UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MUSE: Um Ambiente para Modelagem de Aplicações Multimídia Interativas com Tradutor para E-LOTOS

por

LUCIANO PASCHOAL GASPARY

Dissertação submetida à avaliação, como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Profª. Drª. Maria Janilce B. Almeida
Orientadora

Porto Alegre, julho de 1998.

CIP- CATALOGAÇÃO NA PUBLICAÇÃO

Gaspary, Luciano Paschoal

MUSE: Um Ambiente para Modelagem de Aplicações Multimídia Interativas com Tradutor para E-LOTOS / por Luciano Paschoal Gaspary. – Porto Alegre: CPGCC da UFRGS, 1998.
106f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Curso de Pós-graduação em Ciência da Computação, Porto Alegre, BR-RS, 1998. Orientadora: Almeida, Maria Janilce B. de

1. Aplicações multimídia interativas. 2.Sincronização. 3. Modelos de Autoria. 4. Verificação. 5. Validação. 6. E-LOTOS. 7. Ambiente de autoria. I. Almeida, Maria Janilce B. de. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profa. Wrana Panizzi
Pró-Reitor de Pós-Graduação: Prof. José Carlos
Diretor do Instituto de Informática: Prof. Roberto Tom Price
Coordenador do CPGCC: Profa. Carla Maria Dal Sasso Freitas
Bibliotecária-Chefe do Instituto de Informática: Zita Prates de Oliveira

Agradecimentos

Gostaria de agradecer em primeiro lugar à profa. Janilce Almeida pela sua orientação, bem como pelo inestimável apoio e incentivo durante todo o curso de mestrado.

Agradeço de forma especial à Ana Cristina pelo carinho, compreensão e paciência neste período.

Ao amigo Lisandro Granville gostaria de agradecer pelo companheirismo e parceria nos últimos seis anos.

Aos colegas do projeto DAMD: Marcelo Soares, Luciane Trarbach, Daniel Vieiro e Ana Carolina Hermann gostaria de deixar expresso meu agradecimento pela colaboração com os trabalhos realizados neste último ano.

Um agradecimento também especial aos meus pais, João Carlos e Liane, e às minhas irmãs, Anelise e Cristine, por todo apoio e compreensão que sempre tiveram.

Sumário

LISTA DE FIGURAS 6

LISTA DE TABELAS 8

RESUMO 9

ABSTRACT 10

1 INTRODUÇÃO 11

2 MODELAGEM DE APLICAÇÕES MULTIMÍDIA INTERATIVAS 13

2.1 Aspectos básicos de sincronização em sistemas multimídia 14
2.1 Modelos de sincronização 19
2.1.1 Modelo baseado em intervalos 19
2.1.2 Modelo baseado em linha temporal 22
2.1.3 Modelos baseados em fluxo de controle 23
2.1.4 Sincronização baseada em eventos 28
2.2 O padrão MHEG-5 29

3 AUTORIA DE APLICAÇÕES MULTIMÍDIA E FERRAMENTAS 33

3.1 Sistemas de autoria baseados em grafos 33
3.1.1 IconAuthor 34
3.1.2 Editor I-HTSPN 35
3.2 Sistemas de autoria baseados em linha do tempo 36
3.2.1 Macromedia Director 37
3.3 Sistemas de autoria baseados em scripts 38
3.4 Sistemas de autoria baseados em estruturas 39
3.4.1 CMIFed 40
3.4.2 EBS 42

4 MODELO DE AUTORIA PROPOSTO 45

4.1 Símbolos gráficos do modelo 47
4.2 Estruturação lógica 48
4.2.1 Cenários e grupos 48
4.2.2 Ícones de início e fim 49
4.2.3 Representação de componentes compartilhados 50
4.2.4 Suporte ao reuso de cenários e grupos 50
4.3 Sincronização Temporal 51
4.3.1 Apresentação seqüencial e paralela 51
4.3.2 Duração de eventos e delay 52
4.3.3 Mecanismo de transição entre cenários 52
4.3.4 Pontos de sincronização 53
4.3.5 Instantes de sincronização 54
4.3.6 Ícones de composição 55
4.4 Sincronização espacial 55
4.5 Um exemplo de utilização do modelo 56

5 TRADUÇÃO DAS ESPECIFICAÇÕES PARA E-LOTOS 57

5.1 Características e operadores da TDF E-LOTOS 57

5.1.1 Declaração de tipos de dados 58

5.1.2 Declaração de processos 59

5.1.3 Expressões de comportamento 60

5.1.4 Declaração de módulos 64

5.2 Representação em E-LOTOS de aplicações multimídia interativas 64

5.2.1 Estruturação do processo raiz e declaração dos dados 65

5.2.2 Representação de grupos 66

5.2.3 Representação de cenários 69

5.2.4 Representação de componentes multimídia e restrições 71

6 O AMBIENTE MUSE 74

6.1 Funcionalidades básicas 74

6.1.1 Repositório de componentes multimídia 75

6.1.2 Visão hierárquica da aplicação 77

6.1.3 Palheta de ícones 78

6.1.4 Visão espacial dos cenários 81

6.1.5 Segmentação de componentes multimídia dependentes do tempo 82

6.1.6 Reuso de grupos e cenários 83

6.1.7 Geração automática de código E-LOTOS 86

6.2 A implementação 86

6.2.1 Classes de janela 87

6.2.2 Classes para a consistência da especificação 88

7 CONCLUSÕES E TRABALHOS FUTUROS 93

ANEXO 1 A BIBLIOTECA DE TIPOS BÁSICOS 95

ANEXO 2 APLICAÇÃO EXEMPLO EM E-LOTOS 97

BIBLIOGRAFIA 104

Lista de Figuras

FIGURA 1.1 - Estrutura do projeto DAMD 12
FIGURA 2.1 - Sistemas multimídia locais e com recursos distribuídos 13
FIGURA 2.2 - Sincronização de componentes independentes do tempo 14
FIGURA 2.3 - Sincronização a nível de LDU 15
FIGURA 2.4 - Aplicação com diversos elementos multimídia 15
FIGURA 2.5 - Modelo de referência de sincronização 16
FIGURA 2.6 - Aplicação utilizando a interface oferecida pelo nível de mídia 16
FIGURA 2.7 - Aplicação desenvolvida no nível de fluxo 17
FIGURA 2.8 - Aplicação utilizando a interface oferecida pelo nível de mídia 18
FIGURA 2.9 - Relações temporais entre dois objetos 20
FIGURA 2.10 - Operações da extensão do modelo baseado em intervalos 21
FIGURA 2.11 - Especificações segundo modelo baseado em intervalos 21
FIGURA 2.12 - Especificação segundo modelo baseado em linha temporal 22
FIGURA 2.13 - Especificação segundo modelo hierárquico 23
FIGURA 2.14 - Sincronização no modelo baseado em pontos de referência 24
FIGURA 2.15 - Especificação segundo modelo pontos de referência 25
FIGURA 2.16 - Relações de intervalos representadas em OCPN 26
FIGURA 2.17 - Variações temporais na apresentação de componentes multimídia 26
FIGURA 2.18 - Semântica de sincronização do modelo TSPN 27
FIGURA 2.19 - Aplicação representada no modelo HTSPN 28
FIGURA 2.20 - O *engine* MHEG 29
FIGURA 2.21 - Estrutura básica das aplicações MHEG-5 30
FIGURA 2.22 - Hierarquia de classes do padrão MHEG-5 31
FIGURA 2.23 - Definição de um cenário simples em MHEG-5 32
FIGURA 3.1 - Grafos informais e formais 34
FIGURA 3.2 - Interface gráfica do IconAuthor 35
FIGURA 3.3 - Interface do Editor I-HTSPN 36
FIGURA 3.4 - Paradigma baseado em linha temporal 37
FIGURA 3.5 - Grade temporal do Macromedia Director 38
FIGURA 3.6 - Aplicação descrita através de um *script* 39
FIGURA 3.7 - Autoria baseada em estruturas 40
FIGURA 3.8 - Visão hierárquica do CMIFed 41
FIGURA 3.9 - Visão dos canais para a seqüência Rota da Caminhada 42
FIGURA 3.10 - Janela principal do EBS 43
FIGURA 3.11 - Relações de sincronização entre objetos 44
FIGURA 4.1 - Estrutura de uma aplicação multimídia interativa 47
FIGURA 4.2 - Conjunto de ícones suportados pelo modelo 48
FIGURA 4.3 - Estruturas básicas para navegação aplicações multimídia interativas 49
FIGURA 4.4 - Componentes compartilhados entre cenários e grupos 50
FIGURA 4.5 - Reutilização de cenários e grupos 51
FIGURA 4.6 - Apresentação seqüencial e simultânea 51
FIGURA 4.7 - Utilização de *delay* 52
FIGURA 4.8 - Transição de cenas 53

FIGURA 4.9 - Pontos de sincronização e políticas de disparo 54
FIGURA 4.10 - Sincronização de um vídeo com legendas 55
FIGURA 4.11 - Representação de um exemplo completo 56
FIGURA 5.1 - Esqueleto da estrutura principal de uma aplicação em E-LOTOS 65
FIGURA 5.2 - Modelagem do GrupoInicial em E-LOTOS 66
FIGURA 5.3 - Representação de grupos com estrutura de rede 67
FIGURA 5.4 - Representação de componentes compartilhados 68
FIGURA 5.5 - Representação de cenários 69
FIGURA 5.6 - Código de uma transição simples 70
FIGURA 5.7 - Cenário com mais de um ponto de saída e com restrições 70
FIGURA 5.8 - Código de uma transição simples 71
FIGURA 5.9 - Representação de objetos multimídia simples 73
FIGURA 5.10 - Objetos multimídia em cenários com mais de uma transição 73
FIGURA 5.11 - Objetos multimídia envolvidos com pontos de sincronização 73
FIGURA 6.1 - Poder de expressão *vs.* facilidade de uso em ambientes de autoria 74
FIGURA 6.2 - Interface gráfica do ambiente de autoria 75
FIGURA 6.3 - Adição de componentes ao repositório 76
FIGURA 6.4 - Gerenciamento dos componentes da aplicação 76
FIGURA 6.5 - Propriedades de conteúdo do componente selecionado 76
FIGURA 6.6 - Visão hierárquica da aplicação 77
FIGURA 6.7 - Visão modular baseada no contexto da especificação 77
FIGURA 6.8 - Acesso agilizado aos grupos e cenários da aplicação 78
FIGURA 6.9 - Palheta de ícones 78
FIGURA 6.10 - Associação do componente ao ícone de apresentação 79
FIGURA 6.11 - Definição das transições 81
FIGURA 6.12 - Visão espacial do cenário Blocagem das Rodas 82
FIGURA 6.13 - Segmentação de componentes dependentes do tempo 82
FIGURA 6.14 - Reagrupamento de segmentos 83
FIGURA 6.15 - Reuso de grupos e cenários 84
FIGURA 6.16 - Reutilização do grupo Capítulo1 84
FIGURA 6.17 - Componente com mesmo nome já presente no repositório 85
FIGURA 6.18 - Mensagem de transição inexistente 85
FIGURA 6.19 - Atualização automática das transições possíveis 86
FIGURA 6.20 - Resultados da pré-compilação 86
FIGURA 6.21 - Classes de janela do ambiente 88
FIGURA 6.22 - A classe clsEspecificação 89
FIGURA 6.23 - A classe clsElemento 89
FIGURA 6.24 - A classe clsGrupo 90
FIGURA 6.25 - A classe clsCenário 90
FIGURA 6.26 - As classes clsÍcone 91
FIGURA 6.27 - A classe clsArco 91
FIGURA 6.28 - Diagrama ER das classes do ambiente 92

Lista de Tabelas

TABELA 5.1 - Objetos monolíticos utilizados para representar objetos multimídia.. 72
TABELA 5.2 - Restrições descritas em E-LOTOS 72
TABELA 6.1 - Componentes multimídia suportados pelo ambiente 76
TABELA 6.2 - Propriedades dos ícones 80

Resumo

É notável o avanço da utilização de aplicações multimídia nos diversos setores da atividade humana. Independente da área, seja ela educação ou entretenimento, a possibilidade de agregar recursos dinâmicos como áudio e vídeo aos já largamente utilizados como texto e imagem acarreta em benefícios aos usuários destas aplicações. Além disso, com a popularização da Internet, há uma crescente demanda pela sua execução em ambientes distribuídos.

Este trabalho teve como objetivo desenvolver MUSE, um ambiente gráfico para modelagem de aplicações multimídia interativas. Através de uma interface gráfica avançada e de um novo modelo de autoria de alto nível, é possível a criação de sistemas complexos de forma rápida e intuitiva. O modelo de autoria proposto neste trabalho e adotado pelo ambiente prevê a possibilidade de os elementos que constituem a aplicação estarem dispersos em uma rede de computadores, permitindo a definição de limiares aceitáveis de atraso e componentes alternativos.

Pela grande expressividade do modelo, no entanto, podem ser geradas especificações com inconsistências lógicas e temporais. Por esta razão, o ambiente provê ainda especificações E-LOTOS - uma extensão temporal de LOTOS - utilizadas para fins de análise e verificação, permitindo a validação dos requisitos temporais das aplicações definidas pelo autor. A formalização das especificações através de uma TDF, além de viabilizar sua validação, provê descrições sem ambiguidades, que podem ser alternativamente utilizadas por autores que sejam familiarizados com a técnica de descrição formal.

Este trabalho é parte do projeto DAMD (*Design de Aplicações Multimídia Distribuídas*) dentro do programa PROTEM fase 2, que tem por objetivo fornecer uma metodologia que cubra o ciclo completo das aplicações multimídia distribuídas e que permita a um autor não especializado em métodos formais desenvolver essas aplicações naturalmente.

Palavras-chave: aplicações multimídia interativas, sincronização, modelos de autoria, verificação, validação, E-LOTOS, ambiente de autoria.

Title: "MUSE: An Interactive Multimedia Applications Specification Environment with Translator to E-LOTOS"

Abstract

It is notable the advance of multimedia applications utilization in several fields of human activity. Independent from the area, whether education or entertainment, the possibility to aggregate dynamic resources like audio and video to the ones already widely used like text and image results in benefits to the users of such applications. Besides, with the popularization of the Internet, there is an increasing demand for their execution in distributed environments.

This work presents MUSE, a graphical environment for modeling interactive multimedia applications. Through an advanced graphic interface and a new high-level authoring model, it is possible to create complex systems in a fast and intuitive way. The authoring model proposed in this work and adopted by the tool deals with media objects distributed in a computer network, allowing the definition of acceptable delay thresholds and alternative media objects.

Due to the large expressiveness of the model, however, specifications can be generated with logical and temporary inconsistencies. For this reason, the tool also provides E-LOTOS specifications used with the purpose of analyzing and verifying the applications aiming at validating the temporal requirements defined by the author. The formalization of the specifications by means of a TDF, beyond making their validation possible, provides descriptions free of ambiguities, which may be alternatively used by authors familiarized with the formal description technique.

This work is part of DAMD (Distributed Multimedia Applications Design) project, sponsored by the Brazilian research council. Its main objectives are to provide a methodology to completely cover the distributed multimedia applications development cycle and to allow authors who are not expert in formal methods to easily develop their applications.

Keywords: interactive multimedia applications, syncronization, authoring models, verification, validation, E-LOTOS, authoring environment.

1 Introdução

Os anos 90 têm sido marcados pela utilização de aplicações multimídia em diversos setores da atividade humana tais como educação, medicina e entretenimento. Estas aplicações tem se sofisticado cada vez mais ao longo do tempo, sendo que atualmente são executadas em ambientes distribuídos, operando transparentemente em plataformas hetorogêneas. Considerando-se este novo contexto, passa-se a tratar com aspectos como a garantia da qualidade de serviço (QoS - *Quality of Service*) e a definição de formatos de transferência comuns, objetivando a criação de sistemas abertos.

A possibilidade de os componentes que fazem parte da aplicação multimídia estarem dispersos em uma rede impacta não apenas os pontos citados acima, como também o processo de criação e modelagem das aplicações. Os usuários devem fornecer ao ambiente de autoria informações como restrições temporais, definindo limiares aceitáveis de atraso aos elementos que compõem o sistema e estabelecendo a apresentação de componentes alternativos. Torna-se possível definir, por exemplo, que um áudio deva ser executado em um determinado tempo e que, caso este tempo seja excedido, um texto será apresentado em seu lugar. Em outra situação, este componente alternativo pode ser utilizado quando a estação em que a aplicação esteja sendo executada não apresente os recursos mínimos necessários, como uma placa de som.

A definição destas restrições é realizada com base em um modelo de sincronização, que dita as regras sobre como os componentes de uma aplicação podem ser relacionadas no tempo. Vários modelos de sincronização têm sido propostos [BLA96], aumentando sensivelmente a flexibilidade e o poder de expressão das especificações. Vale ressaltar que o termo especificação é utilizado neste documento significando o processo de modelagem das aplicações multimídia. Em razão desta ampla expressividade dos modelos, no entanto, as especificações resultantes podem ser fonte de incoerências, não havendo garantia sobre a consistência lógica e temporal dos componentes multimídia envolvidos.

