. Marina Teresa Pires Vieira (orientadora)
DC/UFSCar

Prof. Dr. Mauro Biajiz
DC/UFSCar

Prof. Dr. Eduardo Simões de Albuquerque
UFGO

São Carlos
Agosto de 1999.

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

M141mu

Macedo, Marcos Roberto.
MHEG-5/SMIL : um ambiente de integração entre os padrões / Marcos Roberto Macedo. -- São Carlos : UFSCar, 2007.
136 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 1999.

1. MHEG-5. 2. SMIL. 3. Banco de dados multimídia. 4. Regras de transformação. I. Título.

CDD: 005.1 (20ª)

Aos meus pais
Paulo e Maria

À minha irmã Leila

À minha tia Jandira

AGRADECIMENTOS

À Deus pela vida e pelas oportunidades.

À Profa. Dra. Marina Teresa Pires Vieira em especial, pela orientação durante todo o decorrer deste trabalho, por sua dedicação, compreensão e incentivo.

Aos meus pais Paulo e Maria, minha irmã e meus familiares que estão sempre prontos a me auxiliar e compreender.

Aos amigos da pós-graduação pelo companheirismo e amizade.

Aos amigos da república “Santa Casa”, Mário (Marião), Ildeberto (Betão), Wagner (Finin), Luiz Carlos (Jiraya), Wanderley (Wand), Henrique, e à *Dona Ana* pela amizade concedida.

Ao estimável “Elvis” pela infinita alegria, pois o “cachorro” ainda é o melhor amigo do homem.

Aos docentes, funcionários do Departamento de Computação e amigos do Grupo de Banco de Dados da UFSCar, pela oportunidade para que eu pudesse realizar este trabalho.

À CAPES pelo auxílio financeiro concedido.

RESUMO

Este trabalho apresenta um ambiente de integração entre os padrões: MHEG-5 e SMIL. O padrão MHEG-5 define a sintaxe e a semântica de um conjunto de objetos que podem ser utilizados para interoperabilidade de aplicações multimídia entre plataformas com recursos mínimos. Já o padrão SMIL disponibiliza sincronismo de mídias nos documentos *Web*. Para integrar esses dois padrões foram desenvolvidos dois conversores. Com essa integração, as aplicações multimídia podem ser apresentadas tanto no ambiente *Web,* usando SMIL, como em ferramentas de autoria segundo o padrão MHEG-5. Para realizar as conversões foi desenvolvida uma base de regras, que tem como objetivo, armazenar as regras de conversões entre os padrões, possibilitando que haja dinamismo no processo de conversão. Como base de teste para as conversões, foi utilizado o banco de dados multimídia do Servidor de Objetos Multimídia (SOMm) do projeto AMOM, aonde foi desenvolvido um módulo de conexão entre esse banco de dados e o ambiente *Web*. Este trabalho também envolveu a proposta de um ambiente que integra, o SOMm e a Web, cujo título é MAW (*Multimedia Application WebBuilder*).

ABSTRACT

This work presents an integration environment between the standards MHEG-5 and SMIL. The MHEG-5 standard defines the syntax and the semantics of a group of objects which can be used for interoperability of multimedia applications among platforms with minimal resources. On the other hand, the SMIL standard provides synchronized media in the Web documents. In order to integrate the two standards two translators were developed. With this integration the multimedia applications can be presented either in a Web environment, using SMIL, or in authoring tools following the MHEG-5 standard. A rule database was developed to carry out the translations; which has as an objective to store the translating rules between the standards allowing for dynamism in the translation process. As a test database for the translations a “Multimedia Object Server” multimedia database was used in the AMOM project, where a connection module was developed between these database an the Web environment. This work has also involved the proposal of an environment which integrates the SOMm and the Web, whose title is MAW (Multimedia Application WebBuilder).

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

CAPÍTULO 1

Introdução

1.1 Considerações Iniciais

A *Web* é atualmente um dos ambientes mais utilizados para a difusão de informações. Assim como outros ambientes, a *Web* permite a utilização de recursos multimídia para a formatação de informações. Com a crescente utilização da *Web*, uma grande quantidade de informações tem sido e continuam sendo publicadas, permitindo que essas informações sejam utilizadas por diversos usuários. Assim, a maioria das páginas *Web* atualmente contém, além dos textos, outros tipos de mídia, como imagens, sons e animações. Com essa grande quantidade de informações seu gerenciamento se tornou complexo e manualmente inviável, trazendo em questão a utilização de sistemas de gerenciamento de banco de dados (SGBDs) para armazenar e recuperar as informações contidas nas páginas *Web*.

Com o advento de aplicações que exigem o tratamento de dados complexos, os SGBDs tiveram de se adaptar à essa nova realidade, dando origem aos SGBDs Relacional-Objeto e SGBDs Orientados a Objeto (SGBDOOs). Como um banco de dados multimídia deve fornecer funcionalidades para o tratamento de características específicas de cada tipo de mídia; e novas funcionalidades para essas mídias podem surgir a cada momento, o SGBD que gerencia o banco de dados deve possibilitar a adição dessas novas funcionalidades para possibilitar um tratamento adequado às mídias. Devido a esse e outros fatores, os SGBDs Orientados a Objetos demonstraram ser os mais adequados para o tratamento de informações multimídia.

Diversos padrões internacionais como o HyTime [Newcomb91], MHEG [Effelsberg95] [ISO96a] e Premo [Herman96] tratam aplicações multimídia, mas nenhum considera o ambiente da *Web*. Atualmente encontramos apenas páginas escritas em HTML na *Web* para representar as aplicações multimídia. Contudo, a linguagem HTML é bastante limitada em relação a desenvolvimento de aplicações multimídia, sendo uma das limitações a sincronização. Para disponibilizar sincronismo de mídias na *Web*, foi proposto pela *WWW Consortium* a definição de um novo padrão para documentos multimídia. Esse padrão é

denominado SMIL (*Synchronized Multimedia Integration Language*) [Liu98], [Smil98a], [Smil98b], é baseado em XML [Bray98] e fornece algumas funcionalidades básicas para incluir dados multimídia contínuas, como vídeo e áudio, em documentos *Web*.

O padrão MHEG-5 [Effelsberg95], [ISO96a] é utilizado em diversos trabalhos, principalmente aos relacionados à TV-Interativa. Um dos trabalhos que utilizam o padrão MHEG-5 para criação e manipulação de aplicações multimídia, é o SOMm (Servidor de Objetos Multimídia) [SantosM97a], [Teixeira95], [Vieira96]. O SOMm é o núcleo do projeto AMOM (***A****utoria e* ***M****anipulação de* ***O****bjetos* ***M****ultimídia*), que tem como objetivo prover um ambiente para suporte à autoria, armazenamento e manipulação de aplicações multimídia.

Este trabalho teve como objetivo a integração dos padrões MHEG-5 e SMIL, para possibilitar a transformação de aplicações multimídia, desenvolvidas com base em um desses padrões, para o outro padrão, para poderem ser manipuladas.

1.2 Objetivo da Pesquisa

Como citado anteriormente, este trabalho tem como objetivo principal a integração entre os padrões MHEG-5 e SMIL. Para isso foram desenvolvidos dois conversores: um para as conversões de aplicações MHEG-5 para documentos SMIL e o outro para as conversões de documentos SMIL para aplicações MHEG-5. Como auxílio às conversões foi desenvolvida uma base de regras de conversão que são utilizadas pelos conversores, objetivando torná-los independente da implementação, isto é, ao invés de declarar as regras de conversão na implementação dos conversores, estas são armazenadas em um banco de dados, possibilitando sua manutenção.

Como parte deste trabalho foi proposto um ambiente denominado MAW (*Multimedia Application WebBuilder*), do qual os conversores fazem parte, integrando-o com o SOMm. O objetivo do MAW é dar suporte ao desenvolvimento e apresentação de aplicações multimídia através da *Web*. Além dos conversores, outros dois módulos desse ambiente também foram implementados, como parte deste trabalho, para suporte à realização de tarefas específicas no ambiente do MAW.

1.3 Organização do Trabalho

Para o desenvolvimento deste trabalho foram abordados vários assuntos relacionados com padrões para representação de aplicações multimídia, representação de

aplicações multimídia na *World Wide Web*, gerenciamento de informações multimídia através de SGBDOOs e conexão entre banco de dados e *Web*. Esses assuntos são tratados neste trabalho como segue. O capítulo 2 é dedicado ao tratamento de aspectos referentes à Internet e *Web*. No capítulo 3 são apresentadas informações referentes às características de aplicações multimídia, tipos de mídias existentes, e também o uso de aplicações multimídia na *Web*. O capítulo 4 apresenta as diferentes abordagens de conexão encontradas entre banco de dados e *Web*. O capítulo 5 apresenta conceitos gerais sobre os padrões MHEG-5 e SMIL. O capítulo 6 apresenta o projeto AMOM, assim como seus dois módulos: o SOMm e o MAW. Na especificação do MAW, são relatados todos os módulos, dando-se uma maior ênfase no módulo de conexão e de autenticação de usuários, que foram desenvolvidos neste trabalho. O capítulo 7 é dedicado à especificação dos módulos de conversão, assim como da base de regras criada. O capítulo 8 foi reservado para relatar aspectos de implementação realizados no trabalho. As conclusões finais e os trabalhos sugeridos/futuros são apresentados no capítulo 9.

CAPÍTULO 2

Internet e *World Wide Web*

A Internet é atualmente um dos ambientes mais utilizados para a difusão de informações, permitindo entre seus serviços a utilização de recursos multimídia para a formatação de informações. Com a sua crescente utilização, uma grande quantidade de informações tem sido e continua sendo publicada, permitindo que essas informações sejam utilizadas por diversos usuários. Além de fornecer informações aos usuários, a Internet possui vários outros serviços. Os serviços e as características sobre Internet são especificadas nas seções seguintes.

2.1 Internet

Existem redes de computadores sob diversas formas há muitos anos. Uma das características mais importante da Internet é a de ser “rede de redes”. A Internet conecta não apenas computadores isolados, mas redes inteiras de computadores. Essa é a origem do nome: Internet = ***inter*-*net***work, “entre redes”. A Internet é uma associação de redes, que trocam informações, seguindo um único padrão [Damski96].

A Internet nasceu por volta de 1969, com o estabelecimento de um protocolo para comunicação entre computadores e redes de pesquisa que participavam do fundo militar, patrocinados pelo Departamento de Defesa (DOD) dos Estados Unidos. A principal característica da então Arpanet, era o uso de um protocolo comum de interconexão chamado TCP/IP (***T****ransfer* ***C****ontrol* ***P****rotocol* / ***I****nternet* ***P****rotocol*). Com seu crescimento, particularmente na década de 1980, a Internet tornou-se o meio favorito de pesquisadores, professores e estudantes para se comunicarem por meio de computadores [Damski96].

Durante muitos anos a utilização da Internet ficou restrita a instituições de ensino e à pesquisa. Nos últimos anos tem havido uma crescente abertura da Internet para utilização com fins comerciais, objetivando o fornecimento e o uso de prestação de serviços.

2.1.1 Recursos oferecidos pela Internet

Se, sob o ponto de vista físico, a Internet é uma conexão entre redes, para o usuário aparece como um grupo de serviços disponíveis para intercâmbio de informações entre computadores ou indivíduos conectados à Internet. Esses serviços são descritos resumidamente abaixo [Damski96]:

- **Correio eletrônico.** Permite que qualquer usuário da Internet envie mensagens (texto) para outro usuário.

- ***File Transfer Protocol* (FTP).** Permite a transferência eficiente de arquivos entre computadores.

- **Listas de distribuição de mensagens.** São serviços de distribuição de mensagens eletrônicas entre membros de uma lista de usuários interessados em um assunto específico. As mensagens são distribuídas por alguns computadores centrais que gerenciam o processo de distribuição. As mensagens são enviadas aos usuários "assinantes" da lista, que as recebem como qualquer outra mensagem eletrônica.

- **Usenet.** É outra maneira de distribuir mensagens eletrônicas, distintas das listas de distribuição, na qual as mensagens são espalhadas, a princípio, por todos os computadores da Internet, e colocadas à disposição de todos os usuários, agrupadas por assunto. As mensagens ficam armazenadas localmente, e são lidas quando o usuário assim desejar, não sendo portanto remetidas diretamente. Existem atualmente milhares desses grupos em nível internacional.

- **Gopher.** Sistema de obtenção de informação orientado por menus, no qual os arquivos disponíveis são indexados.

- ***World Wide Web* (*WWW, W3 ou Web*).** Com características multimídia (texto, imagem, áudio, vídeo, etc.) permite a obtenção de diversas informações na Internet. A *Web* é a coqueluche atual da Internet, e responsável por um grande aumento no tráfego de informações.

- **Telnet.** Sistema que permite que sua máquina possa ser um terminal de outra máquina na Internet. Para isso o usuário deve ter uma "conta" (*login*) na máquina destinatária.

- **Talk.** Serviço de comunicação interativa e em tempo real entre dois usuários da Internet.

- **Finger.** Permite obter informações sobre um usuário específico da Internet - por exemplo, quando foi à última vez que sua conta foi usada.

Uma nova tecnologia chamada Internet2, foi desenvolvida com intuito de solucionar alguns aspectos relevantes que surgiram com o uso da Internet, como por exemplo, o problema da demora de acesso às informações. Na seção seguinte é especificado o surgimento, algumas características e a arquitetura física da Internet 2.

2.1.2 Internet 2

O projeto Internet2 [Internet2] passou a ser, neste momento, o primeiro passo (e talvez o mais importante) no novo empreendimento americano. Em janeiro de 1997, mais de 100 universidades americanas já haviam assumido compromisso formal com a participação no projeto. Atualmente o consórcio Internet2 conta com o apoio e a participação não só do grupo inicial de universidades, mas também de centros de pesquisa, agências do governo e membros da indústria dedicados ao desenvolvimento de novas tecnologias Internet de alto desempenho.

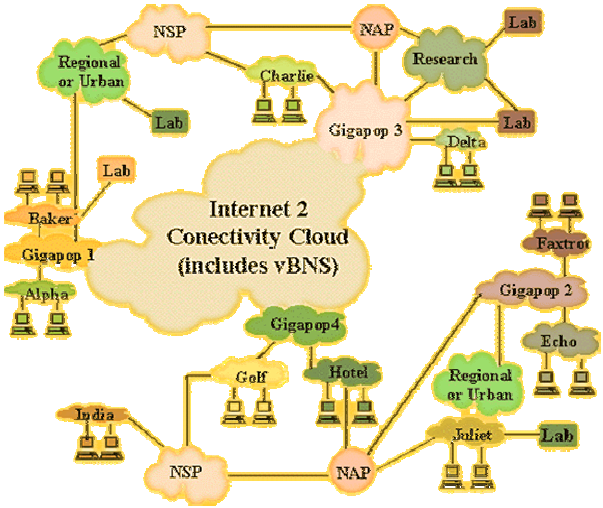


FIGURA 2.1 – ARQUITETURA FÍSICA DA INTERNET 2

Diversas aplicações para redes de alta velocidade estão sendo desenvolvidas na Internet2, sendo que muitas delas já se encontram em fase de teste. A arquitetura física da Internet2 é ilustrada na figura 2.1. No momento, algumas das principais linhas de pesquisa desenvolvidas para a aplicação de serviços em rede de alto desempenho são:

- Bibliotecas digitais com capacidade de reprodução de imagens de áudio e vídeo de alta fidelidade; oferta de imagens com alta resolução, com reprodução quase imediata na tela do computador e novas formas de visualização de imagens digitais;

- Ambientes colaborativos que englobam laboratórios virtuais com instrumentação remota; desenvolvimento de tecnologias para debates virtuais em tempo real e com utilização de recursos multimídia, em alta velocidade e de aplicação simplificada;

- Novas formas de trabalho em grupo, com desenvolvimento de tecnologias de presença virtual e colaboração em 3D;

- Telemedicina, incluindo diagnóstico e monitoração remota de pacientes;

- Projeção de telas de computadores em três dimensões, através da utilização da *ImmersaDesk* (espécie de grande tela de TV que projeta as imagens em 3D);

- Controle remoto de microscópios eletrônicos para pesquisas médicas.

Demonstrações dos novos potenciais da Internet2 vêm sendo apresentadas desde o ano de 1998 em vários eventos e *workshops*, promovidos com o intuito de sensibilizar não só a comunidade acadêmica como também diversos setores de indústria e até mesmo o governo (recentemente foi feita uma apresentação para diversos senadores americanos).

Em seu estágio atual a Internet2 utiliza o vBNS (***v****ery high performance* ***B****ackbone* ***N****etwork* ***S****ystem*), ou *backbone* de alta velocidade, da *National Science Foundation*. A velocidade máxima oferecida pelo vBNS é de 622 Mbps.

De uma forma geral não se conhece ainda o limite do que é tecnicamente possível. Pode-se dizer então que o foco principal da Internet2 reside no desenvolvimento de

aplicações avançadas com uso intensivo de tecnologias multimídia em tempo real. Como resultado de todo o movimento de mobilização da comunidade acadêmica para a retomada da liderança no âmbito da nova geração da Internet, foi criada em 1º de outubro de 1997 a ***U**niversity **C**orporation for **A**dvanced **I**nternet **D**evelopment* (UCAID). A UCAID é uma organização sem fins lucrativos cujo objetivo é orientar o avanço e desenvolvimento da Internet2. Esta corporação, inicialmente constituída por três universidades americana líderes no setor de pesquisa, tem como missão orientar os estudos e descobertas relativas às aplicações em todas as áreas do conhecimento, bem como em engenharia e ferramentas para redes eletrônicas de alto desempenho.

A arquitetura física da rede eletrônica que dá suporte à Internet2 inclui a implantação de GigaPOPs – pontos de presença com velocidade de tráfego da ordem de Gigabits, conforme mostrado na figura 2.1. A função principal do GigaPOP é o gerenciamento da troca do tráfego Internet2 de acordo com especificações de velocidade e qualidade de serviços previamente estabelecidos através da rede. Cada GigaPOP irá concentrar e administrar o tráfego de dados originados e destinados a um conjunto de universidades e centros de pesquisa localizados em uma mesma região geográfica. A troca de dados entre os GigaPOPs é realizada atualmente por uma rede de alto desempenho mantida pela *National Science Foundation*. Essa rede possui restrições quanto ao tipo de tráfego que transporta, permitindo seu uso apenas para as instituições acadêmicas participantes da Internet2.

Os GigaPOPs possuem políticas locais para a aceitação de conexões, que deverão ser negociadas entre as partes envolvidas. Ressalta-se, no entanto, que além de oferecer os requisitos técnicos necessários, a função do GigaPOP consiste também em separar o tráfego entre membros participantes da Internet2 e as instituições localmente conectadas, mas cujo tráfego deve ser desviado para a Internet comercial. Consequentemente, todo GigaPOP possui no mínimo, duas conexões: uma para a Internet2 e a outra para a Internet comercial.

2.2 *World Wide Web* (WWW, W3 ou Web)

A *Web* é uma rede virtual na Internet, que torna os serviços disponíveis na Internet totalmente transparentes para o usuário e ainda possibilita a manipulação da informação multimídia [Damski96]. A arquitetura da *Web* foi proposta em 1989 e está ilustrada na figura 2.2. A figura mostra os padrões em comum: URL *(**U**niversal **R**esource **L**ocator)* e HTTP (***H**yper**T**ext **T**ransfer **P**rotocol)*, com negociação de formatos de tipos de dados.

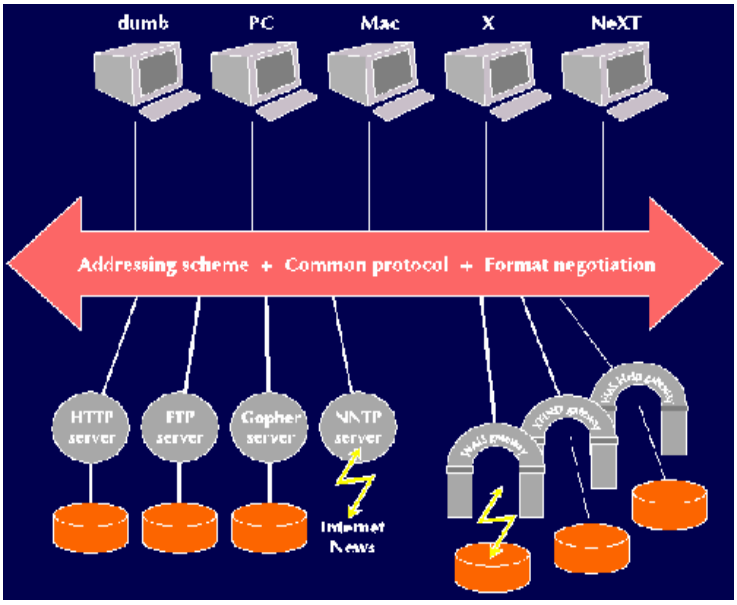


FIGURA 2.2 – ARQUITETURA ORIGINAL DA WWW DE 1990 [BERNERS-LEE94]

A *Web* surgiu para suportar várias abordagens, onde incluem [Berners-Lee94]:

- A idéia de um mundo de informação ilimitado, em que todos os itens possuem uma referência para outras informações que podem ser recuperadas.

- Um sistema de endereços (URI - ***U****niversal* ***R****esource* ***I****dentifiers*), onde é criado esse mundo de possibilidades, apesar de protocolos diferentes.

- Um protocolo de rede (HTTP) que é usado por servidores nativos, fornecendo maior desempenho.

- Uma linguagem marcada que todo cliente *Web* requisita, e é usada para a transmissão de conteúdos básicos como texto, menus e informações de ajuda *on-line* simples através da rede.

- Um corpo de dados disponíveis na Internet usando todos ou alguns dos itens listados anteriormente.

Existem vários programas de navegação para a Internet. Esses programas são também chamados de *browsers*. São capazes de apresentar os documentos, e também formatá-los dependendo da mídia recebida. Os mais conhecidos são *Mosaic*, *Netscape*, *Internet Explorer*, *HotJava* e *Lynx* (para terminais baseados em caracteres).

Em [Berners-Lee96], Tim Berners-Lee define três metas a serem atingidas na *Web*. A primeira envolve o aperfeiçoamento da infra-estrutura, para fornecer mais serviços funcionais, robustos e eficientes. A segunda é intensificar a *Web* como meio de comunicação e interação entre pessoas. E a terceira é permitir a interação entre máquina e *Web*, fornecendo um mecanismo capaz de analisar a *Web* e de resolver alguns possíveis problemas.

Com o surgimento da *Web*, as aplicações multimídia começaram a se integrar com esse ambiente, oferecendo recursos para os usuários. Em [Doorn94] são definidas basicamente duas formas de integração de aplicações multimídia e *Web*, sendo elas:

- Integração da *Web* em aplicações (usar *Web* como parte de um programa).

- Integração de aplicações na *Web* (embutindo *scripts* numa página *Web*).

A integração da *Web* em aplicações surgiu com a necessidade de oferecerem recursos específicos da *Web* para disponibilizar informações aos usuários. Um exemplo dessa forma de integração é demonstrado em [Doorn94], utilizando uma linguagem *script* chamada *Tcl-script*. Esse exemplo é ilustrado abaixo.

```
web .w -home "http://www.cs.vu.nl/"
button .b -text "Back" -command {.w back}
pack .w
pack .b
```

Para desenvolver aplicações na *Web*, isto é, adicionar funcionalidades na *Web*, foi necessário criar linguagens *scripts*. Atualmente, os documentos são escritos em linguagens marcadas, como HTML (***H**yper**T**ext **M**arkup **L**anguage)*, e para adicionar as funcionalidades, deve-se criar novas *tags* (comandos) que serão embutidas juntamente com a página *Web*. Um exemplo dessa integração é o uso de uma *tag* chamada <*hush*> que foi desenvolvida para a *Tcl-script* descrita em [Doorn94]. Abaixo é ilustrado um exemplo dessa integração.

```
<hush tcl="button $this.b -text {Hello}; pack $this.b" >
</hush>
```

Com o desenvolvimento de aplicações na *Web*, torna-se necessária a utilização de métodos de segurança a fim de impedir que usuários não autorizados utilizem informações fornecidas pelas páginas. Para isso foram criados vários métodos para a segurança de informações na *Web*. Essa abordagem é feita na seção seguinte.

2.2.1 Segurança

O crescente interesse da comunidade de usuários da Internet na utilização da *Web* como mecanismo de prestação de serviços, inclusive com intuito comercial, e como arquitetura de base para a implementação de grande diversidade de aplicações multimídia baseadas no modelo cliente/servidor, levanta questões e traz problemas não resolvidos de forma satisfatória, pelas primeiras gerações da tecnologia. Entre eles, destaca-se o problema da segurança. Em muitas aplicações *Web* é requisitado que os *browsers* e os servidores se autentiquem/certifiquem mutuamente, e que seja garantida privacidade das interações, isto é, que não seja possível a terceiros a consulta de informações trocadas.

Os requisitos (serviços) considerados relevantes e essenciais para a segurança são: privacidade nas transações, autenticação de servidores/serviços, autenticação de clientes/utilizadores e integridade das transações.

Em [Simão95] é definida uma panorâmica de alguns esquemas de segurança propostos. Esses esquemas são descritos resumidamente abaixo, sendo eles:

- **Autenticação Básica** – é o único esquema de autenticação e serviço de segurança integrado oficialmente na versão do HTTP/ 1.0, tendo de ser obrigatoriamente suportado por clientes/servidores que estejam em conformidade com a norma. É um esquema que trata apenas da autenticação de clientes perante servidores, ou seja, é um mecanismo de autenticação aplicável a controle de acessos. Baseia-se no conceito de segredo partilhado, que toma a forma de uma *password* semelhante às de *login*. Quando um cliente requer acesso a um documento protegido com esse esquema de autenticação, o servidor o obriga a enviar juntamente com o pedido HTTP o seu identificador de cliente (*userId*) e a sua *password*. Na recepção, o servidor valida o pedido do cliente consultando uma base de dados local. Se a *password* conferir considera-se o usuário autenticado, caso contrário é retornado uma mensagem de erro. É semelhante tecnologicamente aos serviços de **Telnet** e **FTP**, e oferece, portanto um nível de segurança equivalente, isto é, muito baixo.

- **Servidor com Chave Pública** – para suplantar o maior defeito do esquema de autenticação básica, foi proposta uma versão segura que utiliza criptografia de chaves públicas. Nesse esquema, a informação de autenticação enviada pelo cliente é protegida criptograficamente, impedindo que a *password* possa ser

"roubada". Além disso, as respostas do servidor são criptadas com uma chave fornecida e gerada pelo cliente a fim de se garantir a privacidade. Nesse esquema, os servidores possuem um par assimétrico (chave pública, chave privada), da qual os clientes usam a parte pública para criptar a informação de autenticação. Quando o cliente tenta acessar pela primeira vez um documento protegido sem fornecer qualquer informação de autenticação, o servidor envia-lhe uma indicação de erro contendo a sua chave pública. Uma vez conhecida a chave pública do servidor, o cliente reenvia o pedido incluindo as informações de autenticação criptadas assimetricamente. Na recepção, o servidor decripta a informação de autenticação com a sua chave privada (*userId* e *password*), como na autenticação básica e verifica se o endereço IP do cliente confere com o do pedido. Se algumas das verificações falharem, o servidor envia uma indicação de erro de autenticação ao cliente. Se todas sucederam, o cliente é considerado autenticado.

- **Segurança Baseada no RIPEM/PGP** – é uma das soluções mais simples, mais efetivas, utilizada para providenciar segurança na *Web*. Consiste em utilizar alguns programas de segurança disponíveis, como o PGP e o RIPEM, para proteger as transações HTTP. Esses programas utilizam a criptografia de chaves assimétricas para providenciar os serviços de segurança. Esse esquema é verdadeiramente simples, pois delega o tratamento dos problemas de segurança para outra aplicação. É, contudo limitado, pois requer a invocação de um programa executável para cada pedido/resposta. Além disso, não suporta mecanismos de negociação, devido principalmente ao fato de ter por base, aplicações que foram desenvolvidas para comunicação unidirecional. O esquema foi proposto inicialmente por Tony Sanders da *Berkeley Software Design Inc.,* e integrada no servidor HTTP Plexus/3.0, e mais tarde no *browser* Mosaic (a partir da versão 2.2) e no servidor de HTTP desenvolvido pela NCSA (*National Center for Supercomputing Applications*).

- ***Digest Access Authentication*** **(DAA)** – segue o modelo de interação semelhante à autenticação básica. Quando um cliente solicita o acesso a um documento protegido, e não apresenta informação de autenticação ou essa informação é inválida, o servidor retorna-lhe uma indicação de erro com os dados que ele poderá utilizar para se autenticar corretamente. Entre esses dados constam: um identificador *reino* e*,* opcionalmente, uma lista de URI/URL designada de domínio.

O identificador *reino* serve apenas para o cliente identificar qual *password* deve ser usada. O domínio serve para informar ao cliente que outros URIs/URLs requerem autenticação, permitindo assim diminuir o número de transações falhadas. Usando esses dados, o cliente reconstrói o pedido HTTP adicionando-lhe a informação de autenticação exigida pelo servidor, que inclui um cabeçalho para a autenticação, o identificador do cliente (*userId*) e outros dados que tornam possível ao servidor validar o pedido.

- ***Mediated Digest Authentication*** **(MDA)** – é uma generalização do esquema DAA acima citado. Quando o número de servidores/serviços disponíveis na *Web* aumenta o pré-estabelecimento de uma *password* secreta entre cada par (cliente, servidor) torna-se impraticável. Os servidores de autenticação (SA) são responsáveis por gerarem pedido de chaves de sessão secretas e por autenticar clientes e servidores. Cada cliente/servidor pode registrar-se em um ou mais SA, sendo necessário, para que uma transação se concretize que o cliente e o servidor envolvido estejam registrados em apenas um SA comum.

- ***Secure HTTP*** **(S-HTTP)** – é o mais abrangente em relação aos acima citados. Suporta múltiplos métodos de estabelecimento de chaves, múltiplos formatos para as mensagens protegidas com criptografia, aplicação de vários serviços de segurança de forma ortogonal e, múltiplos algoritmos de criptografia. Os pedidos e as respostas HTTP enviadas por clientes e servidores são encapsulados em mensagens exteriores. Essas mensagens são constituídas por duas partes: um cabeçalho, com uma sintaxe semelhante a um cabeçalho HTTP normal, e um corpo, onde o verdadeiro pedido/resposta HTTP se encontra encapsulado segundo um dos formatos padrão para mensagens protegidas através de criptografia. O receptor de uma mensagem usa informação do cabeçalho externo e outra informação específica do formato de encapsulamento para recuperar o pedido/resposta original, procedendo à sua decodificação e decriptação, e à verificação da informação de segurança aplicada à mensagem.

- ***Shen*** – desenvolvido pela CERN. Baseia-se na categorização dos dados em cinco classes, de acordo com o seu *status* de segurança. É um dos esquemas que oferece funcionalidade para proteção dos direitos autorais, permitindo definir um tempo de expiração depois que os dados foram transmitidos, para serem eliminados.

- ***S**ecure **S**ockets **L**ayer* (SSL) – é independente de protocolo, que providencia um canal de comunicação seguro num protocolo de transporte orientado para conexão (por exemplo, TCP). Depois do estabelecimento da conexão de transporte, entra-se numa fase de *handshake*, onde as entidades comunicantes são autenticadas, e negociam entre si uma chave de sessão secreta juntamente com os algoritmos de criptografia a utilizar. A partir daí, todos os dados enviados pelo canal de comunicação são protegidos criptograficamente.

Atualmente, a maioria das páginas *Web* é constituída de critérios de segurança, sendo essas páginas denominadas de sites seguros.

Na seção seguinte são relatadas algumas das linguagens de programação existentes nas páginas *Web*.

2.2.2 Linguagens Abordadas em uma Página *Web*

Esta seção tem como objetivo esboçar algumas das principais linguagens de programação que podem ser encontradas nas páginas *Web* distribuídas na Internet. Entre as principais destacam-se HTML, XML, e as linguagens *scripts*: JavaScript, DHTML, ASP, e outras. Atualmente surgiu a linguagem SMIL que tem como objetivo disponibilizar sincronismo na *Web*. O SMIL será tratado em um capítulo exclusivo, especificamente no capítulo 5. As linguagens HTML, XML, Java (JavaApplet) e JavaScript são especificadas a seguir.

a-) HTML (*HyperText Markup Language*)

HTML é a linguagem de marcação mais utilizada na Internet e que devido à sua simplicidade foi amplamente difundida [December95]. Essa linguagem é um sistema para descrever documentos, isto é, é definida a partir da linguagem SGML (***S**tandard **G**eneralized **M**arkup **L**anguage*) e possui características adicionais de hipertextos através de *hyperlinks*. O código HTML é escrito em formato ASCII, e isso é uma grande vantagem, uma vez que ASCII pode ser lido por qualquer plataforma (IBM, Macintosh, UNIX, etc.). Por ser uma linguagem de marcação, sua maior preocupação é com a estrutura dos documentos e não com a aparência. A linguagem consiste em texto comum e códigos especiais chamados *tags*, que, na verdade, são os comandos da linguagem, sendo que esses comandos

permitem a inclusão de textos, imagens e referências para outros documentos (*hyperlinks*) [Ramalho97].

HTML não é uma linguagem de *layout,* mas sim uma linguagem usada para marcar a parte estrutural de um documento (parágrafos, listas, etc). Utilizando essas marcações, os programas, que renderizam os documentos HTML, os mostram de maneira legível; esses programas são chamados de *browsers*. Essa organização permite a separação entre a especificação estrutural de um documento e sua aparência formatada nos *browsers*.

Assim como outras linguagens, HTML possui uma estrutura básica para seus programas. Para que um *browser* interprete corretamente o programa, ele deve possuir alguns comandos básicos. Alguns *browsers* até dispensam o seu uso, porém, é melhor assumir tais comandos como parte fundamental do programa. Um programa HTML possui três partes básicas: a estrutura principal, o cabeçalho e o corpo do programa. A figura 2.3, mostra esse esquema da estrutura e os comandos associados a ela. Todo programa HTML deve iniciar com o comando <HTML>, sendo possível a inclusão de informações para o cabeçalho do documento, que é especificado na área de cabeçalho através do comando <HEAD>. A maioria dos comandos HTML é colocada na área do corpo do programa delimitada pelo comando <BODY>.

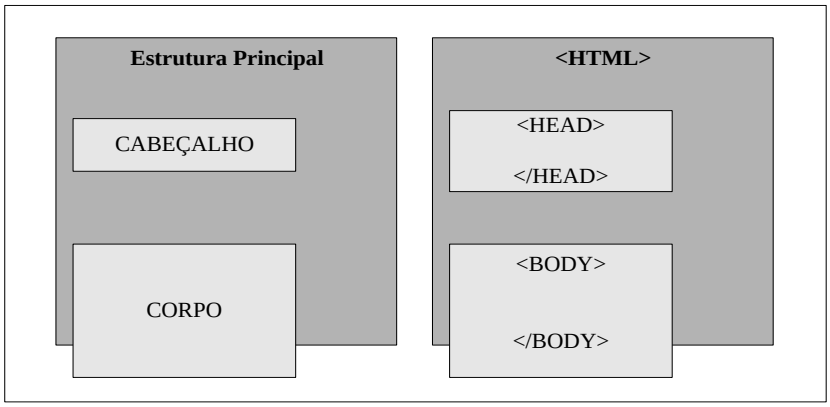


FIGURA 2.3 – ESTRUTURA DE UM DOCUMENTO HTML [RAMALHO97]

Na figura 2.4 é mostrado um exemplo de arquivo HTML apresentado no *browser Netscape*. Esse exemplo mostra duas imagens que possuem *links* para outras páginas *Web.* O código fonte desse arquivo HTML é demonstrado abaixo. A figura 2.5 e 2.6 são exemplos de páginas que podem ser encontradas na *Web.*

```
<html>
  <head>
    <title> Exemplo de HTML </title>
  </head>
  <body>
    <h1>Títulos disponíveis</h1>
    <A HREF= "gp.htm" >  <img align=middle src="w95gp03.gif"></a>
       Windows 95 Guia Prático
    <A HREF="gi.htm"> <img align=middle src="w95gi03.gif"></a>
       Iniciação ao Windows 95<p><hr>
       Clique na capa do livro que deseja ver seu resumo.
  </body>
</html>
```

FIGURA 2.4 – EXEMPLO DE ARQUIVO HTML

FIGURA 2.5 – EXEMPLO DE PÁGINA HTML
URL: http://www.ufscar.br

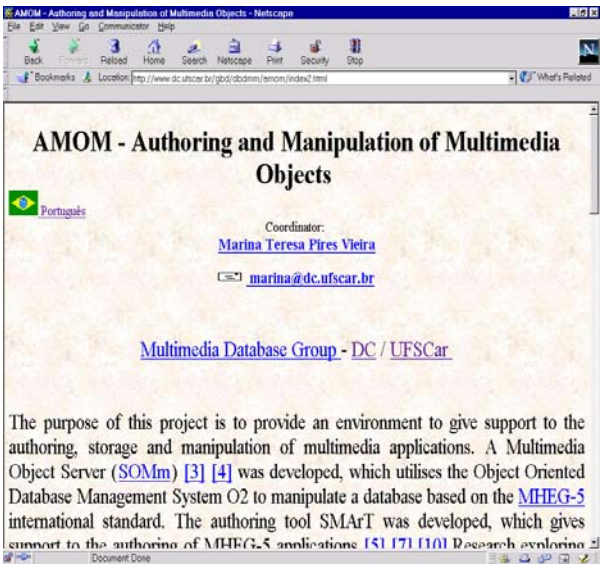


FIGURA 2.6 – EXEMPLO DE PÁGINA HTML
URL:http://www.dc.ufscar.br/gbd/dbdmm/amom/

b-) XML (*eXtensible Markup Language*)

Na metade de 1996, um grupo de aproximadamente 80 peritos juntou forças com o *World Wide Web Consortium (W3C)* para formar um grupo de trabalho sob a chefia de Jon Bosak da *Sun Microsystems*. A meta era desenvolver uma linguagem de marcação que tivesse o poder e a generalidade da SGML e, ao mesmo tempo, fosse fácil de ser implementada na *Web*. Essa linguagem de marcação teria de fazer o seguinte [Light99]:

- Dar suporte à marcação generalizada na *Web*;
- Produzir documentos que idealmente fossem válidos de acordo com as regras da SGML;
- Fornecer suporte para *hyperlinks* que fossem altamente compatíveis com a abordagem URL e
- Fornecer um mecanismo de folha de estilo genérico e poderoso.

A primeira façanha desse grupo de trabalho foi desenvolver uma especificação inicial da linguagem para XML, a qual foi divulgada em novembro de 1996 na Conferência SGML em Boston, EUA. Um segundo esboço foi publicado em março de 1997. Logo após, em abril de 1997, o primeiro esboço da especificação de *hyperlinks* XML foi publicado. Em 1° de julho de 1997, essa organização foi formalizada nos padrões das linhas da W3C. O Conselho de Revisão Editorial (ERB) W3C SGML tornou-se o Grupo de Trabalho W3C XML (GT) e agora segue as linhas dos grupos de trabalho da W3C. Este GT W3C XML se encarregou da formalização do padrão XML. Na mesma data, o atual Grupo de Trabalho SGML transferiu suas funções para o GT XML, que mudou seu nome para Grupo de Interesse Especial W3C XML.