Uma alternativa seria utilizar diretamente uma técnica de descrição formal (TDF) para descrever as aplicações, viabilizando sua análise e assim garantindo sua consistência. Esta seria a melhor opção se todos os autores fossem familiarizados com estas ténicas, o que não ocorre pela alta complexidade inerente às mesmas. Evidencia-se, assim, a necessidade de se contar com um modelo estruturado de alto nível para especificar aplicações multimídia. Ao mesmo tempo, deseja-se que o resultado desta especificação seja traduzido para uma TDF, sob a qual seja possível aplicar métodos de verificação e simulação. A formalização das especificações, além de viabilizar sua validação, provê descrições sem ambiguidades, que podem ser alternativamente utilizadas por autores que conhecam a técnica de descrição formal.

Dentro deste contexto, considerando alguns conceitos presentes em MHEG-5 [ISO96] e alguns modelos de sincronização, criou-se um modelo de autoria de aplicações multimídia. Baseado neste modelo, desenvolveu-se o ambiente MUSE (*Interactive Networked MUltimedia Applications Specification Environment*), que permite ao usuário definir, em alto-nível e com elevado poder de expressão, uma apresentação multimídia em conformidade com o padrão MHEG-5. A adoção de MHEG-5 permite o compartilhamento de informação multimídia sem se preocupar com a plataforma ou sistema operacional utilizado, viabilizando a especificação e

desenvolvimento de aplicações portáveis. Para viabilizar o processo de validação das especificações, o ambiente permite traduzi-las para a TDF E-LOTOS [GAS98a] [GAS98b] [GAS98c] [GAS98d].

Este trabalho faz parte constituinte do projeto PROTEM (DAMD - *Design de Aplicações Multimídia Distribuídas*), que tem por objetivo fornecer uma metodologia que cubra o ciclo completo das aplicações multimídia distribuídas e que permita a um autor não especializado em métodos formais desenvolver essas aplicações naturalmente. Em linhas gerais, o projeto foi desenvolvido de acordo com a figura 1.1. O ambiente MUSE de certa forma centraliza o processo que compreende modelagem e apresentação das aplicações finais. As especificações criadas pelo usuário são validadas e os resultados obtidos são apresentados a ele de forma bastante legível no próprio ambiente. O processo especificação-validação se repete até que as incoerências sejam eliminadas. Após esta etapa, geram-se aplicações MHEG-5, que podem ser executadas pelo *engine*.

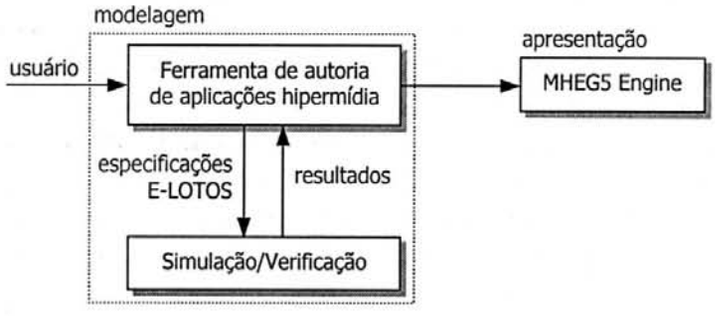


FIGURA 1.1 - Estrutura do projeto DAMD

O capítulo 2 introduz o estudo de aspectos relevantes em aplicações multimídia como o conceito de sincronização neste tipo de aplicações e alguns modelos de sincronização propostos na literatura. O capítulo 3 ilustra algumas ferramentas de autoria existentes. O capítulo 4 apresenta o modelo de autoria proposto. No capítulo 5 são abordados aspectos básicos da TDF E-LOTOS e o estudo do mapeamento do modelo de autoria para esta linguagem. O capítulo 6 apresenta o ambiente e as estruturas internas empregadas, e o capítulo 7, as considerações finais.

2 Modelagem de Aplicações Multimídia Interativas

Sistemas multimídia são aqueles que integram diversos tipos de informação como texto, imagem, animação, vídeo e áudio. Esta é a definição mais popular para esta classe de aplicações. A literatura especializada no assunto, no entanto, prefere defini-los como a associação de pelo menos um texto ou imagem, denominados componentes multimídia discretos ou independentes do tempo, com pelo menos um vídeo ou áudio, os chamados componentes contínuos ou dependentes do tempo. Estes sistemas são classificados como interativos, quando fornecem mecanismos para que o usuário possa controlar sua execução.

Em outra classificação, os sistemas multimídia podem ser categorizados como locais ou com recursos distribuídos. Os sistemas multimídia locais se caracterizam por não utilizar recursos além dos presentes localmente [FLU95]. Estes sistemas são equipados com todo o aparato necessário como microfones, câmeras e não se utilizam de recursos remotos de armazenamento. Alguns exemplos desta classe de aplicações são:

- *Treinamento individual baseado em computador (CBT-Computer-Based Training)*: técnicos e engenheiros aprendem operações de manutenção através de documentação multimídia presente em estações de trabalho. O documento é estático e inteiramente armazenado em recursos locais como CD-ROMs.
- *Educação individual baseada em computador (CBE)*: alunos seguem cursos ou praticam exercícios sobre determinado assunto utilizando aplicações multimídia instaladas inteiramente em computadores pessoais.

Os sistemas multimídia com recursos distribuídos referem-se a aplicações separadas em sub-sistemas que se comunicam através de redes de computadores (vide figura 2.1). Duas razões justificam esta classe de aplicações:

- Algumas aplicações são genuinamente distribuídas, uma vez que seu objetivo é permitir a comunicação remota entre indivíduos; um exemplo são as aplicações de videoconferência.
- Por razões econômicas, recursos multimídia como dispositivos de armazenamento podem ser alocados em servidores, viabilizando compartilhamento de informação. Estes servidores são acessados remotamente pelos sistemas de apresentação.

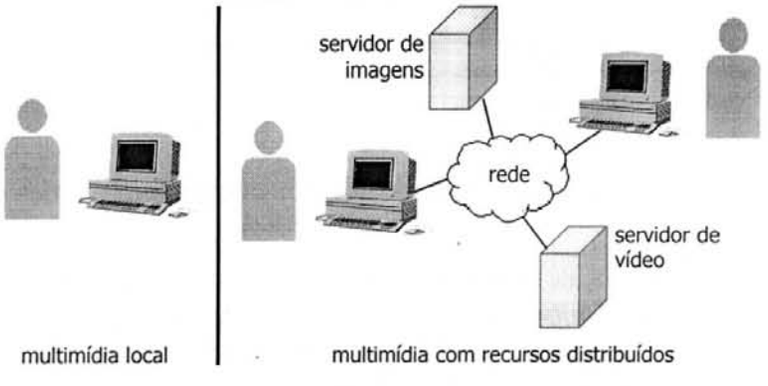


FIGURA 2.1 - Sistemas multimídia locais e com recursos distribuídos

Este trabalho aborda o processo de especificação de aplicações multimídia interativas com recursos distribuídos. Nas próximas seções são apresentados tópicos de interesse a esta classe de aplicações.

2.1 Aspectos básicos de sincronização em sistemas multimídia

O termo sincronização, quando relacionado com sistemas multimídia, refere-se à relação temporal existente entre os diferentes componentes que compõem estes sistemas [BLA96]. Como já mencionado anteriormente, estes componentes são classificadas como dependentes ou independentes do tempo. Os componentes dependentes do tempo, representados por seqüências de áudio e vídeo, são caracterizados pelo fluxo de informações que os compõem. No caso de um vídeo, por exemplo, este fluxo é representado por seus diversos quadros associados a um tempo fixo de apresentação. Os componentes que independem do tempo, por sua vez, não apresentam tal característica, sendo constituídos por elementos de natureza estática como imagem e texto.

Existem duas formas básicas de sincronização: intra-mídia e entre mídias. A sincronização intra-mídia refere-se às relações de tempo entre várias unidades de apresentação de uma única mídia dependente do tempo. A relação temporal estabelecida entre os quadros de uma seqüência de vídeo é um exemplo deste tipo de sincronização. Por outro lado, a sincronização entre mídias objetiva ordenar temporalmente a apresentação de dois ou mais componentes multimídia distintos, sejam eles dependentes ou independentes do tempo. A seguir são apresentadas várias formas de sincronização entre mídias, considerando tanto os componentes independentes como os dependentes do tempo.

A sincronização entre elementos independentes do tempo é simples, uma vez que eles apresentam um comportamento bem conhecido representado por um tempo inicial e final de apresentação. Como exemplo deste tipo de sincronização, poderia-se ter uma apresentação de três *slides*, com tempos individuais de 10 segundos. No quarto segundo de cada um deles, começa a ser apresentado um texto explicativo, que permanece no cenário até o fim do período de apresentação do respectivo *slide* (vide figura 2.2).

FIGURA 2.2 - Sincronização de componentes independentes do tempo

A sincronização de elementos dependentes do tempo introduz o conceito de LDU (*Logical Data Units*) [BLA96]. LDUs constituem as unidades em que um componente multimídia pode ser logicamente dividido para fins de sincronização. Para uma mesmo componente, esta divisão pode ser realizada de várias formas. Por exemplo, um vídeo normalmente é sincronizado a nível de quadros. Para efeitos de sincronização, no entanto, a sua LDU pode tanto ter tamanho de 1 quadro ou ser representada por um conjunto de 10 quadros. Como ilustração da sincronização de

elementos dependentes do tempo, poderia-se citar a apresentação de uma seqüência de vídeo onde seja apresentada uma pessoa e um áudio separado representando sua voz; é desejável que o áudio esteja completamente sincronizado com os movimentos bucais da pessoa. Na figura 2.3, é apresentado um esboço da sincronização a nível de LDU. No caso do vídeo, a LDU é representada por um quadro. O áudio, por sua vez, tem sua LDU definida pela unidade *sample*. Este tipo de sincronização é também conhecida como intra-fluxo.

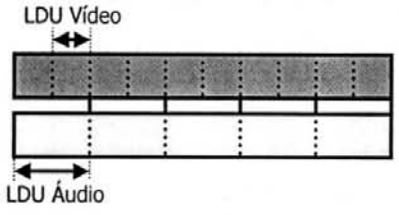


FIGURA 2.3 - Sincronização a nível de LDU

Pode-se ter, ainda, a composição das duas categorias de componentes dispostos na mesma aplicação. A figura 2.4 apresenta um exemplo de aplicação multímidia que agrega elementos dependentes e independentes do tempo. Inicialmente um vídeo e um áudio correspondente são visualizados. A seguir, uma seqüência de três *slides* é apresentada. Por fim, aparece uma animação sendo que, a partir de um certo ponto da mesma, um segundo áudio é executado. Os componentes independentes do tempo podem ser vistos como uma única LDU, passíveis de sincronização com LDUs dos componentes multimídia dependentes do tempo.

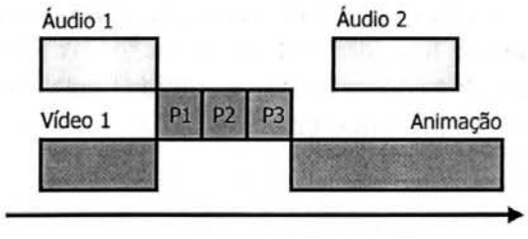


FIGURA 2.4 - Aplicação com diversos elementos multimídia

A sincronização é tratada e manipulada em diversos níveis, abrangendo: sistema operacional, sistema de comunicação, bancos de dados e aplicações. Por esta razão, a literatura [BLA96] propõe um modelo de referência que trata a sincronização em quatro níveis: nível de mídia, de fluxo, de objetos e de especificação (vide figura 2.5). Cada um dos níveis oferece mecanismos de sincronização disponibilizando uma interface apropriada, utilizada diretamente por uma aplicação ou na implementação da interface do nível superior. Níveis mais altos oferecem maiores abstrações em relação à programação e a aspectos relacionados com qualidade de serviço. Os níveis serão explicados com a utilização de um exemplo onde se deseja apresentar legendas em momentos predefinidos da execução de um vídeo. Os fragmentos de código foram extraídos de [BLA96].

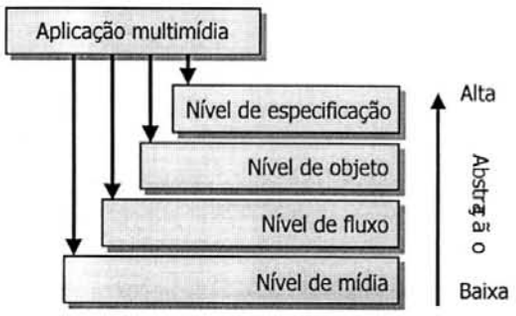


FIGURA 2.5 - Modelo de referência de sincronização

a) Nível de mídia

Este nível é tratado pelo sistema operacional e pelos níveis inferiores dos sistemas de comunicação; se preocupa em manipular fluxos individuais de componentes multimídia com o objetivo de evitar *jitter*[1] durante sua apresentação. Aplicações que operam neste nível manipulam os componentes multimídia contínuos como seqüências de LDUs. A abstração oferecida neste nível se restringe a uma interface independente de dispositivo com operações como *read(devicehandle, LDU)* e *write(devicehandle, LDU)* e é encontrada, por exemplo, em sitemas como o *kernel* de áudio/vídeo *ActionMediaII* ou em dispositivos de áudio *SunSPARC*.

Uma aplicação neste nível, portanto, deve prover mecanismos de sincronização intra-mídia utilizando-se de mecanismos de controle de fluxo entre o dispositivo que está produzindo e o que está consumindo este fluxo. Para configurar um fluxo de mídia contínua a aplicação executa um processo conforme apresentado na figura 2.6. O processo lê e apresenta LDUs, enquanto houver dados a serem lidos do arquivo correspondente ao vídeo. A apresentação das legendas sincronizadas com instantes pré-determinados do mesmo é obtida pela constante verificação de identificadores específicos presentes em cada LDU.

```
window = open ("videodevice");      // Cria uma janela de saída para o vídeo
movie open ("file");                // Abre o arquivo de vídeo
while ( not eof (movie)) {
  read(movie, &ldu);                // Lê uma LDU
  if (ldu.time=20)                  // Inicia a apresentação das legendas sincronizadas
    print ("Subtitle1")
  else
    if (ldu.time=26)
      print ("Subtitle2")
      write (window, ldu);          // Apresenta a LDU
}
close(window)                       // Fecha a janela
close(movie)                        // Fecha o arquivo
```

FIGURA 2.6 - Aplicação utilizando a interface oferecida pelo nível de mídia

[1] O efeito do *jitter* na apresentação de mídias contínuas são intervalos indesejáveis durante sua apresentação.

b) Nível de fluxo

O objetivo deste nível é manter as relações temporais entre vários fluxos de componentes multimídia, respeitando seu entrelaçamento. Neste nível são abordados problemas como garantir a sincronização dos movimentos da boca de um locutor com sua fala, representados respectivamente por um fluxo de vídeo e outro de áudio. Este tipo de sincronização é denominada intra-fluxo. Este nível aborda também a sincronização entre-fluxo. Neste caso, procura-se sincronizar grupos de fluxos entre si.

As operações mais comuns invocadas por aplicações neste nível são *start(stream)*, *stop(stream)*, *create_group(list-of-streams)*, *start(group)* e *stop(group)*. A sincronização intra e entre-fluxo é obtida pela definição de eventos em pontos específicos dos fluxos envolvidos. Durante a execução, quando estes pontos são atingidos, dá-se início a uma ação que normalmente corresponde a apresentação de outro fluxo ou de um componente independente do tempo. Na figura 2.7 a ação associada a cada um dos eventos é a apresentação de legendas (componentes independentes do tempo). Como pode ser observado, LDUs individuais não são mais visíveis.

```
open digital video
alias ex                // Cria descritor do vídeo
load ex video.avs               //Associa arquivo ao descritor do vídeo
set cuepoint ex
  at 20 return 1                // Define evento 1 para legenda 1
set cuepoint ex
  at 26 return 2          // Define evento 2 para legenda 2
set cuepoint ex on              // Ativa eventos cuepoints
play ex                 // Inicia a apresentação
switch readevent() {            // Manipulação de eventos
  case 1: display ("Subtitle1")         // Caso ocorra evento 1, apresenta legenda 1
  case 2: display ("Subtitle2")         // Caso ocorra evento 2, apresenta legenda 2
}
```

FIGURA 2.7 - Aplicação desenvolvida no nível de fluxo

c) Nível de objetos

Suporta a sincronização entre componentes multimídia dependentes e independentes do tempo, bem como a interação do usuário. Ao contrário dos níveis anteriores, permite abstrair a distinção existente entre os tipos de componentes multimídia. Neste nível, a aplicação é vista como uma apresentação onde todas as relações temporais entre os componentes envolvidos já estão definidos; ele simplesmente recebe a especificação do nível superior e é responsável pelo escalonamento da apresentação como um todo. Deve-se ressaltar, ainda, que este nível se propõem a eliminar o *gap* entre as necessidades para execução de uma apresentação sincronizada e serviços orientados a fluxo de mídia. As funções deste nível permitem a apresentação de componentes independentes do tempo. O tratamento dos componentes contínuos é feito, neste nível, com chamada às funções do nível de fluxo.