XML e HTML têm uma origem em comum, o SGML. Ao contrário da linguagem HTML, no entanto, XML não estabelece como um determinado elemento deve ser visualizado. Seu objetivo é armazenar as informações de forma organizada. A idéia é que um arquivo XML possa ser apresentado em mídias diferentes – um mesmo material, por exemplo, pode receber determinado tratamento gráfico para a *Web* e outra formatação para ser impresso. Por suas características, XML necessita de um intermediário para que os dados sejam visualizados. Nesse princípio de implantação da nova linguagem, as opções ainda são precárias, mas os fabricantes e o *World Wide Web Consortium (W3C)*, trabalham

para implementar padrões complementares ao XML. Um deles é o XSL (*eXtensible Stylesheet Language*), ainda em discussão no W3C.

XML é ideal não apenas para transmissão de dados de servidores para *browser*, mas também para enviar dados de aplicações para aplicações e de máquinas para máquinas [Sall98]. XML não trabalha com *tags* fixas como HTML, permitindo que cada autor crie o seu próprio conjunto de *tags*. Então, para criar *tags* no XML, é necessário seguir algumas regras, tais como, cada elemento deve ser composto obrigatoriamente por uma *tag* de abertura e outra de fechamento (<tag> </tag>),elementos vazios devem ser encerrados por uma barra (<tag/>), entre outras. As definições das novas *tags* criadas devem estar contidas em uma DTD (*Document Type Definition*), bem como as suas inter-relações. Dessa forma, um documento XML é interpretado seguindo as regras definidas em sua DTD. Diversos tipos de aplicações foram reestruturadas através do uso de XML, as principais envolvem EDI (*Electronic Data Interchange*), utilização de banco de dados na *Web*, tradução de idiomas, etc. Além de reestruturar, novas funcionalidades a esses tipos de aplicações são fornecidas através da criação de novas linguagens, como é o caso de SMIL (*Synchronized Multimedia Integration Language*), MathML, CML (*Chemical Markup Language*).

XML e HTML operam em níveis diferentes de generalidade, então geralmente não estarão em competição direta. HTML tem como objetivo particular exibir páginas *Web* on-line, com *hyperlinks*, e XML é um perfil da SGML, o que significa que pode dar suporte a um intervalo ilimitado de aplicações.

Um arquivo XML pode ser incluído dentro de um HTML de forma estática (um simples *link* para um arquivo XML) ou de forma dinâmica (ativados por um CGI). No exemplo abaixo, cada elemento do arquivo XML (teste.xml) foi deslocado para ser exibido em uma célula da tabela do arquivo HTML (teste.html). A vantagem dessa solução é que o arquivo HTML traz apenas a estrutura gráfica e não os dados para a tabela HTML.

Exemplo: Dados Estruturados em XML (teste.xml)

```
<?xml version= ‘1.0’?>
<clientes>
  <cliente>
    <nome> Joao da Silva </nome>
    <telefone> 999-9999 </telefone>
    <endereco> Rua Z, 99 </endereco>
    <email> jsilva@provedor.om.br  </email>
  </cliente>
</clientes>
```

Arquivo HTML para visualizar as informações contidas no arquivo XML (teste.html)

```
<HTML>
 <HEAD>
  <TITLE>Teste</TITLE>
    <XML ID= "Clientes" SRC = "teste.xml" > </XML>
 </HEAD>
 <BODY bgcolor=#ffffdd>
   <TABLE>
    <TR>
     <TD valign=top>
      <IMG SRC = http://www.dc.ufscar.br/gdb/dbdmm/amom/logotipo.gif"/>
     </TD>
     <TD valign=middle> <font face=verdana ize=4 color=darkred>
        Relacao de clientes</font>
     </TD>
    </TR>
   </TABLE>
   <TABLE DATASRC= "#Clientes" BORDER= "1" WIDTH = "100%" >
    <THEAD>
     <TR>
      <TH WIDTH = "30%" BGCOLOR= "darkblue">
        <font face=arial size=2 color=white>Nome</font>
      </TH>
      <TH WIDTH = "20%" BGCOLOR= "darkblue" align= "right">
        <font face=arial size=2 color=white>Telefone</font>
      </TH>
      <TH WIDTH = "30%" BGCOLOR= "darkblue" >
        <font face=arial size=2 color=white>Endereco</font></TH>
      <TH WIDTH = "20%" BGCOLOR= "darkblue" align= "right">
        <font face=arial size=2 color=white>E-Mail</font>
      </TH>
     </TR>
     <TR>
      <TH VALIGN=TOP WIDTH= "30%" >
        <font face=arial size=2><B><SPAN DATAFLD = "nome"> </B> </FONT>
      </TH>
      <TH VALIGN=TOP WIDTH= "20%" align= "right" >
        <font face=arial size=2><SPAN DATAFLD = "telefone"> </SPAN> </FONT>
      </TH>
      <TH VALIGN=TOP WIDTH= "30%" >
        <font face=arial size=2><SPAN DATAFLD = "endereco">  </FONT>
      </TH>
      <TH VALIGN=TOP WIDTH= "20%" align= "right" >
        <font face=arial size=2><SPAN DATAFLD = "email"> </SPAN> </FONT>
      </TH>
     </TR>
    </THEAD>
   </TABLE>
 </BODY>
</HTML>
```

Esse exemplo pode ser visualizado no *Internet Explorer 5.0*, uma vez que esse *browser* suporta parcialmente XML.

c-) Java

A linguagem Java é totalmente orientada a objetos. Tudo em Java é um objeto, exceto alguns tipos básicos como os números [Cornell98] [Arnold97] [Java97]. Java é uma linguagem de programação desenvolvida pela Sun Microsystems com o intuito de ser uma linguagem multiplataforma. A facilidade de multiplataforma ocorre porque os programas Java são compilados para os chamados *bytecodes* Java, que são projetados para serem executados em uma máquina abstrata, a máquina virtual Java. Dessa maneira uma máquina virtual Java pode ser implementada em qualquer plataforma computacional, e é ela quem interpreta as instruções para o código da máquina na qual o programa está sendo executado.

Os códigos Java são escritos utilizando a norma Unicode, uma norma internacional de conjunto de caracteres, através da qual pode-se utilizar, nos programas, caracteres especiais, como letras gregas. A linguagem de programação possui sua sintaxe e palavras chaves bem parecidas à linguagem C++.

Quando um programa Java é executado utilizando um *browser* (Netscape, Internet Explorer, Opera, etc.) esse programa é chamado de *JavaApplet*. Um *JavaApplet* possui basicamente as mesmas funcionalidades de um programa Java, a diferença é que um *applet* não pode executar diretivas de E/S (no *cliente Web)* e tem que ser derivada da classe *Applet* da linguagem Java. A especificação de um *JavaApplet* é abordada na seção seguinte.

c-1) JavaApplet

Os aplicativos em Java são executados a partir de uma linha de comandos fazendo com que seu interpretador interprete os *bytecodes* contidos num arquivo de classe. Os *applets*, por outro lado, são executados dentro de uma página *Web* por meio de um *browser* preparado para usar Java, como o *Netscape 2.0* ou o *Internet Explorer 3.0* em diante [Cornell98].

Para executar *applets*, deve-se criar dois arquivos: um com o código fonte da programação (.java) e outro contendo indicadores que executam o programa (.html). Abaixo é demonstrado um exemplo de um programa *JavaApplet*.

```
import java.awt.*;
import java.applet.*;
public class Hello extends Applet
{
      Font f = new Font("System", Font.BOLD, 18);
      public void paint (Graphics g)
      {
            g.setFont(f);
            g.drawString ("Hello", 25, 50);
      }
}
```

O programa acima especificado, emitirá a mensagem "Hello" nas posições (25, 50) da página *Web*. Para disponibilizar um programa *JavaApplet* na *Web*, é preciso escrever um arquivo HTML incluindo informações sobre o *applet*. Esse arquivo HTML possuirá uma *tag* que é específica para determinar um *applet*. Abaixo é ilustrada a *tag* <APPLET> chamando o programa *JavaApplet* criado acima.

```
<APPLET CODE="HELLO.CLASS">
</APPLET>
```

c-2) JavaScript

Uma outra linguagem encontrada nas páginas *Web* é *JavaScript*. Ela é interpretada pelo *browser* e o seu código fonte é um tipo de hóspede dentro de um programa HTML. No mesmo arquivo .htm ou .html em que estão os comandos básicos da linguagem HTML, o código *JavaScript* é inserido de maneira a ser interpretado quando necessário [Ramalho97].

Existem duas formas de incluir o código *JavaScript* dentro do documento HTML. A primeira é embutir o código *JavaScript* inteiramente dentro do documento e a segunda é criar um arquivo externo com os comandos *JavaScript*, que é referenciado por um comando dentro do programa HTML, que o carrega automaticamente. Embutir o código dentro do programa HTML tem a vantagem de deixar tudo à mão para a edição do código. É ideal quando o código *JavaScript* vai ser usado apenas naquele documento. Quando o código é utilizado por vários arquivos, então é interessante deixá-lo em arquivo separado.

Para embutir um código *JavaScript* no programa HTML, deve-se usar a *tag*:

```
<SCRIPT LANGUAGE= "Javascript">
..... código
</SCRIPT>
```

Os códigos são colocados através de comentários, pois nem todos os *browsers* suportam a linguagem *JavaScript*.

Na figura 2.7, é demonstrado um exemplo de arquivo HTML acrescentado de comando *JavaScript*, executado pelo *browser* Netscape. Esse exemplo mostra apenas uma caixa de mensagem. O código fonte desse arquivo HTML é demonstrado após a figura.

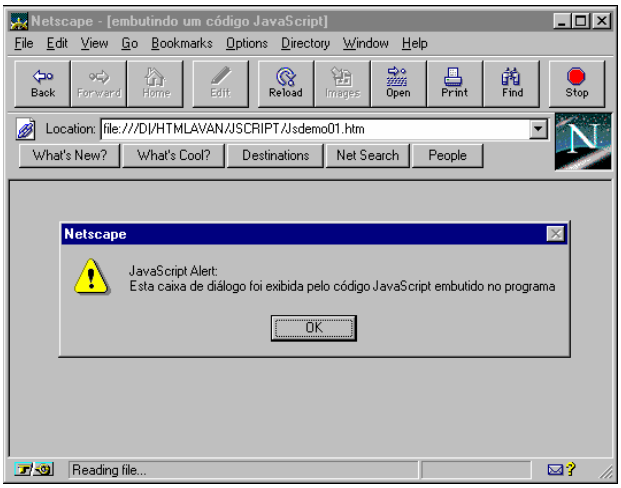


FIGURA 2.7 – ARQUIVO HTML USANDO JAVASCRIPT

Código fonte do arquivo HTML contendo uma programação em *JavaScript*

```
<html>
  <head>
    <title>embutindo um código JavaScript</title>
    <script language="JavaScript">
      <!--
      alert( "Esta caixa de diálogo foi exibida pelo código JavaScript
embutido no programa" )
    //-->
    </script>
  </head>
  <body>
  </body>
</html>
```

2.3 Considerações Finais

Neste capítulo foram tratados assuntos referentes a Internet e *Web*. O capítulo iniciou-se com a seção 2.1, que contém a especificação do termo “Internet”, e também contém especificação de alguns recursos oferecidos pela Internet. Na seção 2.2 é definido o termo *Web*; é realizada uma panorâmica sobre os diversos esquemas de segurança existente para a *Web*. Ainda nessa seção, são definidas as principais linguagens de programação encontradas em uma página *Web*. Esse capítulo explanou vários conceitos empregados no ambiente *Web*. Todas as páginas *Web* são compostas por mídias, como: texto, áudio, imagens, etc.

Para tratar de aplicações multimídia na *Web*, foram abordados diversos assuntos no capítulo 3. É utilizado também o termo hipermídia para as aplicações multimídia na *Web*. Os termos multimídia e hipermídia e seus contextos são relatados no capítulo 3.

CAPÍTULO 3

Multimídia e Hipermídia

A definição tradicional do termo “hipertexto” descreve que ele é um sistema para tratar texto simples adicionado de *links*. A partir disso, vários sistemas incluiram também a possibilidade de trabalhar com gráficos e outras mídias, definindo-se então em usar o termo *hipermídia* para expressar os aspectos multimídia de seus sistemas hipertextos [Nielsen90].

Em [Harrison95], Harrison define *hipermídia* como sendo o potencial para revolucionar o desenvolvimento e estilo de aplicações, bem como a sofisticação dessas aplicações. É definido também, que um sistema *multimídia* é um sistema onde as páginas podem incluir muitos tipos de mídia. A principal propriedade que caracteriza sistemas multimídia em geral é a incorporação de mídia contínua como vídeo, áudio, animação e tipos convencionais de dados, como textos, imagens e gráficos. Portanto, um sistema *hipermídia* é definido como um sistema multimídia que possui *hyperlinks* entre suas “páginas”.

3.1 Dados Multimídia em Banco de Dados

Há alguns anos atrás, sistemas de gerenciamento de banco de dados manipulavam apenas tipos de dados como caracteres e inteiros. Estruturas de registro eram simples e de fácil manuseio para representar os dados. Recentemente, com o surgimento de aplicações multimídia, os sistemas de gerenciamento de banco de dados sofreram mudanças para suportar esses tipos de dados complexos, exigindo-se uma evolução no critério de armazenamento e manipulação.

Em [Pazandak97] e [Klas97] são relacionados os tipos de dados multimídia que podem ser armazenados, tais como:

- *Texto*. É reduzida a representação de caracteres, sendo que pode representar uma ampla quantidade de texto, estruturado na forma de livros, por exemplo, capítulos, seções, subseções, parágrafos e tipos de letras.

- *Gráficos*. Incluem desenhos e ilustrações utilizando um alto nível de descrição como CGM, Pict e PostScript. Esse tipo de dado pode ser armazenado em um modo estruturado no banco de dados. Pode-se consultar o conteúdo através de metadados, como linhas, círculos e arcos (por exemplo, “Buscar todos os gráficos que contenham um círculo”).

- *Imagens*. Incluem figuras e fotografias que são definidas através de formatos padrão, como Bitmap, JPEG e MPEG. A representação de armazenamento de imagens é uma tradução direta da imagem, pixel por pixel.

- *Animação*. É uma seqüência temporal de imagem ou dado gráfico. Especifica a ordem em que gráficos ou imagens serão mostrados. As imagens e os gráficos são construídos e organizados independentemente. Para cada imagem ou gráfico é determinado um período de tempo, e esse tempo pode variar por animação (por exemplo, pode ter duas imagens por segundo ou trinta imagens por segundo).

- *Vídeo*. É um conjunto de dados fotográficos seqüencializados temporalmente. Esses dados são produzidos através de um dispositivo como um registrador de vídeo digital. Os dados são divididos em unidades chamadas *frames*. Cada *frame* contém uma imagem fotográfica simples. Em muitos casos, vídeos registram de 24 a 30 *frames* por segundo (fps).

- *Áudio*. Em contraste com os tipos anteriormente apresentados, o tipo áudio possui características temporais. Uma interpretação compreensiva de um áudio é baseada em sua relação com uma escala constante e progressiva de tempo. A escala de tempo associa os dados de áudio, ou mais precisamente os componentes atômicos de um *stream* de áudio, com sua interpretação correta em algum ponto no tempo. Algumas manipulações, como execução e gravação, sempre serão associadas com uma escala de tempo. Dados de áudio geralmente utilizam técnicas de compressão devido à grande quantidade de dados.

- *Tipos Compostos*. Dado multimídia composto é criado através da combinação de tipos de dados multimídia básicos com outros dados multimídia básicos. Tipos podem ser misturados fisicamente para formar um novo tipo. Uma mistura física resulta em um novo formato de armazenamento, no qual os dados como áudio e

vídeo são misturados. Uma mistura lógica define um novo tipo de dado, mas mantém tipos de dados e formatos de armazenamento individuais.

- *Apresentações*. São objetos compostos complexos que também descrevem sincronização de dados multimídia para efeitos de modificação e apresentação dos dados. Descrevem uma ordenação temporal, onde se pode determinar que a apresentação mostre o vídeo v1, depois o vídeo v2 e assim por diante. Ou pode ser muito mais complexo, especificando como o usuário, sistema, e a aplicação de interação determinarão à apresentação final.

Uma aplicação ou documento multimídia comporta todos esses tipos de mídia. Com a utilização de sistemas hipermídia, os documentos (principalmente páginas HTML) estão se constituindo em grandes aplicações que estão sendo divulgadas na Internet através da *Web*.

3.2 Aplicações Multimídia na *Web*

As aplicações multimídia têm se tornado um dos componentes fundamentais para a interação de usuários e sistemas. Com o decorrer do tempo, as aplicações multimídia sofreram mudanças e adaptações para melhoramento e adaptação aos novos tipos de mídia.

O grande número de aplicações na *Web* é voltado ao uso de mídias especiais como as citadas na seção anterior. Documentos hipermídia funcionam como recipientes para objetos hipermídia, que são entidades transparentes de rede com uma variedade de tipos de mídia, como MPEG, GIF, RTF, etc. Um sistema hipermídia pode reunir os seguintes requisitos [Yu97] :

1. Sincronização temporal de alto nível entre objetos multimídia.

2. Provisão de sincronização espacial dependente de tempo dentro da arquitetura de documentos hipermídia.

3. Integração e manuseio transparente de vários tipos de mídia.

4. Ser transparente perante a rede, para recuperação de documentos e objetos.

5. Ser simples, mas possuir interface de usuário poderosa para manipular as informações temporais e espaciais dentro de documentos hipermídia.

Com a divulgação desses documentos na *Web*, os usuários requisitam mais recursos para o processamento de mídias nas páginas. Um desses recursos é a sincronização, onde estão envolvidos tempos para a apresentação das mídias, isto é, um documento multimídia é caracterizado não somente por seu conteúdo de natureza diversa: estáticas (textos, gráficos, imagens) ou dinâmicas (áudio, vídeo, animação), mas também pela organização temporal de seus elementos.

Os documentos tradicionais como os especificados em HTML, têm *layout* espacial estático e também elementos estáticos como textos ou imagens. Outros elementos de um *layout* como *video-clip* ou *clip* de áudio possuem procedimento temporal, mas não possuem métodos para especificar relações temporais entre elementos. Portanto, conforme citado no item 1 dos requisitos para um sistema hipermídia, deve-se definir a sincronização temporal para o documento que contém os objetos multimídia.

Dessa forma, uma outra classificação significativa para os tipos de dados multimídia pode ser feita através de suas relações temporais, assim um dado multimídia pode ser **dependente** (áudio, vídeo e animação) ou **independente** (texto, imagens, gráficos) em relação ao tempo. Uma escala de tempo é necessária para se associar dados dependentes com sua correta interpretação em qualquer ponto no tempo determinado pelos componentes atômicos do dado. Por exemplo, um vídeo tem como componentes atômicos *frames* que correspondem a intervalos iguais na escala do tempo. O tamanho de cada intervalo é determinado pela velocidade de gravação, como um vídeo de 30 *frames* / segundo.

Para utilização dos diversos tipos de mídias é necessário seguir as restrições de cada tipo básico (textos, gráficos, imagens, áudio e vídeo), bem como seguir novas restrições que surgem para os tipos compostos, além das restrições relacionadas com o aspecto temporal. Dentro desse contexto a principal restrição será tratada no próximo tópico, e diz respeito à sincronização dos dados.

3.3 Sincronização Multimídia

Dois tipos de sincronização entre mídias são citados em [Ehley95]: **contínua** e **dirigida por eventos**. A **sincronização contínua** relaciona-se com mídias contínuas como áudio e vídeo e requer constante monitoramento. Por exemplo, a sincronização dos

movimentos dos lábios com a fala em um vídeo. Já a **sincronização dirigida por eventos** é mais flexível e relaciona-se com apresentações do tipo slide.

Além do controle da sincronização entre mídias em uma apresentação multimídia, um outro tipo de sincronização diz respeito à disponibilidade das mídias, ou seja, ao se utilizar mídias de origens diferentes, como em um ambiente distribuído, é necessário realizar um controle sobre a disponibilidade da mídia no momento da apresentação, pois podem ocorrer atrasos na chegada dos dados devido à sobrecarga no ambiente de rede. Dentro desse contexto, foram propostas diferentes técnicas de sincronização multimídia conforme apresentadas em [Ehley95]. Essas técnicas implementam o controle da sincronização em vários locais na origem e no destino dos dados dependendo dos requisitos. A sincronização multimídia é classificada primeiramente como distribuída ou local. A sincronização distribuída implementa sincronização baseada em protocolos de redes e relaciona-se com vários computadores, enquanto que a sincronização local é usada para um único computador.

A maioria dos estudos realizados sobre sincronização multimídia foi direcionada para redes distribuídas porque quanto maior a distância que um dado tem de percorrer, maior será a chance de ocorrer atrasos. Devido a limitações das tecnologias das redes, e como dados multimídia possuem grande volume de dados, a maior parte das propostas de sincronização utiliza como base redes *broadband* (BISDN e ATM); isso ocorre porque esse tipo de rede possui, além da grande velocidade, um controle sobre a qualidade do serviço.

Já no que concerne à sincronização local, é necessário atentar à questão da armazenagem dos dados, pois devido ao tamanho de certos dados multimídia é necessário utilizar mídias com grande capacidade para o armazenamento, como *disk arrays* e *jukeboxes*. Dessa forma, controles adicionais para os dados têm de ser aplicados para que a mídia seja recuperada e apresentada de forma correta. Por exemplo, um vídeo, por possuir (normalmente) mais de 1GB, tem de ser recuperado de forma a não comprometer a qualidade da apresentação.

3.4 Páginas Dinâmicas HTML

Com a grande popularidade de servidores e *browsers Web*, os *sites* têm publicado seus bancos de dados na Internet/Intranet. Isso permite aos usuários recuperar e explorar itens do banco de dados de uma forma “dinâmica” com páginas HTML via *browsers Web*

padrão como: *Mosaic, Netscape* ou *Internet Explorer*. O conteúdo de uma página dinâmica é gerado através da inicialização de uma consulta ao banco de dados.

Uma vez que a página dinâmica HTML é gerada, é criada automaticamente uma área de *cache* local no cliente *Web*. Alterações nos itens do banco de dados não refletem numa página dinâmica HTML, e nem no armazenamento local (área de *cache* no cliente *Web)*. Quando um cliente referencia uma página dinâmica HTML gerada anteriormente, é então executada a recuperação do conteúdo da página na área de *cache* local, e caso haja modificações da página atual (isto é, página que pode conter mídias que contenham alterações realizadas no banco de dados) com a contida na área de *cache* local, serão automaticamente realizadas cópias dos dados atuais, isto é, dos itens do banco de dados, para a área de *cache* local, sendo esse processo denominado **problema de coerência de página** [Si98].

Atualmente existem três maneiras de abordagem para tratar o problema de coerência de página:

- Manualmente, recarregando à página do servidor *Web*, isto é, executando novamente a consulta.

- Através de *browsers Web* que são configurados com uma duração de *refresh*. Então uma página dinâmica pode ser recarregada após a duração de *refresh* configurada.

- Através de *browsers Web* que permitem uma página dinâmica ultrapassar a duração de *refresh* configurada, usando a *tag* <*meta*> ou o campo de cabeçalho *express.*

O problema de coerência de página, é semelhante ao problema de *cache* coerente em ambiente de banco de dados distribuído convencional. Em um *browser Web*, tanto a memória *caching*, como o armazenamento *caching* é empregado.

Quando o usuário inicializa a recuperação de uma página onde o seu conteúdo pode ser estático ou dinâmico, diferentes ações serão inicializadas pelo cliente, dependendo do tipo de solicitação [Si98]. Se o usuário inicializar a solicitação através do botão “*Back*” ou “*Forward*”, o cliente recuperará as páginas da memória ou do armazenamento *cache*, caso a

página tenha sido recuperada anteriormente. Se o usuário inicializa a solicitação através do botão "*Reload*", o cliente contata o servidor *Web* e recupera a página do servidor apenas se o servidor indicar que a página foi modificada desde a última recuperação. Se o cliente inicializa a solicitação através da combinação "*Shift-Reload*", o cliente recuperará a página de um servidor, desconsiderando se a página foi modificada. Se a página solicitada é gerada dinamicamente, ela será gerada novamente pela execução da consulta recuperada.

Em [Si98] é apresentada uma arquitetura para geração de páginas dinâmica HTML. A estrutura do processo para manter a coerência da página é distribuída entre um cliente *Web* e um servidor *Web*. A figura 3.1 ilustra um cliente *Web* contendo dois processos: *cache monitor* e *cache refresher*.

A primeira função do *cache monitor* é monitorar o estado de todos os itens do banco de dados que são capturados em uma página dinâmica HTML. Quando a consulta é inicializada para gerar página dinâmica, o *cache monitor* simplesmente informa ao *cache refresher* sobre a consulta. Durante a seção, quando o usuário está explorando o conteúdo da página dinâmica, o *cache monitor* continua monitorando o estado dos itens do banco de dados e informa ao *cache refresher* sobre todos os itens de *cache* inválido, isto é, aqueles em que o tempo de *refresh* (estimado de acordo com a probabilidade de mudança de um item no banco de dados) expirou. Quando algum item é modificado, o *cache monitor* informa ao *cache refresher* que contata o servidor *Web* para servir a solicitação. Então são retornados os itens de banco de dados alterados e *refresh times* para o *cache monitor* que altera a cópia *cache* para o armazenamento local no cliente.

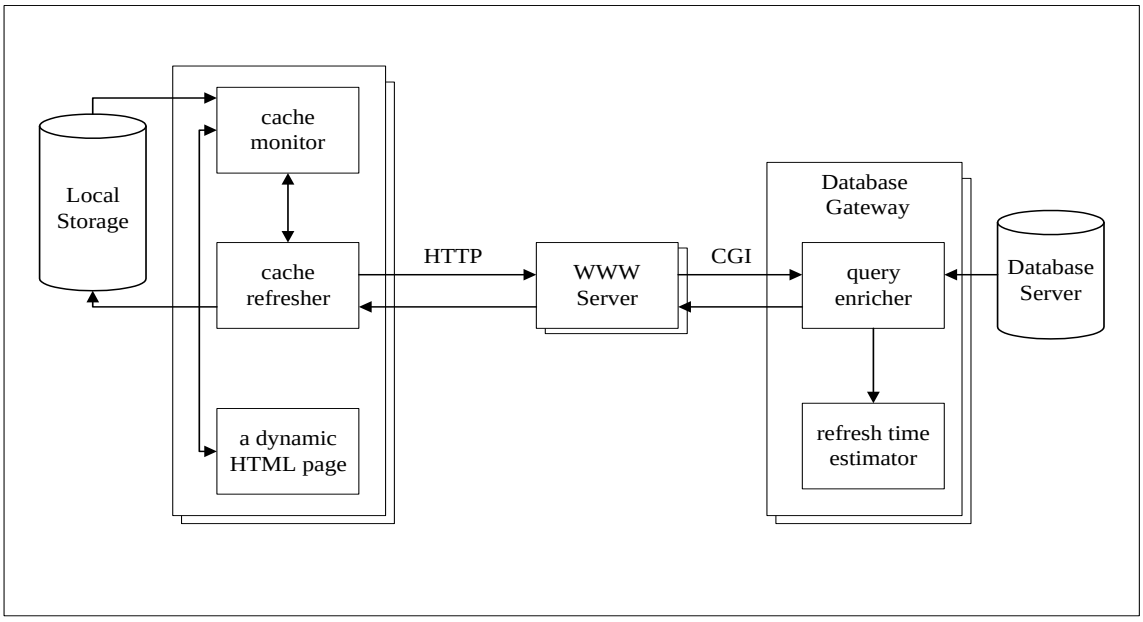


FIGURA 3.1 – ARQUITETURA E ESTRUTURA DE PROCESSOS PARA MANTER PÁGINAS COERENTES [SI98]

O servidor *Web* recebe a solicitação de um cliente e contata o *database gateway* através de um CGI para atender a solicitação. O processo do *database gateway* contém um *query enricher* e um *refresh time estimator*. O *query enricher* estende uma consulta de um cliente incluindo informações adicionais para que o *cache monitor* possa inicializar o processo que controla o tempo de *refresh*. O *refresh time estimator* estima um tempo de *refresh* para cada item do banco de dados de um servidor de banco de dados em resposta à consulta.

A tecnologia de *caching* é uma boa maneira para aperfeiçoar acesso rápido para informações *online* na *Web*. Se a informação é recuperada de um servidor *cache* próximo, diminui o caminho de acesso para a informação, e reduz a latência. Também minimiza acessos duplicados para a mesma informação, reduz o tempo para localizar os servidores primários e melhora a resposta do servidor [Nabeshima97].

3.5 Considerações Finais

Este capítulo teve como objetivo principal a caracterização de sistemas hipermídia e multimídia. Para isso, foram citados inicialmente alguns conceitos sobre esses sistemas encontrados em [Nielsen90] e [Harrison95]. Atenção especial foi dada às mídias que são encontradas em banco de dados, tais como: textos, gráficos, imagens, animações, vídeos, áudios, tipos compostos e apresentações.

Foram discutidos também fatores relacionados às aplicações multimídia na *Web*, sendo citados alguns requisitos abordados em [Yu97], que podem reunir uma aplicação hipermídia. A seção 3.3 foi destinada ao tratamento de sincronização de mídias, onde são citados os dois tipos de sincronização encontrados em [Ehley95]: a contínua e a dirigida por eventos.

Finalizando este capítulo foi citado o conceito, e as três abordagens existentes sobre o problema de coerência de página, que ocorrem em páginas *Web* consideradas “dinâmicas”. É citado como exemplo de processo de geração de páginas dinâmicas, o trabalho encontrado em [Si98], que apresenta uma arquitetura e toda a estruturação de processos distribuídos entre um cliente e um servidor *Web.*

No próximo capítulo, algumas considerações relevantes para a utilização de mídias armazenadas em banco de dados em páginas *Web*, são discutidas. Também são

especificados os diversos tipos de conexão existente entre SGBDs e *Web*, dando como suporte e exemplos, alguns trabalhos encontrados na literatura.

CAPÍTULO 4

Tipos de Conexão entre Bancos de Dados e *Web*

A integração da tecnologia de banco de dados e *Web*, denominada *Internet Databases*, vêm aumentando consideravelmente, e conseqüentemente a complexibilidade de armazenamento e gerência de objetos na *Web*. A *Web* possibilita a distribuição de informações, disponíveis em um banco de dados, a um grande número de usuários. Esses usuários podem acessar essas informações remotamente (Internet), ou apenas através de uma corporação (Intranet) e/ou também entre grupos corporativos (Extranet).

Devido a essa enorme potencialidade de integração, surgiram, ao longo dos últimos anos, diversos *softwares* para realizar a integração entre SGBDs e *Web*. Esses *softwares* são conhecidos como *gateways* ou *middlewares*. Quando um *gateway* é ativado, sua principal função é executar, junto ao SGBD, o pedido que lhe foi solicitado e devolver ao servidor *Web* ou, em alguns casos, diretamente ao cliente *Web*, o resultado do pedido em um formato especificado, normalmente (atualmente) em HTML.

Neste capítulo são apresentadas as principais formas de utilização de banco de dados na *Web*, ou seja, como são feitas as conexões entre o ambiente *Web* e os SGBDs. São também mostradas as vantagens e desvantagens da utilização de cada uma das abordagens. Como bibliografias principais para este tópico foram utilizadas [CGI95], [Cornell98], [Crawford97], [Denny98], [Ehmayer97], [Kim96] e [Mohseni96].

4.1 CGI (*Common Gateway Interface*)

As páginas *Web* são estáticas e para acessar um SGBD através delas, é necessária a utilização de um protocolo padrão chamado CGI [CGI95]. CGI é uma interface padrão para a execução de programas externos ao servidor *Web*. A interface descreve regras definindo como os programas externos são inicializados pelo servidor *Web* e como os dados gerados pelos programas são retornados ao servidor *Web*.

O CGI funciona da seguinte maneira, um *browser* requisita a execução de um programa no site do servidor. Essa aplicação externa realiza o acesso ao SGBD, utiliza os parâmetros de entrada, processa e devolve os resultados para o servidor, que devolve para o *browser* requisitante. Esse tipo de programa permite que ele seja feito em qualquer linguagem (as mais usadas são Perl, C, Delphi e Visual Basic). Essa é uma das grandes vantagens do CGI, a liberdade de escolha da linguagem e do banco de dados.

Existem algumas desvantagens acerca da utilização de CGI; a principal reside no fato de que cada chamada cria um processo no servidor, assim, com muitas chamadas simultâneas, o servidor pode ficar sobrecarregado. Isso pode ser simplificado através da utilização de apenas uma sessão para acesso ao SGBD. Quanto à questão de várias chamadas, isso pode influir também na questão de concorrência na utilização do SGBD. Isso ocorre devido à natureza do protocolo HTTP, pois cada vez que o CGI é executado ele abre uma nova sessão e processa a chamada. Nesse contexto vários problemas de concorrência podem ocorrer. Um dos problemas relevantes diz respeito ao fato de um *browser*, ao fazer uma segunda chamada ao servidor, são criados um novo processo e uma nova sessão, pelo fato do servidor não ter como identificar a chamada.

Outro problema relaciona-se com a ocorrência de uma eventual desconexão (*crash*) do *browser*; com isso, como não há uma conexão direta entre o *browser* e o SGBD, este manterá uma sessão aberta mesmo após o *browser* ter se desconectado.

Uma forma de diminuir esses problemas é através da utilização de *cookies*, que são mecanismos usados tanto pelo servidor quanto pelo *browser* para armazenar informações acerca do estado atual do *browser*. Com isso, pode-se, por exemplo, recarregar as informações do *browser* antes de se criar uma nova sessão no SGBD.

4.2 API (*Application Program Interface*)

APIs são formadas por um conjunto de funções que possibilitam estender as funcionalidades dos servidores *Web* e melhorar a performance de aplicações externas ao servidor *Web*. Nesta arquitetura, uma transação *Web* se inicia com uma requisição do cliente *Web* para a execução de uma aplicação API, que na maior parte dos casos compartilha recursos de memória com o servidor *Web*. O servidor *Web* inicia a execução da aplicação API se ela ainda não estiver ativa em memória e repassa os dados enviados pelo cliente *Web* de acordo com a implementação API do servidor *Web*, ou seja, não se usa a interface CGI.

O desempenho é potencialmente bom, pois o servidor *Web* e a aplicação API compartilham recursos de memória e, devido ao fato dos programas APIs ficarem residentes, eles são executados mais rapidamente do que os programas CGI. A principal desvantagem do uso de APIs, reside no fato de que a tarefa de codificação é demasiadamente complexa, pois o programador necessita possuir bom conhecimento sobre o funcionamento e sobre a implementação da API de determinado servidor *Web*. Dessa forma, a arquitetura é muito dependente da interface API do servidor *Web*, além disso, a aplicação não é portável para outros tipos de servidores *Web*.

4.3 SSI (*Server Side Include*)

Uma outra forma de se conectar um SGBD com a *Web* é utilizar um servidor que consiga tratar os comandos direcionados ao SGBD. Assim os comandos podem ser vinculados a comentários em HTML ou utilizando *tags* especiais, sendo que ambos são ignorados pelo *browser* e o servidor faz o acesso ao SGBD quando disponibiliza os documentos ao *browser*. Como as funções estão armazenadas no servidor, este mecanismo é chamado de SSI (*Server Side Include*).

Os problemas relacionados a transações são os mesmos que os já citados para CGI. Outra desvantagem é que a conexão com o SGBD só existe enquanto está sendo feita a interpretação dos arquivos HTML. Além disso, não há padrões para o acesso ao SGBD, assim um servidor dedicado é necessário. Uma vantagem é que implementando o SSI utilizando padrões como ODBC (*Open Database Connectivity*), pode-se conectar a qualquer SGBD relacional (Oracle, Sybase, Informix, etc). Outra vantagem, e talvez a principal, é que o desempenho é potencialmente maior, pois o acesso é feito diretamente ao SGBD.

Fazer com que o SGBD interprete HTML, pode ser outro mecanismo de conexão, assim ocorre uma integração do servidor com o SGBD. A principal vantagem desse mecanismo é que pode ser feito um controle maior sobre a concorrência e sobre transações, pois o SGBD é quem recebe as chamadas.

4.4 Java via JDBC

A utilização de JDBC (*Java Database Connectivity*), trouxe novas possibilidades para a conexão. Também chamado de CSI (*Client Side Includes*), este mecanismo permite que *JavaApplets* acessem SGBDs. Nesse contexto, os componentes de conexão funcionam em qualquer *browser* que suporte a utilização de Java/JavaScript. O mecanismo utiliza o

princípio de o *browser* carregar, via *Web* ou localmente, um programa dedicado chamado *JavaApplet* ou *applet*. O *applet* será o responsável pela manipulação do banco de dados. Com isso não é necessário utilizar um servidor para o acesso ao SGBD, tendo em vista que o controle da sessão é feito pelo *browser*.

A utilização desse mecanismo faz com que o controle de transações e concorrência não sejam mais o principal problema. Além disso, com a utilização de Java, todas as suas facilidades são também compartilhadas, como a independência de plataforma e a integração com dados HMTL. Outra grande vantagem é a interatividade que Java trata para os dados *Web*, apesar de que para se utilizar recursos mais poderosos de interatividade é necessário uma programação um pouco mais robusta. Além disso, JDBC foi concebido de forma a permitir o acesso a vários SGBDs, facilitando a integração de SGBDs relacionais.

Devido à questões de segurança, os *applets* (para versão 1.0.2 e posteriores) podem se conectar somente com o servidor ao qual eles foram executados. Com isso o servidor só poderá se conectar com um banco de dados que esteja fisicamente na mesma máquina.

A integração entre *applets* e HTML é realizada pelo intercâmbio de informações entre *applets* e JavaScript, permitindo assim que JavaScript receba as informações e HTML as apresente. Quando um documento HTML é carregado, os comandos do *script* são executados pelo *browser* antes do HTML ser interpretado, sendo que o controle do SGBD é de responsabilidade do *applet*. Como o *applet* permanece persistente no ambiente do *browser*, ele pode servir a qualquer número de documentos utilizando apenas uma conexão com o SGBD. Para o acesso ao SGBD, o protocolo de conexão nativo pode ser usado pelo *driver* JDBC, e para aplicações nas qual a segurança dos dados é crítica, mecanismos de criptografia podem ser usados.

4.5 Java via *Servlets*

Servlet é uma arquitetura desenvolvida pela *Sun* e tem como principal característica ser um conjunto de classes 100% Java que funciona basicamente como se fosse CGI. Dessa forma, um *servlet* é uma versão *server-side* de um *applet*, ou seja, um pequeno código em Java que é carregado por um servidor *Web*, e usado para tratar requisições de clientes da mesma forma que CGI .

Uma grande desvantagem da arquitetura CGI reside no fato de se criar uma conexão para cada chamada do cliente. Assim, para um site que é pouco acessado (por volta de centenas de acessos diários) não há grandes problemas. Porém para sites nos quais ocorre

um acesso por segundo, esse tipo de arquitetura é demasiadamente problemático, pois, somente para se criar uma sessão, acessar o SGBD e retornar o resultado para o cliente geralmente leva-se mais de um segundo. Já no caso da arquitetura de *servlets* é possível se criar uma única sessão para todos os clientes.