A figura 2.8 apresenta a aplicação descrita com a utilização da interface do nível de objetos. Na verdade, este é um exemplo de aplicação MHEG. O escopo deste padrão é a representação codificada de aplicações multimídia. Uma implementação possível para o nível de objeto é um MHEG *engine*.

```
Composite {          // Objeto composite
start-up link        // Como iniciar a apresentação
viewer start-up          //Associa arquivo ao descritor do vídeo
viewer-list:
Viewer1:
  reference to Component1          // Vistas virtuais em objetos compostos
Viewer2:
  reference to Component2
Viewer3:
  reference to Component3
Component1           // Objeto do composite
  reference to content "movie.avs"
Component2
  reference to content "subtitle1"
Component3
  reference to content "subtitle2"
Link1                // Relações temporais
  "when timestone status
  of viewer1 becomes 20 then start Viewer2"
Link2
  "when timestone status
  of viewer1 becomes 26 then start Viewer3"
```

FIGURA 2.8 - Aplicação utilizando a interface oferecida pelo nível de mídia

d) Nível de especificação

É um nível aberto, não apresentando uma interface específica. É composto por aplicações e ferramentas que permitam especificar a sincronização entre os componentes de uma apresentação. Exemplos destas ferramentas são editores e sistemas de autoria. Ainda neste nível estão localizadas as ferramentas responsáveis por converter especificações para um formato no nível de objetos. O nível de especificação é também responsável por mapear requisitos de QoS do nível do usuário às qualidades oferecidas na interface do nível de objetos.

Métodos para especificação de sincronização podem ser classificados nas seguintes categorias:

- *Especificações baseadas em intervalos*: permite a especificação de relações entre intervalos de tempo que correspondem à apresentação de componentes multimídia;
- *Especificações baseadas em eixos*: posiciona os componentes de uma aplicação em eixos temporais, determinando automaticamente a ordenação dos mesmos;
- *Especificações baseadas em controle de fluxo*: o fluxo da apresentação é sincronizado com base no estabelecimento de pontos de sincronização;
- *Especificações baseadas em eventos*: eventos na apresentação de componentes disparam ações de apresentação de outros componentes.

O trabalho desenvolvido se enquadra no nível de especificação. Através de um modelo de alto nível, o usuário define a sincronização dos componentes que envolvem a aplicação. Estas especificações são posteriormente mapeadas para o padrão MHEG-5 e apresentadas pelo respectivo *engine*, que representa o nível de objetos. Deste modo, a partir do tópico a seguir enfatiza-se a problemática da sincronização pertinente ao nível de especificação. No próximo item são analisados alguns dos modelos de sincronização apontados acima.

2.1 Modelos de sincronização

O problema de sincronizar a apresentação de dados, a interação do usuário e os dispositivos físicos se resume a satisfazer relações de precedência temporal sobre requisitos reais de tempo. Estas relações podem ser estabelecidas com a utilização de modelos de sincronização que, para atender a definição de sistemas complexos, devem considerar alguns requisitos [COU96]:

- O modelo deve suportar consistência dos objetos (componentes multimídia) e manutenção das especificações de sincronização. Os componentes devem ser considerados como uma unidade lógica na especificação.
- O modelo deve fornecer abstração tal do conteúdo do componente multimídia, de modo que seja possível, de um lado, fazer-se referência a uma parte dele e, por outro, encará-lo como uma única unidade lógica.
- Deve ser possível representar sincronização baseada em interações do usuário.
- Todos os tipos de relações de sincronização devem ser facilmente descritos.
- Deve ser possível integrar componentes dependentes e independentes do tempo.
- A definição de requisitos de QoS deve ser suportada.
- Níveis hierárquicos da sincronização devem ser suportados para habilitar a manipulação de cenários de sincronização grandes e complexos.

Existem vários métodos e modelos que objetivam permitir a especificação de sincronização em aplicações multimídia que são classificados conforme foi exposto no final da seção anterior e que serão apresentados a seguir.

2.1.1 Modelo baseado em intervalos

A concepção de um modelo baseado em intervalos leva em consideração a duração da apresentação de um componente multimídia que é denominada *intervalo* [BUF94]. Com base neste conceito, dois componentes quaisquer podem ser sincronizados de treze maneiras diferentes. Estas relações podem ser reunidas em apenas sete casos conforme apresentado na figura 2.9, uma vez que seis destes casos correspondem às relações inversas. Por exemplo, *after* é a relação inversa de *before*.

Apresentações multimídia simples podem ser completamente definidas a partir destas sete relações [WAH94]. A necessidade de expressar sincronizações mais complexas, no entanto, resultou na definição de um modelo estendido, capaz de representar vinte e nove relações distintas. Estas são definidas como disjunções das relações básicas. Para simplificar o processo de especificação, foram definidos dez operadores para a manipulação destas relações (vide figura 2.10). A concatenação das vinte e nove relações em apenas dez operadores é viabilizada pelo fato de vários casos poderem ser combinados em uma única operação, analogamente ao que ocorre no modelo original.

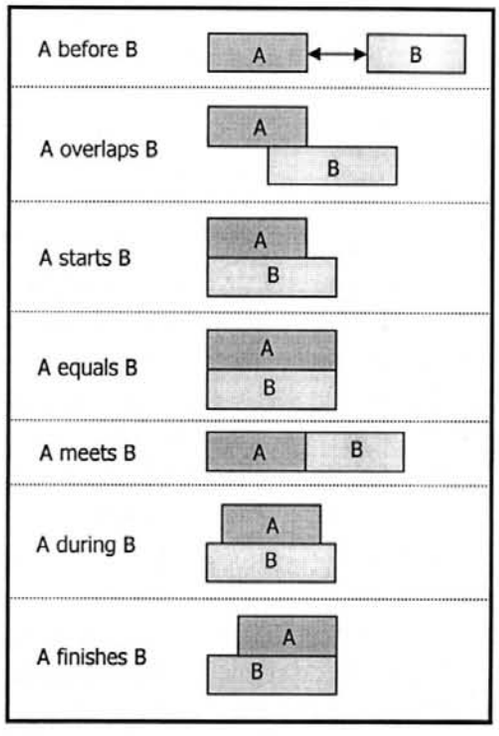


FIGURA 2.9 - Relações temporais entre dois objetos

A duração da apresentação de A ou B, bem como do atraso δ_1, pode não ser conhecida no momento da especificação. Este é o caso, por exemplo, quando se deseja representar a interação de um usuário, onde não seja estabelecido o tempo máximo de espera (atraso) para sua ação. Além disso, deve-se ressaltar que as operações *beforeendof*, *delayed*, *startin*, *endin*, *cross* e *overlaps* não podem ter o atraso δ_1 igual a zero, pois representariam relações temporalmente incoerentes.

Alguns exemplos de como representar aplicações multimídia utilizando este modelo são apresentados na figura 2.11. A apresentação de uma seqüência de três *slides* simultaneamente a um áudio pode ser representada conforme apresentado na figura 2.11a. A sincronização fina ou *bucal* é apresentada na figura 2.11b. Apresenta-se ainda uma aplicação que integra componentes dependentes e independentes do tempo (figura 2.11c); esta última aplicação é a mesma apresentada na figura 2.4.

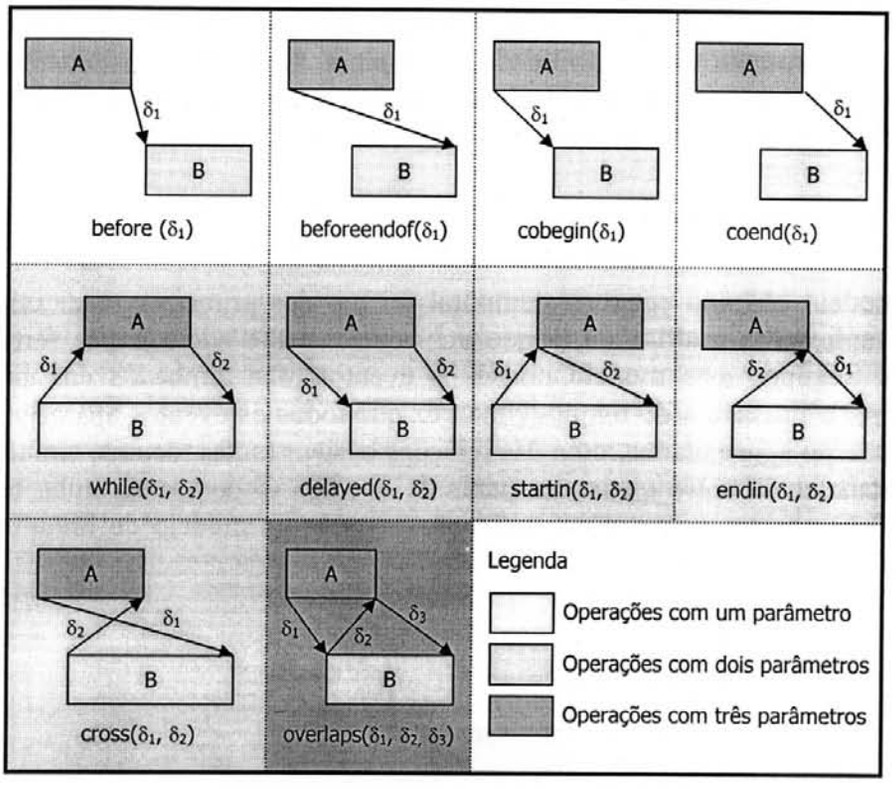


FIGURA 2.10 - Operações da extensão do modelo baseado em intervalos

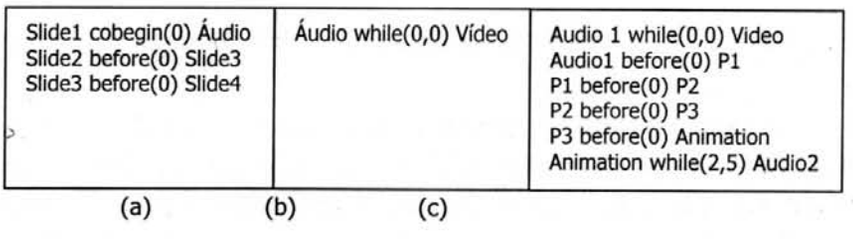


FIGURA 2.11 - Especificações segundo modelo baseado em intervalos

O modelo baseado em intervalos é bastante flexível, apresentando um elevado grau de expressividade. A combinação dos operadores pode resultar na criação de novas relações. Esta maleabilidade do modelo, contudo, pode ser fonte de especificações incoerentes. Considere-se, por exemplo, dois vídeos, A e B, relacionados com o operador *not parallel*. A apresentação de B, no entanto, está associada também com uma interação do usuário através de uma relação *before(0)*. No momento da execução desta aplicação, A está sendo apresentado; quando a interação do usuário é realizada, o vídeo B está apto para também ser apresentado satisfazendo a relação *before(0)* mas, em contrapartida, não pode ser iniciado pois infrigiria a relação *not parallel* estabelecida com A. Por esta razão, faz-se necessária a aplicação de métodos que permitam tratar estas inconsistências durante ou previamente à execução da aplicação.

Não é possível especificar diretamente relações entre sub-unidades de componentes multimídia. Estas relações são obtidas com a utilização de *delays*, como mostrado no exemplo anterior (figura 2.11b) ou através da divisão dos objetos em segmentos menores.

Pode ser bastante difícil entender as especificações criadas, dependendo de sua complexidade. Especificações maiores se utilizam de um elevado número de relações, o que compromete sua legibilidade e propicia a criação de incoerências. Estes problemas ficam bastante evidenciados também pela ausência de uma forma para representar graficamente as especificações.

2.1.2 Modelo baseado em linha temporal

O modelo baseado em linha temporal foi um dos primeiros a ser utilizado para definir aplicações multimídia, sendo até hoje bastante adotado por ferramentas de autoria. Segundo este modelo, todos os eventos[2] são alinhados em um eixo que representa a passagem do tempo. Uma vez que todos os eventos aparecem na ordem em que serão apresentados, uma das relações básicas (antes, depois, simultaneamente a) é estabelecida entre quaisquer pares de eventos dispostos na linha temporal. A figura 2.12 abaixo, ilustra a especificação baseada no modelo de linha temporal da descrição apresentada na figura 2.4.

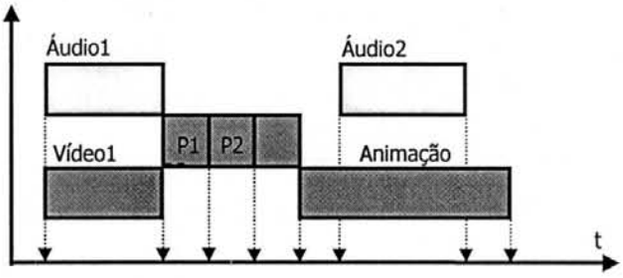


FIGURA 2.12 - Especificação segundo modelo baseado em linha temporal

Este modelo é de fácil entendimento, provendo um alto grau de abstração das especificações. É, também, de fácil manutenção pela independência mútua dos objetos que compõem uma especificação; a exclusão de um componente multimídia da linha temporal não interfere na sincronização estabelecida entre as que restaram nela. Embora simples e bastante intuitivo, é deficiente para representar relações como interação do usuário e apresentação de componentes cuja duração seja defina por um intervalo como as dependentes do tempo, por exemplo.

Esta deficiência pode ser ilustrada em uma situação onde uma imagem deva ser apresentada até que ocorra uma interação do usuário, quando a apresentação de uma outra imagem é realizada. O instante inicial da apresentação da imagem é conhecido durante o processo de especificação da aplicação. Seu instante final, contudo, depende da interação e, conseqüentemente, não pode ser previsto. Verifica-se, assim, a impossibilidade de modelar este cenário utilizando esta abordagem tradicional.

Existem várias propostas de extensão para este modelo, a fim de permitir a especificação de cenários interativos [HIR95]. Estas, no entanto, imprimem um razoável grau de complexidade nas especificações, fazendo com que o modelo perca sua característica mais importante que é a simplicidade de compreensão do comportamento temporal da aplicação.

---

[2] Pontos nos quais é possível sincronizar a apresentação de mídias. Normalmente são considerados eventos o início e fim de sua apresentação.

2.1.3 Modelos baseados em fluxo de controle

Nas especificações baseadas em fluxo de controle, o fluxo da apresentação de threads concorrentes é sincronizado em pontos pré-definidos da aplicação. Os principais modelos baseados nesta abordagem são o hierárquico, pontos de referência e redes de Petri temporizadas.

Modelo hierárquico

O modelo hierárquico é baseado principalmente nas operações seqüência e paralelo. A especificação é vista como uma árvore, onde os nodos internos representam um operador de apresentação serial, paralela ou algum outro tipo de sincronização. Este operador indica como será tratada a sub-árvore que dele se origina. Uma ação pode ser atômica ou composta. Uma ação atômica manipula a apresentação de um componente multimídia específico (vídeo, som, imagem) ou de uma interação, enquanto que ações compostas são representadas pela combinação de operadores de sincronização e ações atômicas. A figura 2.11 ilustra a especificação apresentada na figura 2.4, agora representada segundo o modelo hierárquico.

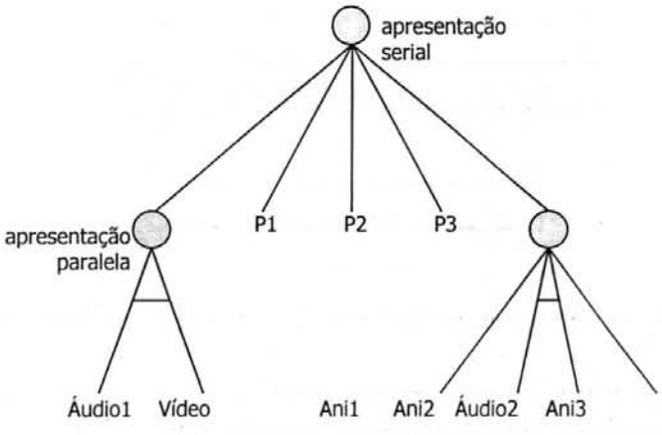


FIGURA 2.13 - Especificação segundo modelo hierárquico

A introdução de atrasos como possíveis ações permitem a modelagem de um número maior de comportamentos. Como exemplo, pode-se citar a modelagem de atrasos em apresentações seriais e a apresentação postergada de objetos em sincronizações paralelas.

As estruturas hierárquicas são facilmente manipuláveis, uma vez que permitem dividir a especificação em diversos níveis. Esta é uma característica desejável para facilitar a modelagem de especificações muito grandes. Por outro lado, ao contrário do modelo baseado em linha temporal, as relações temporais não são representadas de forma natural, sendo difícil perceber a ordem em que os componentes serão apresentados.

A utilização da estrutura hierárquica restringe a sincronização de ações a seu início ou seu fim. Isto significa, por exemplo, que a apresentação de legendas durante alguns momentos de um vídeo requer que o mesmo seja dividido em diversos componentes consecutivos, definindo uma ação composta. Como pode ser visualizado na figura 2.13, a animação é dividia em três partes a fim de que possa ser sincronizada com o áudio. Esta é uma desvantagem do modelo; o objeto animação não é mais visto

como uma unidade abstrata única. Caso fosse desejado sincronizá-lo com algum outro objeto, já não seria possível estabelecer uma relação que levasse em conta seu início e fim, pois agora seu início e final origininais estão associados a objetos distintos: Ani1 e Ani3, respectivamente.

Modelo baseado em pontos de referência

Na sincronização baseada em pontos de referência os diferentes objetos são considerados seqüências de LDUs. Os tempos de início e fim da apresentação de um objeto, bem como os tempos de início de cada LDU são denominados pontos de referência. A sincronização entre dois objetos quaisquer é obtida pela conexão de pontos de referência de ambos os objetos (vide figura 2.14); estas conexões são denominadas pontos de sincronização. A apresentação dos sub-objetos associados a um ponto de sincronização deve ser feita quando estes pontos forem atingidos; sendo assim, as relações temporais são estabelecidas sem referência explícita ao tempo.