As principais vantagens da arquitetura de *servlets* sobre CGI são:

- **Independência de plataforma:** por ser 100% Java um *servlet* pode ser executado em qualquer plataforma, sem necessidade de recompilação. Os *scripts CGI* feitos em Perl também podem ser executados em outras plataformas, já CGIs escritos em outras linguagens como C, C++ e Delphi não são portáveis.

- **Desempenho**: *servlets* necessitam ser carregados somente uma vez, enquanto que programas CGI têm de ser lidos a cada requisição. O método **init()** permite a execução de ações, como conexão com o SGBD, no momento de inicialização e o reuso dela através da chamada ao *servlet*. Assim, ao invés de conectar com o SGBD várias vezes, a conexão ocorre somente uma vez. Somente esse fato já compensa a questão do código Java ser mais lento que um código nativo.

- **Modularidade**: *servlets* são modulares, cada *servlet* pode executar uma tarefa específica, sendo que se pode ligar um *servlet* a outro. Também chamado de corrente de *servlets* (*servlet chaining*), essa técnica possibilita que o resultado de um *servlet* possa ser processado por outro *servlet*. Uma vantagem é que um *servlet* não precisa saber que está sendo usado dessa forma, ou seja, sua construção é a mesma de um servlet comum.

4.6 Usando Visualizadores Externos

A utilização de visualizadores externos diz respeito à utilização dos dados no formato em que são armazenados; assim, no momento de leitura, um visualizador externo é executado. Nesse mecanismo o controle é passado do *browser* para o visualizador; sendo assim, o visualizador é quem faz o acesso ao SGBD. A utilização desse mecanismo satisfaz todo o controle de transações e concorrência. Além disso, benefícios relativos a questões de desempenho também são importantes, pois o servidor simplesmente responde à requisição do *browser* enviando um documento codificado no formato MIME (*Multipurpose Internet Mail Extension*). O *browser* ao receber o documento não possui capacidade de interpretá-lo,

assim executa um visualizador externo. O problema é que a integração da *Web* com o SGBD fica distante, e é necessário instalar o visualizador em cada site.

4.7 Estendendo a capacidade do browser utilizando *Plug-Ins*

Plug-ins são programas que estendem as capacidades dos *browsers*, de forma a possibilitá-los a tratar diferentes dados, como por exemplo, a apresentação de um vídeo. Os *plug-ins* são produzidos de forma a utilizar APIs fornecidas pelos *browsers*, por essa razão cada *plug-in* depende da plataforma e do *browser* a ser utilizado. Uma outra desvantagem de se utilizar *plug-ins* é que eles têm de ser instalados em cada *browser*. Uma vantagem de se utilizar *plug-ins* é que não é necessário ativar um programa externo, com isso o controle sobre os dados continua sendo feito pelo *browser*.

4.8 Serviços baseados em *Proxy*

Este mecanismo utiliza servidores *proxy* HTTP para fazer o acesso ao SGBD. A idéia central está em redirecionar a requisição do *browser* através de um servidor *proxy* que faz o acesso ao SGBD e envia os resultados ao *browser*. Os comandos de manipulação do SGBD têm de estar vinculados com os documentos HTML; o servidor *proxy* os reconhece e os executa através de um *parser*, na requisição do *browser*.

4.9 Exemplos de Conexão entre SGBDs e *Web*

Conforme citado na seção anterior, existem diversos tipos de conexão entre SGBDs e *Web*. Nesta seção são citados alguns trabalhos encontrados na literatura, que fazem conexão entre SGBD e *Web*. Cada trabalho utiliza um paradigma diferente de SGBD, isto é, desde SGBD relacional até SGBD objeto-relacional, e orientado a objetos.

4.9.1 Banco de Dados Relacional / *Web*

Em [Nguyen96] é proposta uma solução para criar aplicações *Web* que acessam SGBD Relacional (SGBDR), sendo abaixo citadas as suas principais características:

1. Novas aplicações devem ser fáceis de construir, preferivelmente eliminando esforços significativos de codificação que pode estar envolvida.

2. Aplicações devem ser de fácil manutenção e de facilidade para adaptação a novas versões HTML.

3. A capacidade da linguagem HTML para criação de formulários de entradas e de saída deve estar disponível para o desenvolvedor de aplicações. O ideal é fornecer ao usuário um editor HTML visual para a construção de formulários HTML.

4. A capacidade da linguagem SQL para acessar o SGBDR deve ser disponível usando uma ferramenta de consulta visual para a construção de consultas SQL necessárias para acessar o SGBDR.

5. Deve existir um mecanismo para transferir variáveis de entradas de um cliente *Web* (isto é, o usuário) para a consulta SQL (ou consultas) para acesso ao SGBDR.

6. Deve existir mecanismo para substituir o resultado de uma consulta SQL em formulários HTML para a visualização do resultado.

7. Permitir consultas adicionais e formulários HTML, possivelmente através de *hyperlinks* embutidos nos relatórios.

Também é proposta uma solução para criação de aplicações *Web* que acessam SGBDRs usando um paradigma de *layout* de página, que encapsula HTML e SQL. Para interligar HTML e SQL é proposto um mecanismo de substituição de variáveis que fornece substituição de variáveis de linguagens híbridas entre entradas HTML e caracteres de consulta SQL, bem como entre colunas resultantes de SQL e saídas HTML. Esse mecanismo tem sido utilizado em projetos e implementações do sistema denominado *DB2 WWW Connection*. Os desenvolvedores de aplicações desenvolvem formulários HTML e comandos SQL, e os armazenam em arquivos (chamados macros) no servidor *Web*. Esses macros conseguem ser processadas em um *engine run-time DB2 Connection*. Já que *DB2 Connection* usa HTML nativo e linguagens SQL, várias ferramentas podem ser utilizadas para a criação de formulários HTML e para geração de consulta SQL. O usuário final apenas interage com o sistema através de *clicks* para navegar entre formulários que acessam banco de dados.

Quando o usuário utiliza um formulário de entrada de dados, essas informações são enviadas até o servidor *Web* através de uma variável de ambiente chamada *QUERY_STRING*. Na figura 4.1, é ilustrado o fluxo de dados usando CGI, onde são

apresentados dois diferentes cenários, que mostram que um programa chamado *DB2 WWW* é inicializado duas vezes com entradas diferentes. Na primeira entrada (1) o usuário através do *Web browser* solicita ao *Web Server* o fornecimento de formulários. O *Web Server* envia a solicitação do usuário ao *DB2 WWW*, e este retornam um formulário de entrada HTML (2). Quando o usuário necessita consultar informações contidas no banco de dados, ele envia uma solicitação ao *Web Server*, que reenvia ao *DB2 WWW* (3). Para consultar os dados requisitados pelo usuário, o *DB2 WWW* realiza uma conexão com o *DB2 database*. A consulta é retornada para o *DB2 WWW* através de resultados em SQL, onde são transformados em formulários de relatórios ao usuário (4).

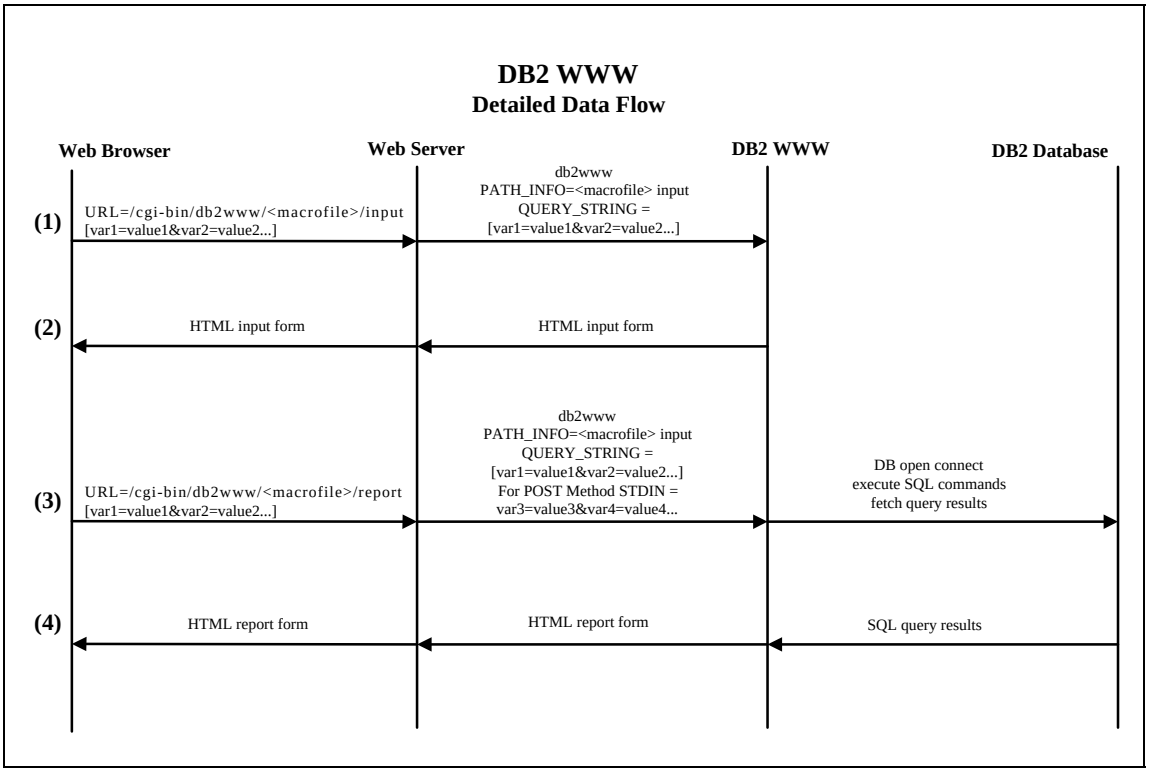


FIGURA 4.1 – O FLUXO DE DADOS USANDO CGI E SGBDR [NGUYEN96]

O *DB2 WWW Connection* (*DB2 WWW*) pode ser usado para acessar banco de dados IBM DB2 sob diversas plataformas. Na figura 4.2, é ilustrada uma visão geral do ambiente em que o *DB2 WWW* é executado. O *DB2 WWW* pode ser inicializado em uma página HTML basicamente de duas formas:

```
1. <A HREF =
       http:[{web-server}]/cgi-bin/db2www[.exe]/
       {macro-file}/{cmd}[?name=val& . . . ]>
```

```
2. <FORM METHOD = "post" ACTION =
       http:[{web-server}]/cgi-bin/db2www[.exe]/
       {macro-file}/{cmd}[?name=val& . . . ]>
```

onde,

{web-server} : é o nome de um servidor *Web*.

{macro-file} : é o nome do arquivo que armazena um arquivo macro definido pelo desenvolvedor de aplicações. Veja que na figura 4.3, existem editores HTML e ferramentas de consultas que podem ser usados pelos desenvolvedores para a criação de novos macros.

{cmd} : é opção de entrada ou saída. Se {cmd} é *input*, então a seção de entrada HTML do *macro-file* é processada, caso contrário, isto é, se {cmd} for *report*, a seção de relatório do *macro-file* é executada.

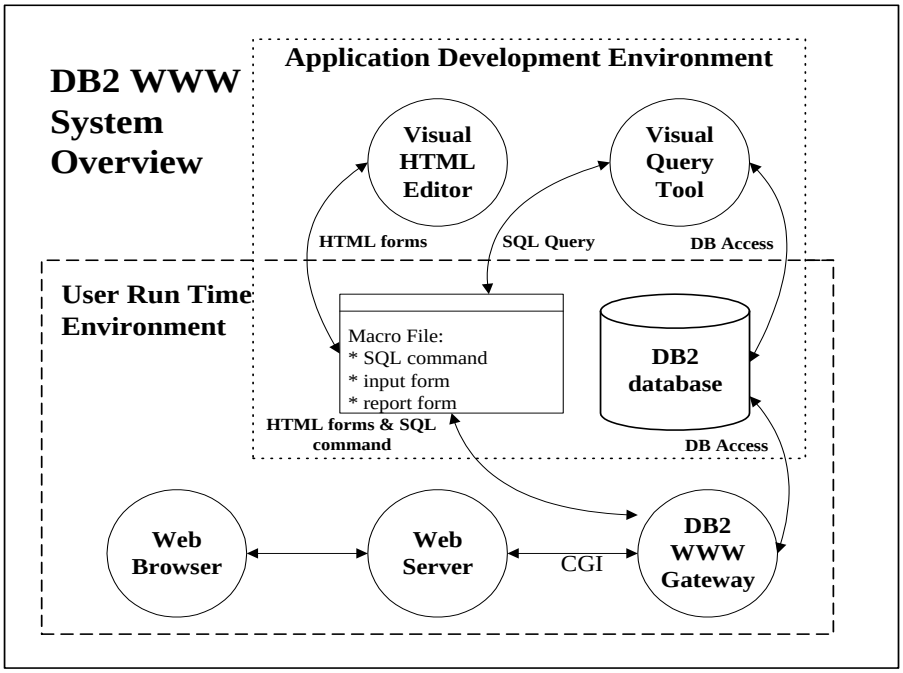


FIGURA 4.2 – VISÃO GERAL DO SISTEMA DB2 WWW [NGUYEN96]

O *DB2 WWW system* é dividido em: *Application Development Environment* e *User Run Time Environment*. No *Application Development Environment*, o usuário pode definir formulários HTML através de um editor HTML visual. O usuário também pode consultar e gerar comandos SQL através de uma ferramenta de consulta. Tanto as consultas SQL como os formulários HTML são armazenados em *macro-file*. Já no *User Run Time Environment*, o usuário, através do *Web Browser,* pode consultar dados enviando uma solicitação ao *Web Server*. Quando o *Web Server* recebe a solicitação, ele a envia ao *DB2 WWW Gateway* através de um programa CGI. O *DB2 WWW Gateway* acessa o *DB2 database* e retorna a consulta ao usuário através de um *macro-file* que contém formulários HTML juntamente com comandos SQL.

4.9.2 Banco de Dados Objeto-Relacional / *Web*

Para permitir que usuários modelem e manipulem dados não convencionais, de forma eficiente, e que usem tecnologias orientadas a objetos para desenvolvimento de aplicações, com todos os benefícios de SQL e com todas as características de domínio comercial, recentemente quase todos os vendedores de banco de dados relacionais têm redirecionado suas estratégias para objeto-relacional e, conseqüentemente, suas arquiteturas de servidores de banco de dados [Härder97]. O uso dessa técnica, objeto-relacional, surgiu devido a várias razões:

- Suporte para um complexo e extensível conjunto de tipos de dados para gerenciamento de informações técnicas, multimídia, dados espaciais e temporais.

- A habilidade para permitir conexões *Web* dinâmicas: facilidade para definir HTML como novo tipo de dado que permite que o servidor de banco de dados o entenda, pesquise-o, e gere dinamicamente páginas HTML.

- Suporte para usuários através de uma interação amigável e controle de consistência. Atualmente, regras ativas têm sido uma das principais propriedades de técnicas objeto-relacional.

- Uma arquitetura flexível e extensível, que seja facilmente tolerada para demandas de domínio específico.

Em [Härder97] são especificados dois exemplos de conexão entre sistemas de gerenciamentos de banco de dados objeto-relacional (SGBDOR) e *Web*. Esses dois exemplos de conexão são: CGI e *JavaApplets.*

CGI

O CGI permite ao servidor *Web* transferir dados submetidos de um *browser Web* para um programa residente no servidor (veja figura 4.3, seta (1) e (2)), para executá-lo, e transmitir a saída resultante para o *browser*. Cada programa localizado no servidor pode ser um cliente de banco de dados que estabelece uma conexão de banco de dados, submetendo uma consulta para o servidor de banco de dados (3), recebe o resultado (4) e converte em uma página HTML (5), que então é transferido para o *browser* (6).

Apesar das desvantagens do CGI citadas anteriormente, pode-se citar algumas vantagens fornecidas pelo acesso a SGBDOR via programas CGI, que são:

1. Utilizando formulários e tabelas HTML, um ***GUI*** (*graphical user interface*) para entrada e visualização de dados pode ser desenvolvido de uma maneira direta.

2. Para estabelecer um banco de dados *front-end*, uma única página HTML é suficiente para a transferência para o *browser*.

3. Por haver apenas programas CGI e páginas HTML residentes no servidor *Web*, todas as alterações podem ser feitas em um único local, tendo assim, custos mínimos de administração.

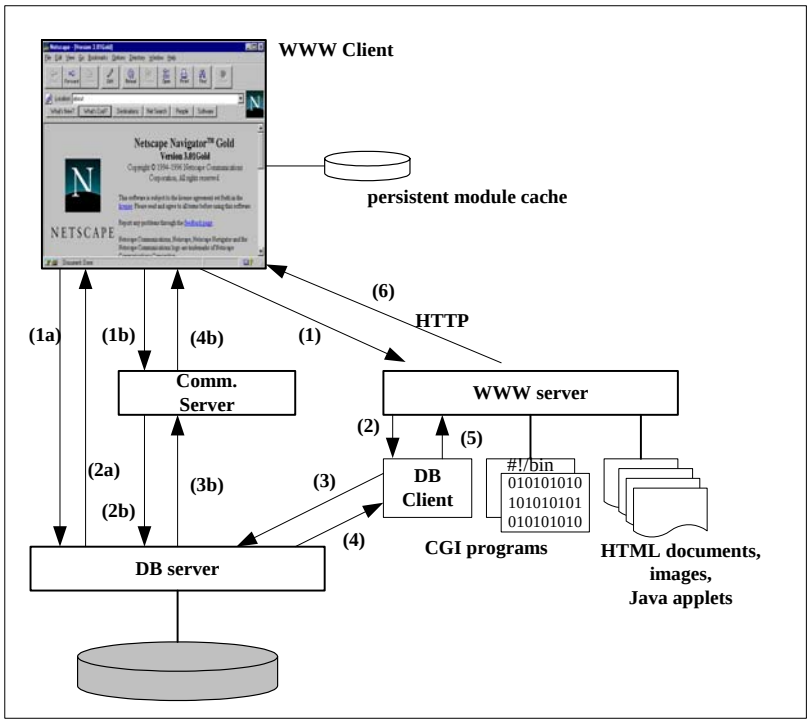


FIGURA 4.3 – VISÃO GERAL DE ACESSO A BANCO DE DADOS ATRAVÉS DA WEB [HÄRDER97]

JavaApplet

O *JavaApplet*, isto é, um programa Java embutido em uma página HTML é transferido para o *browser* utilizando uma conexão normal HTTP ((1) e (6), veja figura 4.3), e é capaz de realizar uma conexão para um servidor de banco de dados residente numa máquina que tenha um servidor *Web*. Existem várias maneiras diferentes para acessar um banco de dados através de um *applet* [Härder97]:

- A primeira necessita a utilização do JDBC, uma chamada ao nível de interface (CLI) baseada em SQL, definida pela linguagem Java. Alguns *drivers* JDBC permitem ao *applet* conectar diretamente aos servidores de banco de dados (1a e 2a), outros usam um Servidor de Comunicação especial localizado entre o *applet* e o servidor de banco de dados (1b a 4b).

- Outra técnica é baseada em CLIs proprietárias, por exemplo, MsqlJava para mSQL ou J/OCI para Oracle.

- Usando um *applet* como um cliente CORBA (*Common Object Request Broker Architecture*). Nesse caso, o Servidor de Comunicação ilustrado na figura 4.3, será um ORB (*Object Request Broker*).

Um programa *JavaApplet*, oferece várias vantagens para o acesso ao banco de dados: por causa da conexão direta com o banco de dados, possibilita ter transações "longas", isto é, consistindo de mais de uma consulta; possibilita suportar aplicação lógica em *sites* clientes, isto é, o servidor *Web* se aliviará de algumas tarefas de processamento.

Para aplicações com poucas freqüências, HTML e programas CGI são adequados. Os programas Java podem ser integrados em páginas HTML, e a interface do usuário e o *browser Web* pode ser usado para manipulação de dados, para aplicações lógicas e controle de *workflow*. Pode também fazer o uso de ambas as técnicas descritas acima.

4.9.3 Banco de Dados Orientado a Objetos / *Web*

Alguns sistemas de gerenciamento de dados banco de dados orientados a objeto (SGBDOOs) realizam conexão direta entre os banco de dados e a *Web*, como exemplo, *Jasmine* e *Poet*. Nessa seção é exemplificada uma arquitetura criada para suportar conexão entre um SGBDOO e a *Web*, descrita em [Yang96].

Em [Yang96] é proposta uma arquitetura para um sistema que suporta visões customizadas dinamicamente. Visões *Web* são estruturas abstratas impostas na *Web*, onde diferentes usuários podem ter visões diferentes de uma mesma *sub-web*; o mesmo usuário pode ter diferentes visões em diferentes tempos. A arquitetura proposta que está ilustrada na figura 4.4, consiste de três componentes principais: *Http Implementation*, *View Processor* e o *Object-Oriented Database*. Esses componentes são descritos abaixo.

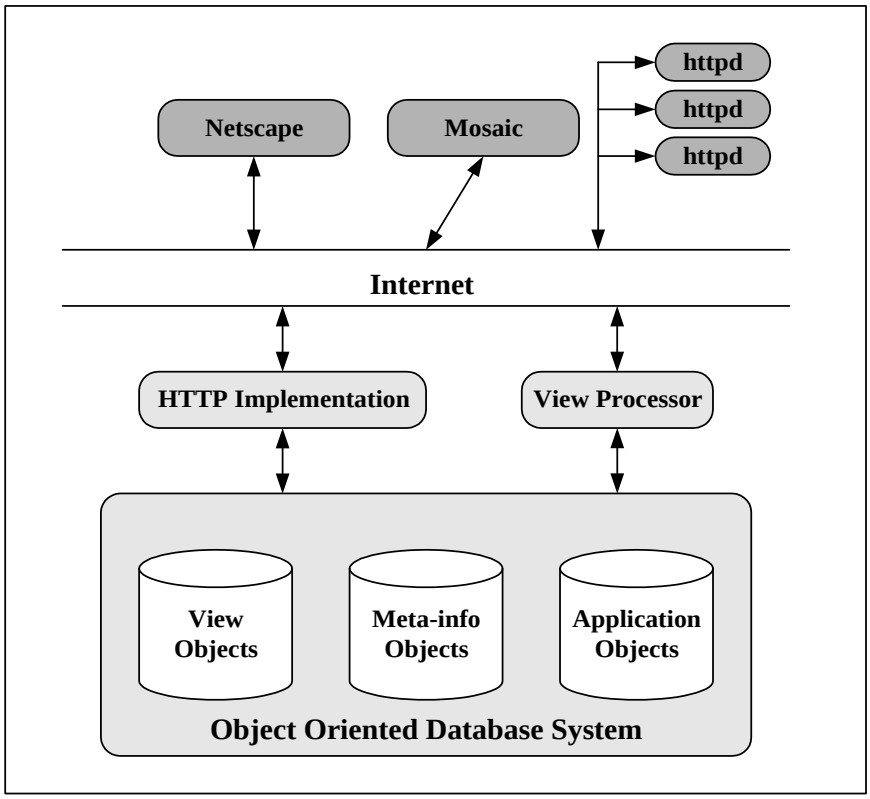


FIGURA 4.4 – ARQUITETURA DE UM SISTEMA DE SUPORTE A VISÕES WEB [YANG96]

- ***HTTP Implementation*:** Esse componente recebe solicitações de *browsers* através do protocolo HTTP. Envia essas solicitações a outras partes do sistema para a obtenção de dados apropriados e os envia como resposta para os *browsers*.

- ***Object-Oriented Database*:** Três conjuntos diferentes de objetos são armazenados nos três bancos de dados: *View Objects, Meta-info Objects e Application Objects*.

  - ***View Objects*:** Esses objetos armazenam definições de visões. Uma visão é definida em um formato HTML e inclui consultas a *objectbase* embutido que gera *hyperlinks* para objetos *web*.

  - ***Meta-Information Objects*:** Esses objetos modelam uma parte da *web* de interesse. Meta-informação pode incluir informações sobre o objeto *web* correspondente que são representadas como atributos de *meta-information objects*

- ***Application Objects*:** Esses objetos são aplicações específicas e não são armazenadas na *web*.

- ***View Processor*:** O *view processor* é uma máquina de processamento central do sistema. Faz a leitura de definições de visões de um *objectbase* e executa as consultas embutidas para obter os objetos necessários para construir uma visão. Contata os servidores HTTP remotos para recuperar objetos *web* e consultas em *objectbase* local para objetos de aplicação. Para suportar alterações de dados na *web* dinamicamente, o *view processor* deve ser capaz de manusear situações como, por exemplo, não disponibilizar objetos *web* por causa de falhas da rede.

A arquitetura do sistema [Yang96] é ampliada com *métodos embutidos* em objetos *Web*. Um método embutido é definido em um formulário de uma "tag desconhecida" em HTML. As especificações HTML definem que "tags desconhecidas" são ignoradas pelos *browsers*, portanto não são vistas através de um acesso normal. Mas quando o objeto *Web* é acessado via um *View Processor*, este entenderá como informação trazida por *métodos embutidos*, que podem ser um valor de atributo que descreve características de um objeto *Web*, ou uma operação de alteração em cada banco de dados orientado a objetos. Abaixo é mostrado um exemplo simples.

```
<TITLE> Exemplo de Metodos Embutidos </TITLE>
  <H1> Exemplo de Metodos Embutidos </H1>
   <EMETH type=attribute name=autor value=jyang@cs.columbia.edu>
   <EMETH system=DKWEB type=operation name=NewComment
      npara=2 para1=from para2=content method=GET
      operator=http://www.cs.columbia.edu/~jyang/cgi=bin/sendmail.cgi>
```

Para a construção de uma Visão *Web*, são necessários vários passos, sendo eles:

1. ***Aprovação da Solicitação***: O *HTTP Implementation* recebe uma solicitação de um *browser* cliente em um formulário que ativa um método POST ou GET. Métodos GET são usualmente usados para obtenção de visões pré-definidas e métodos POST para modificar visões. O *HTTP Implementation* traduz a solicitação em consultas para banco de dados orientado a objetos (OODB) ou comandos de gravação, e envia a consulta ou gravação para o OODB ou para o *View Processor*, dependendo do tipo de solicitação.

2. ***Recuperação/Alteração de Visão***: O OODB obtém solicitações do *HTTP Implementation* e recupera/altera objetos visão. Se a solicitação é a alteração de uma visão, o OODB informa ao *HTTP Implementation* o resultado e replica para o *browser* cliente, finalizando a operação. Caso contrário, os dois passos seguintes são invocados:

    1. ***Construção de Visão***: Depois que o objeto visão é recuperado de um *objectbase*, este é enviado ao *View Processor*, que analisa a definição da visão e recupera informações necessárias em *objectbase* local ou na *Web*. O *View Processor* consulta o *objectbase* para a localização da origem dos objetos.

    2. ***Resposta***: O *View Processor* então fornece a visão processada para o *HTTP Implementation*, e a visão processada é enviada até o *browser* cliente.

Abaixo é demonstrado um exemplo de como objetos *view* e *meta-information* são definidos:

```
View : : superclass ENTITY;
      owner             : User;
      public            : boolean;
      content           : text;
end

MetaInfo : : superclass ENTITY;
      autor             : string;
      URL               : string;
end

# example de subclasses de MetaInfo

DocRoot : : superclass MetaInfo;
      System            : string;
      chapters          : set_of Chapter;
end

Chapter : : superclass MetaInfo;
      chapter_number    : integer;
      module            : link Module;
      sections          : set_of Section;
end
```

No exemplo acima, é definida uma classe *View* que é herança da classe *ENTITY*. Na classe *View* são definidos três atributos: *owner, public* e *content*. Também é definida uma classe chamada *MetaInfo* que possui uma herança da classe *ENTITY*. A classe *MetaInfo* possui dois atributos: *autor* e *URL*. Completando o exemplo acima, são definidas duas subclasses da classe *MetaInfo*: *DocRoot* e *Chapter*. A *DocRoot* possui dois atributos:

system e *chapters* (que é um conjunto de objetos *Chapter*). A *Chapter* é definida para conter várias seções. Seus atributos são: *chapter_number*, *module* e *sections*.

No *objectbase*, a definição da visão é armazenada para cada instância da visão. O atributo ***content*** então contém a definição da visão escrita em um HTML embutido. Abaixo é um exemplo de definição de visão:

```
<TITLE> Example of View Definition </TITLE>
 <H1> Example of View Definition </H1>
   <HR>
    This example view is a view to the software engineering environment.
    The view contains the chapters and sections of the design document.
   <P>
  <DKOV_QUERY bind doc_root DOC_ROOT where(doc_root.system =="samplesys");>
  <DKOV_QUERY bind chap CHAPTER where (member doc_root.chapters);>
    The document pf <DKOV_QUERY print doc_root> system contains the
    following chapters;
  <DKOV_QUERY print chap>
```

No exemplo acima é criada uma visão que mostra um documento e seus respectivos capítulos. A visão é inicializada através da definição de duas variáveis: *doc_root* e *chap*, que é o documento e seus capítulos respectivamente. Todos os comandos para definir a visão, são definidos através da *tag* inicial *DKOV_QUERY*. Na final da definição da visão, o documento e seus capítulos são ilustrados através do comando *print*.

Como outro exemplo de conexão entre SGBDOO e *Web*, pode-se citar os recursos oferecidos pelo *O_2Web* [$O_2$96]. A arquitetura do *O_2Web* é ilustrada na figura 4.5. O *O_2Web* compreende três principais elementos:

1. *O_2Web gateway*. É um programa CGI instalado no servidor Web, que verifica o arquivo que contém o nome ou o endereço da máquina que está rodando o *O_2Web dispatcher*, por exemplo, “/etc/o2web”.

2. *O_2Web dispatcher*. É responsável por procurar na rede a máquina em que está rodando o *O_2Web server.*

3. *O_2Web server*. É o elemento que realiza a conexão com o *O_2 server*.

Com esses três componentes, é possível realizar consultas ao banco de dados orientado a objetos *O_2*. O sistema *O_2Web* realiza suas funcionalidades de acordo com as seguintes fases:

1. O cliente através de um *browser* na *Web*, envia o URL em formato HTTP. Esse URL contém uma consulta OQL.

2. O servidor HTTP passa a consulta para o *O_2Web gateway*.

3. O *O_2Web gateway* conecta a um *O_2Web dispatcher* que está sendo executado em uma área de rede local.

4. O *O_2Web dispatcher* informa ao *O_2Web gateway* qual o *O_2Web server* que ele deve se conectar.

5. O *O_2Web gateway* conecta ao *O_2Web server* informado.

6. O *O_2Web server* executa a consulta especificada no URL, e transforma os resultados dessa consulta em HTML.

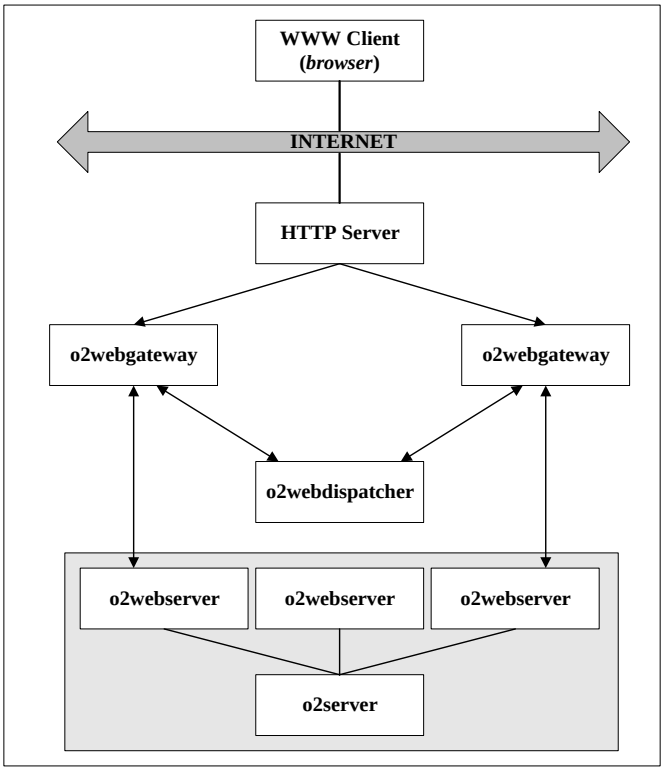


FIGURA 4.5 – ARQUITETURA DO O_2WEB [$O_2$96]

O *O_2Web* pode ser utilizado em diferentes níveis, que são responsáveis pela geração do resultado final apresentado ao cliente através de arquivos HTML:

1. Modo completamente automático (modo genérico). O *O_2Web server*, gera automaticamente código HTML, como resultado de uma consulta contida no URL.

2. Modo parcial, onde se pode programar partes globais da geração de código HTML. Um cabeçalho e um rodapé podem ser inseridos nas páginas. Pode também permitir eventos como: conexão ou desconexão com o servidor, ou ocorrência de erros.

3. Modo manual, que permite total controle sobre a geração de código HTML para cada classe do esquema.

4.10 Considerações Finais

Considerando a existência de várias formas de realizar conexão entre SGBDs e *Web*, um estudo literário aprofundado foi necessário para a obtenção de conhecimentos sobre as técnicas atualmente aplicadas, assim como tentar viabilizar para o projeto, alguns desses critérios abordados.

Este capítulo se iniciou com uma abordagem das várias técnicas aplicadas para realizar conexão entre SGBDs e *Web*, citando algumas vantagens e desvantagens.

Foram citados também neste capítulo, vários exemplos de projetos encontrados na literatura, que possuem critérios que abordam conexão entre SGBD e *Web*. Esses exemplos foram classificados de acordo com três paradigmas de SGBD: relacional, objeto-relacional e orientado a objetos. A contribuição desses exemplos foi na realização de uma análise sobre os critérios de implementação adotados, políticas de controle do fluxo de informações, distribuição da implementação entre o cliente e o servidor *Web*, utilização / criação de *tags* com funcionalidades específicas que podem ser empregadas em documentos *Web* (que serão interpretadas por programas específicos existentes nos servidores *Web)* e também sobre os critérios de armazenamento das informações a serem disponibilizadas. Toda essa análise foi determinante para a elaboração e construção da arquitetura de conexão do projeto, que será explicada no capítulo 6 e no apêndice 1.

Enfim, este capítulo contribuiu muito para o desenvolvimento do projeto, pois aborda os critérios entre SGBDs e *Web*. Como o projeto tem como um dos objetivo a geração e a obtenção de documentos SMIL, é necessário que estes sejam fornecidos no ambiente *Web*.

Além de realizar conexão entre SGBD e *Web*, o projeto tem como principal objetivo a integração entre os padrões SMIL e MHEG-5. Para isso foi necessário um estudo aprofundado sobre o contexto abordado pelos dois padrões. O próximo capítulo tem como objetivo explanar os estudos realizados sobre esses dois padrões. Também serão citados nesse capítulo, os critérios de sincronização adotados em ambos os padrões.

CAPÍTULO 5

Padrões MHEG-5 e SMIL

Atualmente a utilização de aplicações multimídia vem se destacando em várias áreas, como: medicina, educação, propaganda/marketing, indústrias, e outros. Para a construção e apresentação de uma aplicação multimídia, normalmente utiliza-se um padrão. Entre os diversos padrões internacionais existentes podemos citar: HyTime [Newcomb91], MHEG [Effelsberg95] [ISO96a] e Premo [Herman96]. O padrão HyTime é derivado de SGML, sendo que adiciona mecanismos para especificar *hyperlinks* e possui listas de informações multimídia em tempo e espaço. Os padrões MHEG e Premo propõem uma abordagem orientada a objetos, em que uma apresentação multimídia é definida por uma hierarquia de objetos que podem ser executados em um *engine* de apresentação.

Com o crescimento do uso da *Web* para publicação de informações multimídia, grande volume de aplicações está sendo criado. As páginas criadas para a *Web* são na sua maioria escritas em HTML. Essa linguagem é limitada em alguns aspectos, como por exemplo, não possuir funcionalidades para realizar sincronização entre mídias. Para solucionar esse problema e outros encontrados em páginas *Web*, foi criado pela *WWW Consortium* o padrão SMIL [Liu98] [Smil98a] [Smil98b].

Este capítulo apresenta com detalhes os dois padrões: MHEG-5 e SMIL.

5.1 Padrão MHEG

Com a evolução das tecnologias surgem alguns aspectos problemáticos em relação às aplicações multimídia, pois os desenvolvedores criam aplicações que podem ser executadas perante uma determinada plataforma (normalmente a que ele está utilizando), numa determinada linguagem de programação, sistema operacional entre outros. Os desenvolvedores, no momento da autoria da aplicação, utilizam apenas os recursos disponíveis, inviabilizando a execução dessas aplicações em outras plataformas. Um outro grande problema surge em relação aos fabricantes do mercado multimídia, pois eles adotam um padrão próprio para as mídias, linguagens, etc.

Visando solucionar esses problemas, a ISO (*International Organization of Standardization*) e a IEC (*International Electrotechnical Comission*) propuseram um padrão denominado MHEG (*Multimedia and Hypermedia Information Coding Expert Group*) através do documento 13522, cujo título geral é *"Information Technology - Coding of Multimedia and Hypermedia Information"*. Esse padrão define uma estrutura hierárquica de classes que permite o intercâmbio de instâncias dessas classes entre plataformas computacionais heterogêneas ou não, possibilitando que as aplicações sejam portáveis entre essas plataformas.

O documento 13522 é dividido em partes responsáveis pela determinação de exigências em cada nível de intercâmbio entre aplicações.

No MHEG-1, o nível de objetos MHEG é especificado no documento 13522-1 [ISO94a], denominado (*MHEG Object Representation - Base Notation ASN.1*). É uma das partes mais abrangentes para intercâmbio de objetos multimídia/hipermídia, pois detalha sincronização, implementação, etc. Pode ser utilizada para definir objetos contendo informações sobre mídias, relacionamentos entre objetos, comportamento dinâmico de objetos, otimização e manipulação de objetos em tempo real.

O documento 13522-2 [ISO97] especifica os objetos MHEG utilizando SGML .

No documento 13522-3 [ISO94c] é definido o correspondente MHEG-3 (*MHEG Script Interchange Representation*) e é proposta uma interface para entidades MHEG e uma forma de troca de *scripts*.

Já no documento 13522-4 [ISO94b] são especificados os procedimentos de registro dos tipos de dados MHEG.

No documento 13522-5 é tratado o nível de aplicação, denominado MHEG-5 (*Support for Base-Level Interactive Applications*) [ISO96a]. Nesse documento são especificadas a semântica e a sintaxe de intercâmbio na forma final para objetos MHEG-5. Esses objetos podem ser utilizados no domínio de aplicações multimídia interativas em uma arquitetura cliente/servidor.

No documento 13522-6 [ISO96b] é definido o padrão MHEG-6 denominado *"Support for enhanced interactive applications"*. O escopo desse documento é definir a semântica e a

representação codificada na forma final para o intercâmbio de aplicações multimídia interativa avançada.

Dentre esses padrões citados acima, o MHEG-5 (que é relatado no documento 13522-5) será especificado nas próximas seções. O objetivo dessa especificação é fornecer um maior entendimento sobre seu uso para a criação/apresentação de aplicações multimídia. Com isso, é definido todo o contexto que o MHEG-5 abrange, permitindo que tenhamos mecanismos para sua integração com o padrão SMIL.

5.1.1 Padrão MHEG-5

O objetivo do MHEG-5 é definir a sintaxe e a semântica de um conjunto de objetos que podem ser utilizados para interoperabilidade de aplicações multimídia entre plataformas com recursos mínimos.