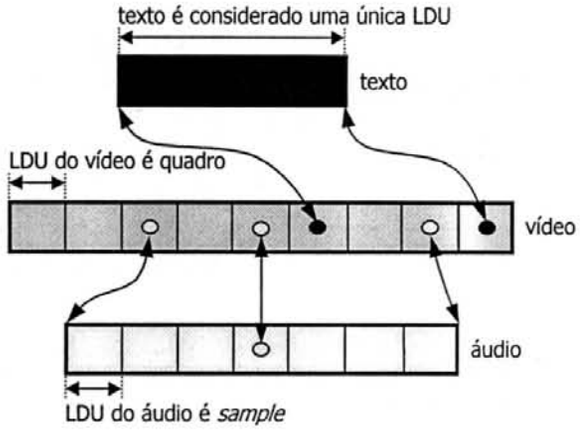


FIGURA 2.14 - Sincronização no modelo baseado em pontos de referência

Da mesma forma como o modelo baseado em linha do tempo, a sincronização pode ocorrer a qualquer instante da apresentação destes objetos. O modelo suporta a sincronização em diversos níveis de granularidade. A sincronização fina, por exemplo, é obtida com a utilização de um elevado número de pontos de sincronização. A especificação de uma aplicação onde um vídeo esteja sincronizado com um áudio representando a fala do locutor que aparece neste vídeo requer um alto grau de precisão. Neste caso, poderia-se estabelecer pontos de sincronização a cada quadro do vídeo. Se este tipo de sincronização não fosse necessário, contudo, poderia-se configurar pontos de sincronização a cada dez quadros, por exemplo. Como ilustração deste modelo tem-se, na figura 2.15, a aplicação descrita previamente na figura 2.4.

Uma desvantagem deste modelo é a necessidade de haver mecanismos para detectar inconsistências. Além disso, a sincronização baseada em pontos de referência não permite a especificação de atrasos nas apresentações. Este problema, entretanto, é abordado em várias propostas. Uma delas apresenta um operador explicito para representar atrasos intencionais.

O modelo pontos de referência possibilita a criação de hierarquias. Isto poderia ser feito simplesmente considerando um conjunto de objetos sincronizados como um objeto único, considerando o início do primeiro objeto e o final do último como pontos de referência.

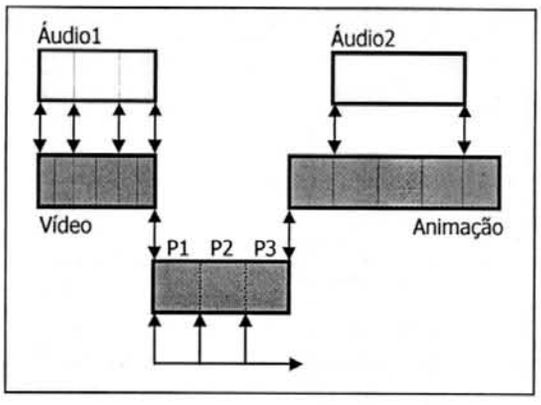


FIGURA 2.15 - Especificação segundo modelo pontos de referência

Redes de Petri temporizadas

Redes de Petri permitem modelar naturalmente e de forma hierárquica cenários multimídia interativos. Os lugares expressam o processamento de cada unidade de informação e as transições especificam a ordem lógica de apresentação destas unidades. A informação pode ser expressa em diferentes níveis de granularidade. Um vídeo, por exemplo, pode ser modelado por um único lugar ou por vários lugares (um para cada quadro ou para cada dez quadros), dependendo do tipo de sincronização que se deseje estabelecer. O comportamento dinâmico da aplicação é modelado pelo progresso da ficha pelos lugares.

A semântica das redes de Petri foi originalmente estendida para viabilizar a representação de cenários multimídia. Incorporou-se ao modelo o conceito de duração para os lugares. Resumidamente, as regras para a evolução destas redes são as seguintes:

- Uma ficha adicionada a um novo lugar é bloqueada pelo tempo determinado (duração da apresentação do componente multimídia ou LDU) para aquele lugar.
- Uma transição é disparada se todos os lugares de entrada possuem uma ficha e todos já tiverem sido desbloqueados.
- Se uma transição é disparada, a ficha é removida do lugar de entrada e posicionada no lugar de saída.

Em uma das propostas de redes de Petri temporizadas, o OCPN (*Object Composition Petri Nets*), cenários são modelados de acordo com uma abordagem de estruturação baseada na utilização recursiva das sete relações fundamentais de intervalos apresentadas anterioremente na seção 2.1.1. A figura 2.16 apresenta a representação em OCPN destas relações.

O modelo TSPN (*Temporal Streams Petri Nets*) [DIA93] estende ainda mais a funcionalidade das redes, incorporando o conceito de intervalo de validação temporal (IVT) aos arcos. A duração de um componente multimídia passa a ser representada não mais por um valor exato associado a um lugar, mas por três valores distintos associados ao arco que o sucede, representando respectivamente sua duração mínima, nominal e máxima de apresentação. A semântica associada ao TSPN considera, assim, as variações temporais possíveis em ambientes multimídia com recursos distribuídos.

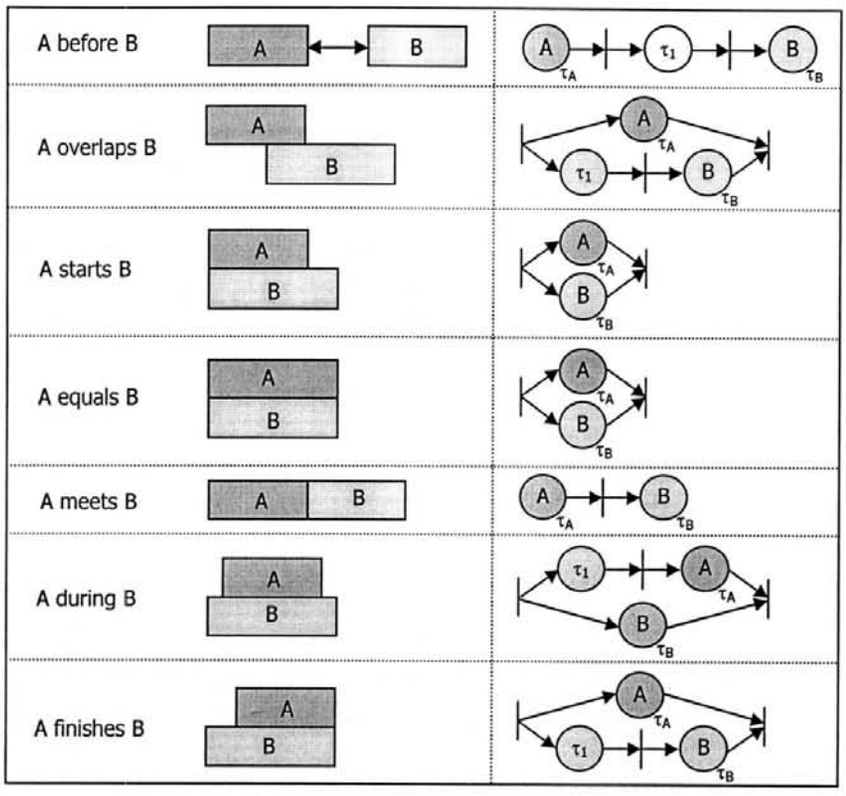
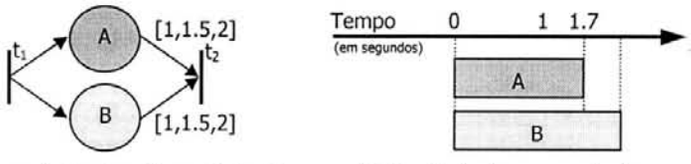


FIGURA 2.16 - Relações de intervalos representadas em OCPN

Pelo fato de o modelo possibilitar a representação de tempos de duração variáveis, indicados pelo IVT, a definição de requisitos estreitos de sincronização pode, em muitos casos, não ser garantida no momento da execução. A figura 2.17a ilustra uma aplicação onde dois componentes multimídia, A e B, devem ser apresentados simultaneamente. Sua apresentação deverá durar no mínimo 1 e no máximo 2 segundos, sendo que 1.5 segundos corresponde à duração ideal. O efeito resultante da apresentação (figura 2.17b) mostra que B continuou sendo apresentado durante 0.7 segundos após o final da apresentação de A. Possivelmente a rede interligando o ambiente de execução e o local onde o componente multimídia B está armazenado estava congestionada naquele instante, impedindo que ele fosse apresentado à sua taxa nominal.



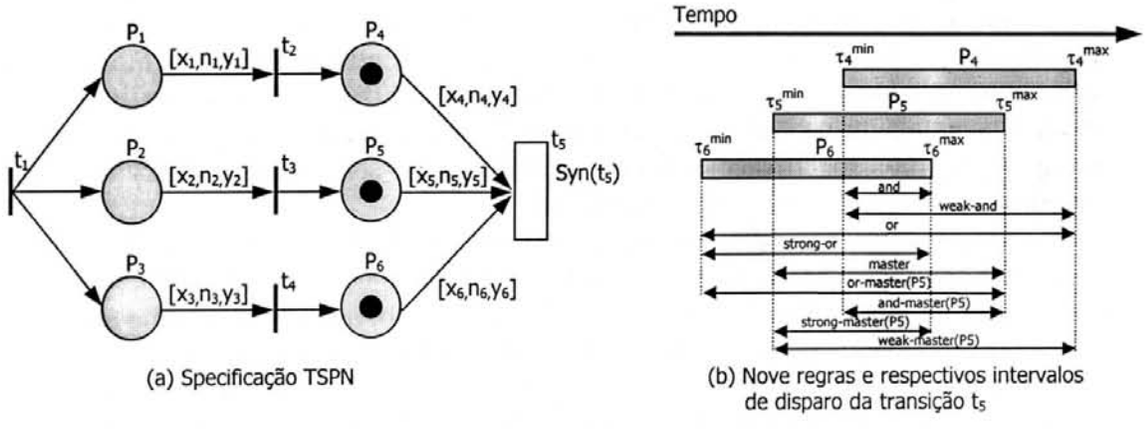
(a) Apresentação simultânea de A e B (b) Resultado da apresentação

FIGURA 2.17 - Variações temporais na apresentação de componentes multimídia

Em alguns casos o ocorrido pode ser perfeitamente aceitável; em outros, poderia ser encarado como uma violação da semântica temporal desejada. Considerando estes aspectos, o modelo TSPN permite associar diferentes regras de sincronização às transições preservando a semântica temporal de alguns arcos privilegiados de acordo com as seguintes estratégias:

- *Estratégia de sincronização dinâmica strong-or*: baseada no primeiro arco a atingir sua duração máxima admissível. Esta estratégia garante a correção temporal de pelo menos um arco, o primeiro a atingir seu respectivo tempo máximo. Do ponto de vista da execução, esta sincronização pode envolver a terminação forçada de processos (componentes multimídia) cujas apresentações estejam atrasadas. A utilização desta estratégia na transição t_2 do exemplo anterior permitiria o disparo da transição tão logo o primeiro elemento, fosse ele A ou B, concluísse sua apresentação, causando a imediata interrupção da apresentação de outro componente multimídia.
- *Estratégia de sincronização dinâmica weak-end:* baseada no último arco, ou seja, no último a alcançar sua duração máxima admissível, garantindo sua correção temporal. Do ponto de vista da execução, esta sincronização pode envolver o atraso proposital de processos que estejam sendo apresentados normalmente. A transição t_2 regida por esta estratégia só seria disparada ao final da apresentação de A e B.
- *Estratégia estática:* baseada em um arco pré-determinado, normalmente denominado mestre. Durante a execução, a sincronização mestre pode envolver o atraso proposital de processos que terminem antes dele e a terminação abrupta de processos em andamento no momento em que o arco mestre torna possível a evolução. Se no exemplo anterior fosse escolhido o componente A como mestre, então a transição t_2 seria disparada imediatamente após a apresentação de A, acarretando a interrupção abrupta de B caso sua apresentação não tivesse sido concluída.

Estas três estratégias de sincronização originam nove regras, obtidas pela combinação de intervalos de validação temporal dos arcos de entrada de uma transição (vide figura 2.18).



(a) Specificação TSPN

(b) Nove regras e respectivos intervalos de disparo da transição t_5

FIGURA 2.18 - Semântica de sincronização do modelo TSPN

Vale lembrar ainda o modelo HTSPN (*Hierarchical Time Streams Petri Nets*) [SEN95]. Este modelo aproveita as características do TSPN e agrega a ele mecanismos para a estruturação lógica dos cenários multimídia. As especificações são hierarquizadas, sendo que os níveis superiores provêm uma visão geral da aplicação e os níveis inferiores oferecem detalhamento sobre partes específicas da mesma. Três níveis distintos podem ser observados:

- *Nível de sincronização lógica*: dedicado à especificação dos caminhos possíveis para a aplicação. Esta especificação é realizada com base em uma rede TSPN onde os lugares representam geralmente composições e os arcos a ordem em que elas são apresentadas. Na prática, as composições podem ser vistas como cenários da aplicação.
- *Nível de sincronização composição*: utilizado para especificar relações temporais entre os elementos que fazem parte do cenário (sincronização entre mídias).
- *Nível de sincronização atômica*: utilizado para modelar sincronização intra-mídia.

A figura 2.19 ilustra a especificação apresentada na figura 2.4, agora representada segundo o modelo HTSPN.

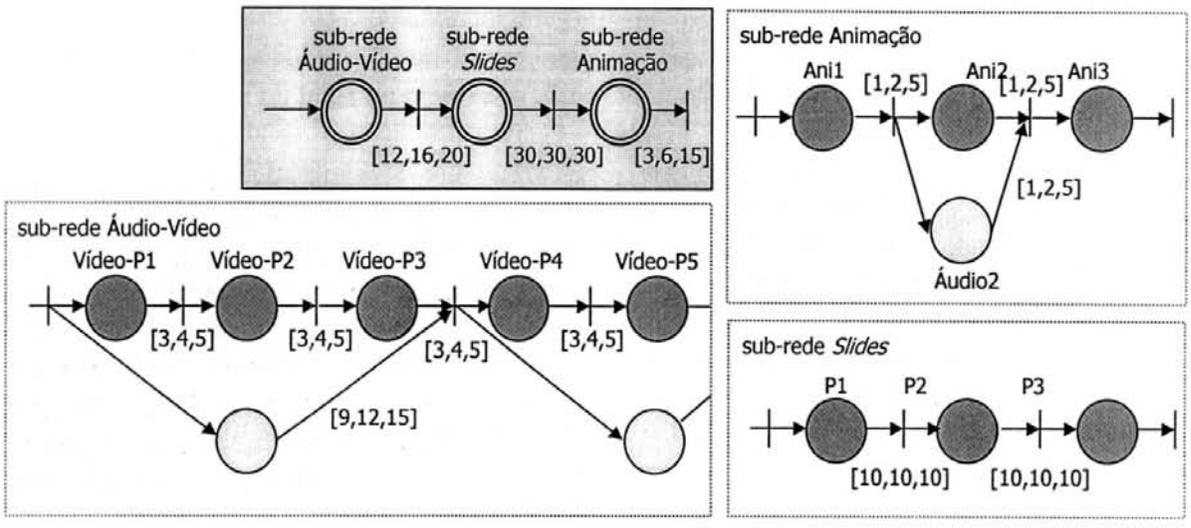


FIGURA 2.19 - Aplicação representada no modelo HTSPN

As redes de Petri permitem especificar todos os tipos de sincronização importantes na modelagem de cenários multimídia interativos. Dentre os modelos até agora apresentados, este é o que provê o maior poder de expressão e flexibilidade. Sua maior desvantagem, contudo, é a complexidade; em se tratando de redes de Petri, a manipulação de especificações muito grandes pode ser bastante dificultada, em virtude do problema da explosão de estados.

Os modelos baseados em redes de Petri são bastante adequados tanto para modelar a interação do usuário como atrasos intencionais. Em algumas abordagens desta classe de modelos como o OCPN e TSPN, contudo, não se obtém uma abstração suficiente dos objetos uma vez que, para fins de sincronização, muitos deles precisam ser divididos em sub-unidades. No HTSPN este problema é resolvido graças ao mecanismo de hierarquização das especificações. Na figura 2.19, por exemplo, a sub-rede Áudio-Vídeo modela a sincronização de um vídeo com um áudio, sendo que ambos os componentes são segmentados para que o relaciomanento temporal entre eles possa ser definido. Para o restante da aplicação, no entanto, o vídeo é tratado como uma única unidade abstrata; na sub-rede principal, ele é representado por um único lugar.

2.1.4 Sincronização baseada em eventos

Na sincronização baseada em eventos, ações são realizadas com base na ocorrência de eventos de sincronização. Algumas ações possíveis são o início e o

encerramento da apresentação de um componente multimídia, por exemplo. Os eventos que habilitam estas ações podem ser externos (gerados por um temporizador ou por uma interação do usuário) ou internos à apresentação (gerados por um componente multimídia dependente do tempo ao atingir uma LDU específica). Este modelo de especificação é facilmente estendido através da criação de novos eventos [BLA96].

Uma desvantagem deste tipo de sincronização é a dificuldade para manipular cenários grandes. O usuário invariavelmente se perde ao tentar manipular as especificações, tornando sua criação e manutenção bastante complicadas. O padrão MHEG-5 para representação de objetos multimídia é baseado neste modelo de sincronização.

2.2 O padrão MHEG-5

O padrão MHEG-5 foi desenvolvido para suportar a distribuição de aplicações multimídia interativas através de uma arquitetura cliente/servidor em plataformas heterogêneas. Seu objetivo é definir a sintaxe e a semântica de um conjunto de classes a fim de que objetos possam ser compartilhados entre diversas aplicações sem haver uma preocupação com a plataforma ou sistema operacional utilizado. O intercâmbio de objetos MHEG pode ocorrer através de uma rede ou através da utilização de dispositivos de armazenamento. O módulo responsável pelo intercâmbio dos objetos faz parte do *engine* MHEG; este módulo tem por objetivo interpretar estes objetos, representados segundo um formato interno, para uma estrutura ASN.1 e vice-versa [ISO96].

A figura 2.20 ilustra como é realizado o intercâmbio entre dois sistemas MHEG sendo executados sob plataformas heterogêneas. Quando o sistema B solicita um objeto MHEG ao sistema A, o codificador MHEG do sistema A converte as informações de um formato interno compatível com A para o formato definido pelo MHEG, de acordo com a linguagem definida para o intercâmbio. Ao receber o objeto, o decodificador MHEG do sistema B interpreta as informações contidas nesse objeto, convertendo-as para um formato compatível com B.

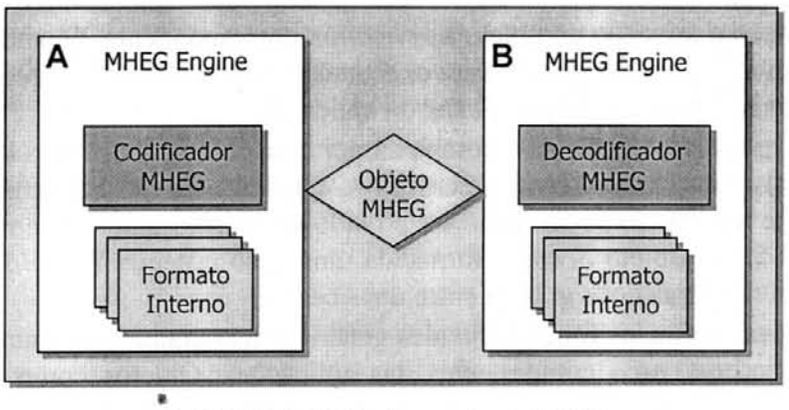


FIGURA 2.20 - O *engine* MHEG

Aplicações MHEG-5 são definidas como uma hierarquia de classes de acordo com uma abordagem orientada a objetos. Em linhas gerais, uma aplicação é constituída de objetos *Scene*, que representam os diversos cenários que a constituem. Cada uma das cenas, por sua vez, é composta de um grupo de *Ingredients*, que podem

tanto representar informação como gráfico, som e vídeo, quanto *links* e ações. Estes últimos são responsáveis pelo controle da apresentação de acordo com a ocorrência de eventos internos (determinado instante da apresentação atingido) ou externos (pressionamento de um botão, por exemplo). A navegação em uma aplicação é feita através de transições entre cenas. A forma de representação das aplicações MHEG-5 é textual. A figura 2.21 abaixo, contudo, apresenta um esquema gráfico meramente ilustrativo da estrutura básica das aplicações MHEG-5.

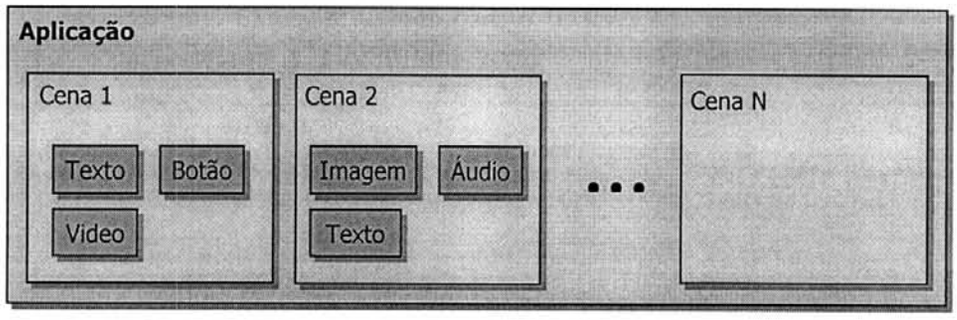


FIGURA 2.21 - Estrutura básica das aplicações MHEG-5

Conforme já mencionado, o padrão se utiliza de várias classes para a definição completa de uma aplicação multimídia [ISO96]. A figura 2.22 apresenta a hierarquia estabelecida entre elas. As principais classes são:

- *Root*: esta é a classe base abstrata para a maioria das classes MHEG-5. Sua principal funcionalidade é prover semântica para comportamentos genéricos MHEG-5 (ativação, desativação, preparação e destruição de objetos). A ativação de um objeto consiste da realização de uma série de ações particulares ao tipo do objeto. A ativação de um objeto *Visible*, por exemplo, resultaria no início de sua apresentação. A classe *Root* fornece ainda mecanismos para identificação dos objetos.
- *Application*: esta classe é composta por grupos de objetos da classe *Ingredient*. Somente um objeto *Application* pode ser ativado a cada vez pelo *engine* MHEG e, ainda, outros objetos não podem ser ativados enquanto não houver um objeto *Application* ativo. Um *engine* MHEG-5 ocioso inicia uma aplicação pela preparação e ativação do objeto *Application* correspondente. Quando este objeto se torna ativo, automaticamente será executada a ação *OnStartUp*, que pode ser usada para ativar o primeiro objeto *Scene* da aplicação.
- *Scene*: esta classe também é composta por grupos de objetos da classe *Ingredient*. O objetivo da classe *Scene* é permitir a apresentação de um cenário multimídia. Somente um objeto *Scene* pode ser ativado ao mesmo tempo dentro de um *engine* MHEG-5. Ao objeto *Scene* é fornecida uma ação especial *TransitionTo* que torna possível uma transição gráfica entre duas cenas.
- *Ingredient*: define as funcionalidades genéricas associadas aos objetos que são, por fim, descritos como componentes das aplicações. Objetos como *Bitmap, Video, Text, Button, Audio, Pallete, Font, CursorShape* são instanciados de classes que herdam a classe *Ingredient*. Esta classe define atributos usados para determinar se um objeto deve ou não ser iniciado já no estado ativo, seu formato de codificação, seu conteúdo e tamanho, e se o objeto deve se manter em apresentação durante transições de cenas.
- *Links*: a dinâmica das aplicações está embutida em seus objetos *Link*, os quais são constituídos de duas partes: uma condição e um objeto *Action*. Quando a condição

torna-se verdadeira, significando que ocorreu o evento especificado, o *link* é disparado e o objeto *Action* correspondente é ativado. Eventos podem representar a mudança de estado de objetos ou seus atributos como *IsAvailable* e *IsRunning*, ou podem ser gerados externamente, como *UserInput*, *NewChar* e *TimerFired*, por exemplo.

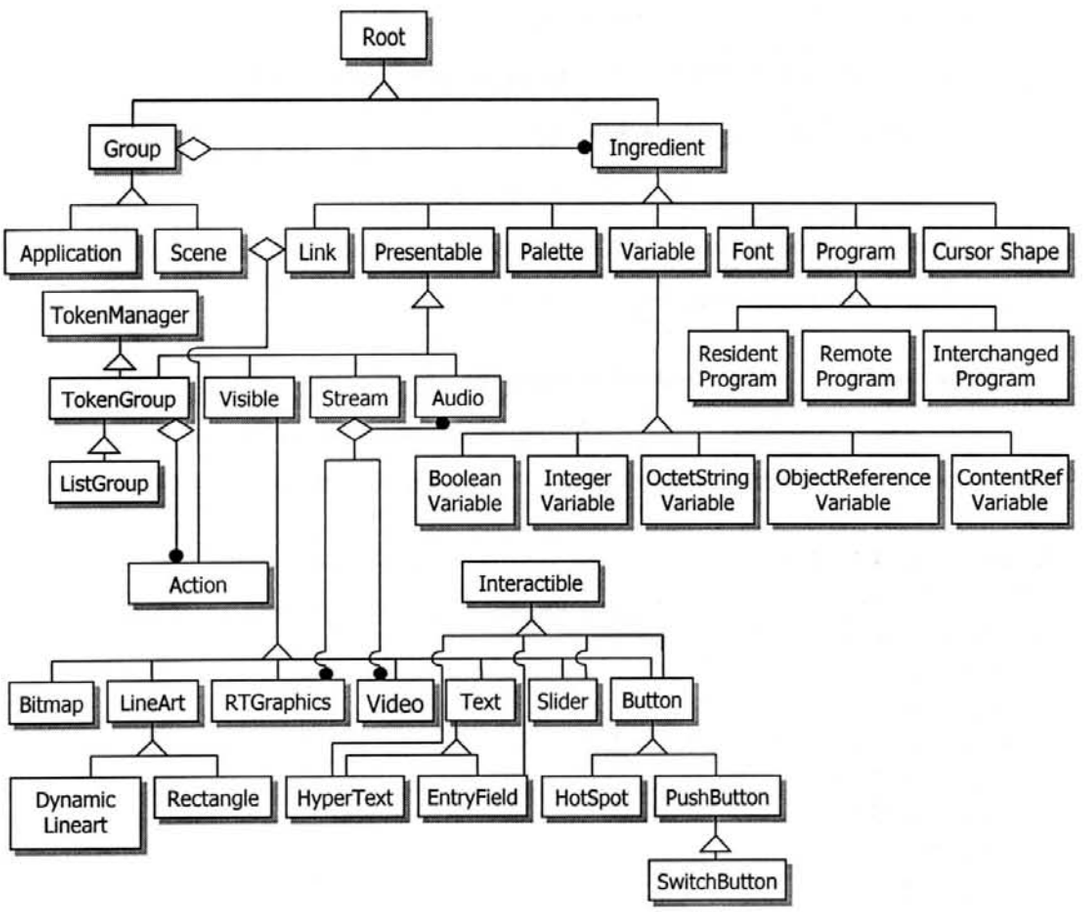


FIGURA 2.22 - Hierarquia de classes do padrão MHEG-5

A figura 2.23 modela um cenário multimídia bem simples que apresenta uma imagem e um texto. O usuário, ao pressionar o botão esquerdo do mouse,dispara uma transição da cena atual (InfoScene1) para InfoScene2.

```
(scene:InfoScene1                    //Definição do objeto Scene
  <other scene attributes here>
  group-items:
    (bitmap: BgndInfo                //Definição do objeto Bitmap
      content-hook: #bitmapHook
      original-box-size: (320 240)        //Características de apresentação da imagem
      original-position: (0  0)
      content-data: referenced-content: "InfoBngd"
    )
    (text:                           //Definição do objeto Text
      content-hook: #textHook
      original-box-size: (280 20)        //Características de apresentação do texto
      original-position: (40 50)
      content-data: included-content: "1. Lubricate..."
    )
  links:
    (link: Link1                     //Definição do objeto Link
      event-source: InfoScene1
      event-type: #UserInput                //Disparado pela interação do usuário
      event-data: #Left
      link-effect: action: transition-to: InfoScene2        //Efeito: transição para InfoScene2
    )
)
```

FIGURA 2.23 - Definição de um cenário simples em MHEG-5

3 Autoria de Aplicações Multimídia e Ferramentas

A autoria de aplicações multimídia normalmente é realizada mediante a utilização de ferramentas que provêm ao usuário mecanismos que permitem agrupar elementos como gráficos, textos, vídeos, sons e animações. Embora estes sistemas possam ser completamente definidos e implementados a partir de linguagens de programação ditas de baixo nível como C e Pascal, este processo é cada vez mais realizado com a utilização de sistemas de autoria. Estes oferecem inúmeras facilidades ao usuário, tornando o processo de definição simples e rápido. Mais do que isso, possibilitam que indivíduos que não tenham profundo conhecimento em informática possam criar suas especificações naturalmente.

Os sistemas de autoria devem fornecer funcionalidades que permitam, sobretudo, o estabelecimento de relações temporais entre os elementos que fazem parte da aplicação. Para tal, estes sistemas normalmente suportam um modelo de sincronização como os apresentados na seção anterior. A adoção de um destes modelos influencia diretamente o paradigma sob o qual as aplicações serão especificadas. Em [BUL95] são apresentados os quatro principais. São eles:

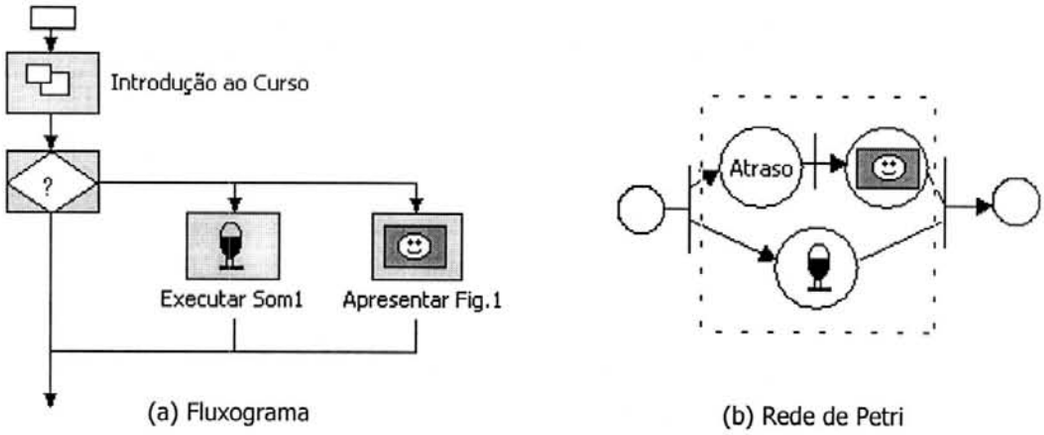
- *Autoria baseada em grafos*, onde diagramas descrevem o fluxo da apresentação;
- *Autoria baseada em linha do tempo*, onde o fluxo da apresentação é definido com base em um eixo temporal;
- *Autoria baseada em scripts*, que se utiliza de uma representação textual para descrever a posição e a temporização de objetos multimídia;
- *Autoria baseada em estruturas*, que agrupa objetos com base no contexto da apresentação.

3.1 Sistemas de autoria baseados em grafos

Esta abordagem se utiliza de diagramas para representar o fluxo e as relações dos objetos multimídia que fazem parte da aplicação. Os grafos podem ser descritos formal ou informalmente. Grafos informais (figura 3.1a) são utilizados unicamente para prover uma visão global do ordenamento dos objetos. Grafos formais (figura 3.1b), por sua vez, também ilustram as interações entre objetos mas viabilizam, adicionalmente, a aplicação de métodos de análise que possibilitam provar ou determinar propriedades dos comportamentos especificados.

A autoria baseada em grafos pode prover especificações poderosas. Ao mesmo tempo, especificações complexas, sejam elas baseadas em grafos formais ou não, acabam sofrendo do problema da explosão de estados, onde as especificações tendem a ficar ilegíveis com o aumento do número de objetos e das relações estabelecidas entre eles. Para gerenciar esta complexidade, interfaces sofisticadas podem ser apresentadas ao usuário, contendo funcionalidades como *zoom-in* e *zoom-out,* permitindo a ele ter uma visão abrangente de toda a aplicação ou de um segmento específico da mesma.

Mesmo com estas funcionalidades, a utilização desta abordagem é limitada, especialmente no caso de grafos informais. Aplicações grandes acabarão sendo representadas em diversas páginas ou telas e invariavelmente a visão integrada será perdida ou extremamente prejudicada. Este problema é ainda maior em sistemas de autoria baseados em grafos formais, em virtude da complexidade inerente aos formalismos.



(a) Fluxograma (b) Rede de Petri

FIGURA 3.1 - Grafos informais e formais

Existem várias ferramentas de autoria que operam com base neste paradigma. IconAuthor é um sistema baseado em grafos informais como os apresentados na figura 3.1a. O editor I-HTSPN, por outro lado, basea-se em redes de Petri. A seguir são apresentadas estas ferramentas.

3.1.1 IconAuthor

A ferramenta IconAuthor opera com uma linguagem visual baseada em ícones [KOE92]. Estes, quando interligados, determinam o fluxo de controle da aplicação. A figura 3.2 apresenta a interface gráfica do IconAuthor. À esquerda, pode-se observar a barra de ícones; para adicionar um novo ícone à aplicação basta arrastá-lo da barra para a área de especificação, que pode ser visualizada à direita na mesma figura. Na realidade, o processo de autoria é similar aos procedimentos realizados em uma linguagem de programação convencional, com o adicional de apresentar uma sintaxe gráfica.

Cada ícone tem uma função única bem determinada. No entanto, graças a existência de ícones de composição, é possível criar coleções reusáveis deles. Pela funcionalidade que desempenham, podem ser enquadrados nas seguintes categorias:

- *Fluxo*: controlam o fluxo de execução da aplicação, indicando os caminhos pelos quais ela seguirá. Nesta classe de ícones, encontram-se os de decisão, laço, menu.
- *Interação*: ícones especiais, cuja função é viabilizar a interação do usuário. Podem representar, por exemplo, entrada de dados via teclado ou *mouse*.
- *Saída*: controlam as saídas geradas pela aplicação, podendo ser informações gráficas ou textuais disponibilizadas na tela ou em outro dispositivo qualquer.
- *Multimídia*: permitem manipular diversos tipos de componentes multimídia como vídeo, imagem, texto e som.

Como pode ser observado, a ferramenta dispõe de um elevado número de ícones. Contudo, por representarem uma única funcionalidade e sendo ela bem definida, torna-se fácil trabalhar normalmente com eles. Por outro lado, pela natureza primitiva das funções a eles associadas, os grafos tendem a ser grandes e de difícil entendimento e manutenção.