Os maiores objetivos do MHEG-5 são os seguintes:

- Prover um ambiente padrão para o desenvolvimento de aplicações multimídia.
- Definir uma forma final de representação codificada para o intercâmbio de aplicações entre plataformas de diferentes tipos.
- Prover as bases para um nível de conformância concreta.
- Permitir que o programa executável do *engine* (*runtime engine*) do cliente seja relativamente pequeno e simples de implementar.
- Permitir o desenvolvimento de uma ampla variedade de aplicações, inclusive provendo o acesso à bibliotecas externas.
- Garantir que o código de uma aplicação seja seguro.
- Promover o desenvolvimento rápido de aplicações através da disponibilização de primitivas de alto nível de um paradigma declarativo para o desenvolvimento de aplicações.

Engines são responsáveis pela interpretação de objetos MHEG-5, interagindo através de documentos codificados em linguagem ASN.1 (*Abstract Syntax Notation One*) [ISO94a] ou textual.

Todo MHEG-5 *engine* deve possuir um codificador/decodificador que mapeie as classes MHEG-5 para uma linguagem que facilite a troca das aplicações multimídia e vice-versa, além de possuir mecanismos que interpretem as informações multimídia segundo suas exigências de apresentação. Através dessa interpretação, o *engine* permite a apresentação das cenas de uma aplicação para o usuário.

O padrão MHEG-5 foi desenvolvido para suportar a distribuição de aplicações multimídia interativas numa arquitetura cliente/servidor.

A idéia básica de uma aplicação MHEG-5 é que ela é constituída por objetos compartilhados por diversas cenas. Cada cena contém um conjunto de objetos que devem ser apresentados num determinado momento, coordenadamente. Esses objetos que compõem uma cena são chamados de ingredientes. Uma aplicação MHEG-5 é composta por cenas e objetos comuns a todas as cenas. Uma cena contém um grupo de objetos utilizados para apresentar as informações com comportamentos baseados em disparos de eventos. Uma única cena é apresentada de cada vez. A navegação através da aplicação corresponde à realização de transições entre as cenas.

a-) Estrutura de Classes MHEG-5

Na figura 5.1, é apresentada a estrutura hierárquica das classes do padrão MHEG-5, representada através da notação de Rumbaugh [Rumbaugh91]. Um resumo da notação é mostrado na figura 5.2

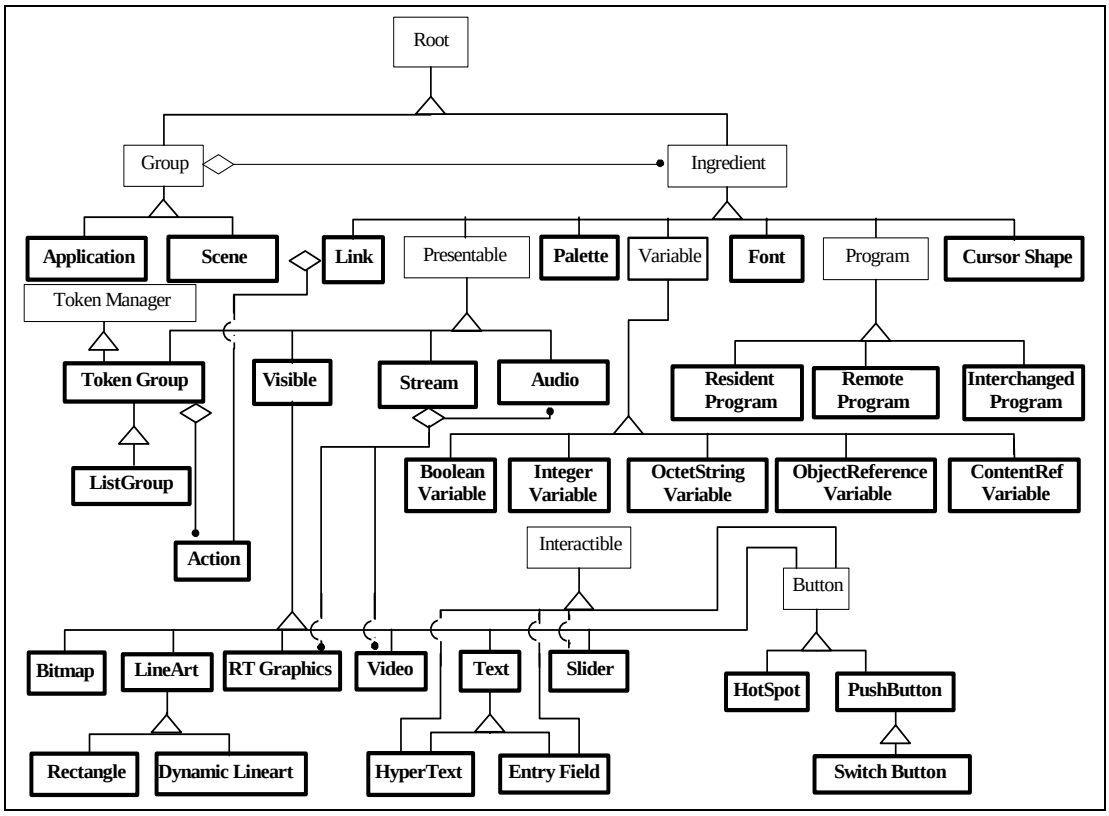


FIGURA 5.1 – ESTRUTURA DE CLASSES MHEG-5 [VIEIRA96]

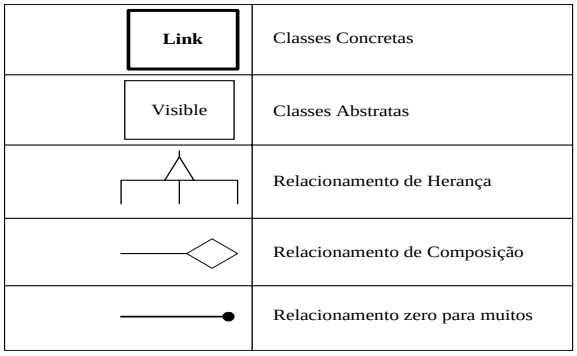


FIGURA 5.2 – LEGENDA UTILIZADA NA REPRESENTAÇÃO DA ESTRUTURA DE CLASSES MHEG-5

b-) Definição das Classes MHEG-5

Conforme é ilustrado na figura 5.1, a estrutura de classes MHEG-5 é consideravelmente complexa. Abaixo são apresentadas algumas classes que compõem a hierarquia de classes MHEG-5 com o objetivo de fornecer uma visão geral sobre o padrão. Em [SantosM97c] encontram-se detalhadas todas as classes da figura 5.1.

. Root

Root é uma classe abstrata que especifica atributos e comportamentos genéricos das demais classes da hierarquia, tais como *Group* e *Ingredient.*

. Application, Scene e **Group**

Os objetos da classe *Application* agrupam objetos da classe *Ingredient*. Somente um objeto *Application* pode estar ativo de cada vez. Uma aplicação MHEG-5 é iniciada através de seu objeto *Application* correspondente. A classe *Scene* contém os objetos que devem ser apresentados coordenadamente em um determinado instante. Essas classes são generalizadas na classe *Group* que é uma classe abstrata.

. Ingredient

Esta classe contém a representação da informação (gráfico, áudio, vídeo, etc.) de cada objeto que compõe uma cena. Esta classe é especializada em classes que mantém informações sobre objetos apresentáveis ao usuário (*Presentable*), informações sobre comportamento dos objetos (*Link*), chamadas a código procedural que executa alguma tarefa específica que não seja facilmente expressa dentro do paradigma do MHEG-5 (*Program*), informações sobre valores de variáveis para serem usadas, por exemplo, como parâmetros para ações (*Variable*), fontes (*Font*), tabelas de cor (*Palette*) e cursores (*CursorShape*). As subclasses *Font*, *Palette* e *CursorShape* podem ser referenciadas por objetos de outras classes, como por exemplo, um objeto de texto (*Text*) pode referenciar um objeto fonte (*Font*).

. Link e **Action**

Os objetos destas classes representam o comportamento embutido nos ingredientes das cenas e aplicações, baseado em disparo de eventos (por exemplo, o botão esquerdo do *mouse* ao ser clicado, ativa um som). A classe *Action* contém as ações elementares que podem ser aplicadas aos objetos das diversas classes do MHEG-5.

. Presentable

Esta classe contém objetos que podem ser apresentados ao usuário como parte de uma cena. Os objetos apresentáveis ou são visíveis (*Visible*) ou são audíveis (*Audio*), ou uma combinação dos dois (*Stream*). As classes *TokenManager*, *TokenGroup* e *ListGroup* contêm informações que possibilitam a navegação entre os objetos a serem apresentados.

. Visible

Esta classe contém os objetos que podem ser mostrados na tela. Possui mecanismos para representar os vários tipos de objetos visíveis, ou seja: *LineArt* (incluindo sua subclasse *Rectangle*), quando executada desenha uma linha ou um retângulo na tela; *RTGraphics* e *Video* encapsulam um objeto visível que muda em tempo real, como por exemplo um vídeo-clip; a classe *Bitmap*, quando executada, desenha uma figura

bidimensional estática; a classe *Button* juntamente com as subclasses *SwitchButton*, *HotSpot* e *PushButton*, permitem que seja mostrado um botão e a troca de aparência do botão; a classe *Slider* representa, em um formato gráfico, o relacionamento entre um valor e um intervalo, ou seja o valor 1 no intervalo [0..2] é representado graficamente como estando no meio do intervalo especificado. A classe *Text* com as subclasses *EntryField* e *HyperText* permitem apresentar um texto na tela, sendo que a classe *HyperText* suporta o conceito de *hyperlinks* e a *EntryField* permite interação com o usuário através da troca de seus conteúdos. As classes *HyperText*, *EntryField*, *Slider* e *Button* permitem que, em tempo de execução, seja controlada a interação do usuário com seus objetos; este controle é efetuado através da generalização destas classes em sua superclasse *Interactible*.

c-) Tratamento de Sincronização MHEG-5

A sincronização entre mídias, é um dos recursos fundamentais utilizados em modelos multimídia. O padrão MHEG-5 trata o sincronismo através de duas classes: *Link* e *Action*. Os objetos *Link* são utilizados para expressar o comportamento de aplicações MHEG-5 e são compostos de uma condição e de um objeto *Action*, representados pelos atributos *LinkCondition* e *LinkEffect* respectivamente. Quando a condição (*LinkCondition*) é avaliada como verdadeira, o *Link* é disparado, ou seja, as ações contidas no objeto *Action* passam a ser executadas. O *LinkCondition* é composto por três partes: um código correspondente a um evento, uma referência a um objeto a partir do qual o evento deve ser gerado e um valor que especifica um possível parâmetro associado ao evento, conforme especificado abaixo.

LinkCondition:
EventSource : contém a referência ao objeto que originou o evento;
EventType : evento que irá ocorrer e
EventData : dado que o evento produz.

Para que a condição do objeto *Link* possa ser satisfeita é necessário que um evento associado a essas três informações ocorra, além do próprio objeto *Link* estar ativo. *Links* ativos devem fazer parte de um objeto *Scene* ativo ou de um objeto *Application* ativo. Os eventos são declarados na classe *Link* e é gerado a partir do *EventSource* e podem ser: síncronos (*IsAvailable, IsRunning, IsStopped,* entre outros) e assíncronos (por exemplo, *CursorEnter, StreamEvent, TimerFired*).

O atributo *LinkEffect* contém uma referência para um objeto da classe *Action.* Um objeto *Action* permite que um conjunto de ações elementares possa ser executado em uma sequência síncrona; o atributo que armazena essas ações é chamado *ElementaryActions*.

Uma ação elementar consiste de um objeto ao qual a ação é destinada e uma lista de valores representando os parâmetros relacionados à ação. A classe *Action* não é herdeira de nenhuma outra classe MHEG-5, o que significa que objetos *Action* não podem ser referenciados como entidades individuais.

Todo o tratamento dos eventos deve ser controlado pelo *engine* MHEG-5. O *engine* examina o evento que ocorreu e checam os atributos da classe *Link* de todos os objetos *Link* ativos (isto é, da aplicação e das cenas). Após isso, o *engine* valida a associação com o tipo e a fonte do evento em questão. Para cada *link* que satisfaz essa condição, os dados associados do evento são checados com os dados do atributo *EventData* do *link*. Os *links* que satisfazem essa condição são considerados disparados.

Um *engine* MHEG-5 é dirigido pela ocorrência de eventos assíncronos. Na ocorrência de um evento assíncrono, o *engine* deve examinar todos os *links* ativos para determinar o seu disparo. Para cada *link* disparado, as ações elementares do seu atributo *LinkEffect* são armazenadas em uma fila para a sua execução seqüencial. Como resultado direto de um *LinkEffect* sendo executado, eventos síncronos podem ocorrer. Esses eventos são tratados diretamente pelo *engine*, isto é, após a execução de uma ação elementar, o *engine* deve checar se existem *links* disparados como resultado de eventos síncronos que tenham ocorridos. Se for o caso, o *link* e todos os seus efeitos devem ser processados antes que o *engine* proceda para o tratamento de um posterior evento assíncrono enfileirado. Qualquer evento assíncrono que ocorra durante o processamento de outro evento deve ser enfileirado para posterior tratamento. Ações que alteram o contexto de ações que estão sendo processadas influenciarão tanto a fila de eventos assíncronos quanto a fila de ações que estão esperando para processamento. O contexto é alterado pelas ações *TransitionTo*, *Launch, Spawn* e *Quit*. Se essas ações ocorrerem, os eventos assíncronos enfileirados devem ser removidos da fila e ações elementares enfileiradas devem também ser removidas da fila. Algumas particularidades que devem ser observadas são os efeitos contínuos da ação *Run*, que retorna o controle para o tratamento do *link* e continua a apresentação do objeto; e da capacidade de um *link* se desativar (*Deactivate*). O tratamento dessa última ação é adiado até o término do tratamento do *link*.

A interação com o usuário é realizada através de objetos MHEG-5 que pertencem à classe *Interactible* que podem estar num estado "*interacting*", que é assinalado quando o atributo *InteractionStatus* é verdadeiro. Quando um objeto está interagindo nenhum evento de entrada do usuário pode ser gerado. É importante notar que apenas um objeto pode

estar interagindo por vez. Os eventos *CursorEnter* e *CursorLeave* ocorrem quando um cursor entra ou sai de uma área definida por um objeto *Interactible*.

5.1.2 Padrão MHEG-6

No documento 13522-6 [ISO96b] é definido o padrão MHEG-6 denominado *"Support for enhanced interactive applications"*. O escopo desse documento é definir a semântica e a representação codificada na forma final para o intercâmbio de aplicações multimídia interativa avançada. Essas aplicações estendem aplicações coberto pelo ISO/IEC 13522-5 em incorporar funcionalidades como computação (processamento de dados) e comunicação estendida com o ambiente externo, incluindo servidores, dispositivos locais, etc.

A representação codificada definida por essa parte do ISO/IEC 13522 estende a representação codificada definida pela ISO/IEC 13522-5 [ISO96a]. Especialmente, define a representação codificada para o atributo *Content* da classe MHEG-5 *InterchangedProgram.*

A classe *InterchangedProgram* permite a criação de instâncias de objetos cujo conteúdo pode ser um programa (ou referência a ele) em linguagem de *script*. Esses programas podem então compor um dado multimídia e agregar-lhe alto índice de complexidade.

No padrão MHEG-6 , é eleito *Java*^{TM} como a linguagem de *script* para objetos *InterchangedProgram*. Esse padrão define a API (*Application Program Interface)* MHEG-5 que permite ao código de um objeto *InterchangedProgram* invocar facilidades de apresentação do MHEG-5 *engine*.

5.2 Padrão SMIL

Para disponibilizar sincronismo de mídias na *Web*, foi proposto pela *WWW Consortium* a definição de um novo padrão para documentos multimídia sincronizada. Esse padrão é denominado SMIL (*Synchronized Multimedia Integration Language*) [Liu98], [Smil98a], [Smil98b], baseado em XML [Bray98] e fornecem algumas funcionalidades básicas para incluir dados multimídia contínuas como vídeo e áudio em documentos *Web*.

Todos os documentos criados na linguagem SMIL são estruturados através de uma maneira declarativa para a apresentação multimídia. Um documento é dividido basicamente em duas seções: cabeçalho e corpo, representados pelas *tags* <*head*> e <*body*>

respectivamente. O <*head*> contém informações gerais sobre o documento e à definição do layout espacial de apresentação. O <*body*> contém a definição dos objetos e seus relacionamentos temporais, além da especificação de âncoras e elos de navegação.

O layout espacial do documento contém a definição de regiões da janela, onde são exibidos os objetos. Existe uma região que representa a janela inteira onde o documento será exibido, sendo que é necessário que essa região contenha as referências para todas as demais regiões. Os objetos não contêm o conteúdo dos dados associados, e sim uma descrição onde está incluída uma referência (URI) para o conteúdo propriamente dito. Esses objetos podem ser definidos com atributos de início e fim de apresentação, sendo *clip-begin* e *clip-end* respectivamente.

A apresentação de um documento SMIL é estruturada utilizando o conceito de composição. Uma composição pode conter objetos de mídia e outras composições, determinando a forma como seus componentes deverão ser exibidos. Existem dois tipos de composição, a paralela representada pela *tag* <*par*> e a seqüencial <*seq*>. O corpo de um documento SMIL é uma composição seqüencial.

Os objetos de mídia possuem um atributo chamado *dur* que representa a duração, isto é, o tempo de exibição de uma mídia. Os objetos de mídia contínua, como áudio e vídeo, possuem essa duração de forma implícita, derivada de seu próprio conteúdo, enquanto que em objetos de mídia discreta, como texto e imagem, a duração implícita é nula. O atributo *dur* realiza de forma explícita uma duração definida pelo usuário. Quando essa duração for menor que a implícita na mídia, a exibição será interrompida, ao passo que se a duração explícita for maior, a exibição da mídia será prolongada. Além do atributo de duração, os elementos possuem outros dois atributos: *begin* e *end*. Caso um elemento com esses atributos definidos pertença a uma composição paralela, determinará o início e o término de sua apresentação de acordo com o intervalo de tempo especificado. Tanto em composições paralelas quanto em seqüenciais, os atributos *begin* e *end* dos elementos também podem ser especificados em relação ao início e término de qualquer outro elemento de composição.

As composições paralelas também possuem o atributo *endsync*, que determina o término da composição em relação ao término de um de seus componentes. O valor desse atributo pode identificar um dos componentes, indicando que todos os demais serão terminados quando o componente selecionado terminar. O atributo pode ainda assumir os valores *first* (primeiro) ou *last* (último), para especificar o término da composição em relação

ao primeiro ou último componente a terminar a exibição, respectivamente. Caso esse atributo não seja especificado, as composições paralelas terminam pelo último componente. Um mesmo elemento pode ser exibido diversas vezes em seqüência, onde o número de repetições é determinado pelo atributo *repeat*.

É possível ainda especificar sub-regiões (âncoras) temporais e espaciais em objetos de mídia e definir ligações entre elas. As ligações possuem sempre uma âncora de origem e destino, e a navegação nos mesmos é ativada por uma interação do usuário. As ligações podem ser definidas entre sub-regiões de um objeto, elementos, ou combinações dessas possibilidades. Um dos atributos de ligação é o *show*, que controla o comportamento do objeto de origem quando a ligação é percorrida. O valor desse atributo pode ser: *replace*, que especifica que a apresentação do objeto de origem será substituída pela apresentação do destino; *new*, que especifica que a apresentação do objeto de destino acontecerá em outra janela; e *pause*, que especifica que a apresentação do objeto de origem será suspensa e depois retomada, sendo o objeto de destino apresentado em outro contexto.

Apesar dos recursos apresentados acima, SMIL possui algumas desvantagens, como as citadas em [Rousseau98]:

- Composição temporal é baseada numa maneira híbrida, que mistura duas abstrações diferentes: intervalos e pontos de tempo (*time-points*). Um cenário é representado como uma árvore, com operadores temporais *par* (paralelo) e *seq* (seqüencial) que são nós e intervalos correspondentes a objetos multimídia que são folhas. Para a criação de um operador *par* não ambíguo, SMIL define atributos adicionais para referir aos pontos de tempo. Os atributos especificam a semântica exata do operador *par*. Como resultado, um cenário dado pode ser expresso de diversas maneiras diferentes e as especificações podem tornar-se confusas.

- O padrão SMIL fornece um atributo opcional chamado *lipsync* que é aplicado para um grupo de objetos envolvidos com o operador *par*. Isso não é suficiente para especificar sincronização entre mídias.

Além dessas desvantagens, em [Rodrigues99] são relatados também algumas limitações decorrentes da simplicidade do SMIL. Dentre elas, destacam-se:

- O SMIL fornece dois tipos de elementos compostos para definir os relacionamentos temporais, que são as composições paralelas e seqüenciais. Com isso a estrutura lógica do documento é necessariamente a estrutura de apresentação formada por uma hierarquia de composições, paralelas e seqüenciais, aninhadas. Utilizando esse tipo de modelo temporal, algumas alterações nas especificações de sincronização, por exemplo, a inclusão de novos relacionamentos temporais entre objetos, implica às vezes, uma reestruturação de todo o documento.

- O SMIL não provê a facilidade de reutilização de dados e das estruturas de apresentação em outros documentos e nem dentro de um mesmo documento. As estruturas são reutilizadas através de cópias, o que dificulta a manutenção. A linguagem só provê reuso do conteúdo dos objetos de mídia, que são identificados através de referências descritas na forma de URIs, e de layouts de apresentação dentro de um mesmo documento.

- Os elos entre componentes de um documento SMIL são sempre relacionamentos *1:1* ativados através de interação do usuário com o objeto de origem do elo, enquanto as relações síncronas são especificadas nas composições paralelas e seqüenciais. Com isso, não é possível especificar relacionamentos que envolvam tanto eventos síncronos como eventos que dependam da interação do usuário.

- O SMIL não oferece suporte para especificar mudanças de comportamento durante a apresentação de um objeto. Por exemplo, no caso de um áudio, uma especificação do tipo “coloque o volume a *X* db decorrido *y* segundos do início de sua apresentação” não pode ser definida.

- O SMIL não oferece muita flexibilidade para a especificação do comportamento temporal dos objetos, por exemplo, permitindo a definição de durações mínima e máxima de apresentação, ou mesmo de preparação, dos objetos. Essa flexibilidade permitiria a um sistema que controlasse a apresentação ajustar os tempos de tal forma a melhorar a qualidade da exibição e reduzir as chances de ocorrerem inconsistências temporais [Buchanan93] [Kim95].

Um arquivo SMIL (extensão .smi) pode ser criado com algum editor de texto ou processador de palavras. Abaixo são demonstrados alguns exemplos usando o padrão SMIL.

Exemplo 01

```
<smil>
<head>
      <meta name= "author"    content= "Jane Morales" />
      <meta name= "title"     content= "Multimedia My Way" />
      <meta name= "copyright" content= "(c)1998 Jane Morales" />
</head>
<body>
      <seq>
            <audio src= "audio/newsong.wav" />
            <audio src= "audio/oldsong.snd" />
      </seq>
</body>
</smil>
```

No exemplo 01 são ativados dois *clips* de áudio seqüencializados utilizando a *tag* <**seq**>. Já no exemplo 02 são apresentados dois *clips* de áudio paralelos em diferentes tempos, utilizando a *tag* <**par**>. O primeiro áudio (song01.snd) é inicializado imediatamente, mas inicia no *timeline* após 30.4 segundos. O final do primeiro *clip* é 60.4 segundos, o que resulta o tempo total de 30 segundos para esse áudio no *timeline*. O segundo *clip* é atrasado para 28 segundos e depois é inicializado no *timeline* após 2.4 segundos e o seu final é 13.7 segundos, totalizando 39.3 segundos para esse áudio.

Exemplo 02

```
<smil>
<head>
      <meta name= "author"    content= "Jane Morales" />
      <meta name= "title"     content= "Multimedia My Way" />
      <meta name= "copyright" content= "(c)1998 Jane Morales" />
</head>
<body>
      <par>
            <audio src= "song01.snd" clip-begin= "30.4s"
                        clip-end= "60.4s"  />
            <audio src= "song02.ra" delay= "28.s" clip-begin= "2.4s"
                        clip-end= "13.7s" />
      </par>
</body>
</smil>
```

O exemplo 03 mostra a execução de uma seqüência de arquivos, juntamente com a combinação de arquivos paralelos.

Exemplo 03

```
<smil>
<head>
```

```
      <meta name= "author"    content= "Jane Morales" />
      <meta name= "title"     content= "Multimedia My Way" />
      <meta name= "copyright" content= "(c)1998 Jane Morales" />
</head>
<body>
      <seq>
            <anchor href= "http://www.exemplo.br/exemplo1.smil" />
            <par>
                  <anchor href= "http://www.exemplo.br/exemplo2.smil" />
                  <anchor href= "http://www.exemplo.br/exemplo3.smil" />
            </par>
            <anchor href= "http://www.exemplo.br/exemplo4.smil" />
      </seq>
</body>
</smil>
```

E finalizando, o exemplo 04 ilustra um arquivo SMIL onde é escolhida a largura de banda que o computador do usuário estará configurado, através do atributo *system-bitrate* da *tag* <**par**>. Essa largura de banda é especificada em bits por segundo (bit/s). No exemplo, de acordo com a largura de banda, são definidas as mídias que serão executadas.

Exemplo 04

```
<smil>
<head>
      <meta name= "author"    content= "Jane Morales" />
      <meta name= "title"     content= "Multimedia My Way" />
      <meta name= "copyright" content= "(c)1998 Jane Morales" />
</head>
<body>
      <switch>
            <par system-bitrate=75000>
                  <audio src= "audio/newsong1.snd" />
                  <video src= "video/newsong1.avi" />
                  <image src= "lyrics/newsong1.gif" />
            </par>

            <par system-bitrate=47000>
                  <audio src= "audio/newsong2.snd" />
                  <video src= "video/newsong2.avi" />
                  <image src= "lyrics/newsong2.gif" />
            </par>

            <par system-bitrate=28000>
                  <audio src= "audio/newsong3.snd" />
                  <video src= "video/newsong3.avi" />
                  <image src= "lyrics/newsong3.gif" />
            </par>

      </switch>
</body>
</smil>
```

5.3 Considerações Finais

Neste capítulo foram discutidos os dois padrões: MHEG-5 e SMIL. A seção 5.1 foi dedicada ao padrão MHEG, sendo especificados os padrões MHEG-5 e o MHEG-6. Também foi relatado na seção 5.1, o funcionamento de um MHEG-5 *engine*, com intuito de mostrar que a maioria das ações MHEG-5 são executadas somente pelos *engines.* Na

seção 5.2 foi definido o padrão SMIL, apresentando-se suas vantagens e desvantagens. Além disso, são citados exemplos de documentos SMIL.

Atualmente o padrão MHEG-5 é utilizado em vários projetos, principalmente aos que utilizam TV-Interativa. O projeto AMOM (Autoria e Manipulação de Objetos Multimídia) que será apresentado no capítulo 6, tem como implementação um Servidor de Objetos Multimídia (SOMm), que faz uso de um banco de dados MHEG-5, isto é, um banco de dados que implementa as classes do padrão MHEG-5. Esse detalhe e outros são citados no capítulo 6.

CAPÍTULO 6

Autoria e Manipulação de Objetos Multimídia (AMOM)

O projeto AMOM (Autoria e Manipulação de Objetos Multimídia) teve origem como um subprojeto (denominado "Modelagem e Gerenciamento de Objetos Multimídia em Ambientes Distribuídos") do projeto SMmD (ProTeM-CC II - CNPq) [Teixeira95] [Teixeira98]. O objetivo do projeto AMOM é prover um ambiente para suporte à autoria, armazenamento e manipulação de aplicações multimídia. Conforme mostrado na figura 6.1, esse projeto é composto por:

- **Servidor de Objetos Multimídia (SOMm):** É o módulo responsável pela autoria, armazenamento e manipulação de objetos multimídia. Na seção 6.1 são dados mais detalhes sobre esse módulo.

- ***Multimedia Application WebBuilder* (MAW):** É o módulo que trata a integração entre aplicações MHEG-5 e documentos *Web* dando também suporte à autoria de aplicações multimídia através da *Web.* A arquitetura desse módulo foi desenvolvida como parte deste trabalho. Mais detalhes é dado na seção 6.2.

- **Interface WEB:** Esse módulo (em desenvolvimento) fornece uma interface para realizar consultas ao banco de dados multimídia do SOMm, através da *Web*. É utilizado neste módulo o O_2Web [$O_2$96], para facilitar a interação entre os servidores *Web* e o SGBDOO O_2.

- **SGBDOO O_2:** O SGBDOO O_2 foi utilizado para implementar o servidor de objetos multimídia. Através dele são armazenados novos objetos e recuperados aqueles que satisfazem as consultas. As consultas ao servidor são formuladas utilizando a linguagem O_2C e/ou OQL, que pertencem ao ambiente do O_2.

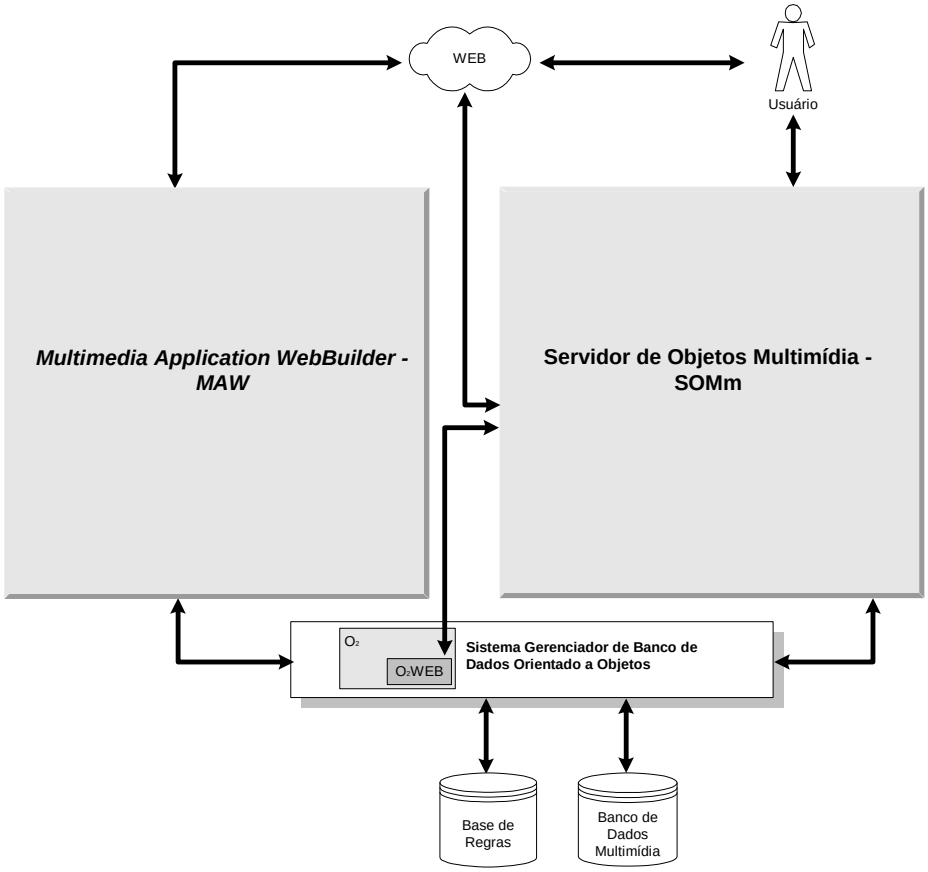


FIGURA 6.1 – ARQUITETURA IMPLEMENTADA NO PROJETO AMOM

Nas seções a seguir são especificados o módulo de Servidor de Objetos Multimídia e o módulo *Multimedia Application WebBuilder* respectivamente.

6.1 Servidor de Objetos Multimídia (SOMm)

O Servidor de Objetos Multimídia é responsável por armazenar e manipular os metadados sobre os dados armazenados nos servidores de mídia. O banco de dados do servidor implementa a estrutura de classes descrita no padrão MHEG-5 (ver figura 5.1), bem como os métodos que permitem a manipulação dos objetos instanciados nessas classes.

O ambiente é composto de quatro módulos (ver figura 6.2):

- Módulo de Criação do Banco de Dados
- Módulo de Apresentação
- Módulo de Recuperação e
- Módulo de *DataMining.*

- **Módulo de Criação do Banco de Dados:** Esse módulo é responsável por alimentar o banco de objetos MHEG-5 com aplicações multimídia. Essas aplicações podem ser originadas de duas fontes distintas: através da ferramenta de autoria do ambiente (SMArT - **SM**mD **A**utho**r**ing **T**ool) [SantosF97a], [SantosF97b], [SantosF98] ou com aplicações importadas, ou seja, aplicações desenvolvidas em outros ambientes e traduzidas para o formato utilizado pelo servidor de objetos multimídia.

- **Módulo de Apresentação:** Uma forma de apresentação das aplicações multimídia pode ser feita através da ferramenta de autoria SMArT. Pode-se também apresentar aplicações e cenas (dessa aplicação) no ambiente *Web*, através do módulo *SmartWeb* que utiliza a ferramenta O_2Web para realizar a conexão entre o servidor HTTP e o ambiente SOMm.

- **Módulo de Recuperação:** Esse módulo é responsável pela recuperação de objetos armazenados no servidor de objetos multimídia. Essa recuperação pode ser efetuada, até o momento, por quatro formas:

    - Através de formulação de consultas envolvendo atributos convencionais, que podem ser de interesse de qualquer usuário ou servir de apoio ao MHEG-5 *engine*;

    - Através de formulação de consultas envolvendo características semânticas, propiciando busca por conteúdo no banco de objetos [SantosM97b] [Vieira97] [Vieira99];

    - Através da formulação de consultas envolvendo características nebulosas [Fornazari99];

    - Através da recuperação de documentos de texto e fala que são partes de aplicações armazenadas no banco de dados [Ribeiro99].

- **Módulo de *DataMining*:** Esse módulo encontra-se em desenvolvimento como tema de dissertação de mestrado. Ele é responsável pela execução do processo de data mining a conjuntos de aplicações MHEG-5, através da filtragem dos dados de acordo com critérios especificados pelo usuário e posterior aplicação de algoritmos que executam as

principais tarefas de data mining sobre esses dados, gerando novos padrões de conhecimentos.

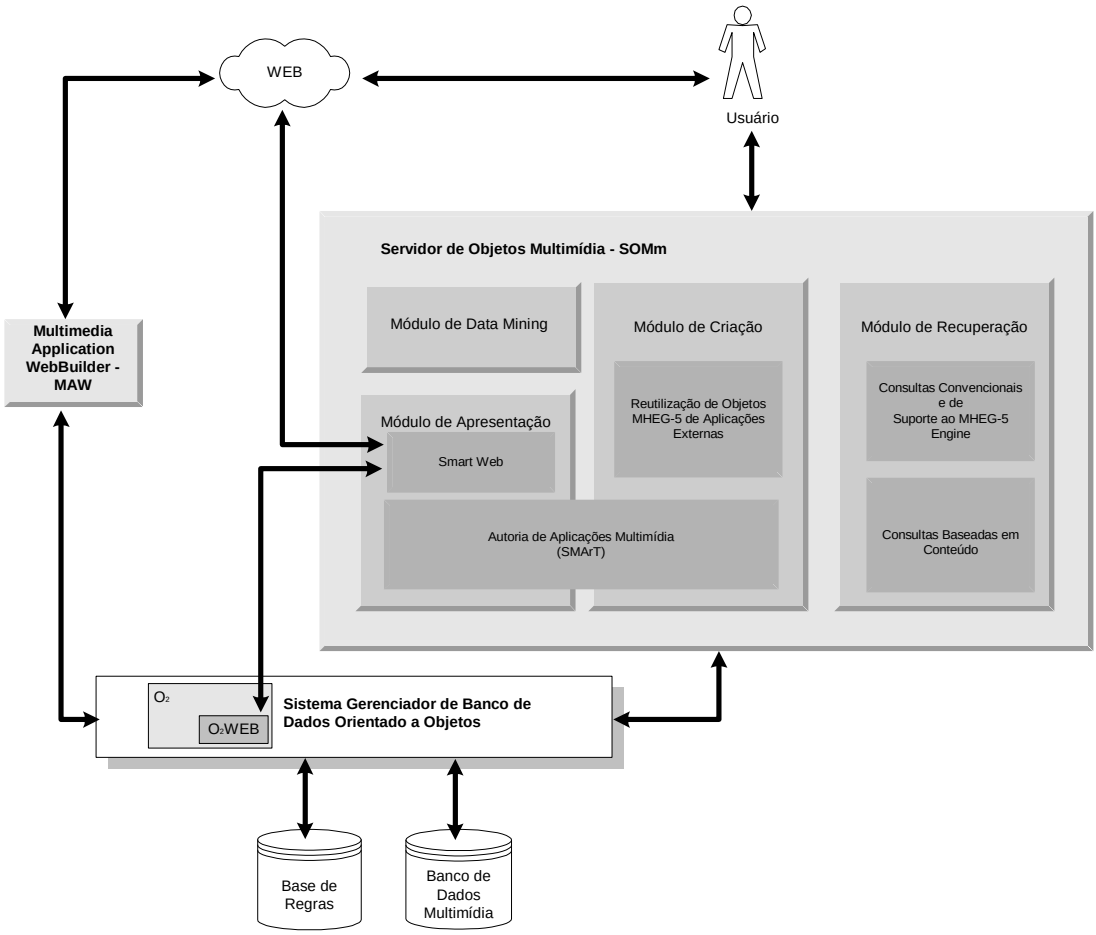


FIGURA 6.2 – ARQUITETURA DO AMBIENTE DO SERVIDOR DE OBJETOS MULTIMÍDIA (SOMM)

6.2 *Multimedia Application WebBuilder* (MAW)

Como o objetivo principal deste trabalho é a integração dos padrões MHEG-5 e SMIL/HTML, foi desenvolvido um ambiente para suporte. Esse ambiente foi denominado de MAW – *Multimedia Application WebBuilder.* Esse ambiente contém um módulo que trata o processo de conversão, que realiza a integração dos padrões. Além desse módulo outros foram criados afim de disponibilizar uma maior interação com o ambiente *Web*.

Os módulos do MAW estão sendo implementados em Java, objetivando sua portabilidade e também para que seja executável em qualquer *browser* Web. A arquitetura do MAW é ilustrada na figura 6.3. Para concretizar a idéia principal deste trabalho (integração dos padrões MHEG-5 e SMIL/HTML) e assim auxiliar nas funcionalidades do MAW, os seguintes módulos foram implementados: autenticação de usuários, conversor de aplicações MHEG-5-SMIL, conversor SMIL-MHEG-5 e módulo de conexão.

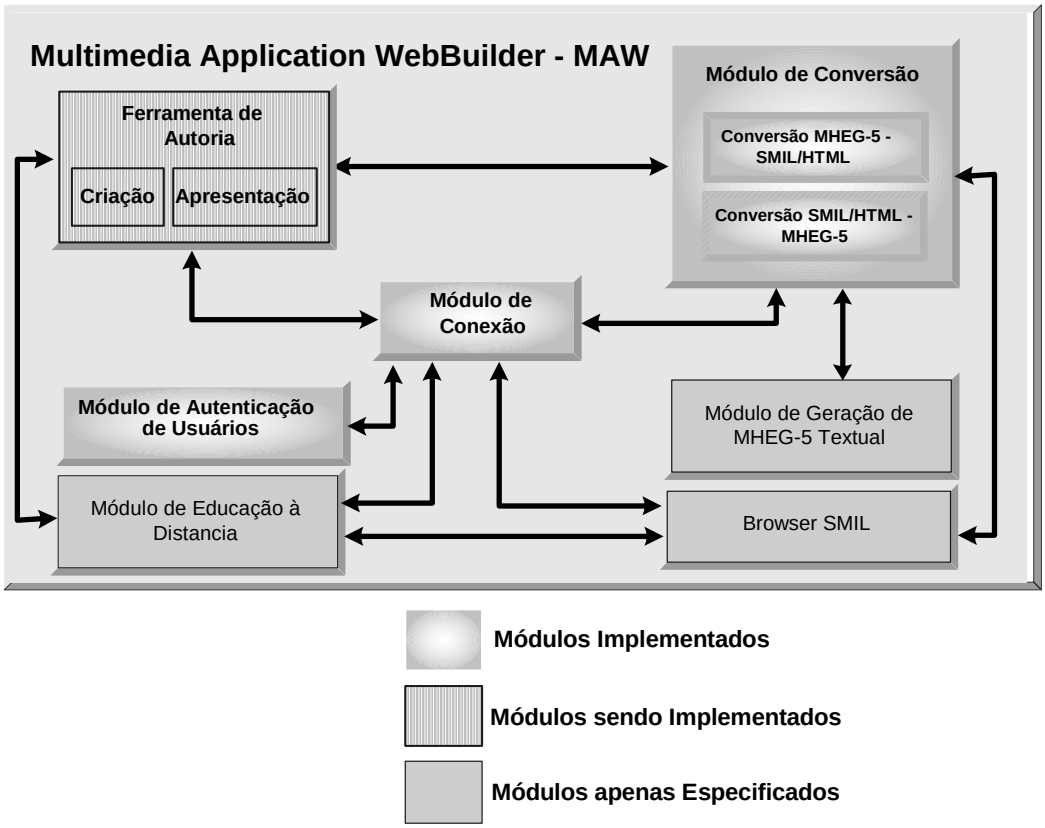


FIGURA 6.3 – ARQUITETURA DO MAW

Conforme ilustrado na figura 6.3, o MAW é composto pelos seguintes módulos: Ferramenta de Autoria, Módulo de Conversão, Módulo de Autenticação de Usuários, Módulo de Conexão, Módulo de Educação à Distância , *Browser* SMIL e Módulo de Geração de MHEG-5 Textual. Esses módulos são especificados a seguir, e o módulo de Conversão é detalhado no capítulo 7.

6.2.1 Ferramenta de Autoria

A ferramenta de autoria do MAW tem como principal característica auxiliar na autoria de documentos SMIL, objetivando o reuso de cenas de aplicações e mídias. A ferramenta visa facilitar o processo de autoria de forma a permitir que autores, sem conhecimento prévio da linguagem SMIL, consigam produzir documentos das mais variadas complexidades. Essa ferramenta encontra-se em desenvolvimento como tema de um outro trabalho de mestrado.

A ferramenta de autoria, através de seu ambiente visual, permitirá ao autor desenvolver novas aplicações e, caso deseje, recuperar do banco de dados (através do módulo de consultas do SOMm) mídias ou cenas de uma aplicação para fins de reutilização.

6.2.2 Módulo de Educação à Distância

Objetiva-se desenvolver um módulo de auxílio à educação à distância, de modo que o autor, sendo um professor, possa produzir aulas (através de aplicações multimídia), e através deste módulo fazer o controle de acesso dos alunos de sua disciplina. Com isso o professor poderá saber quais alunos assistiram às aulas e em quais pontos tiveram mais dúvidas, podendo melhorar a aula aproveitando a facilidade de reuso que a ferramenta de autoria fornecerá.

6.2.3 *Browser* SMIL

Esse módulo será utilizado pelos alunos para assistiram às aulas montadas pelos professores, e também fará todos os controles necessários para gerenciamento das atividades individuais do aluno. Além dessas funcionalidades, esse módulo também será usado pelo módulo de conversão para apresentar as aplicações MHEG-5 convertidas.

6.2.4 Módulo de Geração MHEG-5 Textual

Os usuários poderão gerar suas aplicações multimídia no formato de arquivos MHEG-5 textual. Essa notação é utilizada para que os MHEG-5 *engines* possam executar as aplicações multimídia. Com essa opção, o usuário poderá executar sua aplicação em qualquer MHEG-5 *engine*.

6.2.5 Módulo de Autenticação de Usuários

Para que os módulos do MAW sejam disponibilizados para os usuários, é necessário que haja alguns dispositivos de segurança para acesso ao ambiente. Esses dispositivos devem impedir que pessoas não autorizadas façam acesso ao ambiente do MAW, utilizando-o de forma indiscriminada. Esse módulo tem como objetivo autenticar, isto é, autorizar os usuários a utilizarem o ambiente do MAW. Mais detalhes deste módulo estão especificados no apêndice 2, que trata aspectos sobre implementação.

6.2.6 Módulo de Conexão

Um módulo importante do MAW está relacionado com a conexão com os SGBDs, que tem como objetivo tornar transparente para o autor o processo de busca e armazenamento de objetos em bancos de dados, através da *Web*. Com isso, questões como *string* de conexão e autenticação do usuário são automaticamente gerenciadas pelo módulo. Além disso, o módulo funcionará de forma a possibilitar que o MAW possa ser utilizado em qualquer SGBDOO (O2, Jasmine, Poet, etc), sendo que prioridade será dada aos SGBDs que possuam suporte para conexão com a linguagem Java. Atualmente esse módulo de conexão possui implementado métodos de conexão para o SGBDOO O2. Para os outros SGBDOOs estão sendo realizados estudos para futuras implementações. Para auxiliar esse processo de utilização com vários SGBDs, será implementado um outro módulo chamado de módulo de Inicialização (*Initialization module*), que criará classes e instanciará objetos do banco de dados multimídia do SOMm para o SGBDOO desejado, ou seja, criará instâncias do SOMm. Na figura 6.4 é mostrada a arquitetura de integração entre o MAW e os SGBDs.

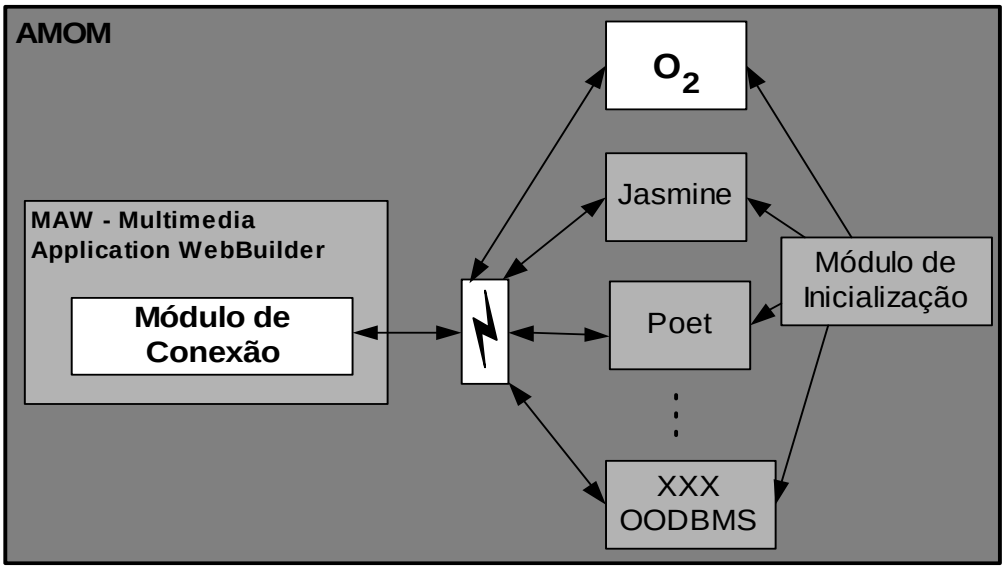


FIGURA 6.4 – CONEXÃO DO MAW COM DIVERSOS SGBDS

No módulo de conexão são contemplados vários métodos utilizados para realizar a conexão. Com o intuito de generalizar diversos tipos de conexões, foram criadas algumas classes que contém alguns métodos genéricos e outros específicos para cada SGBDOO. Toda a implementação da conexão para o SGBDOO O2 foi realizada em Java, uma vez que o objetivo final é tornar esse módulo completamente portável e simples. Para realizar a conexão para o sistema O2, foi necessário criar métodos nativos, uma vez que a versão disponível desse gerenciador, no projeto AMOM, utiliza uma plataforma *SUN*. O SGBDOO

O2 possui métodos específicos para realizar a conexão, sendo estes implementados na linguagem C. Portanto, foi necessário realizar a chamada desses métodos através de métodos nativos escritos em Java.

Na figura 6.5 é mostrado um diagrama de classes (utilizando a notação UML) desenvolvidas para representar os tipos de conexões, entre elas, aquela que o módulo já suporta (que é para o SGBDOO O2), e aquelas que futuramente suportará (Jasmine, Poet, entre outros). As classes do diagrama são: *OODBConnection*, *O2Connection*, *JasmineConnection*, *PoetConnection*, e *XXOODBMSConnection* (esta última representa que, além das classes anteriormente citadas, outras também serão futuramente contempladas) .

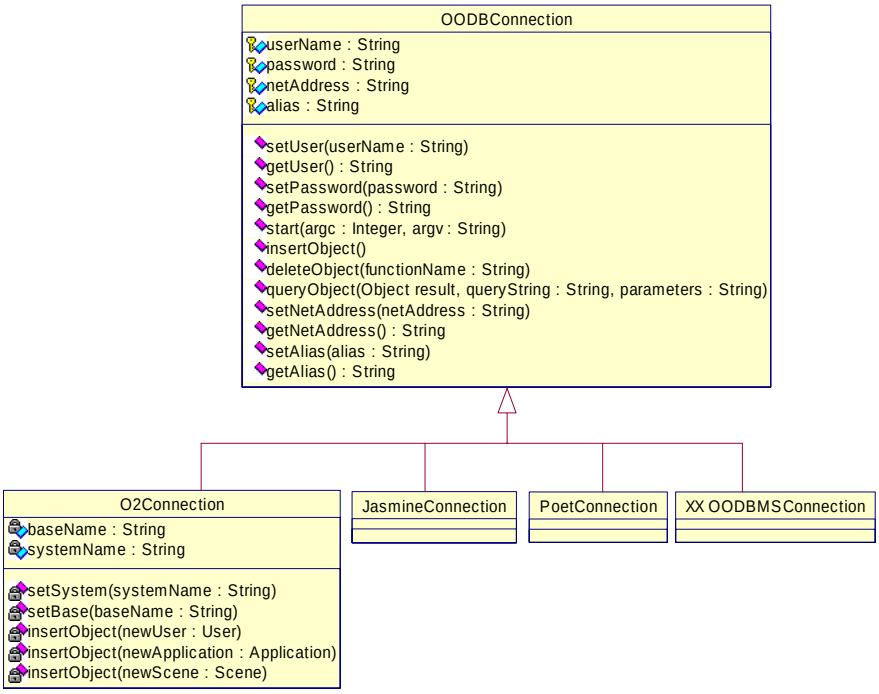


FIGURA 6.5 – ESQUEMA DE CLASSES DO MÓDULO DE CONEXÃO

A classe *OODBConnection* é composta pelos atributos: *userName*, *password*, *netAddress* e *alias*. O atributo *userName* é usado para armazenar os usuários que estão conectados aos SGBDs, juntamente com suas *passwords*. O atributo *netAddress* é usado para configurar o endereço de rede do SGBDOO. Quando surgir a necessidade de várias conexões com SGBDs diferentes ou não, os usuários sabem quais são as conexões que estão em aberto através de um apelido, que é representado através do atributo *alias*. Todos os atributos definidos na classe *OODBConnection* são protegidos, permitindo que somente a própria classe e suas subclasses tenham acesso às informações nelas contidas. Todos os métodos definidos na classe *OODBConnection* são públicos. Para realizar a conexão com um

SGBDOO é necessário ativar os seguintes métodos: *setNetAddress(), setAlias(), setUser(), setPassword e start().* O método *setNetAddress()* é usado para configurar o endereço da rede; o *setAlias()* é ativado para especificar o apelido da conexão; o *setUser()* e *setPassword()* são usados para informar o usuário e sua senha respectivamente; e o *start()* é utilizado para realizar a conexão com o SGBDOO O2, configurando informações como, nome da máquina hospedeira do banco de dados, o caminho, isto é, diretório de instalação do banco de dados e o nome do servidor do banco de dados. Os métodos *getUser(), getPassword(), getAlias()* e *getNetAddress()* são ativados apenas quando os usuários solicitarem, isto é, são métodos adicionais criados para buscar informações sobre a conexão armazenada na classe (objetos de conexão). Para realizar a manipulação de objetos foram criados os métodos essenciais: *insertObject(), deleteObject()* e *queryObject()*. Para inserir objetos em um banco de dados OO é utilizado o método *insertObject()*. Esse método é redefinido nas subclasses a fim de atender as necessidades e especificações de cada SGBDOO. A exclusão de objetos de um banco de dados é realizada através do método *deleteObject()* que por sua vez também deve ser redefinido nas subclasses. Para realizar a atualização de um objeto, é necessário realizar as operações *deleteObject()* e em seguida o *insertObject()*. E finalmente o método mais utilizado pelos usuários de banco de dados, o *queryObject()*. Esse método realiza consultas aos bancos de dados, para buscar objetos que podem, por exemplo, conter mídias e cenas de aplicações multimídia. Esse método pode ser redefinido nas subclasses.

6.3 Considerações Finais

Este capítulo objetivou apresentar o projeto AMOM, que tem como módulos o SOMm (Servidor de Objetos Multimídia) e o *MAW (Multimedia Application WebBuilder*). Foram citadas também, todas as funcionalidades de cada módulo pertencente ao MAW e ao SOMm. Os módulos do MAW implementados como parte deste trabalho são: de conexão, de autenticação de usuários e de conversão. O módulo de conversão é composto de dois sub-módulos: conversão de aplicações MHEG-5 para documentos SMIL/HTML e conversão de documentos SMIL/HTML para aplicações MHEG-5. Esse módulo de conversão é detalhado no capítulo 7.

CAPÍTULO 7

Módulo de Conversão

Neste capítulo são apresentados os dois tipos de conversão, utilizando como suporte às conversões, uma base de regras. Além disso, são relatadas as limitações entre os padrões SMIL e MHEG-5 encontradas no processo de conversão.

A figura 7.1 mostra a esquematização do sistema para realizar a conversão entre aplicações MHEG-5 e documentos SMIL/HTML. Para indicar a interação entre o conversor, o SOMm, a interface em Java e as *packages* iso.mheg5 e *Rule_Db*, foram criados alguns passos que estão enumerados nas setas, e são mostrados na figura 7.1.

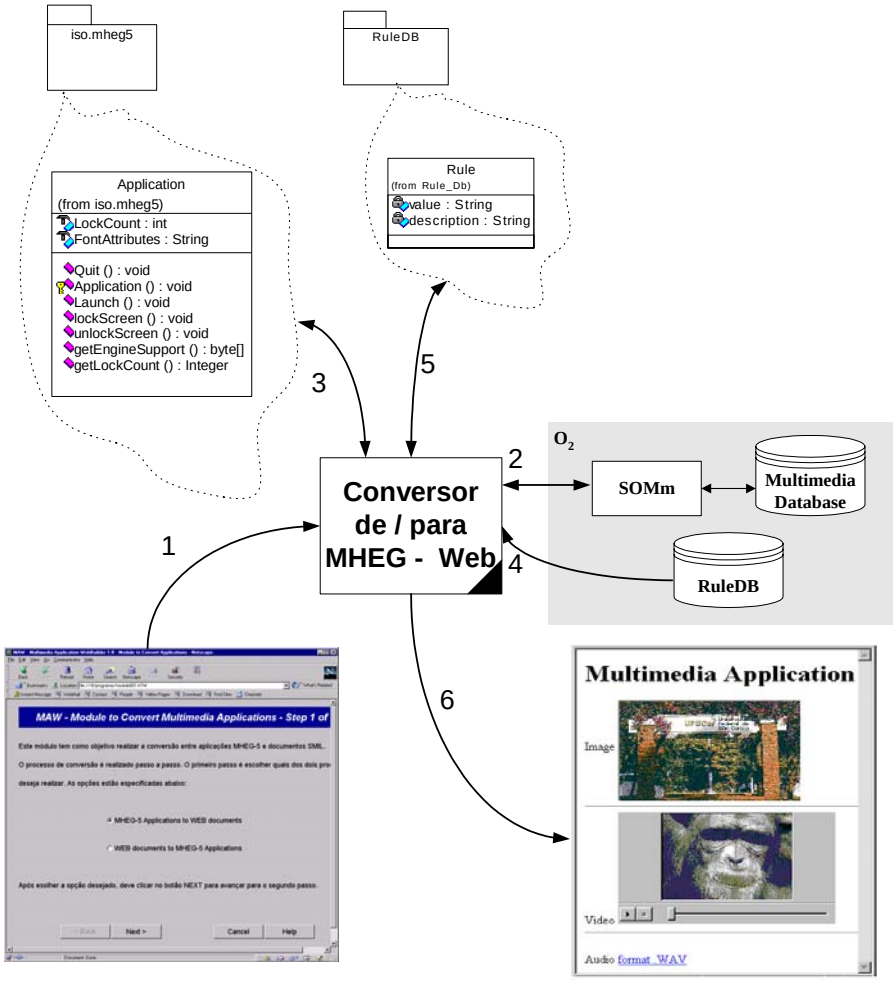


FIGURA 7.1 – ESQUEMATIZAÇÃO DO SISTEMA PARA CONVERSÃO DE APLICAÇÕES MHEG-5 EM DOCUMENTOS SMIL

No passo 3 da figura 7.1 é representada a interação entre o conversor e a *package* denominada iso.mheg5. Essa *package* foi criada para representar todas as classes MHEG-5 e está implementada em Java. O objetivo principal da *package* é auxiliar nas conversões, instanciando objetos nas classes MHEG-5 que a compõem. Em uma consulta de uma aplicação MHEG-5, o conversor busca todos os objetos da aplicação MHEG-5 contida no banco de dados multimídia (através do SOMm), e cria instâncias desses objetos em classes MHEG-5 correspondentes na *package*. O mesmo ocorre para a *package* Rule_Db no passo 5, a qual possui um conjunto de classes que representam regras de conversão. A *package Rule_Db* também está implementada em Java, e é utilizada no processo de conversão de aplicações multimídia.

Com a criação dessas duas *packages,* o conversor pode realizar as transformações utilizando os objetos instanciados nas *packages*, evitando-se assim desperdício de tempo de acesso aos bancos de dados.

No projeto foram tratados dois tipos de conversão de aplicações multimídia: MHEG-5 para SMIL/HTML e vice-versa, isto é, de SMIL/HTML para MHEG-5. A seção 7.1 detalha o primeiro tipo de conversão e a seção 7.2 o segundo. A base de regras dá suporte aos dois tipos de conversão.

7.1 Conversão de Aplicações MHEG-5 para documentos SMIL/HTML

A conversão de aplicações MHEG-5 para documentos SMIL/HTML, para que sejam disponibilizados na *Web*, exige um tratamento de toda a semântica embutida nas aplicações multimídia armazenadas nas classes MHEG-5, conforme citado anteriormente no capítulo 5.

O processo de conversão de aplicações MHEG-5 para documentos SMIL/HTML é realizado através dos passos que correspondem aos números da figura 7.1:

1. O usuário informa a aplicação MHEG-5 para a conversão;

2. O conversor busca a aplicação MHEG-5 no banco de dados multimídia do SOMm;

3. O conversor instancia os objetos que compõem a aplicação MHEG-5 nas classes MHEG-5 correspondentes da *package* iso.mheg5;

4. O conversor busca todas as regras de conversões no banco de dados RuleDB ;

5. O conversor instancia os objetos de cada classe do RuleDB em suas respectivas classes da *package Rule_Db* e

6. O conversor então converte a aplicação MHEG-5 em documento(s) SMIL/HTML, conforme as regras de conversão e logo após o(s) documento(s) está(ão) pronto(s) a ser(em) utilizado(s).

No processo de conversão, é solicitado para que o usuário informe se seja desejável armazenar a aplicação MHEG-5 convertida. Caso o usuário escolha essa opção, os documentos gerados são armazenados no banco de dados multimídia do SOMm, para que, em futuras conversões, esses documentos sejam utilizados, evitando desperdício de tempo no processo de conversão. Esses documentos são armazenados em uma classe chamada File_SMIL_HTML, e sua especificação é dada na figura 7.2. A classe armazena informações como: nome do usuário, nome da aplicação, nome da cena, nome do *browser,* data/hora que foi realizado o armazenamento, o próprio documento e uma composição de todos os documentos pertencentes à aplicação MHEG-5 convertida.

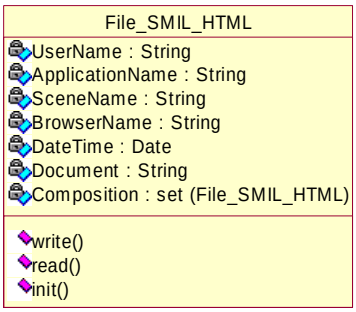


FIGURA 7.2 – DEFINIÇÃO DA CLASSE FILE_SMIL_HTML

Abaixo é definido o algoritmo para realizar a conversão de aplicações MHEG-5 para documentos SMIL/HTML.

Início
1. Mostrar página principal da aplicação
(página HTML do conversor para *login* do usuário)
2. Identificar o Usuário (utiliza métodos do módulo de autenticação de usuários)
3. Receber o nome da aplicação MHEG-5 a ser convertida (***getApplication()***)
4. Caso essa aplicação já esteja convertida então
  4.1. Buscar a aplicação na classe File_SMIL_HTML (***provideFileSMILHTML()***)
Caso contrário
  4.1. Buscar a aplicação MHEG-5 no banco de dados do SOMm (***provideApplication()***)
  4.2. Criar (mapear) as instâncias para classes da *package* MHEG-5 (***copyPackage()***)
  4.3. Buscar no O2 a base de regras para a conversão (***provideRuleDB()***)
  4.4. Criar (mapear) as instâncias da base de regras para classes da *package* RuleDb
    (***copyPackage()***)
  4.5. Para cada instância das classes da *package* MHEG-5 faça
    4.5.1. Selecionar as regras referentes a instância (***selectRule()***)
    4.5.2. Para cada regra de conversão encontrada faça
      4.5.2.1. Realizar a conversão gerando arquivo SMIL/HTML
        (***translateApplication()***)
      4.5.2.2. Ler próxima regra de conversão (***selectRule()***)
    4.5.3. Fim-Para
    4.5.4. Ler próxima instância da classe MHEG-5
  4.6. Fim-Para
  4.7. Caso o usuário tenha escolhido criar vários documentos (páginas) SMIL/HTML então
    4.7.1. Criar os documentos de acordo com cada cena da aplicação
      (***makeDocuments()***)
  4.8. Fim-Caso
  4.9. Caso o usuário escolha opção para armazenar o(s) documento(s) SMIL/HTML então
    4.9.1. Armazenar o(s) documento(s) SMIL/HTML gerados no banco de dados do
      SOMm (File_SMIL_HTML) (***storageApplication()***)
    4.9.2. Realizar *dowload* da aplicação multimídia convertida
  4.10.Fim-Caso
5. Fim-Caso
Fim

No algoritmo são informados, em negrito, os nomes dos métodos que foram implementados para realizar as ações do processo de conversão. A especificação de cada método é dada a seguir, citando o seu nome, a classe a que pertence o tipo de retorno que possui o objetivo e um algoritmo que mostra a funcionalidade do método.

Método....:	***getApplication(String nameApplication)***		
Classe.....:	**Module001**	Retorno:...:	**null**
Objetivo...:	Receber o nome da aplicação MHEG-5 para realizar a conversão		
Algoritmo			
. Recebe o nome da Aplicação MHEG-5, a ser convertida, em documentos SMIL/HTML			

Método....:	***provideFileSMILHTML(String nameFileSMILHTML)***		
Classe.....:	**Module001**	Retorno....:	**File_SMIL_HTML**
Objetivo...:	Recupera a aplicação MHEG-5 já convertida em documentos SMIL/HTML		
Algoritmo			
. Busca a aplicação MHEG-5 já convertida em documentos SMIL/HTML na classe File_SMIL_HTML do RuleDB . Executa o método *queryObject()* da classe O2Connection . Executa o método *read()* da classe File_SMIL_HTML do RuleDB			

Método....:	***provideApplication(String nameApplication)***		
Classe.....:	**Module002**	Retorno....:	**Application**
Objetivo...:	Recupera a aplicação MHEG-5 no banco de dados multimídia do SOMm		
Algoritmo			
. Busca, no banco de dados multimídia do SOMm, a aplicação MHEG-5 solicitada . Executa o método *queryObject()* da classe O2Connection . Executa o método *read()* da classe Application do MHEG-5 . Executa o método construtor da classe Application pertencente à *package* iso.mheg5			

Método....:	***copyPackage()***		
Classe.....:	**Module002**	Retorno....:	**null**
Objetivo...:	Cria instância para todas as classes MHEG-5 correspondente na *package* iso.mheg5		
Algoritmo			
. Cria instância das classes MHEG-5 para a suas correspondentes na *package* iso.mheg5 . Executa o método construtor de cada classe instanciada na *package* iso.mheg5			

Método....:	***provideRuleDB()***		
Classe.....:	**Module002**	Retorno....:	**null**
Objetivo...:	Recupera todas as instâncias de todas as classes pertencentes a RuleDB		
Algoritmo			
. Busca no banco de dados RuleDB todas as instâncias de todas as classes para serem instâncias nas classes da *package* Rule_Db . Executa o método *queryObject()* da classe O2Connection para cada classe da Rule_Db . Executa o método *read()* de cada classe pertencente a Rule_Db . Executa o método construtor de cada classe instanciada na *package* Rule_Db			

Método....:	***selectRule (String nameClass)***		
Classe.....:	**Module003**	Retorno....:	**Vector of (Rule)**
Objetivo...:	Seleciona todas a regras de conversão pertencente à uma classe ou atributo		
Algoritmo			
. Seleciona todas as regras de conversão na classe Rule da *package* Rule_Db para uma classe e/ou para os atributos dessa classe . Executa o método *read()* da classe Rule pertencente à *package* Rule_Db			

Método....:	***translateApplication(String contentRule)***		
Classe.....:	**Module003**	Retorno....:	**String**
Objetivo...:	Realiza a conversão de um atributo ou classe para as *tags* correspondentes em SMIL/HTML.		
Algoritmo			
. Associa conteúdo da regra ao documento . texto= tag.command + parameter.name + rule.value + rule.description . Retorna (texto)			

Método....:	***makeDocuments(String nameDocument)***		
Classe.....:	**Module003**	Retorno....:	**null**
Objetivo...:	Cria vários documentos SMIL/HTML de acordo com a quantidade de cenas contida na aplicação MHEG-5		
Algoritmo			
. Cria documentos SMIL/HTML para as cenas da aplicação MHEG-5			

Método....:	***storageApplication()***		
Classe.....:	**Module004**	Retorno....:	**null**
Objetivo...:	Armazena os documentos SMIL/HTML gerados na classe File_SMIL_HTML		
Algoritmo			
. Armazena os documentos SMIL/HTML na classe File_SMIL_HTML . Executa o método *insertObject()* da classe O2Connection . Executa o método *write()* da classe File_SMIL_HTML do RuleDB			

A seguir é dado um exemplo de conversão de uma aplicação simples MHEG-5 contendo uma imagem, um botão e um *link-action* para mudar de cena.

MHEG-5

```
Application.name = “exemplo”
Bitmap.Content = “mapa.gif”
PushButton.Label= “visualizar”
Link-Effect.Action(:Transition_to (“scene01”))
Link-Effect.Action(:Quit(“exemplo”))
```

O processo de conversão, irá gerar um arquivo SMIL e/ou HTML com as seguintes características:

SMIL	HTML
<smil> <head> meta name = “title” content= “exemplo” </head> <body> <seq> <img src = “mapa.gif” > Scene1 Quit </seq></body></smil>	<html> <title>exemplo</title> <body> <img src=”mapa.gif”> <input type = “button” value=”visualizar”> <input type = “button” onClick=”window.location.href=”scene01”> <input type = “button” onClick=”window.close()”> </body><html>

O quadro a seguir, mostra algumas regras de conversão que foram utilizadas para esse exemplo de conversão. Os atributos cujos nomes iniciam com ":", indicam atributos que armazenam ações.

MHEGClass. name	Attribute. name	Rule. value	Rule. description	Tag. command	Tag. language	Parameter. name
Bitmap	Content	Null	Value_of Content / SRC	<img>	Html	src
PushButton	Label	Null	Value_of Label/Value	<input type = "button" >	Html	value
Link	Link-Effect(Actions)	-	-	-	-	-
Action	:Transition_to	"window. location.href= "	Value_of scene/page	<input type = "button" >	Html	OnClick
Action	:Quit	"window. close()"	-	<input type = "button" >	Html	OnClick
Application	Name	Null	Value_of Name	<title>	Html	-
Bitmap	Content	Null	Value_of Content/SRC	<img>	Smil	src
Action	:Transition_to	Null	Value_of scene/page	<a>	Smil	href
Text	Content	Null	Value_of Content	<text>	Smil	src
Video	Content	Null	Value_of Content	<video>	Smil	src
Application	Name	Null	Value of Name	<region>	Smil	title

7.2 Conversão de documentos SMIL/HTML para Aplicações MHEG-5

Para realizar a conversão de documentos SMIL/HTML para aplicações MHEG-5 é necessário realizar os seguintes passos da figura 7.1:

1. O usuário informa a URL ou o arquivo que contém o(s) documento(s) SMIL/HTML para a conversão, escolhendo a Segunda opção (seta 1);

2. O conversor busca todas as regras de conversões no banco de dados RuleDB (4);

3. O conversor instancia os objetos de cada classe do RuleDB em suas respectivas classes da *package Rule_Db (5)*;

4. O conversor então transforma o(s) documento(s) SMIL/HTML para uma ou várias aplicações MHEG-5, conforme as regras de conversão e instancia as classes que conterão instancias para a(s) aplicação(ões) criada(s) (3) e

5. A(s) aplicação(ões) então é(são) armazenada(s) no banco de dados multimídia do SOMm (2).

Nesse processo de conversão, cada documento pode ser convertido em uma cena de uma única aplicação, ou caso o usuário desejar, pode-se gerar para cada documento uma nova aplicação multimídia.

Abaixo é definido o algoritmo para realizar a conversão de documentos SMIL/HTML para aplicações MHEG-5.

Início
1. Mostrar página principal da aplicação
   (página HTML do conversor para *login* do usuário)
2. Identificar o Usuário (utiliza métodos de autenticação de usuários)
3. Receber o(s) nome(s) do(s) documento(s) *Web* a ser(em) convertida(s) (***getDocument()***)
4. Buscar o(s) documento(s) *Web* de acordo com a URL(s) ou nome do(s) arquivo(s) informado(s) (***provideDocument()***)
5. Mostrar o conteúdo de cada documento *Web* (***displaySource()***)
6. Buscar no O2 a base de regras para a conversão (***provideRuleDB()***)
7. Criar (mapear) as instância da base de regras para classes da *package* RuleDb (***copyPackage()***)
8. Para cada *tag* do documento *Web* faça
   8.1. Selecionar as regras referentes a *tag* (***selectRule()***)
   8.2. Para cada regra de conversão encontrada faça
      8.2.1. Realizar a conversão gerando aplicação MHEG-5 na *package* iso.mheg5 (***translateApplication()***)
      8.2.2. Ler próxima regra de conversão (***selectRule()***)
   8.3. Fim-Para
   8.4. Ler próxima *tag* do documento *Web*
9. Fim-Para
10. Caso o usuário deseja criar várias aplicações MHEG-5 então
   10.1. Criar aplicações MHEG-5 de acordo com os documentos *Web* (***makeApplications()***)
11. Fim-Caso
12. Para cada aplicação MHEG-5 gerada faça
   12.1. Armazenar a(s) aplicação(ões) MHEG-5 gerada(s) na *package* iso.mheg5 para o banco de dados multimídia do SOMm (Application) (***storeApplication()***)
13. Fim-Para
Fim

Os métodos destacados em negrito são detalhados abaixo, com exceção daqueles já especificados na seção anterior.

Método....:	***getDocument(String nameDocument)***		
Classe.....:	**Module021**	Retorno:...:	**null**
Objetivo...:	Receber a URL ou nome do documento SMIL/HTML para realizar a conversão		
Algoritmo			
. Recebe a URL ou o nome do documento SMIL/HTML a ser convertida em aplicações MHEG-5			

Método....:	***provideDocument(String nameDocument)***		
Classe.....:	**Module022**	Retorno....:	**null**
Objetivo...:	Recupera o documento SMIL/HTML de acordo com o arquivo ou URL.		
Algoritmo			
. Busca no arquivo ou na URL o documento SMIL/HTML solicitado			

Método....:	***displaySource(String nameDocument)***		
Classe.....:	**Module022**	Retorno....:	**null**
Objetivo...:	Mostra o código fonte do documento SMIL/HTML		
Algoritmo			
. Mostra o conteúdo de um documento SMIL/HTML.			

Método....:	***translateApplication(String contentRule)***		
Classe.....:	**Module023**	Retorno....:	**MHEG-5 Class**
Objetivo...:	Realiza a conversão de uma *tag* para um atributo ou classe correspondente em MHEG-5		
Algoritmo			
. Associa conteúdo da regra a classe ou ao atributo da classe . classe= MHEGClass.Name + Attribute.Name + rule.description . Retorna (classe)			

Método....:	***makeApplications(String nameApplication)***		
Classe.....:	**Module024**	Retorno....:	**null**
Objetivo...:	Cria várias aplicações MHEG-5 de acordo com a quantidade de documentos SMIL/HTML		
Algoritmo			
. Cria aplicações MHEG-5 para os documentos SMIL/HTML			

Método....:	***storageApplication()***		
Classe.....:	**Module024**	Retorno....:	**null**
Objetivo...:	Armazena as aplicações MHEG-5 nas classes MHEG-5 do banco de dados multimídia do SOMm		
Algoritmo			
. Armazena as aplicações MHEG-5 na classe MHEG-5 do banco de dados multimídia SOMm . Executa o método *insertObject()* da classe O2Connection . Executa o método *write()* da classe Application e Scene			

Esta seção foi dedicada a definição do processo de conversão de documentos SMIL/HTML para aplicações MHEG-5. Como auxílio a esses dois tipos de conversões, foi utilizada uma base de regras que é especificada na próxima seção.

7.3 Base de Regras de Conversão

Para realizar as conversões entre aplicações MHEG-5 e SMIL foi desenvolvida uma base de regras para armazenar as regras de conversões. Observa-se na literatura que a maioria dos métodos de conversão existentes, como os citados em [Doorn94] e [Yang96], trabalham com uma base de regras de auxílio para realizar transformações. Essas bases de regras são normalmente estáticas, isto é, são impostas na implementação de acordo com a situação do momento (regras existentes), e quando surge a necessidade de impor novas regras, estas são inseridas na própria implementação dos conversores. Uma desvantagem desse processo é a atualização contínua do código do conversor mediante novas regras. Com o intuito de evitar esse problema, decidiu-se criar como auxílio ao conversor, uma base de regras dinâmica**.** Essa base de regras é representada por uma estrutura de classes e tipos, para armazenar as regras de conversão. Assim, quando surgir a necessidade de impor novas regras, será necessário apenas inserir (instanciar) novos objetos nas classes, sem que haja alteração no conversor em si.

Para que o conversor obtenha um bom desempenho, e também para mantê-lo atualizado, é necessário que haja flexibilidade de mudanças na base de regras. A base de regras é um dos componentes importantes no processo de conversão, e atualmente possui uma estrutura de dados para suportar linguagens especificadas a partir de SGML, isto é, linguagens que tratam aplicações multimídia de forma textual através de uso de comandos específicos, também chamados de *tags*.

Na figura 7.3 é mostrada a estrutura da base de regras (denominada RuleDB), juntamente com os relacionamentos existentes entres as classes e tipos. A figura utiliza a notação UML para a representação.

Para a definição da base de regras foram definidas as classes: *MHEGClass, Tag, Parameter* e *Rule* e os dois tipos: *Extension_Tag* e *Attribute*.

A idéia geral é ter uma estrutura de classes para armazenar todos os elementos (meta informações) envolvidos nos padrões MHEG-5 e HTML/SMIL e as correspondências existentes entre os elementos desses padrões, para dar suporte à conversão de aplicações, de um padrão para o outro.

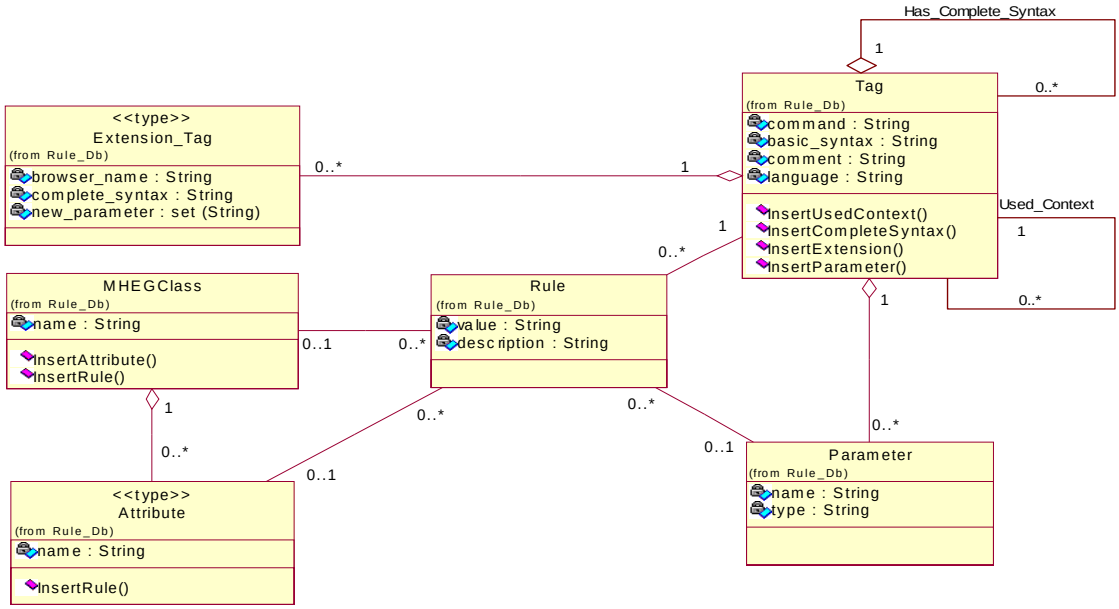


FIGURA 7.3 – ESTRUTURA DA RULEDB PARA ARMAZENAMENTO DE REGRAS DE CONVERSÃO

A estrutura da RuleDB foi implementada em O_2C e Java. A implementação em O_2C foi necessária para criar o banco de dados para armazenar as regras de conversão. Já a sua implementação em Java foi necessária, devido ao fato do conversor utilizar essa estrutura para realizar as conversões, isto é, o conversor busca as regras no banco de dados RuleDB e cria instâncias nas respectivas classes em Java. Com a adoção desse critério, o processo de conversão se torna mais rápido, pois não é necessário acessar a todo o momento o banco de dados de conversão RuleDB.