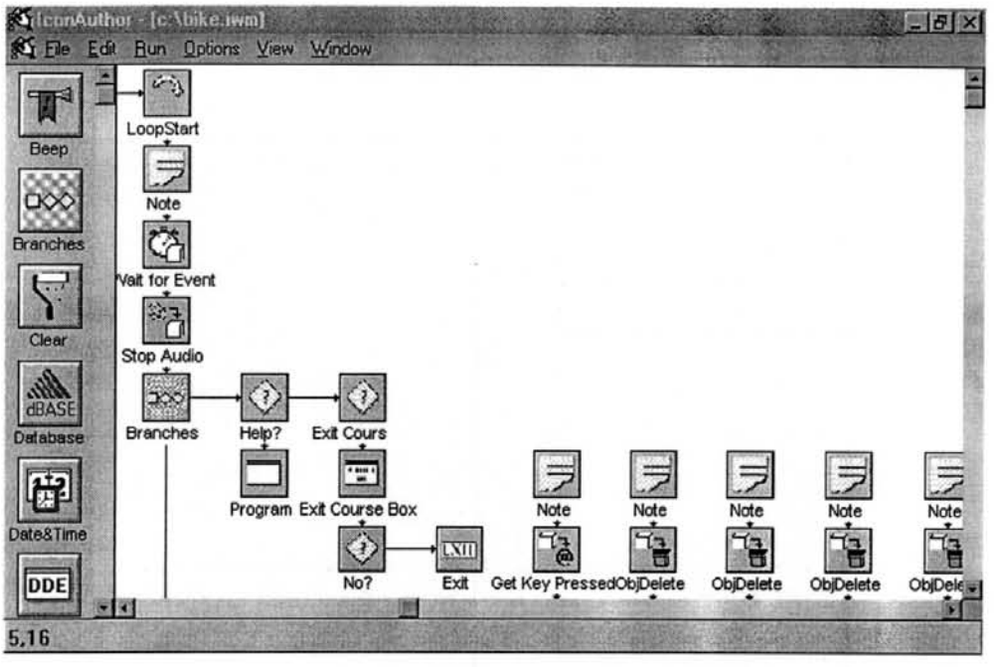


FIGURA 3.2 - Interface gráfica do IconAuthor

A área onde o grafo está sendo construído pode ser aumentada, facilitando o processo de navegação quando este se torna grande. Além disso, grupos de ícones podem ser agrupados, ajudando a controlar sua complexidade. Deve-se ressaltar, no entanto, que a ferramenta não força nenhuma disciplina de programação, sendo possível ao usuário criar estruturadas complexas e não estruturadas.

3.1.2 Editor I-HTSPN

Em [WIL96], os autores propõem uma extensão ao modelo de sincronização HTSPN afim de permitir a especificação completa de documentos hipermídia a partir da especificação das estruturas conceituais, de apresentação e do conteúdo. Esta extensão é uma interpretação do modelo HTSPN, chamada de I-HTSPN (*Interpreted-HTSPN*). O Editor I-HTSPN contempla esta extensão, oferecendo ao autor mecanismos para que ele possa, graficamente, descrever uma aplicação multimídia interativa com recursos distribuídos. A figura 3.3 apresenta a interface do editor.

Os dados que fazem parte da aplicação multimídia como vídeo e imagem são especificados a partir de uma janela denominada *Data Specification Editor*. Esta janela funciona como um repositório de componentes multimídia, provendo meios para a adição e exclusão de objetos a ele, bem como permitindo a atribuição de valores aos objetos que dele fazem parte. Por exemplo, a janela *Data Specification Editor* da figura 3.3 apresenta a lista de objetos que fazem parte de uma determinada aplicação multimídia. Nesta janela, pode-se visualizar a especificação de um áudio localizado em http://www.inf.ufsc.br/~willrich/audio1.au.

A estrutura e o fluxo da aplicação são definidos a partir de uma janela especial que possibilita a edição e manipulação de redes HTSPN. Nesta janela, o autor pode criar os lugares, transições e arcos da rede, bem como editar os atributos associados a estes objetos. Aos lugares podem ser associados componentes previamente adicionados ao repositório. A figura 3.3 ilustra a especificação do lugar *audio1*, sendo

possível verificar que o componente *Audio1Data* está sendo associado a ele. Após a realização de todas as etapas de desenvolvimento, o autor pode requisitar a geração automática de código Java da aplicação correspondente.

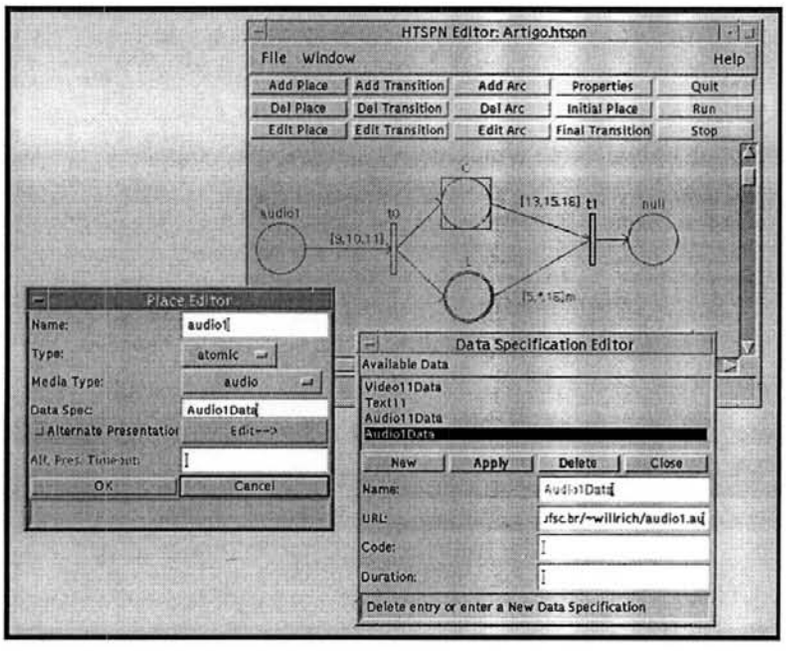


FIGURA 3.3 - Interface do Editor I-HTSPN

Embora bastante poderosa, a ferramenta é de difícil utilização principalmente quando os autores não são da área de informática. Mesmo para os que trabalham nesta área, sua utilização pode ser comprometida por se tratar de um modelo baseado em redes de Petri. O problema clássico da explosão de estados pode tornar uma especificação grande completamente ilegível e de difícil manutenção.

3.2 Sistemas de autoria baseados em linha do tempo

Estes sistemas estão diretamente associados ao modelo de sincronização baseado em linha do tempo apresentado na seção 2.1.2. De maneira geral, o conceito de linha temporal é bastante natural aos seres humanos. Todos os acontecimentos que ocorrem cotidianamente possuem um instante de início e fim bem definidos e acabam sendo projetados na linha temporal representada pela história. A partir desta projeção, torna-se trivial entender as relações estabelecidas entre os diferentes acontecimentos permitindo observar, por exemplo, que um determinado fato B ocorreu após ou simultaneamente a um evento A.

Esta metáfora é apropriada para a definição de sistemas multimídia, pois os elementos que compõem estes sitemas também possuem uma ordenação temporal determinada [BUL95]. A autoria, neste caso, é realizada com o posicionamento de ícones representando componentes multimídia em posições específicas de uma linha do tempo (vide figura 3.4). A duração destes itens pode ser facilmente modificada, simplesmente redefinindo a extensão ocupada por eles nesta linha. A granularidade da linha temporal pode tanto estar associada ao relógio do sistema como a uma referência lógica de tempo escalável. No segundo caso, a criação de aplicações portáveis é facilitada.

A grande contribuição destes sistemas é a possiblidade de se ordenar intuitivamente os elementos no tempo. Não é necessário ser um especialista em informática para criar as aplicações, uma vez que o paradigma se aproxima muito da maneira de pensar das pessoas. Uma de suas desvantagens, em contrapartida, é que mudanças individuais a objetos requerem a reestruturação de vários componentes multimídia associados. Imagine, por exemplo, se o início da apresentação de Imagem3 na figura 3.4 fosse modificado para o instante 2. Se o objetivo fosse que Texto3 e Vídeo1 iniciassem sua apresentação simultaneamente a Imagem3, invariavelmente estes dois componentes também teriam que ser readaptados ao novo contexto. Este tipo de mudanças são raramente realizadas de forma automática, pois é difícil prever se o usuário quer ou não manter as mesmas relações temporais iniciais.

Outro problema é a navegação pela apresentação. Ao ajustar um ponteiro lógico temporal, a maioria dos sistemas multimídia permitem ao usuário final atingir, a qualquer momento, partes específicas da apresentação. Enquanto este tipo de avanço ou retrocesso rápido é bastante intuitivo, o comportamento resultante não é. Ilustrando, se o usuário avançar abruptamente ao instante 2, não fica completamente claro se Texto1, Imagem3, Texto3, Vídeo1, Imagem1 e Música1 ou somente os dois últimos citados serão apresentados.

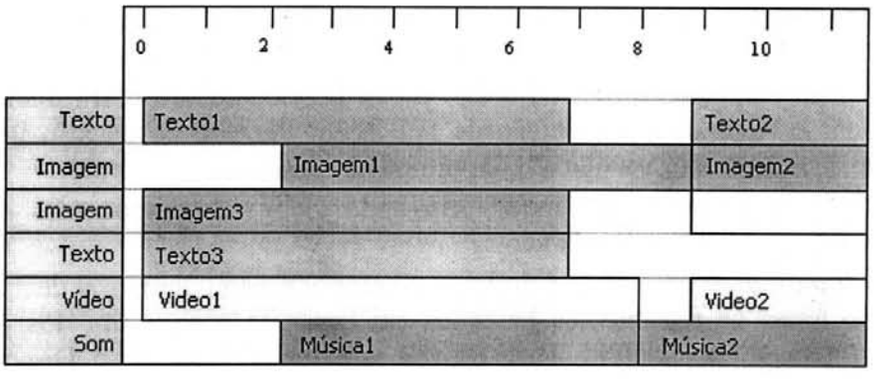


FIGURA 3.4 - Paradigma baseado em linha temporal

Grande parte dos sistemas comerciais existentes atualmente operam sob o conceito de linha do tempo. Entre eles, se destaca o Macromedia Director. Esta ferramenta é apresentada a seguir.

3.2.1 Macromedia Director

O processo de criação de aplicações no Macromedia Director é realizado como se o autor estivesse montando um filme. Sua interface fornece uma planilha com um cronograma associado (figura 3.5). Elementos multimídia como gráficos, sons, transições e scripts são posicionados verticalmente à esquerda nesta planilha como se cada um deles representasse um canal. Cada canal possui o seu cronograma particular, sendo que apenas um elemento multimídia pode ocupar uma posição no cronograma em um dado instante. Estes elementos são encarados como personagens e podem ser manipulados ainda em uma visão denominada Palco, onde se determinam seus posicionamentos em relação ao dispositivo de saída.

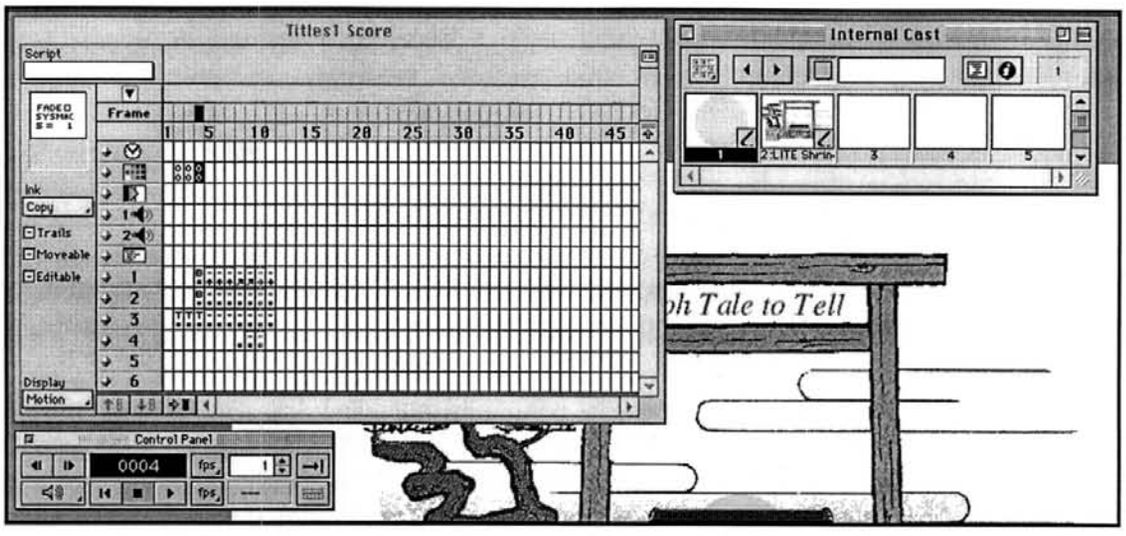


FIGURA 3.5 - Grade temporal do Macromedia Director

Grande parte da funcionalidade do Director resume-se, deste modo, a quatro janelas básicas:

- *Palco (Stage)*: apresenta o resultado da disposição espacial dos elementos exatamente como aparecerão quando a aplicação for executada;
- *Painel de controle (Control Panel)*: provê funcionalidades para iniciar, encerrar, avançar e retroceder a exibição da aplicação. Permite ao usuário verificar se a especificação traduz ou não o comportamento desejado.
- *Cast*: funciona como um repositório onde todas os componentes presentes na aplicação podem ser visualizados;
- *Planilha (Score)*: área onde é definido o comportamento temporal da aplicação.

Como todas as ferramentas baseadas em linha do tempo, um dos probelmas é definir interação do usuário. Para prover esta funcionalidade, o Director provê a linguagem Lingo. Sua utilização, no entanto, faz com que o processo de autoria deixe de ser homogêneo, já que representa um mecanismo auxiliar para suprir as deficiências do modelo de sincronização. Além disso, e mais importante, muitos usuários acabam esbarrando nela ao construir suas aplicações.

3.3 Sistemas de autoria baseados em scripts

Tanto os sistemas de autoria baseados em grafos como os baseados em linha do tempo se utilizam de alguma forma gráfica para descrever a interação entre componentes multimídia em uma aplicação. Enquanto esta abordagem visual é adequada para definir comportamentos de alto-nível, ela não se adequa tão bem na definição de comportamentos complexos. Sistemas baseados em *scripts* ou programas, como também são comumente denominados, fornecem ao autor facilidades em baixo-nível para especificar componentes, temporizações, posicionamentos e interações em uma apresentação [BUL95].

Os esforços de programação variam de acordo com a linguagem utilizada. Estas abrangem desde linguagens de prototipação (*scripts*) até mesmo as de programação convencionais. Alguns dos problemas imediatos deste paradigma são a pouca portabilidade das aplicações criadas e sua limitação aos recursos providos pelo sistema operacional. Além disso, enquanto o modelo é atraente para especialistas,

dificilmente será aceita pela comunidade que opera com multimíia ou tem pouco conhecimento na área de informática. Por esta razão, este paradigma passa a ser utilizado apenas em casos onde requisitos estritos de performance devam ser garantidos.

```
set win = main_win
set cursor = wait
clear win
put background "pastel.pic"
put text "heading.txt" at 10,0
put picture "gables.pic"at 20,0
put picture "logo.pic"at 40,10
put text "contents.txt" at 20,10
set cursor = active
```

FIGURA 3.6 - Aplicação descrita através de um *script*

3.4 Sistemas de autoria baseados em estruturas

Os três paradigmas de autoria vistos anteriormente compartilham uma característicam em comum: definem aplicações em termos de posicionamento e ativação de grupos de componentes multimídia. A abordagem baseada em estruturas, todavia, se preocupa em separar a definição da estrutura lógica da aplicação da definição dos objetos que fazem parte da mesma. Num primeiro momento, o paradigma procura oferecer mecanismos para o autor organizar sua apresentação. Em se tratando de aplicações multimídia, esta organização pode ser feita separando estas aplicações em capítulos ou seções, por exemplo. Apenas após este passo é que o autor passa a se preocupar em adicionar componentes e relacioná-los, processo que pode ser realizado de maneira *top-down* ou *bottom-up* [BUL95].

A navegação da aplicação está tipicamente relacionada com sua estrutura lógica. Referências detalhadas ou associações de conteúdo são normalmente encontradas em níveis mais internos enquanto referências mais gerais e abrangentes são obtidas em níveis superiores. Em aplicações multimídia existe usualmente localidade de interação entre seus componentes, permitindo que se faça agrupamentos entre itens que são estruturalmente relacionados mais frequentemente.

A definição de uma aplicação em termos de uma estrutura lógica explícita pode ser usada para particionar tarefas entre um conjunto de vários autores. A facilidade com que a aplicação pode ser editada e mantida também é aumentada quanto o autor tem uma visão global das relações lógicas entre os itens. No entanto, a abordagem baseada em estruturas não é suficiente para descrever todas as temporizações e requisitos de interação entre elementos de uma aplicação. Como resultado, esta abordagem é normalmente combinada com outras técnicas. Duas ferramentas híbridas que refletem esta situação são o CMIFed e o EBS, apresentadas a seguir.