7.3.1 Elementos do padrão MHEG-5

Para representar as classes do padrão MHEG-5 foi criada a classe *MHEGClass*. Esta classe é composta por um único atributo chamado *name*, que armazena o nome de cada classe MHEG-5. Essa classe possui um relacionamento de associação com a classe *Rule* para representar as regras de conversão que a classe MHEG-5 pode possuir independentemente de seus atributos. O quadro 01 mostra algumas instâncias da classe *MHEGClass* e o quadro 02 apresenta a implementação dessa classe tanto na linguagem Java como na linguagem O_2C do SGBDOO O2.

Quadro 01 – Exemplos de instâncias da Classe MHEGClass

MHEGClass
Name
Application
Scene
Ingredient
Bitmap
Audio
Text
Video
Link
Action

Quadro 02 – Implementação da classe MHEGClass em Java e O_2C

MHEGClass.java	**MHEGClass.o2**

public class MHEGClass {
// attributes
String name;
Vector attributes;
//services
MHEGClass(String x_name) {
name = x_name;
attributes = new Vector();
rules = new Vector();
}
 	
public void InsertAttribute (Attribute x_attributes) {
attributes.addElement(x_attributes); }
 	
public void InsertRule (Rule x_rules) {
rules.addElement(x_rules); }
}
	class MHEGClass inherit Object
	public type
	tuple(name: string,
	attributes: set(Attribute),
	rules: list(Rule))
	end;

Uma classe MHEG-5 pode ou não possuir atributos. Para representar os atributos das classes MHEG-5 foi criado o tipo *Attribute*, que possui o atributo *name* para armazenar o nome do atributo.

A classe *Attribute* possui um relacionamento de associação com a classe *Rule***.** Essa associação representa as regras de conversão para os atributos da classe MHEG-5. No quadro 03 são indicados alguns atributos das classes MHEG-5 e no quadro 04 são apresentadas as especificações em Java e O_2C de *Attribute*.

Quadro 03 – Exemplos de instâncias da Classe MHEGClass *e seus Atributos*

MHEGClass	**Attribute**
Name	**Name**
Application	Name, . . .
Scene	Scene_Coordinate_System, Input_Event_Register, . . .
Ingredient	Object_Identifier
Bitmap	Content, Transparency, Tiling, OriginalTransparency

Quadro 04 – Implementação do tipo Attribute em Java e O_2C

Attribute.java	**type Attribute**

public class Attribute {
// attributes
String name;
Vector content_rule;
//services
Attribute (String x_name) {
name = x_name;
content_rule = new Vector(); }
 	
public void InsertRule (Rule x_content_rule) {
content_rule.addElement(x_content_rule); }
}
	type Attribute :tuple(name: string,
	content_rule: list(Rule));

7.3.2 Elementos dos padrões SMIL e HTML

Para armazenar as informações referentes às linguagens HTML e SMIL, criaram-se a classe *Tag* que armazena todas as informações referentes aos comandos utilizados nessas linguagens. Essa classe é composta pelos atributos: *command, basic_syntax, comment* e *language.*

Os quadros 05 e 06 respectivamente, são dados, exemplos de instâncias da classe *Tag* e a especificação dessa classe em Java e O_2C.

Quadro 05 – Exemplos de instâncias da classe Tag

Tag			
command	**basic_syntax**	**comment**	**language**
A	<A> characters </A>	Null	HTML
BODY	<BODY></BODY>	Null	HTML
BR	 	Null	HTML
HEAD	<HEAD></HEAD>	Null	HTML
HTML	<HTML></HTML>	Null	HTML
TITLE	<TITLE<</TITLE>	Null	HTML
P	<P> characters	Null	HTML
ADDRESS	<ADDRESS>characters</ADDRESS>	Null	HTML
APPLET	<APPLET></APPLET>	Null	HTML
B	<B> characters </B>	Null	HTML
SMIL	<SMIL> </SMIL>	Null	SMIL
LAYOUT	<LAYOUT> </LAYOUT>	Null	SMIL
REGION	<REGION> </REGION>	Null	SMIL
PAR	<PAR> </PAR>	Null	SMIL
SEQ	<SEQ> </SEQ>	Null	SMIL
ANIMATION	<ANIMATION> </ANIMATION>	Null	SMIL
AUDIO	<AUDIO> </AUDIO>	Null	SMIL
VIDEO	<VIDEO> </VIDEO>	Null	SMIL
TEXTSTREAM	<TEXTSTREAM> </TEXTSTREAM>	Null	SMIL

Quadro 06 – Implementação da classe Tag em Java e O_2C

Tag.java	**Tag.o2**

public class Tag {
// attributes
String command;
String basic_syntax;
String comment;
String language;
Vector used_context;
Vector complete_syntax;
Vector extension;
Vector parameters;
//services
	Class Tag inherit Object
	public type
	tuple(command: string,
	basic_syntax: string,
	comment: string,
	language: string,
	extension:set(Extension_Tag),
	used_context: list(Tag),
	complete_syntax: list(Tag),
	parameters: list(Parameter))

Tag (String x_command, String x_basic_syntax, String x_comment, String x_language) {
Command = x_command;
Basic_syntax = x_basic_syntax;
Comment = x_comment;
Language = x_language;
Used_context = new Vector();
Complete_syntax = new Vector();
Extension = new Vector();
parameters = new Vector(); }
public void InsertUsedContext (Tag x_used_context) {
used_context.addElement(x_used_context); }
 	
public void InsertCompleteSyntax
(Tag x_complete_syntax) {
complete_syntax.addElement(x_complete_syntax); }
 	
public void InsertExtension
(Extension_Tag x_extension) {
extension.addElement(x_extension); }
 	
public void InsertParameter (Parameter x_parameters) {
parameters.addElement(x_parameters); }
}
	method
	public init(command: string),
	public write(val: pointer),
	public read(val: pointer)
	end;

O atributo *command* é usado para armazenar o nome de cada comando das linguagens HTML e SMIL, o qual é denominado *tag*. Os elementos básicos para o funcionamento das *tags* são armazenados no atributo *basic_syntax*. A sintaxe completa de uma *tag* é representada através da agregação *Has_Complete_Syntax*.

Além dessa agregação, a classe *Tag* possui um relacionamento de associação chamado *Used_Context*, que representa as *tags* que podem ser utilizadas dentro de um determinado contexto, isto é, as *tags* que podem ser utilizadas a partir de uma *tag* específica. Pode-se observar no quadro 07 alguns exemplos de sintaxe completa e contexto de algumas *tags*.

O atributo *comment* é utilizado apenas para registrar comentários sobre a *tag*. Esses comentários são declarados à escolha do especialista, podendo ser explicativos ou ilustrativos. É necessário também identificar a linguagem a qual a *tag* pertence, sendo que essa informação é registrada no atributo *language***.** Atualmente, o atributo *language* recebe apenas um dos dois valores: HTML ou SMIL, isto é *tags* pertencentes à linguagem HTML ou à linguagem SMIL. A classe *Tag* poderá suportar outras linguagens, desde que estas façam uso de *tags* para descrever seus comandos.

Quadro 07 – Exemplos de instâncias da classe Tag com os atributos used_context e complete_syntax

Tag			
Command	**used_context**	**complete_syntax**	**language**
BODY	<HTML>	<H1> <H2> <H3> <H4> <H5> <H6> <P> <UL> <OL> <DIR> <MENU> <DL> <PRE> <BLOCKQUOTE> <FORM> <ISINDEX> <HR> <ADDRESS>	HTML
BR	<A> <ADDRESS> <B> <CITE> <CODE> <DD> <DT> <EM> <H1> <H2> <H3> <H4> <H5> <H6> <I> <KBD> <LI> <P> <PRE> <SAMP> <STRONG> <TT> <VAR> <DFN> <U>	Null	HTML
P	<ADDRESS> <BLOCKQUOTE> <BODY> <DD> <FORM> <LI>	<A> <IMG> <EM> <STRONG> <CODE> <SAMP> <KBD> <VAR> <CITE> <TT> <B> <I> <BASEFONT> <BLINK> <FONT> <NOBR> <WBR> <APP> <DFN> <U>	HTML
LAYOUT	<HEAD> <LAYOUT> <SWITCH>	<A> <ANCHOR> <ANIMATION> <AUDIO> <BODY> <HEAD> <IMG> <LAYOUT> <META> <PAR> <SEQ> <REF> <REGION> <ROOT-LAYOUT> <SMIL> <SWITCH> <TEXT> <TEXTSTREAM> <VIDEO>	SMIL
BODY	<SMIL> <LAYOUT>	<PAR> <SEQ> <AUDIO> <VIDEO> <TEXT> <IMG> <ANIMATION> <TEXTSTREAM> <REF> <SWITCH> <A>	SMIL

As *tags* podem ou não ser compostas de parâmetros. Esses parâmetros são responsáveis por configurar ou especificar informações adicionais às *tags*. Os parâmetros são, na maioria das vezes, utilizados pelos *browsers*, que os interpretam e os executam.

Os parâmetros podem ser específicos para cada *browser* ou genéricos. Para representar essas possíveis situações, foi desenvolvida a classe *Parameter*, composta de dois atributos: *name* e *type***.** O atributo *name* armazena o nome do parâmetro; o tipo de informação que o parâmetro pode conter é armazenado no atributo *type***.** O tipo do parâmetro pode ser *string, integer* ou *real.* O quadro 08 demonstra alguns exemplos da classe *Parameter* e o quadro 09 mostra a implementação da classe *Parameter* em Java e O_2C.

Quadro 08 – Exemplos de instâncias da classe Parameter

Tag		Parameter	
name	**language**	**name**	**Type**
IMG	HTML	HREF	String
FONT	HTML	SIZE	Integer
HEAD	HTML	TITLE	String
SMIL	SMIL	ID	Integer
REGION	SMIL	TITLE	String
IMG	SMIL	REGION	String
PAR	SMIL	BEGIN	Integer
SEQ	SMIL	DUR	Integer

Quadro 09 – Implementação da classe Parameter em Java e O_2C

Parameter.java	Parameter.o2

public class Parameter {
// attributes
String name;
String type;
//services
Parameter (String x_name, String x_type) {
name = x_name;
type = x_type;
}
}
	class Parameter inherit Object
	public type
	tuple(name: string,
	type: string)
	end;

As *tags* são interpretadas e executadas pelos *browsers*, e cada *tag* pode ou não possuir extensões, isto é, parâmetros específicos que só funcionam em determinados *browsers*. Tentando representar essa situação, foi criado um tipo chamado *Extension_Tag*. Esse tipo representa informações sobre todas as extensões que as *tags* (comandos) possuem. O tipo é composto de três atributos: *browser_name, complete_syntax* e *new_parameter*. O atributo *browser_name* representa o nome do *browser* que possui extensões de *tags*. Para armazenar uma sintaxe nova da *tag*, que são interpretadas apenas por alguns *browsers*, é utilizado o atributo *complete_syntax*. O atributo *new_parameter* é usado para armazenar informações de novos parâmetros dessas *tags*. No quadro 10 são ilustrados alguns exemplos de instâncias. A implementação em Java e O_2C desse tipo é mostrado no quadro 11.

Quadro 10 – Exemplos de instâncias da classe Extension_Tag

Tag		Extension_Tag		
Command	**language**	**browser_name**	**complete_syntax**	**new_parameter**
BGSOUND	HTML	Internet Explorer 3	Null	SRC, LOOP
EMBED	HTML	Netscape Navigator	Null	ALIGN, BORDER, FRAMEBORDER, HEIGHT, HIDDEN, HSPACE, NAME, PALLETE, PLUGINSPACE, SRC, TYPE, VSPACE, WIDTH
EMBED	HTML	Internet Explorer 3	Null	ALIGN, BORDER, FRAMEBORDER, HEIGHT, HIDDEN, HSPACE, NAME, PALLETE, PLUGINSPACE, SRC, TYPE, VSPACE, WIDTH

Quadro 11 – Implementação do tipo Extension_Tag em Java e O_2C

Extension_Tag.java	type Extension_Tag

public class Extension_Tag {
// attributes
String browser_name;
String complete_syntax;
String new_parameter;
//services
Extension_Tag (String x_browser_name, String x_complete_syntax, String x_new_parameter) {
browser_name = x_browser_name;
complete_syntax = x_complete_syntax;
new_parameter = x_new_parameter; }
}
	type Extension_Tag :tuple
	(browser_name: string,
	complete_syntax: string,
	new_parameter: set(string));

7.3.3 Regras de Conversões

A classe *Rule* representa informações sobre as regras de conversões. Essa classe é composta pelos atributos: *value* e *description***.** O atributo *value* armazena informações referentes às conversões que são feitas diretamente, isto é, quando o valor a ser convertido de uma classe MHEG-5 ou de um atributo é automaticamente associado às *tags* ou aos parâmetros das *tags*. Uma outra maneira de representar regras de conversão é através de uma interpretação de comandos, embutindo-se alguma semântica. Essas regras de conversão são armazenadas no atributo *description*. No quadro 12 são ilustrados alguns exemplos de regras de conversão para atributos de algumas classes MHEG-5. A implementação da classe *Rule* em Java e O_2C é mostrada no quadro 13.

Quadro 12 – Exemplos de regras de conversão

MHEGClass. Name	Attribute. name	Rule. Value	Rule. Description	Tag. Command	Tag. language	Parameter. name
Bitmap	Content	Null	Value_of Content / SRC	<img>	Html	src
PushButton	Label	Null	Value_of Label/Value	<input type = "button" >	Html	value
Link	Link-Effect(Actions)	-	-	-	-	-
Action	:Transition_to	"window. location.href= "	Value_of scene/page	<input type = "button" >	Html	OnClick
Action	:Quit	"window. close()"	-	<input type = "button" >	Html	OnClick
Application	Name	Null	Value_of Name	<title>	Html	-
Bitmap	Content	Null	Value_of Content/SRC	<img>	Smil	src
Action	:Transition_to	Null	Value_of scene/page	<a>	Smil	href
Text	Content	Null	Value_of Content	<text>	Smil	src
Video	Content	Null	Value_of Content	<video>	Smil	src
Application	Name	Null	Value of Name	<region>	Smil	title

Quadro 13 – Implementação da classe Rule em Java e O_2C

Rule.java	Rule.o2

public class Rule {
// attributes
String value;
String description;
Tag tag;
Parameter parameter;
//services
Rule (String x_value, String x_description,
Tag x_tag, Parameter x_parameter) {
value = x_value;
description = x_description;
tag = new Tag (x_tag);
parameter = new Parameter(x_parameter); }
}
	class Rule inherit Object
	public type
	tuple(tag: Tag,
	parameter: Parameter,
	value: string,
	description: string)
	end;

Com o desenvolvimento dessa base de regras, as conversões das aplicações MHEG-5 para documentos HTML/SMIL e vice-versa se tornam mais dinâmicas, isto é, não é

necessário alteração no código fonte do conversor para adaptar às novas linguagens de programação para a Web (desde que a nova linguagem utilize *tags* para representar comandos), e sim à instanciação de novas regras na base, que deve ser realizada por um especialista.

Na figura 7.4 é ilustrada a tela do *O2Tools* do O2 que contém uma instância da classe *MHEGClass* juntamente com algumas regras de conversão. Essas instâncias estão armazenadas na base de dados *RuleDB* e são gerenciadas pelo sistema O_2. Observe que na figura 7.4 a classe Bitmap do MHEG-5 possui uma regra de conversão e uma outra regra que é associada para o atributo *Content* dessa classe. Veja na figura também, que a descrição da regra (*Rule.description)* descreve que o valor do atributo será correspondente ao valor do parâmetro *src* da *tag* <img>.

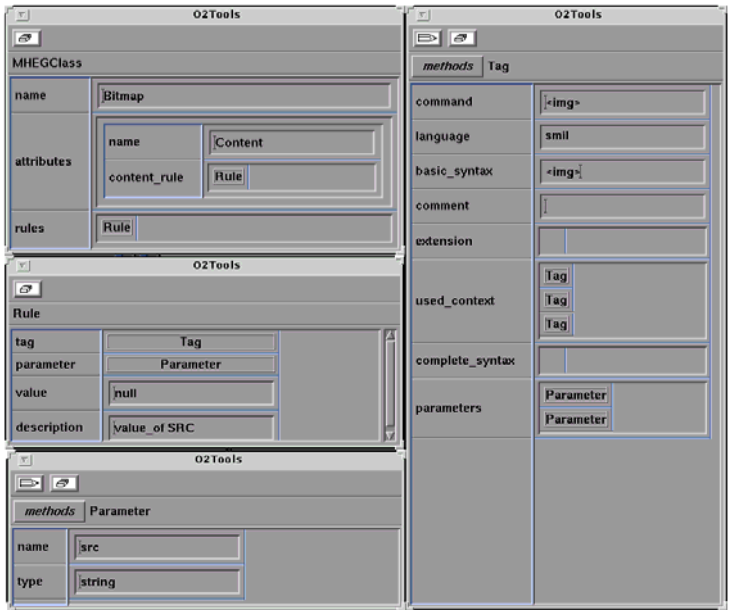


FIGURA 7.4 – INSTÂNCIAS DA CLASSE MHEGCLASS

7.4 Limitações do Processo de Conversão

Essa seção tem como objetivo mostrar quais foram as limitações encontradas no processo de conversão de aplicações MHEG-5 para documentos SMIL/HTML e de documentos SMIL/HTML para aplicações MHEG-5. Um dos primeiros critérios analisados foi a respeito da interação entre o usuário e a aplicação multimídia no momento da apresentação. Os conversores não tratam essa possibilidade, isto é, todas as ações realizadas durante a apresentação são executadas de acordo com as ações já geradas.

Foram analisadas algumas situações que podem ocorrer na conversão, e são especificadas abaixo:

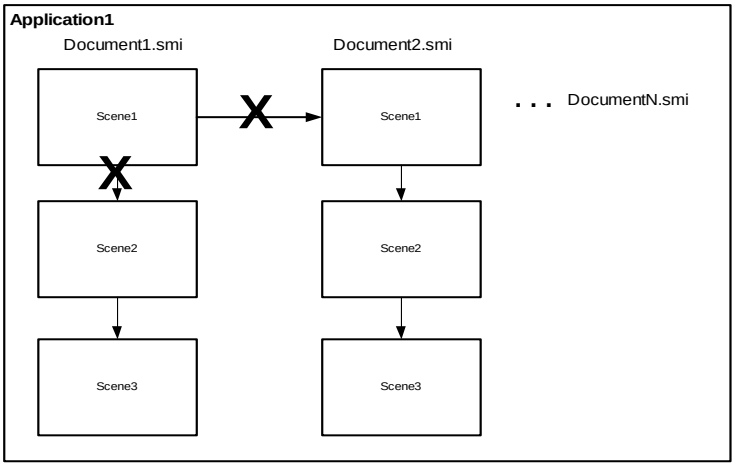
Situação 1: O primeiro critério analisado foi em relação ao tamanho de uma cena em uma aplicação MHEG-5 com os documentos SMIL/HTML. As aplicações MHEG-5 possuem um tamanho fixo. Já os documentos SMIL/HTML podem possuir qualquer tamanho. Apenas os documentos SMIL possuem o critério de construir regiões específicas na tela, caso contrário, não existe limite determinado. Para essa situação foram adotados os seguintes critérios:

- Conversão de Aplicações MHEG-5 para Documentos SMIL/HTML: como não existe limite nos documentos SMIL/HTML, o conversor simplesmente converte a aplicação MHEG-5.

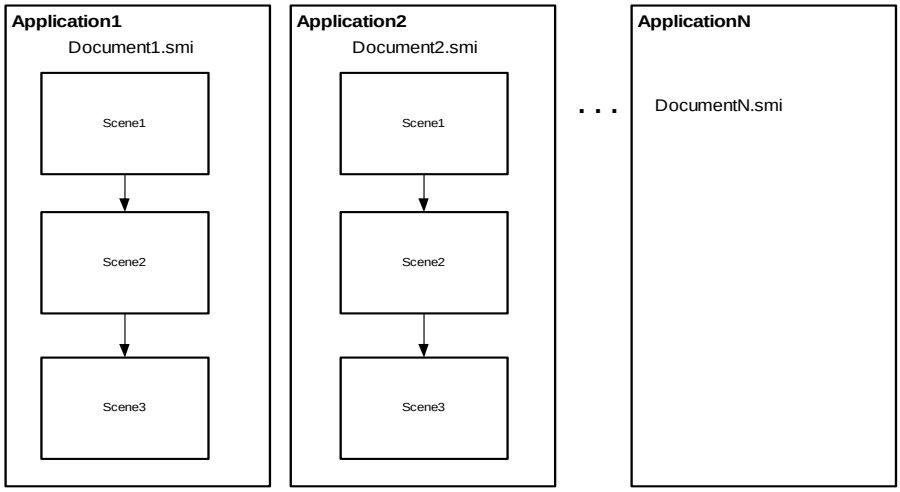
- Conversão de Documentos SMIL/HTML para Aplicações MHEG-5: nesse tipo de conversão podem ocorrer diversas situações, como:

  a-) Usuário escolher a conversão de um documento SMIL/HTML em uma aplicação MHEG-5: nesse caso serão geradas mais de uma cena para a aplicação MHEG-5, somente se o tamanho do documento SMIL/HTML ultrapassar o limite. Caso isso não ocorra, a conversão será realizada, gerando uma aplicação MHEG-5 contendo apenas uma cena.

  b-) Usuário escolher a conversão de vários documentos SMIL/HTML em uma aplicação MHEG-5: nesse caso pode ocorrer que nenhum dos documentos SMIL/HTML ultrapasse o limite determinado. Caso isso ocorra, é gerada uma aplicação MHEG-5 contendo várias cenas que correspondem aos documentos SMIL/HTML. Caso um dos documentos SMIL/HTML ultrapasse o limite, serão geradas uma aplicação para cada documento SMIL. Veja na representação gráfica a seguir, como ficaria a conversão se fosse gerada em apenas uma aplicação MHEG-5.



Como a cena possui apenas uma referência que indica a próxima cena da aplicação, nesse caso teriam de ser gerados duas referências. Como isso não é permitido em aplicações MHEG-5, o conversor cria uma aplicação para cada documento SMIL/HTML. Veja a representação gráfica dessa situação, que é a correta.



c-) Usuário escolher a conversão de vários documentos SMIL/HTML em várias aplicações MHEG-5: nesse caso não ocorre nenhum problema, pois para cada documento SMIL/HTML é gerada uma aplicação MHEG-5, contendo uma ou várias cenas, dependendo do tamanho do documento SMIL/HTML.

Situação 2: Outro critério analisado é em relação as ações MHEG-5 que podem ser mapeadas para os documentos SMIL/HTML e vice versa, quais as ações dos documentos SMIL/HTML que podem ser mapeadas para aplicações MHEG-5. Em relação a temporização entre objetos MHEG-5, é utilizado a ação *SetTimer* e pode estar apenas no

contexto de aplicação e cena. A outra ação que pode ser convertida é a *Transition To*, que representa a referência para outros objetos MHEG-5.

- *SetTimer:* para que essa ação seja gerada para documentos SMIL/HTML, simplesmente é utilizado um dos atributos: *begin* e *end* ou *dur*. Esses atributos podem ser especificados em diversas *tags* SMIL. Para HTML não teria como fazer a conversão de objetos MHEG-5 com temporização.

- Atributos *begin* e *end* ou *dur*: esses atributos tratam a temporização entre mídias, regiões, e outros. Somente em algumas condições é que serão gerados objetos MHEG-5 (ação *SetTimer*). O conversor não realiza a conversão de temporização de **mídias**, pois no SMIL é permitido, mas em MHEG-5 não existe temporização de mídias.

- *Transition To*: para que essa opção seja convertida para documentos SMIL/HTML, o conversor gera uma *tag* de referência (<a>). Para casos em que a referência esteja associada a um botão, a conversão gera também a *tag* <a>, pois não tem como representar botão em SMIL. O mesmo processo ocorre para a linguagem HTML, isto é, gera-se a *tag* <a>.

- *tag* <a>: quando essa *tag* for utilizada em documentos SMIL ou HTML, o conversor simplesmente gera a ação *Transition To* para o objeto MHEG-5 que contiver o *link*.

Situação 3: Algumas classes MHEG-5 não podem ser representadas em documentos SMIL/HTML, por exemplo:

- *CursorShape*: são armazenados vários tipos de representação de cursores. Em um documento SMIL/HTML não tem como modificar a representação do cursor do *mouse*.

- *Action* (parcialmente): conforme citado anteriormente, apenas as ações *SetTimer* e *Transition To* podem ser convertidos em *tags* para os documentos SMIL/HTML. As outras ações MHEG-5 são normalmente representadas pelos MHEG-5 *engines* no momento da apresentação da aplicação ou da cena MHEG-5.

- Button (HotSpot, PushButton, SwitchButton) (parcialmente): são convertidos em *tags* que representam *links* para outros objetos. A sua representação gráfica é perdida, isto é, com o uso de *tag* de referência, é mostrado graficamente apenas o *label* do botão.

Situação 4: Algumas *tags* SMIL e HTML não podem ser representadas em aplicações MHEG-5, e alguns exemplos de *tags* são ilustrados abaixo:

- SMIL:

<animation>: tipo de mídia não suportado no padrão MHEG-5.

<textstream>: tipo de mídia (texto juntamente com *stream*) que também não é suportado no padrão MHEG-5.

<layout>: determina o tamanho do documento SMIL para representar seus conteúdos.

<region>: determina uma área dentro do *layout*.

<switch>: é utilizado quando se deve escolher uma entre duas ou várias opções.

- HTML:

<applet>: representa a chamada de uma programa *JavaApplet.*

<script>: armazena o contéudo de código de programação de linguagens *scripts*.

<area>: determina o tamanho para uma dada área.

<frame>: especifica como é repartido o documento HTML, determinando algumas regiões.

<table>: utiliza o formato de tabela para representar informações.

Enfim, essas são algumas limitações encontradas pelos conversores. Foram mostradas também algumas adaptações feitas para a realização das conversões.

7.5 Considerações Finais

Este capítulo foi dedicado ao processo de conversão de aplicações multimídia. O processo de conversão é classificado em: conversão de aplicações MHEG-5 em documentos SMIL/HTML e conversão de documentos SMIL/HTML em aplicações MHEG-5. A seção 7.1 trata todos os critérios de conversão de aplicações MHEG-5 em documentos SMIL/HTML. Ainda nessa seção, é apresentado um algoritmo especificando o funcionamento do conversor para esse tipo de conversão. Os métodos criados em Java também são citados na seção 7.1. Já na seção 7.2 foi especificado o processo de conversão de documentos SMIL/HTML para aplicações MHEG-5. Assim como na seção anterior, também é especificado na seção 7.2 o algoritmo para esse tipo de conversão, juntamente com os métodos criados em Java. Os dois tipos de conversão utilizam uma base de regras para auxílio às conversões. A especificação de todas as classes e tipos contidos na base de

regras é citada na seção 7.3. A base de regras é composta de: elementos do padrão MHEG-5 elementos do padrão SMIL/HTML e regras de conversão. Para cada classe da base de regras foram criadas e demonstradas instâncias, e também foram citados os códigos fontes em Java e O_2C da implementação de cada classe.

Com a especificação dos conversores citados neste capítulo, fica faltando descrever os critérios de implementação. Esses critérios são relatados no próximo capítulo.

CAPÍTULO 8

Aspectos de Implementação

Neste capítulo são discutidos aspectos sobre a implementação do projeto. Foram implementados os módulos: de conversão, de conexão e de autenticação de usuários, que foram detalhados no capítulo 6, como componentes do ambiente MAW. Inicialmente são abordados na seção 8.1 os aspectos relativos à configuração do ambiente para a execução dos módulos e também são relatadas as características sobre as estruturas de dados criadas para suporte às conversões. Na seção 8.2 são apresentadas as interfaces desenvolvidas para o módulo de conversão, sendo especificados dois exemplos de conversão, sendo o primeiro a conversão de uma aplicação MHEG-5 para documentos SMIL e o segundo de documentos SMIL sendo convertidos para uma aplicação MHEG-5.

8.1 Configuração do Ambiente / Estrutura de Dados

A figura 8.1 mostra a interação entre o cliente *Web* e o servidor *Web*. O cliente *Web* é o usuário do conversor e o servidor *Web* é quem está controlando a interação entre o usuário e o conversor. O conversor foi desenvolvido utilizando-se a linguagem de programação orientada a objeto Java. Para os usuários realizarem as conversões, inicialmente deve-se solicitar um programa *JavaApplet*. Para fazer isso, o usuário deve especificar a URL onde está localizado esse programa. Assim que o *JavaApplet* estiver carregado no *browser*, o usuário poderá então utilizar o conversor. O programa, por sua vez, realiza algumas operações específicas, como realizar conexão com o banco de dados multimídia do SOMm e manipular objetos multimídia. Como o módulo de conexão utiliza-se de métodos escritos em C, foi necessário integrar os programas *JavaApplet,* Java, C e métodos das classes MHEG-5. As informações multimídia que são transferidas entre o cliente *Web* e servidor *Web* são aquelas que foram transformadas pelo conversor. Os programas escritos em Java são responsáveis pelo recebimento e envio de objetos MHEG-5. Já os programas escritos em C e O_2C tratam do armazenamento e recuperação de objetos MHEG-5 do banco de dados do SOMm.

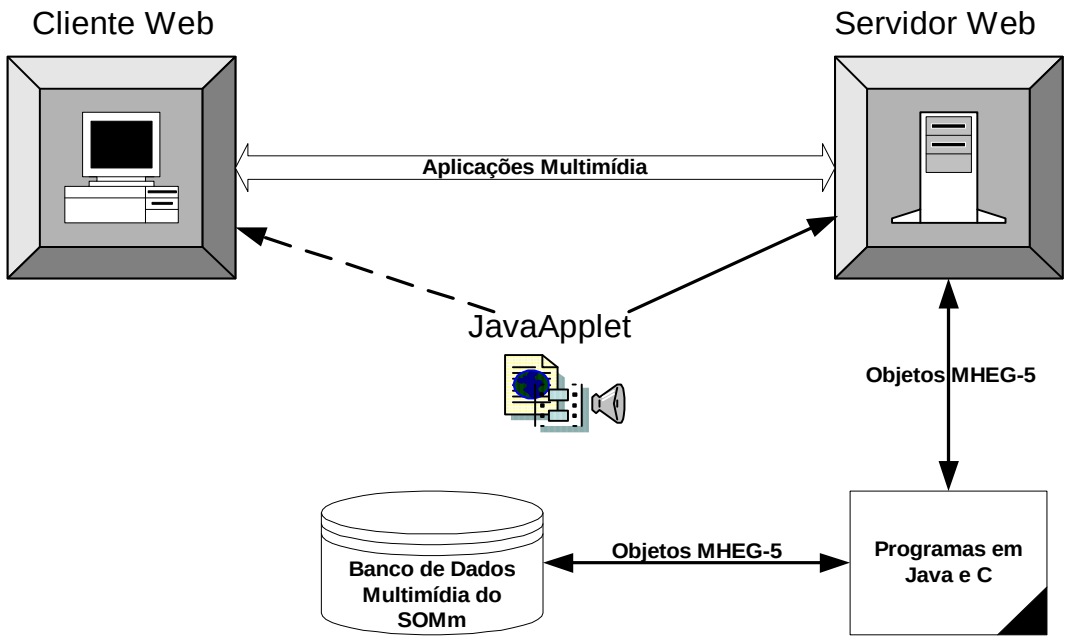


FIGURA 8.1 – ESQUEMATIZAÇÃO DO SISTEMA PARA CONVERSÃO ENTRE APLICAÇÕES MHEG-5 E DOCUMENTOS SMIL/HTML UTILIZANDO A WEB

Após recuperar os objetos MHEG-5 através dos programas em C, os objetos são instanciados em estruturas de dados em C que correspondem ao padrão MHEG-5, e depois são transferidos para os programas escritos em Java, para que possam ser enviados ao *JavaApplet* para realizar a conversão de aplicações. Também foi desenvolvida toda a estrutura de classes do MHEG-5 na linguagem Java, com o objetivo de receber e enviar os objetos MHEG-5 gerados ou fornecidos ao *JavaApplet*. A seguir apresenta-se a especificação da classe *Application,* do MHEG-5, em C, O_2C e Java como exemplo da integração entre as diversas estruturas de classes/dados escritas nas linguagens. Todas as demais classes do padrão MHEG-5 foram especificados dessa mesma forma.

Application.h (estrutura de dados da classe *Application* em C)

```
struct application  {
        struct Root RootApplication;
        struct Group GroupApplication;
        struct Action OnSpawnCloseDown;
        struct Action OnRestart;
        int LockCount;
        struct DefaultAttributesType DefaultAttributes ;
        struct VisibleType  *DisplayStack;
        struct Scene *Scenes;
        o2_Application object_id;
} ;
```

Application.o2 (estrutura da classe *Application* em O_2C)

```
class Application inherit Group public type
   tuple(OnSpawnCloseDown: Action,
         OnRestart: Action,
         LockCount: integer,
         DefaultAttributes:  tuple (  CharacterSet: integer,
                                      Font: tuple(FontName: string, FontReference: string),
                                      FontAttributes: tuple(FontSize: string,  FontStyle: string),
                                      BackgroundColour: string,
                                      TextColour: string,
                                      BitmapContentHook: integer,
                                      InterchangedProgramContentHook: integer,
                                      StreamContentHook: integer,
                                      TextContentHook: integer,
                                      LineArtContentHook: integer,
                                      ButtonColour: string,
                                      HighlightColour: string,
                                      SliderColour: string ) ,
         DisplayStack: list(Visible),
         Scenes: list(Scene))
end;
export schema class Application;
```

Application.java (estrutura da classe *Application* em Java)

```
public class Application extends Group
{
  Action              OnRestart;
  Action              OnSpawnCloseDown;
  int                 LockCount;
  DefaultAttributes   attributes;
  Scene[]             scenes;
  Visible[]           DisplayStack;
}
```

8.2 Módulo de Conversão

A seguir são apresentadas as telas da interface com o usuário para a conversão de uma aplicação multimídia. Na seção 8.21, Conversão de Aplicações MHEG-5 para documentos SMIL, e na 8.22, Conversão de documentos SMIL para Aplicações MHEG-5, são adotadas duas aplicações para ilustrar o processo.

8.2.1 Conversão de Aplicações MHEG-5 para documentos SMIL

A aplicação escolhida para ilustrar o processo de conversão MHEG-5 / SMIL, é composta de 04 cenas. Para o acompanhamento da conversão, será apresentada apenas a primeira cena, pois as demais seguem o mesmo comportamento. A primeira cena é

denominada Scene1 e é composta de 10 mídias, sendo 05 imagens, 03 textos e 02 botões. Para criar a aplicação MHEG-5 no banco de dados multimídia do SOMm, foi utilizada a ferramenta SMArT. A figura 8.2 mostra a tela da ferramenta SMArT que apresenta uma cena dessa aplicação. Conforme mostrado na figura 8.2, a aplicação chama-se "Pan – Americano", e relata características sobre a cidade de *Winnipeg*, onde foi realizado o Pan Americano de 1999.

FIGURA 8.2 – TELA DO SMART APRESENTANDO A SCENE1 DA APLICAÇÃO "PAN-AMERICANO"

Após a autoria dessa aplicação, os objetos MHEG-5 gerados para a Scene1 foram: Pan-Americano (*Application*), Scene1 (*Scene*), Image1 (*Bitmap*), Image2 (*Bitmap*), Image4 (*Bitmap*), Image5 (*Bitmap*), Image6 (*Bitmap*), Text1 (*Text*), Text2 (*Text*), Text3 (*Text*), Button1 (*Button*), Button2 (*Button*) e um objeto da classe *Link* utilizando a *Action :Transition To* . A figura 8.3 mostra a tela do O2Tools do SGBDOO O2 para ilustrar os objetos MHEG-5 gerados para a Scene1 da aplicação "Pan – Americano". Os objetos estão instanciados no atributo *Items* do objeto Scene1. O objeto *Action* é dado como: "Fonte: Button1 -> Efeito: Scene2", isso significa que a ação é disparada pelo objeto *Button1* que é o botão para Próxima cena, e quando for ativado, a Scene2 será mostrada.

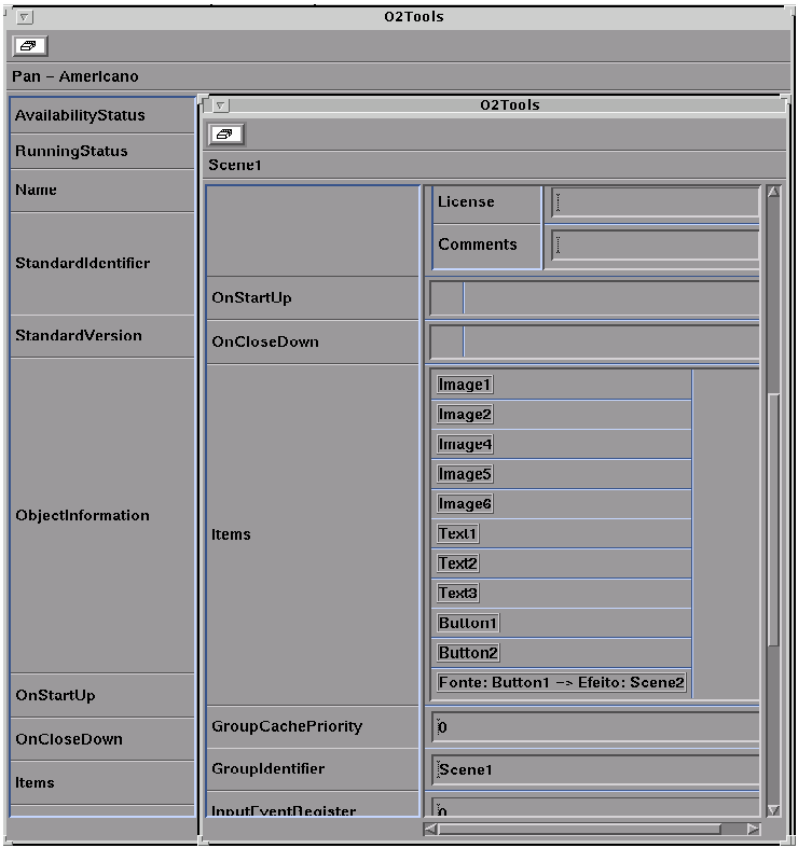


FIGURA 8.3 – TELA DO O2TOOLS CONTENDO OS OBJETOS MHEG-5 GERADOS PARA A SCENE1 DA APLICAÇÃO "PAN-AMERICANO"

A conversão dessa aplicação para documentos SMIL é realizada através de cinco passos, conforme citado anteriormente. A seguir são apresentadas as telas da interface com o usuário para a realização desses passos, aonde são incluídas informações/instruções aos usuários (nesta primeira versão, essas informações foram escritas em português, mas pretende-se convertê-las para instruções mais resumidas em inglês).

Antes de iniciar o processo de conversão, é necessário que o usuário se identifique. Para realizar essa tarefa, o usuário utiliza o módulo de autenticação de usuário, de acordo com as normas citadas no apêndice 2. Caso o usuário esteja autorizado, é apresentado então a tela inicial de conversão. A tela inicial de conversão, ilustrada na figura 8.4, apresenta os dois tipos de conversões disponíveis aos usuários. Em todos os passos serão mostrados os botões: *Back, Next, Cancel* e *Help*. O botão *Back* é ativado quando o usuário desejar voltar um passo, e caso queira avançar um passo deve clicar no botão *Next*. Para cancelar a conversão deve-se utilizar o botão *Cancel* e caso haja dúvidas sobre o passo da conversão, o usuário tem como ajuda *on-line* o botão *Help,* que apresenta um texto explicativo sobre o passo da conversão. Na tela inicial, o usuário, para realizar a conversão

de aplicações MHEG-5 para documentos SMIL, deve simplesmente selecionar a primeira opção. Após isso, deve clicar no botão *Next* para realizar o segundo passo.

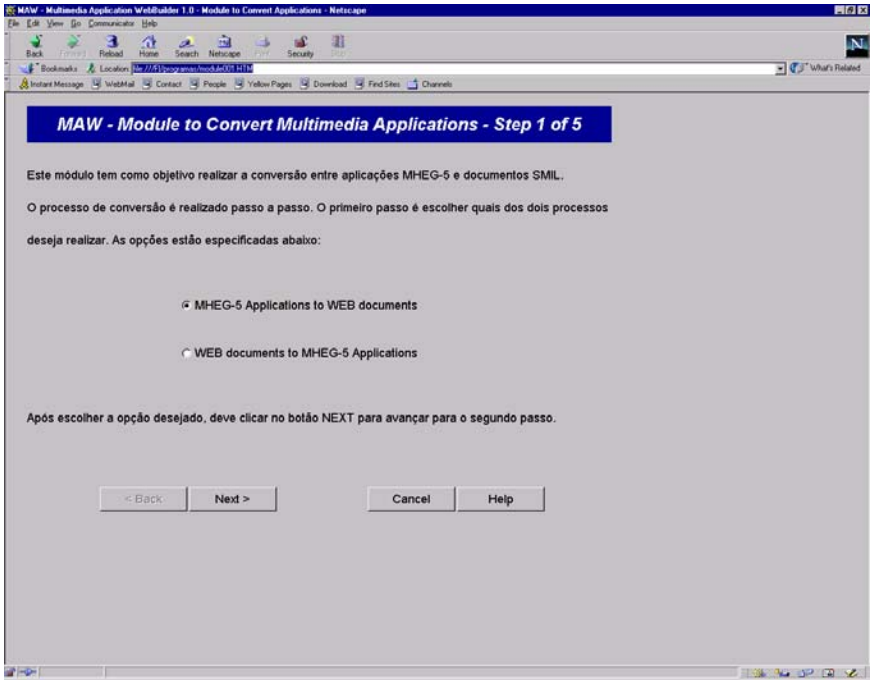


FIGURA 8.4 – TELA DO PASSO 1 DE CONVERSÃO DE APLICAÇÕES MHEG-5 PARA DOCUMENTOS SMIL

A figura 8.5 apresenta a interface do segundo passo de conversão MHEG-5 para SMIL. Nesse passo são disponibilizados os nomes de todas as aplicações MHEG-5 armazenadas no banco de dados multimídia do SOMm. O usuário então escolhe a aplicação de interesse. O botão *Back* possibilita voltar ao passo anterior, e o botão *Next* ativa o terceiro passo.

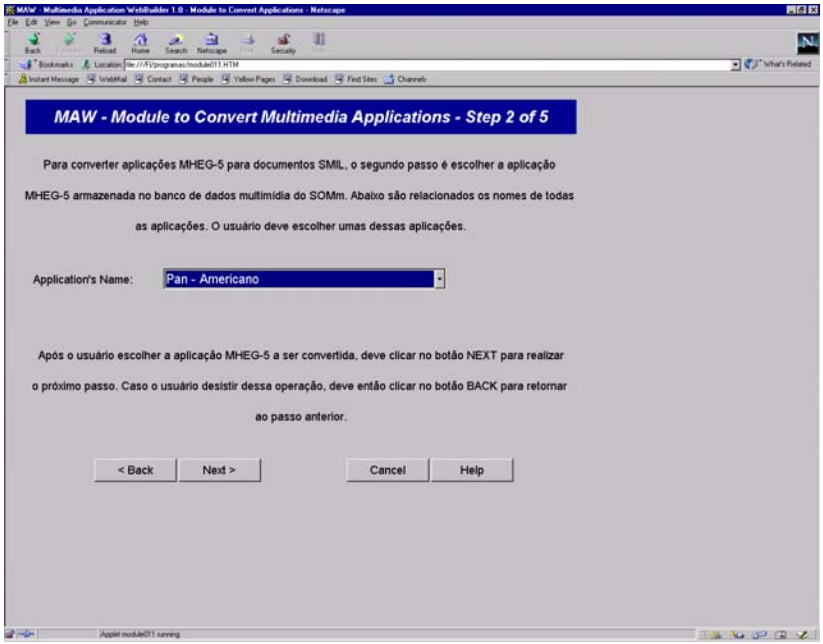


FIGURA 8.5 – TELA DO PASSO 2 DE CONVERSÃO DE APLICAÇÕES MHEG-5 PARA DOCUMENTOS SMIL

No terceiro passo são mostradas algumas informações sobre a aplicação MHEG-5 que foi escolhida no segundo passo da conversão MHEG-5 / SMIL. Essas informações são relevantes para o usuário identificar algumas características da aplicação. Os atributos que apresentam informações sobre as aplicações MHEG-5 são: nome da aplicação *(Application.Name*), quantidade de cenas que a aplicação contém e comentários relacionados à aplicação MHEG-5 (*Application.ObjectInformation.Comments*). Para o exemplo em questão, o terceiro passo do conversor apresenta uma interface conforme descrita na figura 8.6.

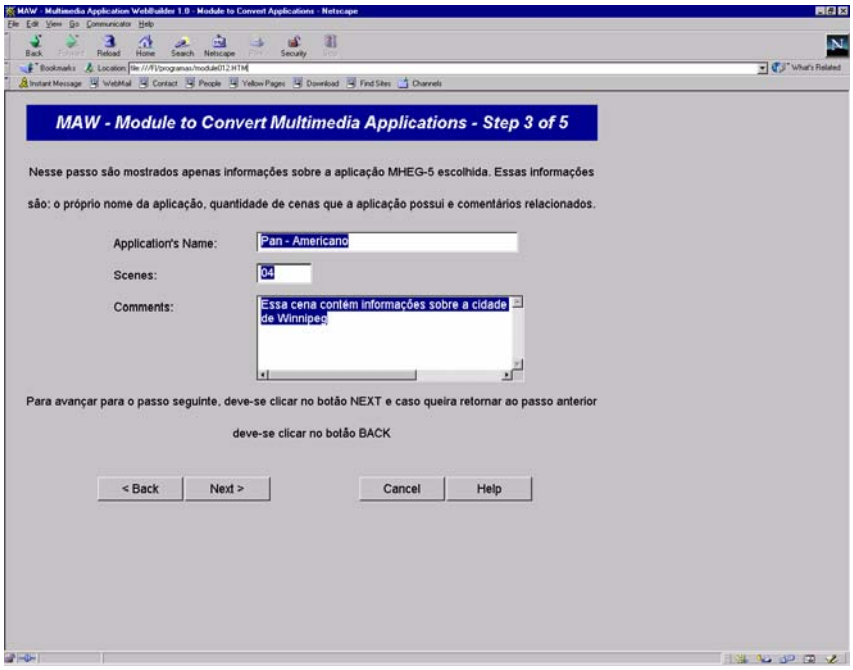


FIGURA 8.6 – TELA DO PASSO 3 DE CONVERSÃO DE APLICAÇÕES MHEG-5 PARA DOCUMENTOS SMIL

No quarto passo (figura 8.7) o usuário deverá escolher entre gerar apenas um ou vários documentos SMIL/HTML. Caso o usuário escolha a primeira opção, será gerado apenas um documento SMIL/HTML contendo todas as cenas da aplicação MHEG-5 escolhida. Caso contrário, o conversor gerará um documento SMIL/HTML para a aplicação e um para cada cena da aplicação MHEG-5. Após essa escolha, o usuário deve clicar no botão **TRANSLATE** para iniciar o processo de conversão da aplicação MHEG-5. Nesse exemplo, escolheu-se para gerar vários documentos SMIL/HTML. Observa-se que, na figura 8.7, é gerado, pelo conversor, um relatório contendo informações sobre o processo de conversão. Esse relatório serve para que o usuário faça um acompanhamento do processo de conversão.

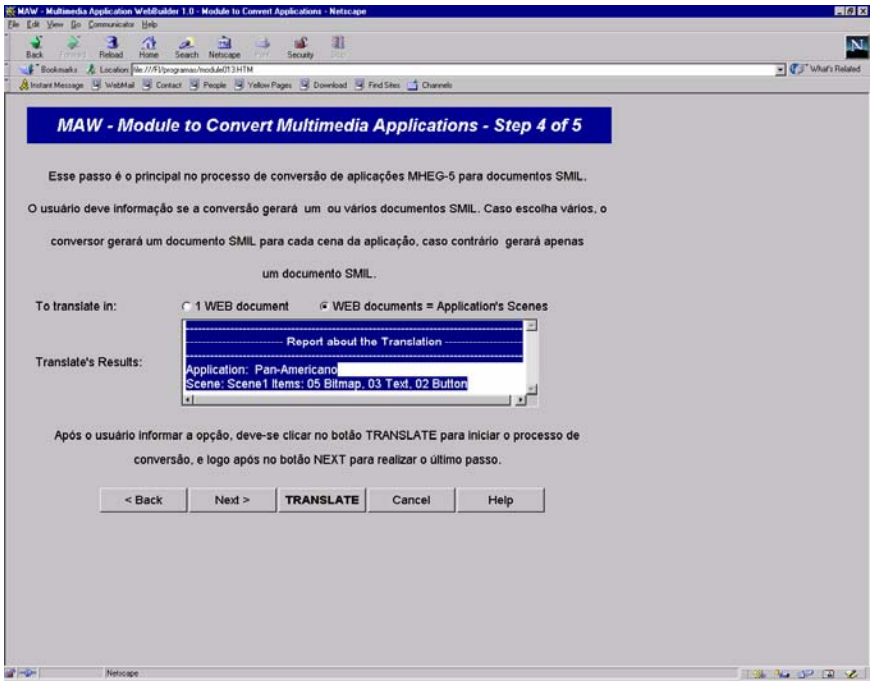


FIGURA 8.7 – TELA DO PASSO 4 DE CONVERSÃO DE APLICAÇÕES MHEG-5 PARA DOCUMENTOS SMIL

No último passo (quinto passo), é disponibilizada ao usuário a opção de armazenar no banco de dados multimídia do SOMm, os documentos SMIL gerados pelo conversor. A figura 8.8 mostra a opção para realizar essa tarefa. Caso o usuário escolha essa opção, o conversor armazenará os documentos SMIL na classe *File_SMIL_HTML* (essa classe foi definida no capítulo 7) do banco de dados multimídia do SOMm. Se o usuário não escolher essa opção, o conversor simplesmente desconsidera essa funcionalidade.

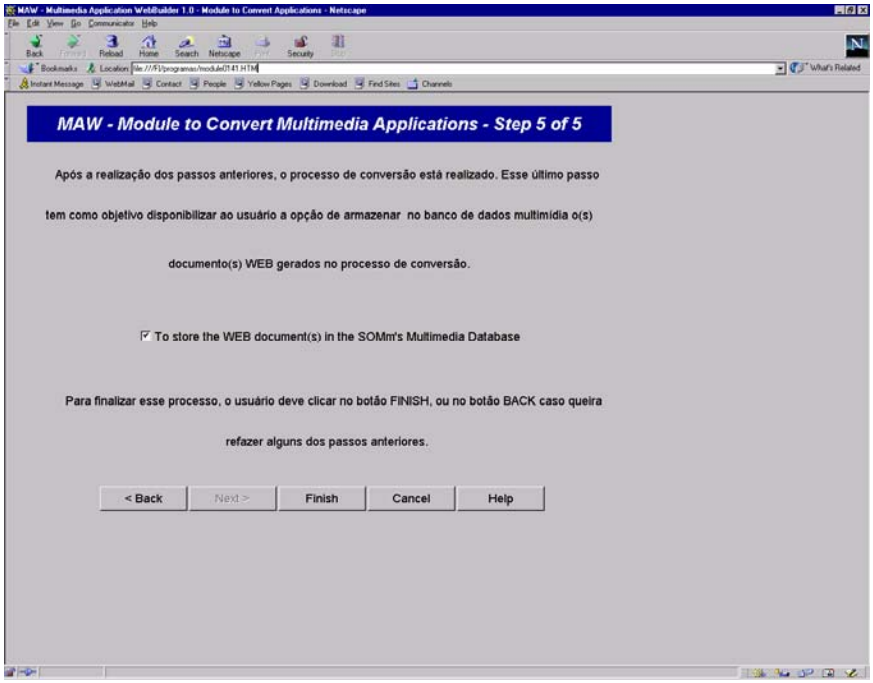


FIGURA 8.8 – TELA DO PASSO 5 DE CONVERSÃO DE APLICAÇÕES MHEG-5 PARA DOCUMENTOS SMIL

Após a execução desses passos, os documentos SMIL/HTML são gerados e podem ser apresentados em qualquer *parser* e/ou *browser* que tenham suporte SMIL/HTML. Veja abaixo o fonte SMIL do documento gerado na conversão. O documento "Pan-Americano.smi" representa a aplicação MHEG-5 convertida, e o documento "Scene1.smi" representa a primeira cena dessa aplicação.

Pan-Americano.smi

```
<smil>
<head>
<meta name= "title" content=  "Pan - Americano" />
</head>
<body>
<seq>
<anchor href= "Scene1.smi" />
<anchor href= "Scene2.smi" />
<anchor href= "Scene3.smi" />
<anchor href= "Scene4.smi" />
</seq>
</body>
</smil>
```

Scene1.smi

```
<smil>
<head>
<meta name= "title"  content= "Scene1" />
</head>
<body>
<seq>
  <img src="Image1.bmp">
  <img src="Image2.bmp">
  <img src="Image4.bmp">
  <text src="Text1.rt"> </text>
  <text src="Text2.rt"> </text>
  <text src="Text3.rt"> </text>
  <img src="Image5.bmp">
  <a href= "Scene2.smi">Proxima</a>
  <a>Anterior</a>
  <img src="Image6.bmp">
</seq>
</body>
</smil>
```

Note-se que, dependendo de suas características, a aplicação poderá estar apenas parcialmente representada em SMIL ou HTML (principalmente nesta última linguagem), conforme tratado na seção 7.4.

8.2.2 Conversão de documentos SMIL para Aplicações MHEG-5

O documento SMIL chamado “Exemplo1.smi”, mostrado a seguir, será convertido em uma aplicação MHEG-5. Esse documento é composto de duas cenas, sendo que a primeira é representa no documento SMIL chamado “Scene1.smi”. O documento “Exemplo1.smi” possui uma temporização entre as cenas, isto é, a segunda cena só será apresenta após 10 segundos, sendo que durante esses 10 segundos, a primeira cena será apresentada.

Os documentos relatam fatos e características sobre as Olimpíadas do ano 2000. Para desenvolver um documento SMIL é necessário apenas um editor de texto comum. Atualmente existem vários *parsers* que interpretam documentos SMIL. Abaixo é mostrado o código fonte SMIL do arquivo “Exemplo1.smi” e “Scene1.smi”.

Exemplo01.smi

```
<smil>
<head>
<meta name= “title” content=  “Sydney 2000” />
</head>
<body>
<seq>
<anchor href= “http://www.dc.ufscar.br/applications/Scene1.smi” begin= “0s” end= “10s” />
<anchor href= “http://www.dc.ufscar.br/applications/Scene2.smi” begin= “10s” />
</seq>
</body>
</smil>
```

Scene1.smi

```
<smil>
<head>
<meta name= “title” content=  “Scene1” />
</head>
<body>
<seq>
<img src="Image1.bmp">
<text src="Text1.rt"> </text>
<img src="Image2.bmp">
<img src="Image3.bmp">
<text src="Text1.rt"> </text>
</seq>
</body>
</smil>
```

Para iniciarmos o processo de conversão, novamente o primeiro passo de conversão é ativado, apresentando as duas opções de conversão. Nesse passo 1, deve-se escolher a

segunda opção, que é a conversão de documentos *Web* para aplicações MHEG-5, conforme mostrado na figura 8.9.

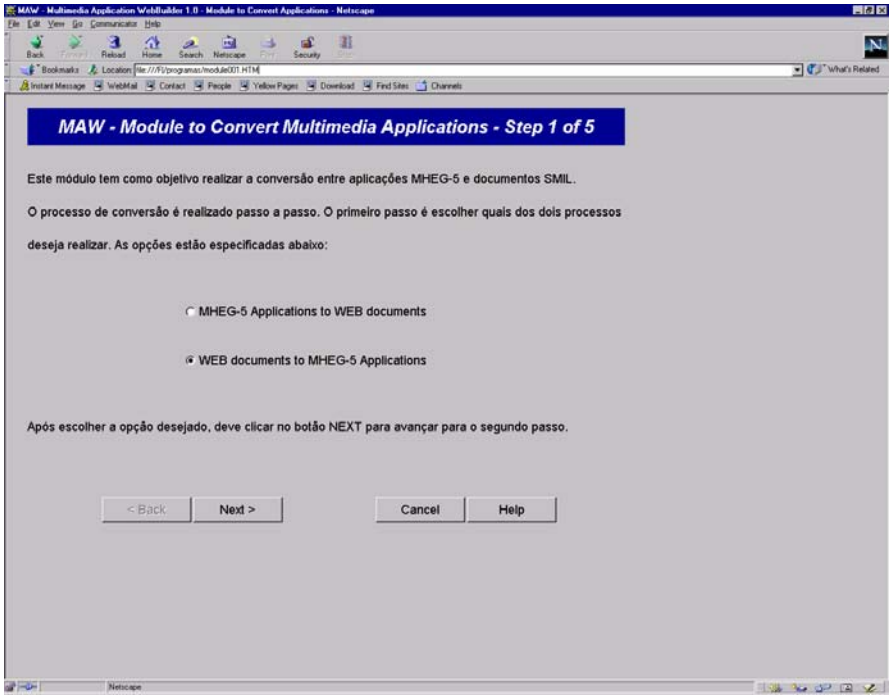


FIGURA 8.9 – TELA DO PASSO 1 DE CONVERSÃO DE DOCUMENTOS WEB PARA APLICAÇÕES MHEG-5

O usuário agora deve informar qual ou quais são os arquivos SMIL, ou qual o endereço que se deve buscar (caso esteja na *Web,* deve-se utilizar a URL) o conteúdo de cada um deles. Pode-se também utilizar as duas opções, isto é, documentos SMIL armazenados em arquivos e documentos localizados em URLs.

A figura 8.10 ilustra a interface do segundo passo, com o exemplo instanciado. Observe que existe um botão *Add,* que é usado toda vez que o usuário tiver mais de um documento a ser convertido.

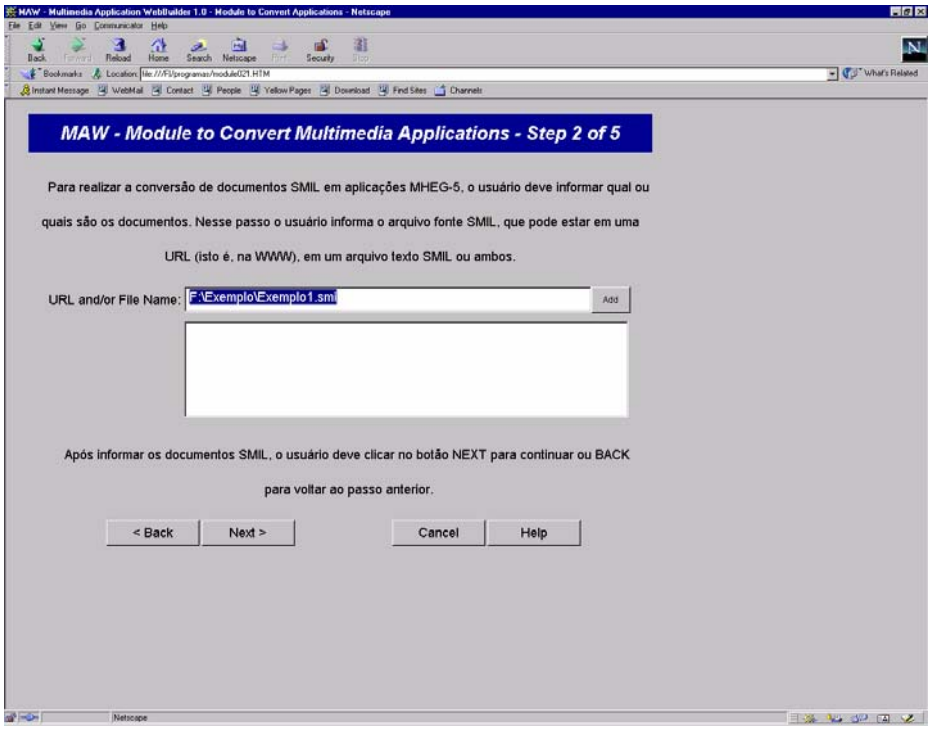


FIGURA 8.10 – TELA DO PASSO 2 DE CONVERSÃO DE DOCUMENTOS WEB PARA APLICAÇÕES MHEG-5

No terceiro passo são mostrados os conteúdos dos arquivos SMIL, conforme mostrado na figura 8.11; essa figura possui os conteúdos SMIL dos arquivos “Exemplo01.smil” e “Scene1.smi”. Após essa visualização, o usuário deve prosseguir clicando no botão *Next*. Caso o usuário tenha especificado vários arquivos e/ou URLs SMIL, o conversor os separa através de uma linha tracejada. Então, para visualizar o conteúdo de cada arquivo/URL SMIL é necessário seguir a ordem em que foram especificados os arquivos/URLs SMIL para a conversão, isto é, seguir a ordem presente na lista de aplicações SMIL a serem convertidas.

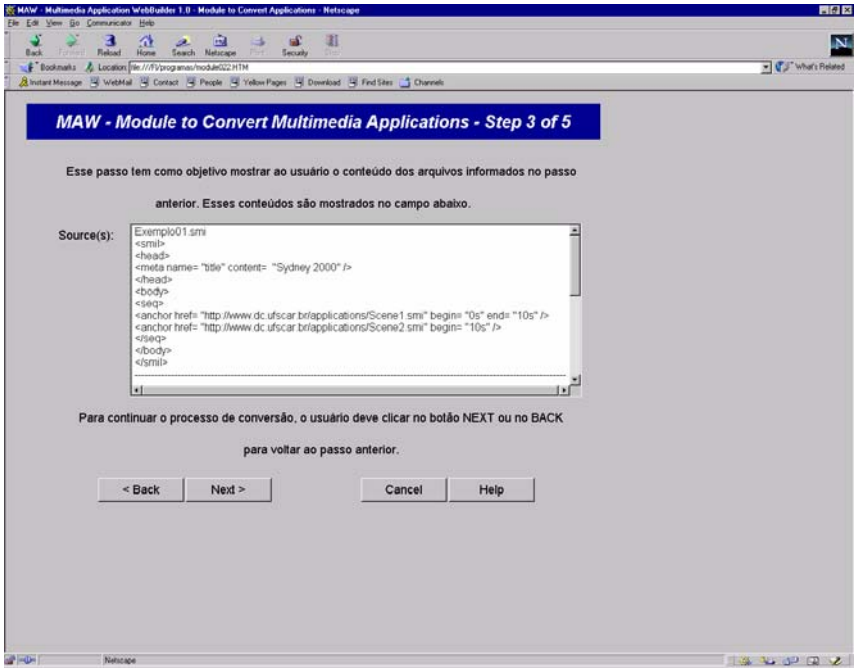


FIGURA 8.11 – TELA DO PASSO 3 DE CONVERSÃO DE DOCUMENTOS WEB PARA APLICAÇÕES MHEG-5

No quarto passo de conversão, o usuário deverá escolher entre gerar uma ou várias aplicações MHEG-5. Caso o usuário escolha a primeira opção, será gerado apenas uma aplicação MHEG-5 para os documentos SMIL escolhidos, criando-se uma aplicação principal e os arquivos sendo considerados como cenas dessa aplicação. Caso contrário, o conversor gerará várias aplicações MHEG-5 sendo uma para cada documento SMIL. Após essa escolha, o usuário deve clicar no botão **TRANSLATE** para iniciar o processo de conversão do documento SMIL. Nesse exemplo, escolheu-se gerar apenas uma aplicação MHEG-5, contendo o documento SMIL “Scene1.smi” como cena principal e o “Scene2.smi” como a segunda cena da aplicação. A figura 8.12 mostra o exemplo (“Sydney 2000”) do arquivo “Exemplo01.smil” sendo convertido para uma aplicação MHEG-5.

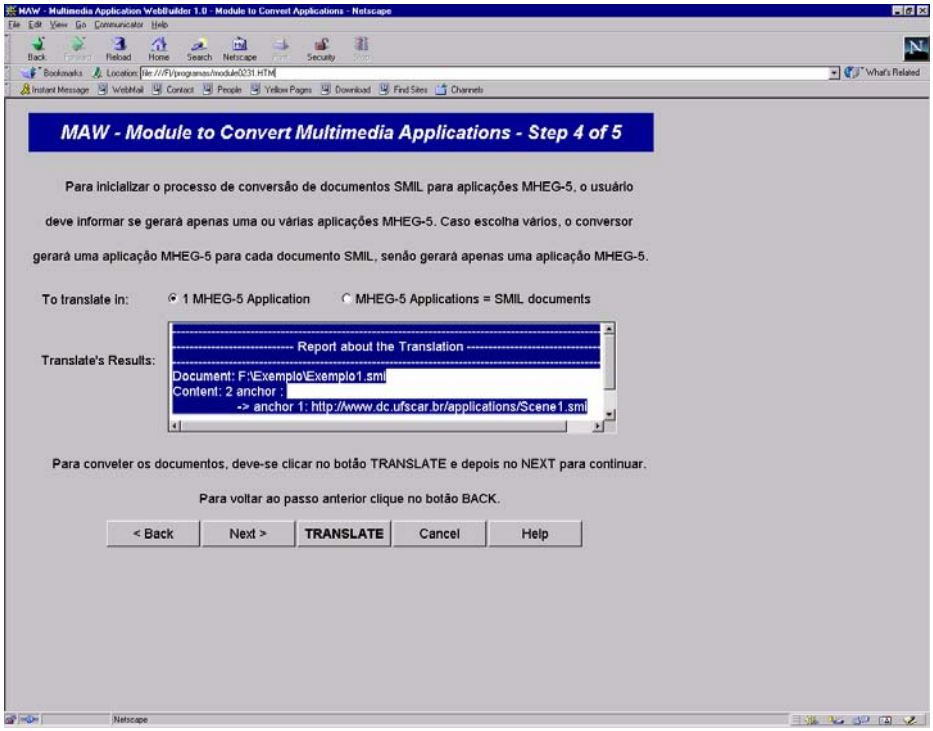


FIGURA 8.12 – TELA DO PASSO 4 DE CONVERSÃO DE DOCUMENTOS WEB PARA APLICAÇÕES MHEG-5

O passo 5 do processo de conversão é simplesmente uma interface mostrando que os passos foram executados com sucesso, e que para finalizar o processo de conversão deve-se clicar no botão *Finish*. A figura 8.13 mostra o quinto passo da conversão. Quando o usuário clica no botão *Finish*, automaticamente a aplicação MHEG-5 é gerada pela conversão, sendo armazenada no banco de dados multimídia do SOMm. Caso o usuário queira refazer algum dos passos, isso é realizado através do botão *Back*.

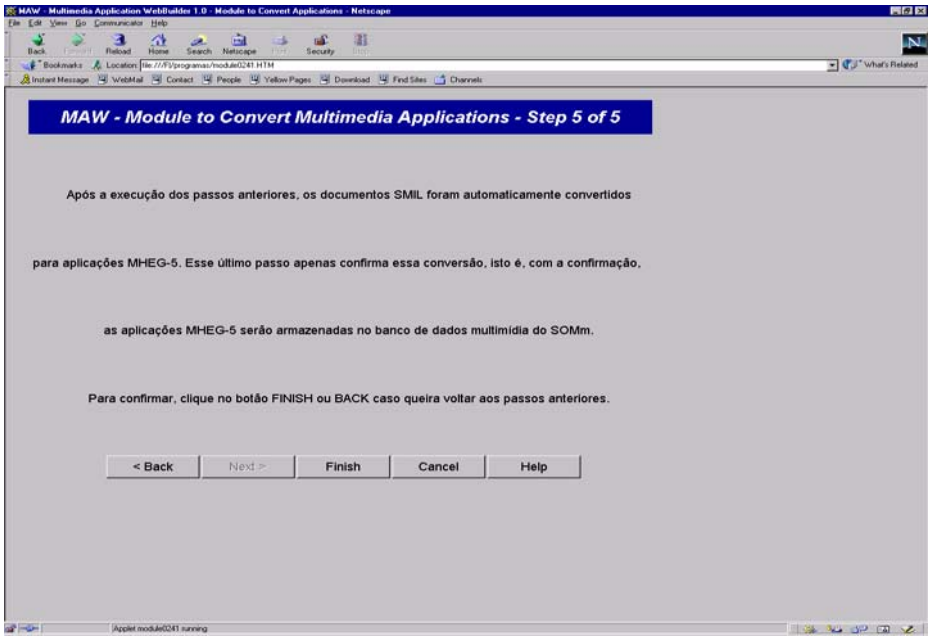


FIGURA 8.13 – TELA DO PASSO 5 DE CONVERSÃO DE DOCUMENTOS WEB PARA APLICAÇÕES MHEG-5

Após a conversão do documento SMIL (“Sydney 2000”), o mesmo pode ser visualizado através da ferramenta SMArT. A figura 8.14 mostra a aplicação MHEG-5 gerada pela conversão.

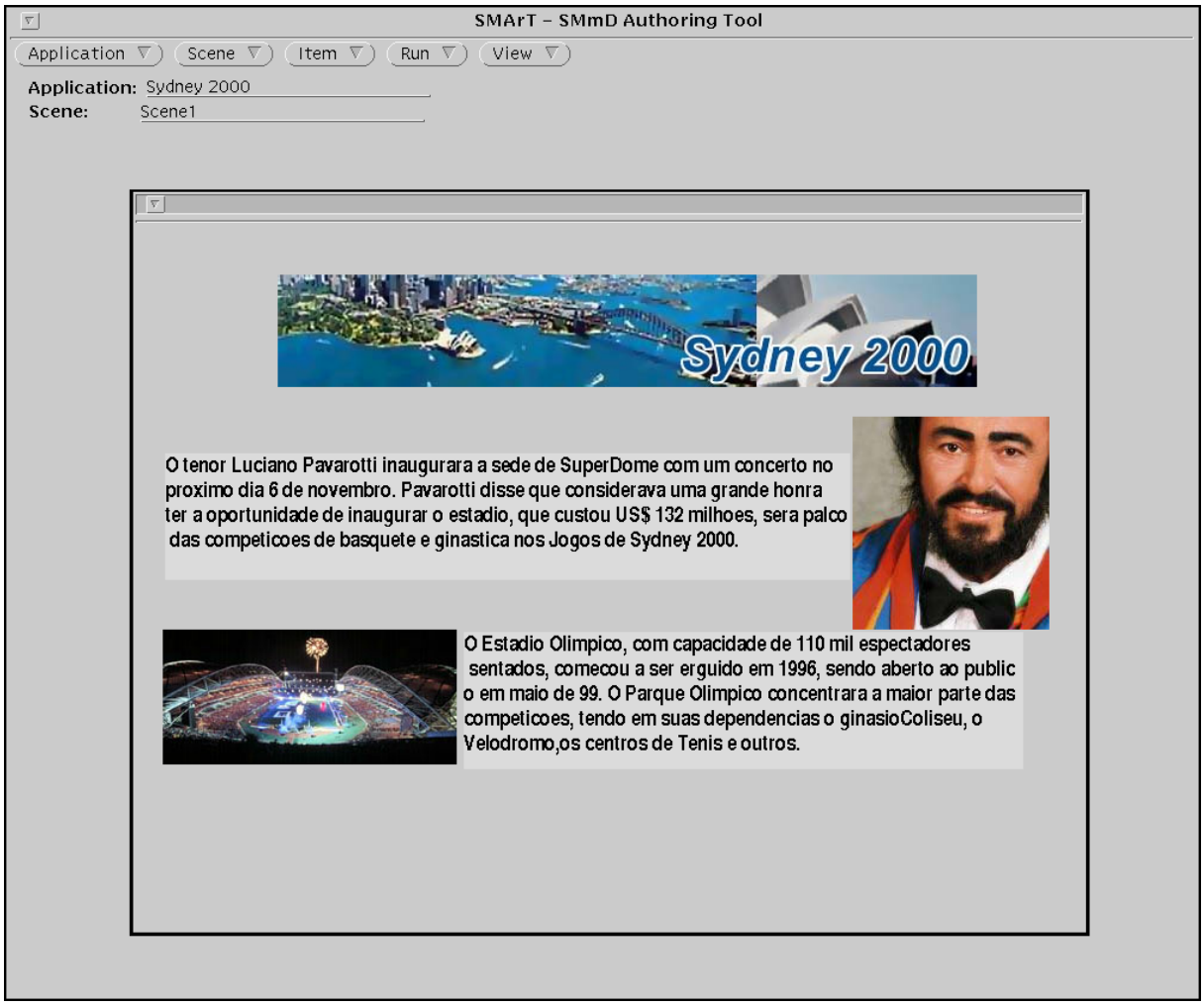


FIGURA 8.14 – TELA DO SMART APRESENTANDO A SCENE1 DA APLICAÇÃO “SYDNEY 2000”

Essa aplicação MHEG-5 está armazenada no banco de dados multimídia do SOMm. Os objetos MHEG-5 gerados foram: Sydney 2000 (*Application*), Scene1 (*Scene*) e Scene2 (*Scene*). Para a Scene1 foram gerados os seguintes objetos: Image1 (*Bitmap*), Image2 (*Bitmap*), Image3 (*Bitmap*), Text1 (*Text*), Text2 (*Text*) e um objeto *Link* com uma *Action :Transition To*. A figura 8.15 mostra, através do O2Tools, os objetos MHEG-5 instanciados pela conversão da aplicação “Sydney 2000”. Observe que foi criada uma instância *SetTimer* no atributo *OnStartUp* da Scene1, com o valor 10.000, que corresponde 10 segundos. Com isso, a primeira cena (Scene1) será apresentada, e após 10 segundos é apresentada a próxima cena.

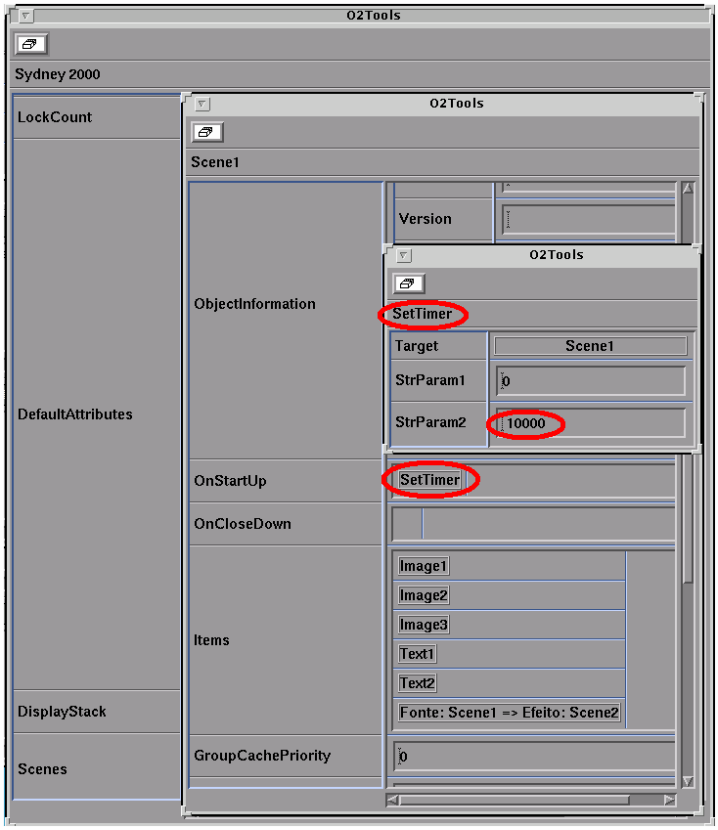


FIGURA 8.15 – TELA DE CONSULTA A OBJETOS MHEG-5 GERADOS PARA A APLICAÇÃO "SYDNEY 2000"

8.3 Trabalhos Relacionados

Algumas propostas surgiram com o intuito de permitir apresentação de aplicações multimídia na Web. Um trabalho relacionado é o sistema Madeus [Jourdan98]. O Madeus é uma ferramenta de autoria e apresentação de documentos multimídia. Possui suporte a documentos SMIL para o ambiente de autoria e apresentação. É composto de um parser SMIL e de um conversor. Esse conversor tem como finalidade transformar as especificações da linguagem SMIL para o modelo Madeus. O modelo Madeus oferece composições para a estruturação lógica do documento e relações temporais e espaciais que podem ser definidas entre os componentes. Entretanto, esse trabalho apresenta limitações, tais como a de não permitir a especificação de mudanças de comportamentos dos objetos envolvidos, durante a sua apresentação.

Para realizar conversão entre documentos Web e aplicações MHEG, tem-se conhecimento de dois projetos: o DAVIC [Davic98] e o Dotrans [Dotrans99]. Em [Davic98] é especificado um parser para realizar as conversões de páginas HTML em aplicações MHEG-5, cuja idéia principal é disponibilizar essas aplicações geradas em terminais. Esses

terminais são compostos de um *engine* MHEG-5 que interpreta os objetos contidos nas cenas MHEG-5. O parser é composto de 3 passos principais: realiza a busca do documento HTML e filtra os elementos HTML (verifica todos os elementos HTML); busca as mídias completando o documento e realiza uma nova filtragem (dessa vez considerando as mídias); e transforma os elementos HTML em objetos de dados, onde esses objetos são convertidos a objetos MHEG-5. O Dotrans [Dotrans99] é um trabalho cuja finalidade principal é a conversão de documentos Web em aplicações MHEG-5 textual. Essa ferramenta é composta de uma interface gráfica e dos mecanismos: de filtragem, pré-processadores de conversão e de exclusão de *tags* que não são suportadas pelo MHEG-5.

Uma das dificuldades encontradas nesse processo de conversão de páginas HTML em aplicações MHEG-5 nos trabalhos citados, é que o tamanho de uma página HTML é ilimitado, enquanto que uma cena de uma aplicação MHEG-5 deve ser baseado no tamanho de uma tela de TV. Para solucionar esse problema, os trabalhos adotaram o critério de dividir o documento HTML em várias cenas MHEG-5. Uma outra desvantagem encontrada nesse processo de conversão, é que nem todos os formatos de dados contidos nas páginas Web podem ser convertidos para MHEG-5, sendo eles *JavaApplet,* controles *ActiveX*, *JavaScript* e outros. Portanto, aqueles que não podem ser convertidos também não podem ser visualizados nativamente por um *engine* ou terminal MHEG-5.