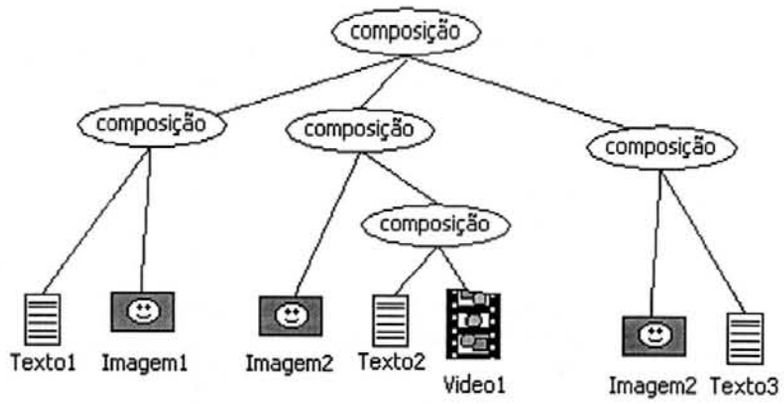


FIGURA 3.7 - Autoria baseada em estruturas

3.4.1 CMIFed

CMIFed foi criado para prover aos autores um ambiente para visualização e manipulação estruturada de aplicações. As tarefas de autoria são realizadas com a utilização de três vistas: a hierárquica, a de canais e a de execução. A visão hierárquica provê ao autor meios para a estruturação da aplicação. Esta pode tanto se realizada de maneira *top-down* ou *bottom-up*, permitindo agrupar componentes ou definir estruturas completamente vazias para serem preenchidas posteriormente. A vista denominada canal apresenta uma linha temporal modificada, permitindo descrever informações relativas a temporização e a utilização lógica de recursos. Informação sobre o fluxo de controle aparece diretamente nesta vista através da utilização de arcos, simbolizando descritores de restrição de sincronização. A terceira vista, por fim, provê ao autor a possibilidade de verificar os resultados de sua especificação da maneira como ela será apresentada ao usuário final.

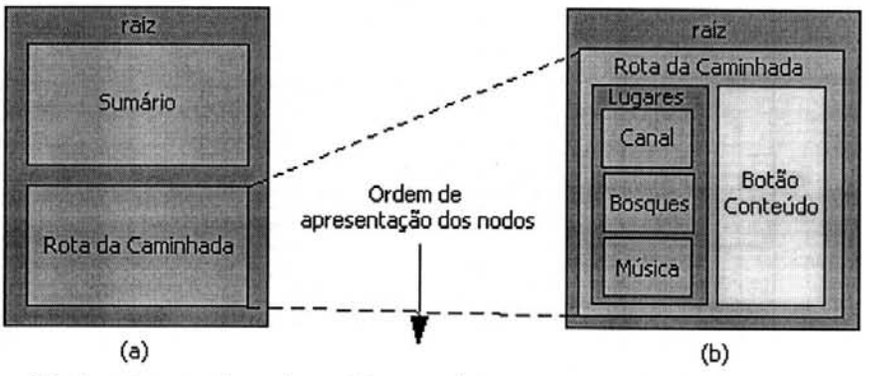
A vista hierárquica

A vista hierárquica é a janela mais importante no processo de autoria, provendo mecanismos para mostrar e manipular a estrutura da aplicação. Nodos folhas representam componentes e nodos não folhas, composições. Estes últimos contém uma coleção de outros nodos de composição e ou componentes. A estrutura hierárquica é representada nesta vista como uma estrutura de blocos. Durante a fase de autoria, cada componente multimídia no diagrama estrutural é associado a um canal[3], que pode representar a tela ou dispositivos de som, por exemplo.

Para possibilitar ao autor especificar as sincronizações temporais de maneira conveniente, dois tipos de composições são suportados: paralelo e seqüencial. É possível, assim, que o autor agrupe componentes a serem apresentados simultaneamente ou um após o outro. Ele não precisa especificar as temporizações neste ponto, uma vez que estas são deduzidas a partir da estrutura hierárquica, ou seja, das durações dos nodos internos à estrutura. Na figura 3.8, as duas caixas Lugares e Botão Conteúdo são apresentadas em paralelo; as três caixas menores aninhadas dentro de Lugares, por sua vez, são executadas sequencialmente uma após a outra. A duração de um nodo composto é derivado pelo sistema. A duração de uma composição serial é a soma das durações dos nodos filhos e a de uma composição

[3] Dispositivo lógico de saída mapeado, pelo módulo executor, para dispositivos físicos de saída

paralela é determinada pelo nodo filho de maior duração. Quando um nodo não tem duração explícita ele é apresentado de acordo com a duração do nodo pai.



(a) As caixas grandes indicam diferentes níveis de estrutura da aplicação. A parte Sumário é apresentada antes de Rota da Caminhada. A pequena caixa branca próxima ao nome do componente indica que existe uma estrutura interna a ele já definida.

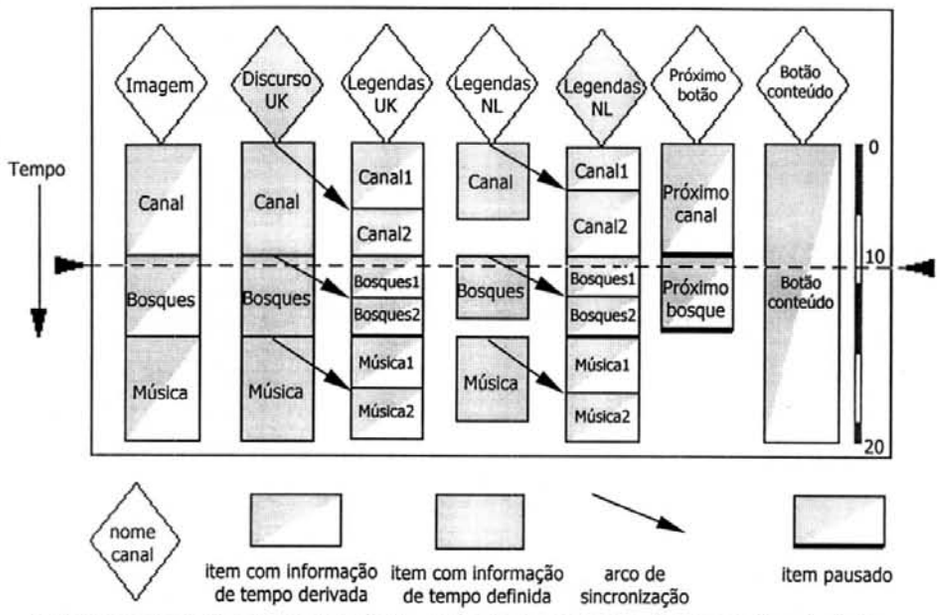
(b) Visão detalhada da cena Rota da Camihada. O nodo Lugares contém três filhos cada um com uma estrutura aninhada já definida. A caixa à direita representa uma mídia (nodo folha da estrutura hierárquica).

FIGURA 3.8 - Visão hierárquica do CMIFed

Vista dos canais

O objetivo desta vista é permitir a representação explícita do tempo e controlar a utilização dos recursos disponíveis. Para tal, é realizado um mapeamento dos componentes multimídia para os respectivos recursos a serem usados para possibilitar sua execução. Uma ilustração da vista dos canais é apresentada na figura 3.9. Ao supor a existência deste nível extra acima dos recursos físicos propriamente ditos, o autor é capaz de descrever aplicações independentes de plataforma. É responsabilidade do executor, otimizado para uma determinada configuração de *hardware*, interpretar os canais lógicos e atribuir os componentes aos respectivos dispositivos de saída disponíveis.

A vista dos canais apresenta as relações temporais derivadas da estrutura definida na vista hierárquica. Os componentes que fazem parte da aplicação (folhas da estrutura hierárquica) são apresentadas com suas durações precisas e com suas relações temporais. Se o autor modificar algum aspecto temporal em qualquer parte da representação, a vista do canal é atualizada para refletir as modificações.



Os losangos na parte superior da figura apresentam os nomes dos canais (canais inativos são sombreados). As mídias atribuídas aos canais são representadas como caixas abaixo dos losangos. A altura da caixa representa a duração da apresentação da respectiva mídia. As caixas completamente sombreadas têm duração explicitamente definidas. As caixas sombreadas pela metade herdam a duração de seu pai na estrutura da apresentação.

FIGURA 3.9 - Visão dos canais para a seqüência Rota da Caminhada

3.4.2 EBS

EBS (*Editor e Browser Gráfico para Sincronização Temporal e Espacial de Objetos Multimídia/Hipermídia*) é parte integrande do ambiente *HySEE* (*HyperProp Show Editor and Executor*), desenvolvido dentro do projeto PROTEM II HyperProp [COS96]. A ferramenta utiliza o Modelo de Contextos Aninhados (NCM) como modelo de estruturação e apresentação de dados, em conformidade com a proposta de padrão MHEG. O sistema permite a definição em forma gráfica da disposição temporal e espacial de um objeto, em relação a um tempo fixo ou a outros objetos, bem como a visualização das composições no tempo e espaço.

Como pode ser observado na figura 3.10 (extraída de [COS96]), o editor é composto basicamente por três visões distintas: *private base*, *time view* e *spatial view*.

A private *base* é um browser do repositório de dados do usuário. Na vista fornecida por ela, nós são representados por círculos, elos por arestas e as composições pela inclusão de círculos e arestas. Através da janela *private base*, o autor pode ver e editar a estrutura dos documentos, e também selecionar o nó que deseja incluir na *time view*. A janela *private base* apresenta apenas a visão estática dos relacionamentos entre os componentes de um documento, não mostrando quais são as relações de sincronismo entre eles.

Para tal, o usuário utiliza a região de interface chamada *time view*. Nela, os nós são representados por retângulos, cujo comprimento indica a duração esperada no tempo para sua exibição. Os objetos são dispostos nas regiões *display channel* e *audio channel*, que correspondem às abstrações dos dispositivos de saída de vídeo e áudio da plataforma de exibição. Os elos são representados por arestas que conectam os nós.

As áreas que correspondem ao *display channel* e ao *audio channel* são representadas por faixas mostradas na *time view*, conforme pode ser visualizado na figura 3.10.

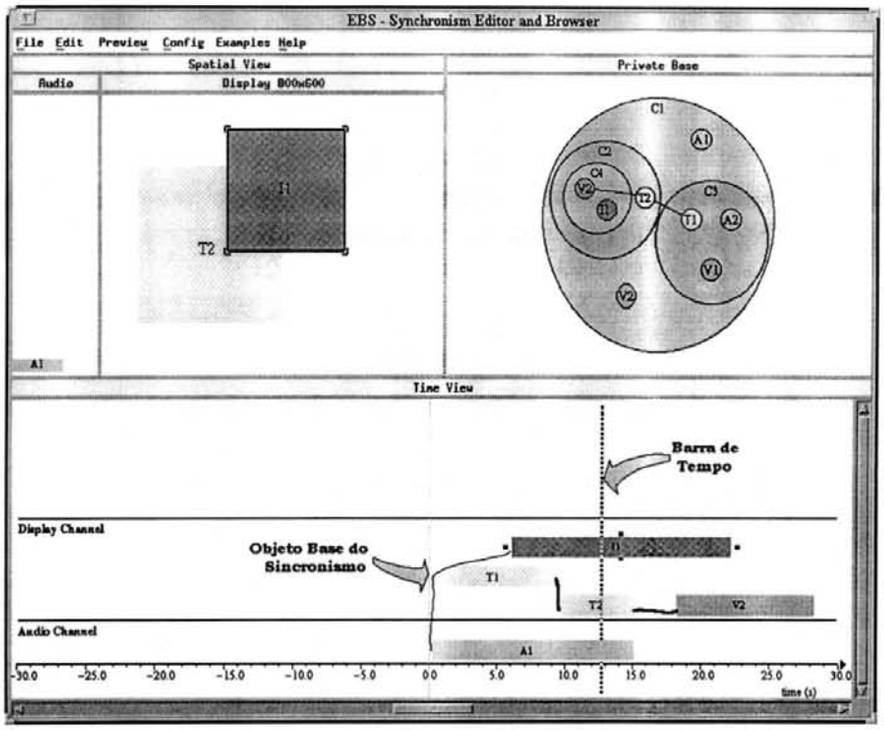


FIGURA 3.10 - Janela principal do EBS

Conforme mencionado anteriormente, a sincronização de objetos multimídia pode ser estabelecida não apenas em relação a um instante fixo, como também em relação a outros objetos. O editor EBS possui operações que permitem definir o sincronismo entre objetos através de relacionamentos de alto nível, tais como *exibir ao término de, exibir iniciando ao mesmo tempo* e *exibir iniciando e terminando ao mesmo tempo.*

Esses relacionamentos implicam na criação automática de elos entre os nós, como será explicado no exemplo a seguir. Considerando a figura 3.10, suponha que o autor defina que os nós V1 e T1 devam ser exibidos iniciando e terminando ao mesmo tempo. Nesse caso, o EBS cria automaticamente dois elos que alinham no tempo os eventos associados ao início e ao fim da exibição dos dois objetos. A figura 3.11 (extraída de [COS96]) mostra os elos criados pelo EBS para definir o alinhamento temporal dos objetos O1 e O2 segundo as três relações de alto nível descritas anteriormente. Note que o relacionamento *iniciar e terminar ao mesmo tempo* exige não apenas a compatibilização do posicionamento no tempo dos objetos, mas também a compatibilização da duração de suas exibições.

Ainda na *time view* há uma barra de tempo móvel (destacada na figura 3.10) que pode ser posicionada em qualquer instante do tempo. A janela *time view* possui um mecanismo de *scroll* para permitir que sejam visualizados todos os instantes da visão temporal. Se a barra de tempo estiver sobre um ou mais nós, estes aparecerão na *spatial view* nos canais em que eles deverão ser exibidos. No exemplo da figura 3.10, a barra de tempo se encontra sobre os *nós* I1, T2 e A1, que aparecem na *spatial view*.

Na *spatial view*, por fim, o autor edita o sincronismo espacial, posicionando os objetos dentro dos canais onde eles serão apresentados. É através deste posicionamento que o autor define, por exemplo, o espaço do *display* que será usado

para exibir uma janela que contém uma imagem gráfica, ou o volume que será usado para exibir um áudio, no instante de tempo definido pela barra de tempo da *time view*.

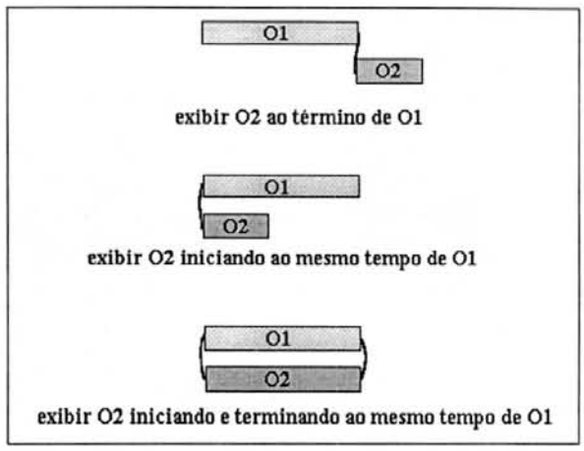


FIGURA 3.11 - Relações de sincronização entre objetos

4 Modelo de Autoria Proposto

O estudo dos modelos de autoria suportados pelas ferramentas atualmente disponíveis no mercado ou no âmbito acadêmico permitiu verificar a diversidade de abordagens presentes no que tange a especificação de aplicações multimídia. Na maioria das vezes, cada um destes modelos agrega algumas características interessantes mas é desprovida de outras, sendo difícil eleger um que atenda integralmente e de forma eficiente as necessidades do autor.

Determinar se eles atendem ou não a estas necessidades pressupõe uma definição clara do escopo das aplicações para as quais foram projetados. Como ilustração, pode-se citar a ferramenta de criação Macromedia Director, originalmente desenvolvida para permitir a especificação de aplicações multimídia interativas locais. A ferramenta opera sob a premissa de que todos os recursos que a aplicação necessitará estarão disponíveis localmente no momento de sua execução. Se a ferramenta simplesmente passasse a atender também aplicações com recursos distribuídos, ela certamente não seria apropriada para definir esta classe de aplicações. A razão é simples: ela incorpora um modelo de sincronização baseado em linha temporal, inadequado para modelar variações temporais na apresentação de componentes multimídia.

Baseado nestas considerações, definiu-se que o modelo de autoria e conseqüentemente o ambiente de criação propostos neste trabalho devem suportar a especificação de **aplicações multimídia interativas com recursos distribuídos em conformidade com o padrão MHEG-5**. Como já foi mencionado anteriormente, este padrão permite o compartilhamento de informação multimídia sem se preocupar com a plataforma ou sistema operacional utilizado, viabilizando a especificação e desenvolvimento de aplicações portáveis. Além de determinar pontualmente a classe de aplicações suportadas pelo modelo, estabeleceu-se algumas características básicas essenciais a serem supridas por ele:

- *Autoria de forma simples e intuitiva*: deve ser possível criar aplicações de forma rápida e facil, mesmo que o autor não seja um especialista em informática. Para tal, é conveniente que o modelo possua uma notação gráfica, comprovadamente mais eficiente para atender aos requisitos de facilidade de compreensão do que as notações textuais. Com o aumento do tamanho das especificações é recomendável que o modelo apresente tanto mecanismos para gerenciar, como para permitir ao usuário abstrair esta crescente complexidade.
- *Facilidade de manutenção*: deseja-se contar com um modelo modular, onde modificações no comportamento lógico e temporal em parte de uma aplicação não afete a restante.
- *Suporte ao reuso*: o modelo deve fornecer estruturas capazes de serem reusadas em aplicações futuras, tornando o processo de desenvolvimento mais produtivo.