O processo de conversão desenvolvido neste projeto possui algumas vantagens em relação aos projetos relacionados. Uma delas, é que o conversor não sofre mudanças em seu código fonte para adaptar-se a novas regras de conversão. Isso é evitado através do uso de uma base de regras, que apenas adiciona novas regras de conversão, sem que o conversor sofra alterações. Os problemas anteriormente citados pelos projetos relacionados, também foram confrontados neste projeto, pois é muito dificil realizar a conversão de funcionalidades embutidas em páginas HTML (*JavaScript*, *JavaApplet* e *ActiveX*) para MHEG-5.

8.4 Considerações Finais

Neste capítulo foram especificados todos os critérios de implementação do trabalho. Foi apresentada a esquematização do sistema para realizar as conversões, isto é, estrutura de informações e do ambiente para suporte à conversões. O capítulo também apresentou as interfaces utilizadas para os módulos de: conversão de aplicações MHEG-5 para documentos SMIL, conversão de documentos SMIL para aplicações MHEG-5. Atualmente o ambiente do MAW, tem os módulos de conversão implementados, juntamente com o

módulo de conexão e autenticação de usuários, que se encontram nos apêndices A e B respectivamente.

CAPÍTULO 9

Conclusões

Neste capítulo são destacadas as considerações finais sobre o desenvolvimento do trabalho. São destacadas também algumas sugestões para trabalhos futuros que envolvem o melhoramento e aperfeiçoamento dos conversores e do ambiente MAW.

9.1 Considerações Finais

Este trabalho apresentou um ambiente de conversões entre os padrões SMIL e MHEG-5. Como auxílio ao processo de conversão foi desenvolvido uma base de regras de conversão, que tem como objetivo armazenar as regras de conversão, possibilitando que haja dinamismo. Ainda como partes desse trabalho foram desenvolvidas um módulo de conexão entre a *Web* e o SGBDOO O2, e um módulo de autenticação de usuários. Como contribuição principal deste trabalho, temos o desenvolvimento dos conversores, cuja técnica utiliza uma base de regras, oferece dinamismo na manutenção das regras existentes, bem como a criação de novas regras. Com as conversões, os usuários podem utilizar aplicações MHEG-5 no ambiente *Web*, e vice-versa, converter as diversar aplicações multimídia disponíveis na *Web* para aplicações MHEG-5 (considerando as limitações já citadas anteriormente na seção 7.4). Uma outra contribuição foi a proposta do ambiente MAW, e a implementação de alguns módulos de suporte, o que propiciou a proposta de novos trabalhos, alguns já em desenvolvimento.

9.2 Sugestões para Trabalhos Futuros

Como continuação deste trabalho (implementação dos módulos especificados no MAW) pode-se citar os seguintes trabalhos:

- Implementação do módulo de Geração de MHEG-5 Textual, sendo que com esse módulo o MAW poderá fornecer aplicações MHEG-5 textual para qualquer MHEG-5 *engine.* Com essa facilidade pode-se também gerar documentos encontrados na *Web*, como páginas em SMIL, em aplicações MHEG-5 textual, utilizando-se os conversores, isto é, transforma os documentos SMIL em aplicações MHEG-5 e depois em MHEG-5 textual.

- Implementação do módulo de Educação à Distância, sendo este módulo especificado para o desenvolvimento de aplicações multimídias voltadas à educação. Além dos critérios de armazenamento, manipulação e recuperação de aplicações implementados no SOMm, este módulo deve possuir todo o tratamento relacionado ao ensino. Essas aplicações poderão ser compostas de métodos especificos, como: controlar ausência de alunos, publicação de aulas, notas e exercícios de aulas, e outros. O conversor interagirá com esse módulo fornecendo e recebendo aplicações multimídia (MHEG-5 ou SMIL).

- Implementação do módulo do *Browser SMIL*, que possuirá todas as características de um *browser Web*, adicionando-se funcionalidades pertencentes ao padrão SMIL. Um dos recursos a ser oferecido pelo *Browser SMIL* é a apresentação de mídias dependentes e independentes de tempo. Incorporação de um *parser* SMIL, para que os usuários possam também desenvolver suas próprias aplicações SMIL.

- Instanciação dos elementos do HTML+TIME [W3C98] na base de regras de conversão. Com isso os conversores poderão gerar aplicações HTML+TIME em aplicações MHEG-5 ou SMIL. Para isso, deve realizar um estudo aprofundado dessa nova linguagem.

- Implementação dos conversores utilizando-se recursos encontrados em *Servlet*, isto é, realizar todo o processamento de conversão no servidor *Web.*

- Implementação do módulo de conexão, abrangendo-se todas as conexões com os SGBDOOs citados anteriormente, como por exemplo: Jasmine, Poet, e outros.

- Realização de análises de desempenho das conversões, com intuito de obter um melhoramento contínuo para os conversores.

Referências Bibliográficas

[Arnold97] Arnold,K;Gosling J.; **Programando em Java ;** *A série Java da Source*; Makron Books, 1997.

[Berners-Lee94] Berners-Lee,T. et al **The World Wide Web**, *ACM Communications,* 37(8): 76-82, 1994.

[Berners-Lee96] Berners-Lee, T. **The World Wide Web: Past, Present and Future**, (URL:http://www.w3.org/People/Berners-Lee/1996/ppf.html), August, 1996.

[Bray98] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M. **Extensible Markup Language (XML) 1.0**, *W3C Recommendation,* (URL:http://www.w3.org/TR/1998/REC-xml-19980210), February – 10, 1998.

[Buchanan93] Buchanan, M. C.; Zellweger, P. T. **Automatically Generating Consistent Schedules for Multimedia Documents**, *Multimedia Systems*, Springer-Verlag, April, 1993.

[CGI95] **"The Common Gateway Interface",** *University of Illinois,* Urbana-Champaign, (URL: http://hoohoo.ncsa.uiuc.edu/cgi/intro.html), 1995.

[Cornell98] Cornell, G.; Horstmann, C. S. **Core Java**, *MAKRON Books do Brasil Editora Ltda*, Editora McGraw-Hill Ltda, 1998.

[Crawford97] Crawford, W.; **Tutorial: Developing Java Servlets**; (URL:http://webreview.com/wr/pub/97/10/10/feature/main.html); October 1997.

[Damski96] Damski, J. C. B.; Valente, A. S. M. **Internet - Guia do Usuário Brasileiro**, *Makron Books do Brasil Editora Ltda*, Editora McGraw-Hill Ltda, 1996.

[Davic98] **DAVIC WWW Gateway v1.1.2 (1998-04) Bridiging the gap between the Internet and DAVIC**, (URL:http://www.infonova.at), 1998.

[December95] December, J.; Ginsburg, M.; **HTML and CGI Unleashed**; Sams.net Publishing;1995.

[Denny98] Denny,R.B.; **SSI, CGI, API or SSS? Choosing the Right Tools for the Job**; (URL:http://solo.dc3.com/white/extending.html); November 1998;

[Doorn94] Doorn, M.; Eliëns, A. **Integrating applications and the World Wide Web**, *Proceedings of the First International World Wide Web Conference*, Geneva, Switzerland/France, May 25-27, 1994.

[Dotrans99] **Dotrans – HTML to MHEG-5 translator for Digital TV**, (URL:http://www/cabot.co.uk/dotrans/index.html), 1999.

[Effelsberg95] Effelsberg, W.; Meyer-Boudnik, T. **MHEG explained**, *IEEE MultiMedia,* 2(1): 26-38, Spring, 1995.

[Ehley95] Ehley, L.; Ilyas, M., Furht, B.; **A Survey of Multimedia Synchronization Techniques;** *A Guided Tour of Multimedia Systems and Applications*; Ed:Borko Furht, Milan Milenkovic, IEEE Computer Society Press, 1995.

[Ehmayer97] Ehmayer, G.; Kappel,G.; Reich , S.;**Connecting Databases to the Web: A Taxonomy of Gateways;***Lecture Notes in Computer Science*;Database and Expert Systems Applications DEXA 97, Springer.

[Fornazari99] Fornazari, F.; Vieira, M. T. P.; Biajiz, M. **Busca nebulosa baseada em conteúdo em uma base de dados de aplicações MHEG-5,** *Anais do V Simpósio Brasileiro de Sistemas Multimídia e Hipermídia,* Goânia – BR, 15 a 17 Junho, 1999.

[Härder97] Härder, T.; Loeser, H.; Zhang, N.; **Supporting Adaptable Technical Information Systems in Heterogeneous Environments - Using WWW and ORDBMS -,** *Eighth International Workshop on Database and Expert Systems Applications, Proceedings DEXA/97,* Toulouse, France, September 1 - 2, 1997.

[Harrison95] Harrison, M. A. **The essential elements of hypermedia** *in MultiMedia Systems and Applications*, Academic Press Ltda, 1995.

[Herman96] Herman, I.; Reynolds, G. J.; Van Loo, J. **PREMO: An emerging standard for multimedia presentation**, *IEEE MultiMedia*, 3(3-4), Fall-Autumn, 1996.

[Internet2] Internet 2, (URL:http://www.rnp.br/i2/i2.html), 1999.

[ISO94a] ISO/IEC 13522-1, **Object Representation, Base Notation (*ASN.1*),** MHEG-1.

[ISO94b] ISO/IEC 13522-4, **Registration Procedures;** MHEG-4.

[ISO94c] ISO/IEC 13522-3, **Script Interchange Representation;** MHEG-3.

[ISO96a] ISO/IEC 13522-5, **Information Technology Coding of Multimedia and Hypermedia Information, Part 5: Support for Base-LevelInteractive Applications**, **MHEG-5 IS Document Pre-release 5,** 1996.

[ISO96b] ISO/IEC 13522-6, **Information Technology Coding of Multimedia and Hypermedia Information, Part 6: Support for Enhanced Interactive Applications**, September, 1996.

[ISO97] ISO/IEC 13522-2, **Information Technology - Codified Representation of Multimedia and Hypermedia Information,** MHEG-2, 1997.

[Java97] **Java and the Java Virtual Machine**; http://www.Java.sun.com;1997.

[Jourdan98] Jourdan, M; Layaïda, N; Roisin, C; Sabry-Ismail, L; Tardiff, L; **Madeus an Authoring Environment for Interactive Multimedia Documents**, *Proceedings of the ACM Multimedia Conference 98*, pp. 267-272, ACM, Bristol, Inglaterra, Setembro, 1998.

[Kim95] Kim, M. Y.; Song, J. **Multimedia Documents with Elastic Time**, *Proceedings of the ACM Multimedia Conference 95*, São Francisco, November, 1995.

[Kim96] Kim, P. **A Taxonomy on the Architecture of Database Gateways for the Web**, September, 1996.

[Klas97] Klas,W.;Aberer,K.;**Multimedia and its Impacts on Databases Systems Architectures**;*Multimedia Databases in Perspective*; Springer;1997.

[Light99] Light, R. **Iniciando em XML***, MAKRON Books do Brasil Editora Ltda,* 1999.

[Liu98] Liu, P. **An Introduction to the Synchronized Multimedia Integration Language**, *IEEE Multimedia* , pp. 84 – 88, October - December, 1998.

[Mohseni96] Mohseni, P. **Web Database – Primer Plus**, *Waite Group Press*, 1996.

[Nabeshima97] Nabeshima, M. **The Japan Cache Project: An Experiment on DomainCache**, *Sixth Internacional World Wide Web Conference,*Santa Clara, California – USA, 7 –11 April, 1997.

[Newcomb91] Newcomb, S. R.; Kipp, N. A.; Newcomb, V. T. **The "HyTime": hypermedia/time-based document structuring language**, *Communications of the ACM,* 34(11): 67-83, November, 1991.

[Nguyen96] Nguyen, T.; Srinivasan, V.; **Acessing Relational Databases from the World Wide Web,** *Proc. ACM SIGMOD Int' I Conf. on Management of Data, ACM Press,* New York, June, 1996, pp. 529 - 540.

[Nielsen90] Nielsen, J. **Hypertext and Hypermedia**, Academic Press INC, 1990.

[$O_2$96] O2 Technology, **O_2Web User Manual,** release 4.6, January, 1996.

[Pazandak97] Pazandak, P.; Srivastava, J. **Evaluating Object DBMSs for Multimedia**, *IEEE MultiMedia,* July-September, 1997.

[Ramalho97] Ramalho, J. A. A. **HTML Avançado**, *MAKRON Books do Brasil Editora Ltda*, Editora McGraw-Hill Ltda, 1997.

[Ribeiro99] Ribeiro, L. C. M.; Vieira, M. T. P. **Busca baseada em conteúdo em documentos de Fala/Texto em um Banco de dados MHEG-5,** *Anais do V Simpósio Brasileiro de Sistemas Multimídia e Hipermídia,* Goânia – BR, 15 a 17 Junho, 1999.

[Rodrigues99] Rodrigues, L. M.; Rodrigues, R. F.; Muchaluat-Saade, D.; Soares, L. F. G. **Acrescendo Facilidades NCM a Documentos SMIL**, *Anais do V Simpósio Brasileiro de Sistemas Multimídia e Hipermídia,* Goiânia, 15 – 17 Junho, 1999.

[Rousseau98] Rousseau, F.; Duda, A. **Synchronized multimedia for the WWW**, *Seventh International World Wide Web Conference,* Brisbane – Australia, April 14-18, 1998. Também publicado em: *Computer networks and ISDN Systems*, pp. 417 – 429, 30(1998).

[Rumbaugh91] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Edy, F.; Lorensen, W. **Object-Oriented Modeling and Design**, Prentice Hall, 1991.

[Sall98] Sall, K. **XML: Structuring Data for the Web: An Introduction**, (URL: http://www.stars.com/Authoring/Languages/XML/Intro/), 3 – March, 1998.

[SantosF97a] Santos, F.C.; Vieira, M.T.P. **Uma Ferramenta de Autoria para Aplicações MHEG-5**. *Anais do III Workshop em Sistemas Multimídia e Hipermídia,* São Carlos - SP, Maio 1997.

[SantosF97b] Santos, F. C.; **SMArT - Uma Ferramenta de Autoria para Aplicações MHEG-5,** *Dissertação de Mestrado, PPG-CC, Departamento de Computação / USFCar,* Setembro, 1997.

[SantosF98] Santos, F. C.; Vieira, M. T. P. **SMArT: Ambiente para autoria de aplicações MHEG-5,** *Anais do IV Simpósio de Hipermídia e Multimídia*, *SBMídia98,* Maio, 1998.

[SantosM97a] Santos, M.T.P.; Santos, F.C.; Vieira, M.T.P. **A MHEG-5 Objects Server for Multimedia Applications**. *Proceedings of the ICAST 97 & ICMIS 97- The 13th International Conference on Advanced Science and Technology in conjunction with The 2nd International Conference on Multimedia Information Systems,* Schaumburg, Illinois, April 1997.

[SantosM97b] Santos, M.T.P.;Vieira, M.T.P.;Figueiredo, J.M. **Extensão de um Banco de Dados de objetos MHEG-5 para suportar Busca por Conteúdo**; Anais do XII Simpósio Brasileiro de Banco de Dados, Fortaleza, Outubro 1997.

[SantosM97c] Santos, M. T. P.; Santos, F. C.; Figueiredo, J. M.; Vieira, M. T. P. **Descrição das Classes do Padrão MHEG-5***, Relatório de Trabalho 01*, Projeto SMmD, 1998.

[Si98] Si, A.; Leong, H. V.; Yau, S. M. T. **Maintaining Page Coherence for Dynamic HTML Pages**, *Proceedings of the 1998 ACM Symposium on Applied Computing,* Atlanta, Georgia, February 27 - March 1, 1998.

[Simão95] Simão, J. P. F. **Segurança na WWW - Panorâmica do Estado da Arte**, *The Portugal WWW National Conference,* Braga - Portugal, 6-8 July, 1995.

[Smil98a] **Synchronized Multimedia Integration Language**, (URL: http://www.real.com/technology/smil), 1998.

[Smil98b] Synchronized Multimedia Working Group of the World Wide Web Consortium, **Synchronized Multimedia Integration Language (SMIL) 1.0 Specification**, *W3C Recommendation*, (URL: http://www.w3.org/TR/REC-smil) , Junho, 1998.

[Teixeira95] Teixeira, C.A.C.; Vieira, M.T.P.; Araújo, R.B.; Trevelin, L.C.; Pimentel, M.G.C.; Sena, C.V.G. **Arquitetura do Projeto SMmD .** *Anais do I Workshop em Sistemas Hipermídia Distribuídos.* (URL: www.icmsc.sc.usp.br/anais/ wshd95), São Carlos, julho,1995, p.30-39.

[Teixeira98] Teixeira, C. A. C.; Pimentel, M. G. C.; Vieira, M. T. P.; Santos, M. T. P.; Abrão, I. C.; Barrére, E.; Baldochi Jr., L. **SMmD - A MHEG-5 Based Distributed Multimedia System**. *In Proceedings of the ProTem-CC Conference - Projects fase II*, Belo Horizonte - MG, Brazil, April 1998, p. 239-260

[Vieira96] Vieira, M.T.P.; Santos, F.C.; Santos, M.T.P. **Especificação de um Servidor de Objetos MHEG-5 para Aplicações Multimídia**, *Anais do XI Simpósio Brasileiro de Banco de Dados* (URL: www.dc.ufscar.br/eventos/sbbd96), São Carlos, Outubro 1996, pp. 231- 245.

[Vieira97] Vieira, M.T.P., Santos, M.T.P. **Content-based Search on a MHEG-5 Standard-based Multimedia Database**. *Proceedings of the QPMIDS DEXA 97,* Toulouse – FR, september 1997.

[Vieira99] Vieira, M.T.P.; Biajiz, M.; Santos, M.T.P.; Ribeiro, L.C.; Fornazari, F.P. **Metadata for Content-Based Search on an MHEG-5 Multimedia Objects Server**, *In Proceedings of the Third IEEE-Meta-Data Conference,* NIH Campus, Bethesda – Maryland, USA, April, 1999.

[W3C98] **Timed Interactive Multimedia Extensions for HTML – HTML+TIME.** *W3C Note*, (URL:http://www.w3.org/TR/NOTE-HTMLplusTIME), September, 1998.

[Yang96] Yang, J.J.; Kaiser G.E. **An Architeture for Integrating OODBs with WWW,** *Proceedings of the Fifth International World Wide Web Conference,* Paris, France, May 6 - 10, 1996.

[Yu97] Yu, J.; Xiang, Y. **Hypermedia Presentation and Authoring System**, *Proceedings of the Sixth International World Wide Web Conference*, Santa Clara Convention Center, California, USA, April 7 –11, 1997.

Apêndice A: Módulo de Conexão

Para implementar o módulo de conexão, foi criada uma classe denominada *O2Connection* que é uma subclasse da *OODBConnection*, conforme mostrado na figura 7.4. Essa classe foi criada com o objetivo de conter informações sobre as conexões realizadas no SGBDOO O2. Na classe *O2Connection* são definidos dois atributos: *systemName* e *baseName.* Para fornecer ao usuário uma flexibilidade de escolha, foram desenvolvidos os métodos *setSystem()* e *setBase()* para a configuração do sistema e da base para o armazenamento e também para as consultas de objetos no banco de dados O2. Portanto, para realizar uma conexão para o SGBDOO O2 é necessário ativar o método *start()*. Após ativar o método *start()*, deve-se ativar o *setBase()* que indica qual a base que será utilizada na conexão. É importante ressaltar que os métodos e atributos da classe *O2Connection* foram definidos como privados, permitindo que apenas os objetos da própria classe tenham acesso.

Prevendo a implementação para o SGBDOO Jasmine e Poet, deixamos especificadas as suas classes de conexão (*JasmineConnection* e *PoetConnection*), permitindo que sejam incluídos atributos e métodos específicos para cada um deles. A implementação da conexão para esses SGBDOOs serão realizadas futuramente.

Para realizar conexão com SGBDOO O2, também foi necessária a redefinição de alguns métodos do *OODBConnection*. Esses métodos se transformaram em métodos nativos, conforme relatado na seção 6.2.6. As especificações em Java das classes *OODBConnection* e *O2Connection* estão ilustradas no quadro 14. Observa-se que existe uma indicação de uma biblioteca chamada *libconnection***.** Essa biblioteca é composta por todos os programas nativos em C que realizam a conexão.

Quadro 14 – Arquivo fonte em Java das classes OODBConnection e O2Connection

OODBConnection.java	O2Connection.java

import java.lang.*;
public class OODBConnection
{
// attributes
protected String userName;
protected String password;
protected String netAddress;
protected String alias;
 	
// methods
public void setUser(String userName) {};
	import java.lang.*;
	public class O2Connection extends OODBConnection
	{
	// OODBConnection attributes
	protected String userName;
	protected String password;
	protected String netAddress;
	protected String alias;
	
	// O2Connection attributes
	private String baseName;

```
public String getUser() {};
public void setPassword(String password) {};
public String getPassword() {};
public void setNetAddress(String NetAddress) {};
public String getNetAddress() {};
public void setAlias(String alias) {};
public String getAlias() {};
public void start(int argc, String argv) {};
public void insertObject(o2_Object, Object) {};
public void deleteObject(String functionName) {};
public void queryObject(result,
     String queryString,  String parameters) {};

}
```

```
private String systemName;

// OODBConnection methods
public native void setUser(String userName);
public native String getUser();
public native void setPassword(String password);
public native String getPassword();
public native void setNetAddress(String NetAddress);
public native String getNetAddress();
public native void setAlias(String alias);
public native String getAlias();
public native void start(int argc, String argv);
public native void insertObject();
public native void deleteObject(String functionName);
public native void queryObject(Object result,
                String queryString,  String parameters);

// O2Connection methods
private native void setSystem(String systemName);
private native void setBase(String baseName);
private native void insertObject(newUser : User);
private native void insertObject(newScene : Scene);
private native void insertObject
                        (newApplication : Application);
static
   {  System.loadLibrary("libconnection");    }
}
```

Para criar um método nativo em Java que chama uma especificação desenvolvida em linguagem C, é necessário realizar alguns passos, sendo eles:

1. Criar o arquivo *.java* contendo os métodos nativos (por exemplo, O2Connection.java);
2. Compilar o arquivo *.java* (javac O2Connection.java);
3. Criar um arquivo de cabeçalho, através do comando *javah O2Connection*;
4. Criar um arquivo *stubs*, com o comando *javah –stubs O2Connection*;
5. Implementar os métodos nativos para cada método;
6. Compilar os métodos nativos em C;
7. Criar uma biblioteca que seja localizada dinamicamente (*libconnection*);
8. Configurar o *path* incluindo a biblioteca gerada e
9. Criar um programa para testar os novos métodos criados (nativos em C).

Para conexão com o SGBDOO O2 foram criados os seguintes programas em C: *start.c, setNetAddress.c, setSystem.c* e *setBase.c*. Observa-se que todos os programas criados possuem as iniciais do programa em Java (que no caso é da classe O2Connection.java), juntamente com o nome do programa em C. O programa *start.c* é o responsável pela configuração dos parâmetros de inicialização para a conexão. No

programa é utilizado um ponteiro chamado **sinit**, para uma instância que possui a seguinte estrutura:

```
typedef  struct {
        char*   sysdir;
        char*   sysname;
        char*   svname;
        char*   swapdir;
        char** libpath;
        char** libname;
} o2_sinit;
```

A definição dessa estrutura está contida no arquivo “o2_c.h” que pode ser localizada no diretório principal do O2. Nessa estrutura são armazenadas informações como o diretório de instalação do O2, nome do sistema, máquina onde o *o2server* está em funcionamento, diretório especial de *swap*, caminho para a busca das bibliotecas e nome das bibliotecas. Também é especificada no *start.c* uma função que inicia a conexão com o O2. Essa função é denominada *o2_link_init()*. Essa função retorna 0 caso tenha conectado, isto é, caso tenha encontrado a base e o diretório de instalação do O2. Caso contrário é retornado um código de erro de inicialização. A especificação do arquivo *start.c* é mostrada abaixo.

start.c

```
#include “o2_c.h”
void O2Connection_start(argc, argv)
int argc; char* argv[];
{
  static 02_sinit o2_sinit =
 {
        netAddress,
        systemName,
        “MachineName”,
        “.”,
        0,0
  };
  #define ALPHA 1
  if (o2_link_init(argc, argv, &o2_sinit, ALPHA) != 0
 {
        printf (“Something wrong to start o2 \n”);
        exit(1);
  }
}
```

Para configurar o diretório de instalação do O2, foi utilizado o método *setNetAddress()* que está especificado no programa *setNetAddress.c*. Esse programa apenas configura o nome do diretório para o programa *start.c.* Assim como esse programa,

o *setSystem.c* também apenas configura um parâmetro para o *start.c*. Esse parâmetro é o nome do sistema do SGBDOO O2 que o usuário deseja conectar.

Para especificar a base que o usuário queira conectar, foi desenvolvido o programa em C chamado *setBase.c*. Sua especificação é dada abaixo, e observa-se que é chamada a função *o2_link_set_base()*, cuja especificação é dada na biblioteca *libo2link.a* do O2.

setBase.c

```
void O2Connection_setBase(baseName)
char* baseName;
{
  o2_link_set_base(baseName);
}
```

Com esses programas em C é possível realizar conexão entre um programa Java e um SGBDOO O2.

Para exemplificar um programa Java para realizar conexão com o SGBDOO O2, simplesmente devemos importar para o programa em Java, a classe *O2Connection* que foi mostrada no capítulo anterior. Após isso, deve-se instanciar um objeto dessa classe e chamar o método *start()* e logo após o *setBase().* Veja no exemplo abaixo, no programa **Example.java**, a qual é instanciada uma conexão que indica qual o diretório de instalação do O2, que nesse exemplo é *"/ExampleDirectory/o2"*, indicando também o nome do sistema a ser inicializado *"ExampleSystem"* e por último é indicado o nome da base *"ExampleBase"*.

Example.java

```
import O2Connection;  // importa a classe O2Connection.java
public class Example
{
   public void executeConnection()
  {
         O2Connection  dbO2 = new O2Connection();
                                                  // instancia um objeto chamado dbO2
          dbO2.setNetAddress("/ExampleDirectory/o2"); // configura o diretório do O2
          dbO2.setSystem("ExampleSystem");       // indica o nome do sistema
          dbO2.start();                           // inicializa a conexão com os dados acima
          dbO2.setBase("ExampleBase");            // indica o nome da base que será usada
 }
}
```

Após estabelecida a conexão com o SGBDOO O2, os usuários podem realizar as operações de inclusão, alteração, consulta e exclusão de objetos no banco de dados O2,

utilizando os métodos *insertObject()*, *deleteObject()* e *queryObject()*. Veja um exemplo abaixo de uma consulta convencional.

```
queryObject (&result, “select  a from a in Application where Name = $1”, application.name);
```

Apêndice B: Módulo de Autenticação de Usuários

Este módulo tem como objetivo realizar o cadastro dos usuários que poderão utilizar a ferramenta de conversão entre aplicações MHEG-5 e SMIL, assim como as outras que serão fornecidas no ambiente MAW. Para utilizar a ferramenta de conversão, é necessário que o usuário inicialmente se identifique através de uma tela principal (mostrada na figura B.1)

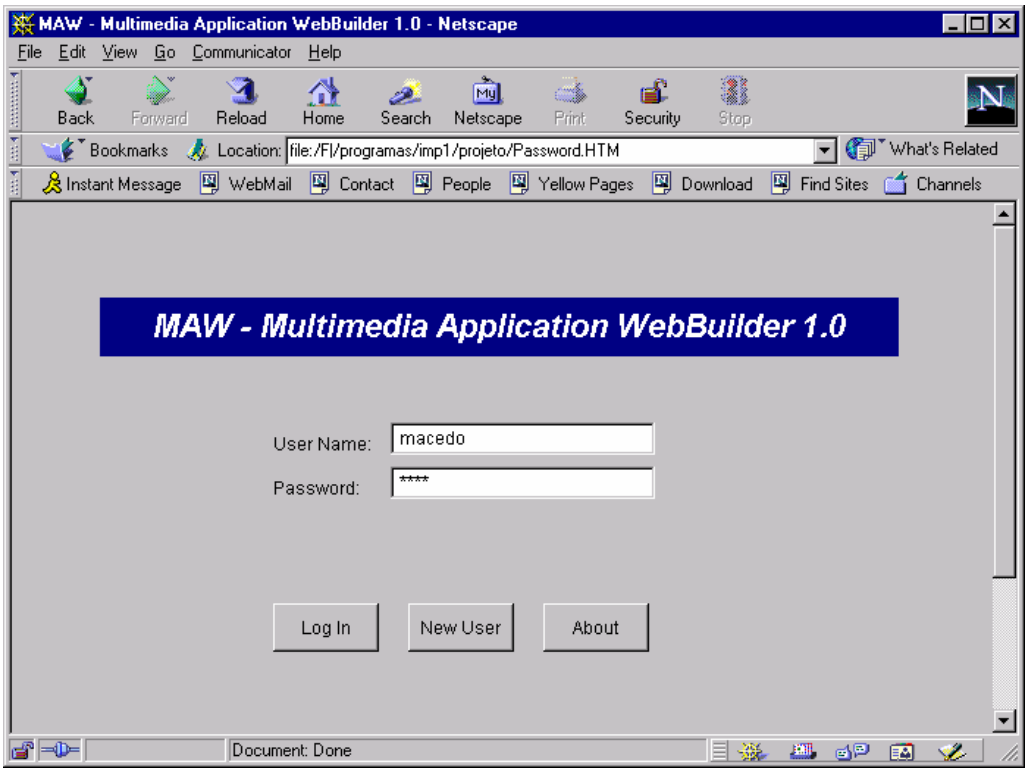


FIGURA B.1 – TELA INICIAL DO PROJETO MAW

Na tela mostrada na figura B.1, observa-se a existência de dois campos: *user name* e *password*. Caso o usuário já tenha utilizado a ferramenta anteriormente, é necessário que o mesmo informe qual é o seu *login* e *password* que foi registrado anteriormente. Após o usuário informar esses dados, o programa faz um busca no banco de dados de usuários do MAW e verifica a existência do mesmo. Caso haja cadastro, e também o prazo de validade de uso esteja em vigência, o programa automaticamente o autenticará e permitirá com que utilize o módulo de conversão. Caso o usuário não esteja cadastrado, ou caso o prazo de uso esteja expirado, então é retornada uma mensagem correspondente à ocorrência, e logo após, é chamada uma página para seu cadastro ou também recadastro. Pode ocorrer também o fato do usuário se registrar sem informar *login* e *password*. Nesse caso, o usuário

deve simplesmente clicar no botão *New User* e, com isso o programa busca a página de cadastro de usuários.

No cadastro do usuário é necessário informar alguns dados, sendo eles, nome, endereço, instituição, *login*, senha, cidade, estado/província, país, e-mail, telefone, áreas de atuação de pesquisas, caso houver, e algumas observações que queira registrar. A tela de cadastro de usuários é mostrada na figura B.2 contendo uma instância de usuário.

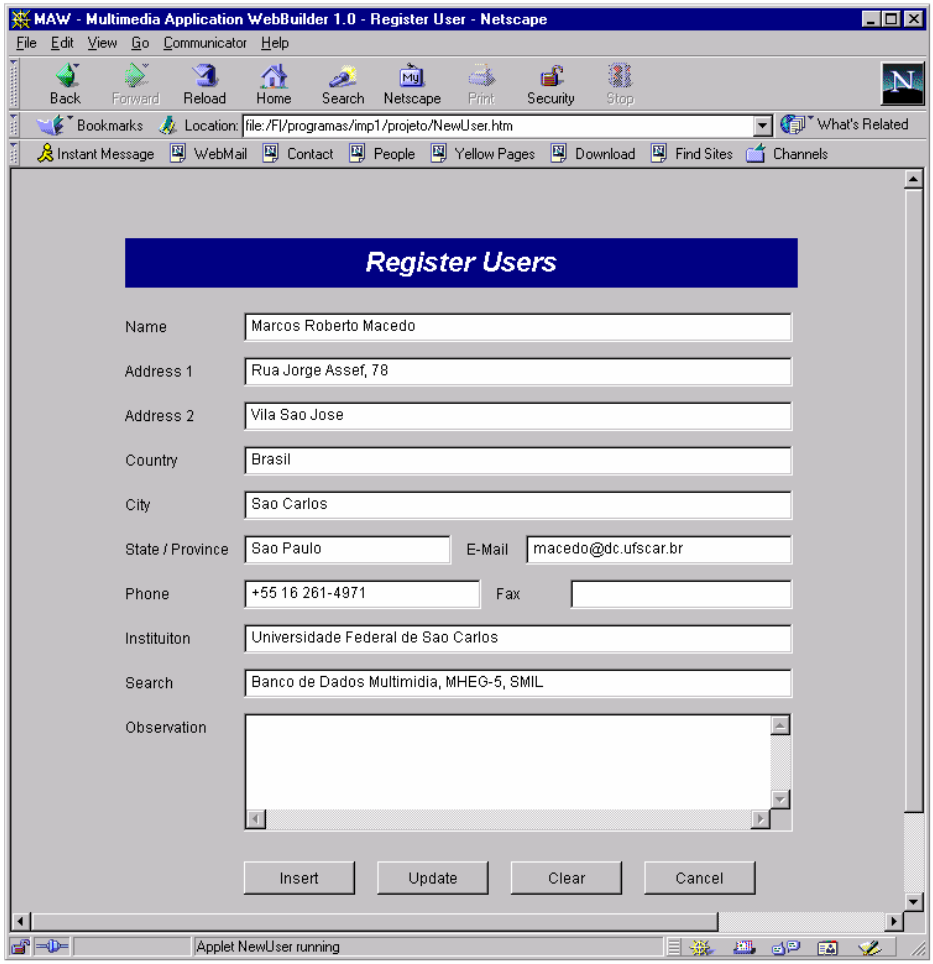


FIGURA B.2 – TELA DE CADASTRO DE USUÁRIOS COM UMA INSTÂNCIA

Após o usuário informar seus dados, é obrigatório informar qual será o *login* e a *password*. Para isso, o usuário clica no botão *Insert* ou *Update* e o programa chama a tela para registro de *login*. Essa tela é mostrada na figura B.3.

FIGURA B.3 – TELA DE CONFIGURAÇÃO DE LOGIN E PASSWORD INSTANCIADA

Após o usuário informar o *login*, o programa deve procurar se existe registrado no banco de dados de usuários algum usuário com o *login* informado. Caso exista, é emitida uma mensagem para que o usuário escolha um outro *login*. Caso contrário, é feita a validação de senha, isto é, verificam-se os campos *password* e *confirm password*. Caso estejam idênticas, o programa registra o *login* e *password* juntamente com uma data de registro do usuário.

Foi desenvolvido um algoritmo para especificar com detalhes toda a funcionalidade sobre cadastro de usuários. Esse algoritmo é apresentado abaixo:

```
Início
. Ler login/password do usuário solicitante
. Estabelecer conexão com o banco de dados O2
. Consultar usuário na classe User
. Caso o usuário não esteja devidamente cadastrado então
        . Validar login do usuário
        . Ler restante das informações do usuário
        . Registrar o usuário na classe User
        . Chamar próxima página da aplicação  (página de conversão)
Caso contrário
        . Mostrar informações do usuário
        . Verificar validade da senha
        . Caso senha no prazo correto então
                . Chamar próxima página da aplicação  (página de conversão)
          Fim-Caso
 Fim-Caso
Fim
```

A instância de usuário apresentado nas telas B.2 e B.3, foram registrados, após as validações, na classe *User* no banco de dados RuleDB. Essa classe armazena todos os usuários que usaram o MAW.

A figura B.4 apresenta a instância criada no ambiente do O2. Para manipular essas informações foi necessário criar três tipos de estruturas de dados: uma na linguagem Java para armazenar o objeto gerado da página; outra estrutura em C (.h) para receber o objeto que foi armazenado em Java; e a última estrutura foi criada para armazenar no banco de dados RuleDB, o objeto que está na estrutura em C.

FIGURA B.4 – INSTÂNCIA DE CLASSE USER MOSTRADA NO AMBIENTE DO SGBDOO O2

Abaixo no quadro 15 é mostrada uma descrição das três estruturas de dados criadas.

Quadro 15 – Estruturas de dados criados para cadastro de usuários

user.java	User.h	User.o2
public class user { String name; String address1; String address2; String country; String city; String phone; String login; String password; String fax;	typedef void* o2_User; typedef void* o2_list_User; typedef void* o2_set_User; typedef struct { char* name; char* address1; char* address2; char* country; char* city; char* phone;	class User inherit Object public type tuple (name: string, address1: string, address2: string, country: string, city: string, phone: string, fax: string, institution: string,

String institution;
String search;
String observation;
String lastdatepassword;
String machineIP;
String email;
String stateprovince;
 	
user(String x_name,
String address1;
String x_address2,
String x_country,
String x_city,
String x_phone,
String x_login,
String x_password,
String x_fax,
String x_institution,
String x_search,
String x_observation,
String x_lastdatepassword,
String x_machineIP,
String x_email,
String x_stateprovince)
{
name = x_name;
address1 = x_address1;
address2 = x_address2;
country = x_country;
city = x_city;
phone = x_phone;
password = x_password;
.
.
.
}
}
	char* fax;
	char* institution;
	char* search;
	char* observation;
	char* login;
	char* password;
	char* lastdatepassword;
	char* machineIP;
	char* email;
	char* stateprovince;
	o2_User object_id;
	} user;
	extern o2_User User_new();
		search: string,
		observation: string,
		login: string,
		password: string,
		machineIP: string,
		email: string,
		stateprovince: string,
		lastdatepassword: string)
		method
		public init(name: string),
		public read(val: pointer),
		public write(val: pointer),
		private provide_password
		(val: pointer)
		end;

Após o usuário informar os dados de cadastros e também o *login* e *password* são então registrados o seu cadastro. Para isso foi necessário criar os seguintes métodos:

1. Criar uma conexão com o banco de dados O2;

2. Instanciar um objeto utilizando a estrutura criada em Java;

3. Repassar esse objeto Java para uma estrutura em C (mediante os métodos nativos escritos em Java) e

4. Executar o método de escrita especificada na classe *User* do banco de dados RuleDB.

No quadro 16 é especificado o código fonte do programa *NewUser.java*, que mostra a conexão com o banco de dados O2 e a chamada de um método para o armazenamento de um usuário. Logo após, é disponibilizada a página do conversor de aplicações.

Quadro 16 – Código fonte do programa NewUser.java

```
void button1_ActionPerformed(java.awt.event.ActionEvent event)
  {
    O2Connection conexao = new O2Connection();
    conexao.start(); // inicializa a conexão com o SGBDOO O2
    user newUser = new user( textField1.getText(),
                                        textField2.getText(),
                                        textField3.getText(),
                                        textField4.getText(),
                                        textField5.getText(),
                                        textField6.getText(),
                                        textField7.getText(),
                                        textField8.getText(),
                                        textField9.getText(),
                                        textField10.getText(),
                                        textField11.getText(),
                                        textArea12.getText() );
    conexao.insertObject(newUser);  // insere um novo usuario na classe User
    String url = new String (getCodeBase() + pagina_senha);
    try
     {
      URL conectar = new URL (url);
      getAppletContext().showDocument (conectar);  // chama a página do conversor
     }
    catch (MalformedURLException erro)
     {
      System.out.println (erro);
     }
  }
```