Considerando todos os aspectos mencionados acima, passa-se a discutir a possibilidade de incorporar um dos modelos de sincronização apresentados na seção 2.1 ao modelo de autoria proposto neste trabalho.

O primeiro modelo a ser abordado é o baseado em intervalos. Este é desprovido de uma representação gráfica, sendo esta uma desvantagem quando se procura um modelo de autoria intuitivo e de fácil utilização. Além disso, e talvez mais importante, não é possível representar em MHEG-5 todas as relações de intervalos apresentadas na figura 2.10, pois neste padrão relações temporais são definidas de

forma procedural, ao contrário do que acontece no modelo, onde elas são descritas de forma declarativa. Para ilustrar melhor o efeito desta diferença, pode-se tomar como exemplo o relacionamento de dois componentes multimídia quaisquer através do operador *beforeendof*(δ_1); não há como modelar em MHEG-5 que a apresentação de um componente multimídia deva ser realizada antes do término da de outro.

Embora seja muito adequado para modelar aplicações multimídia locais, o modelo baseado em linha do tempo, por sua vez, é imediatamente descartado para representar estas aplicações no contexto distribuído. A essência do modelo é justamente conhecer *a priori* os instantes de início e fim de cada um dos componentes que fazem parte da aplicação, o que é impossível quando se trata de componentes que precisam ser buscados em pontos remotos muitas vezes durante a própria apresentação. O modelo é deficiente também na modelagem de interação do usuário. Para tal, acaba sendo necessário a utilização de artifícios extras ao próprio modelo como linguagens de *script*, por exemplo.

O modelo hierárquico também apresenta deficiências. A mais relevante é que o processo de construção e leitura das especificações não é natural, uma vez que não fica clara a ordem em que os componentes serão apresentados. Além disso, o modelo não permite o estabelecimento de alguns tipos de sincronização, o que acaba restringindo sensivelmente o poder de expressão do autor. Na verdade, o conceito de hierarquia é bem mais adequado para representar a estrutura lógica das aplicações do que propriamente para representar a sincronização entre os elementos que a compõem.

Dos modelos de sincronização apresentados na na seção 2.1 restam os baseados em pontos de referência e em redes de Petri. Ambos são adequados para modelar aplicações multimídia interativas com recursos distribuídos. No primeiro deles, no entanto, não é possível mapear as especificações resultantes para MHEG-5. A norma MHEG opera como se os componentes multimídia fossem unidades indivisíveis; o modelo, por sua vez, encara os componentes como seqüências de LDUs.

A exemplo do CMIFed e do EBS, este trabalho propõe um modelo de autoria híbrido que agrega um modelo de autoria baseado em estruturas a um modelo de sincronização baseado em grafos comparável ao de redes Petri. Utilizando o conceito de ícones, criou-se uma representação gráfica simplificada que provê ao autor mecanismos facilitadores ao processo de especificação. A figura 4.1 apresenta um esquema geral do modelo resultante. O nível de estruturação lógica opera sob os conceitos de cenários e grupos, provendo uma visão geral da aplicação. A definição de sincronizações temporais, por sua vez, é realizada em cada cenário sendo possível relacionar e ordenar, com a utilização de um grafo simplificado, a apresentação de componentes multimídia no tempo. Tem-se ainda a sincronização no espaço, onde estes mesmos componentes são organizados levando em consideração seu posicionamento em relação ao dispositivo de saída.

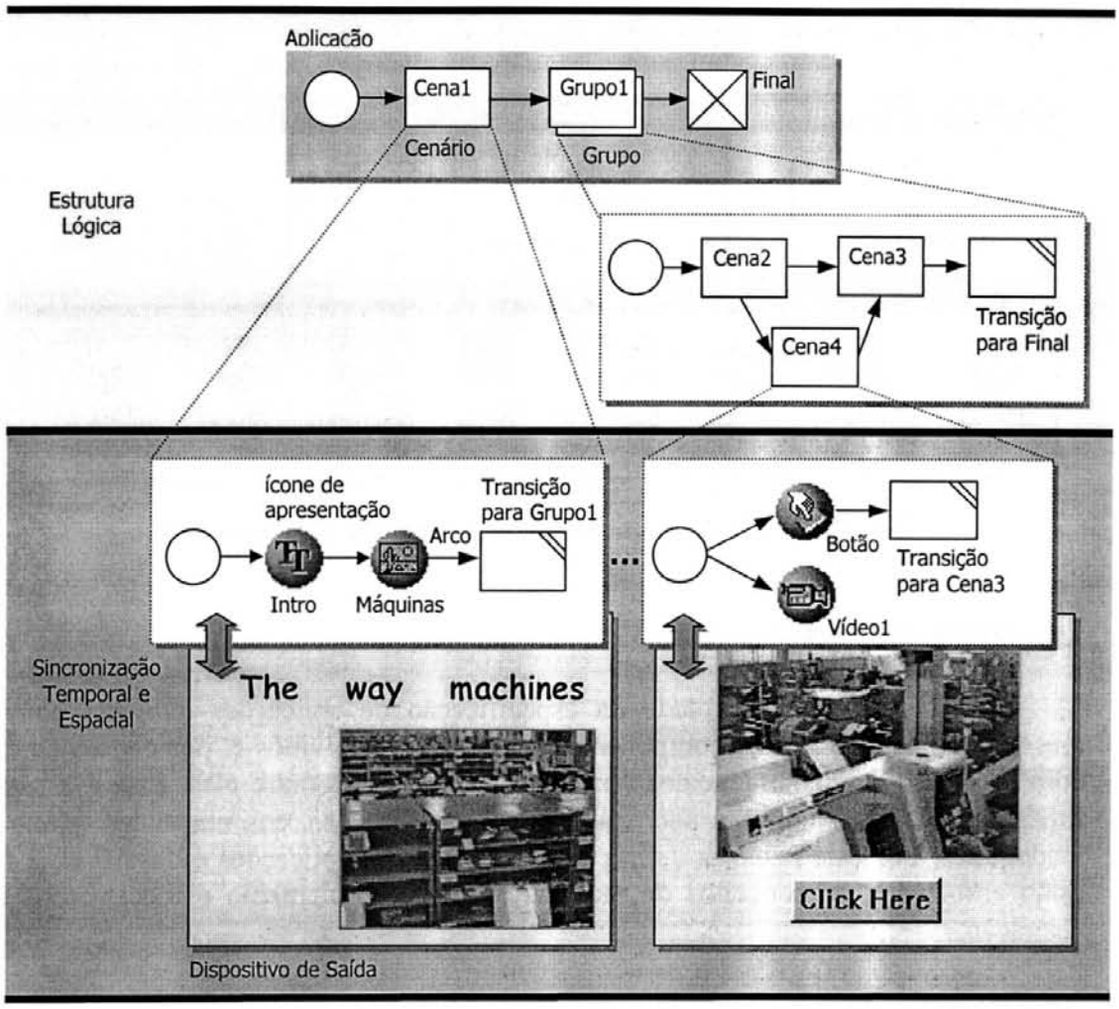


FIGURA 4.1 - Estrutura de uma aplicação multimídia interativa

4.1 Símbolos gráficos do modelo

Antes de abordar as funcionalidades básicas do nível de estruturação lógica, apresenta-se todos os elementos gráficos que podem ser utilizados na especificação das aplicações (vide figura 4.2). De uma maneira geral, os símbolos gráficos podem ser classificados em:

- *Ícones de estruturação*: representam os cenários e grupos. Aparecem apenas no nível de estruturação lógica.
- *Ícones de apresentação*: permitem modelar a apresentação de componentes multimídia. Existe um ícone diferenciado para cada uma delas. Estes podem ser utilizados no nível de estruturação lógica, mas são empregados principalmente na representação da estrutura temporal dos cenários.
- *Pontos de sincronização*: utilizados para modelar diferentes estratégias de sincronização entre componentes multimídia.
- *Transição*: ícone utilizado para representar, na visão temporal, o próximo cenário a ser apresentado.

- *Delimitadores de início e fim*: permitem delimitar o início e fim de uma aplicação, grupo ou cenário. São encontrados em ambos os níveis.

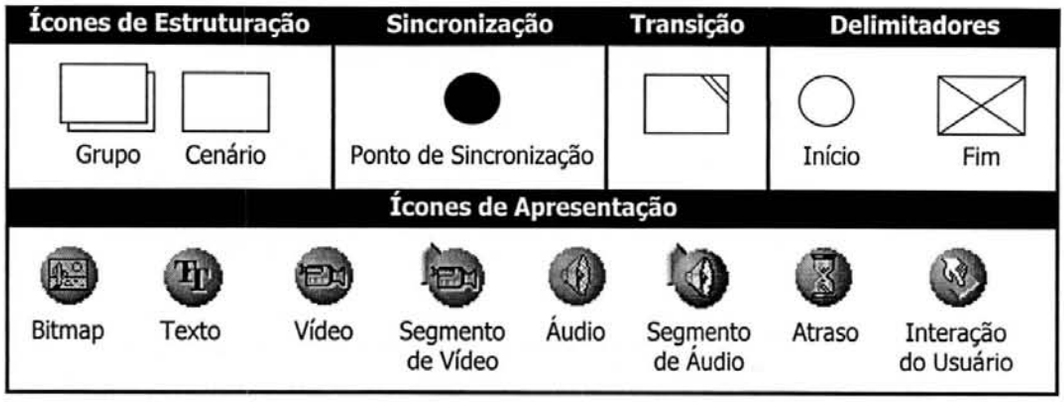


FIGURA 4.2 - Conjunto de ícones suportados pelo modelo

4.2 Estruturação lógica

O grau de complexidade da especificação de aplicações aumenta com o crescimento do número de componentes multimídia envolvidos e, por conseqüência, com os diversos relacionamentos temporais estabelecidos entre eles. Esta é a razão fundamental pela qual a especificação destas aplicações em um único plano é inadequada. Por esta razão, a definição estruturada das aplicações é desejável; a ela estão associados os conceitos de modularidade, encapsulamento e mecanismos de abstração.

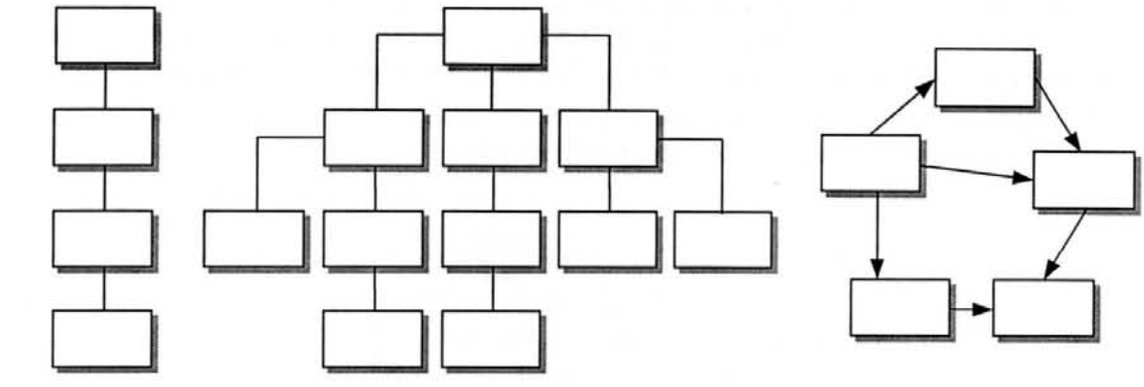
4.2.1 Cenários e grupos

Tendo em vista o problema apontado no parágrafo anterior e utilizando como base a norma MHEG-5, verificou-se que esta se utiliza do conceito de cenas, onde documentos multimídia interativos são organizados como um conjunto de cenários relacionados por eventos que provêm a navegação entre eles.

O conceito de cenas ou cenários foi incorporado ao modelo proposto. Cada um deles pode ser visto como uma *caixa preta* com um comportamento interno que, sob determinadas condições, habilita a apresentação de outros cenários. A utilização deste conceito, contudo, não resolve de todo o problema da complexidade, uma vez que uma especificação com um número elevado de cenários será dificilmente compreendida. Procurando facilitar o entendimento de aplicações muito grandes, criou-se ainda o conceito de grupo de cenários, que permite organizar cenários em grupos, hierarquizando a estrutura lógica da aplicação. A porção superior da figura 4.1 ilustra a estrutura lógica de uma aplicação, composta de quatro cenários (Cena1, Cena2, Cena3 e Cena4) sendo que três deles (Cena2, Cena3 e Cena4), pelo grau de coesão estabelecido entre eles, foram agrupados em Grupo1.

A regra básica para a composição de cenários e grupos é a seguinte: um grupo comporta um número ilimitado de outros grupos ou cenários. Em um cenário, no entanto, não podem adicionados nenhum destes elementos. A estrutura lógica resultante desta hierarquia é uma árvore onde os nodos folhas representam cenários e os internos, grupos.

Ainda na figura 4.1 pode-se observar que o relacionamento entre cenários e grupos é definido com a utilização de arcos que indicam a ordem em que estes serão apresentados. É válido ressaltar que estas ligações indicam um comportamento possível e não obrigatório. Na mesma figura, verifica-se na definição de Cena2 que dependendo dos eventos internos deste cenário, ocorrerá uma transição ou para Cena3 ou para Cena4. Aproveitando este exemplo, deve-se comentar que, por uma própria restrição do MHEG-5, apenas um cenário pode ser apresentado por vez.



(a) Estrutura linear (b) Estrutura hierárquica (c) Estrutura de rede

FIGURA 4.3 - Estruturas básicas para navegação aplicações multimídia interativas

Em [GIN95] são apresentadas as três estruturas de navegação possíveis em aplicações multimídia interativas que são plenamente suportadas pelo modelo: linear, hierárquica e de rede ou grafos, conforme representado na figura 4.3. Para determinar a estrutura mais adequada para uma aplicação, o autor deverá levar em consideração o propósito da mesma. Por exemplo, pode-se escolher uma estrutura linear (figura 4.3a) para a apresentação de material de treinamento, uma vez que é desejável que o usuário estude os módulos de forma seqüencial. A estrutura hierárquica (figura 4.3b) é conveniente, por exemplo, para a apresentação de um livro, onde há uma divisão de capítulos, seções e parágrafos. A estrutura de grafos (figura 4.3c), por fim, é indicada para a organização de informações inter-relacionadas, como em enciclopédias, permitindo ao usuário acessar a mesma informação de diferentes contextos.

4.2.2 Ícones de início e fim

A função dos ícones Grupo e Cenário já foi apresentada. Importantes também são os ícones Início e Fim. O primeiro é utilizado não apenas na estruturação lógica, como também na visão temporal dos cenários para indicar o ponto de partida. O segundo, por sua vez, é utilizado apenas na estrutura lógica, e permite representar pontos de saída da aplicação. Pode aparecer mais do que um ícone de finalização na representação de um determinado grupo. A única exigência é que seja modelado pelo menos um ponto de finalização na aplicação. No caso do ícone de início, por outro lado, a regra é diferente: um único ícone de início deve aparecer obrigatoriamente na representação de cada grupo e cenário. A figura 4.1 reflete estes requisitos. A aplicação modelada apresenta um único ponto de saída (Final) e em todos os grupos e cenários existe um ícone de início.

4.2.3 Representação de componentes compartilhados

Outro ponto importante relativo à estruturação lógica é que normalmente existem componentes multimídia cuja apresentação abrange vários cenários. Esta funcionalidade existe no padrão MHEG-5 e é materializada através da associação de objetos *Ingredient* à classe *Application*. O efeito de sua utilização é que estes são apresentados apresentados ininterruptamente mesmo durante a ocorrência de transições de cenas. A figura 4.4 abaixo ilustra uma aplicação organizada em três cenários (Cena1, Cena5 e Cena6) e um grupo (Grupo1). Simultaneamente a toda a aplicação é apresentado uma imagem (Logo) e durante a apresentação de Grupo1 e Cena5 é executado um áudio (Áudio1). As linhas que permitem representar a duração dos componentes em termos de abrangência de cenário não apresentam as setas para diferenciá-las das setas que indicam uma transição.

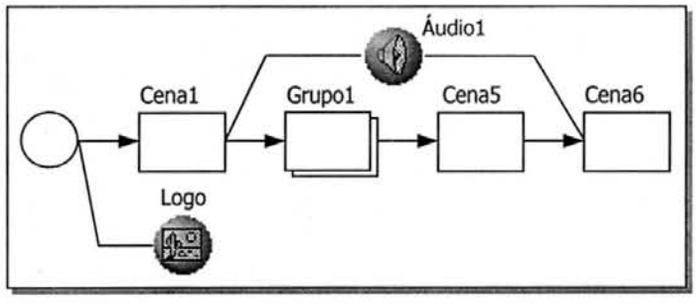


FIGURA 4.4 - Componentes compartilhados entre cenários e grupos

4.2.4 Suporte ao reuso de cenários e grupos

Do ponto de vista da autoria de aplicações, a estrutura proposta possibilita a reutilização de cenários e grupos que venham a se repetir em diferentes especificações. Somado a isto, o modelo viabiliza o desenvolvimento de *templates* - cenários básicos pré-contruídos - que tornam o processo de especificação evolutivo e incremental. Pode-se ter uma série de *templates*, sendo que o processo de especificação, neste caso, se resume a unir estes diferentes cenários, reduzindo drasticamente o tempo de desenvolvimento.

Especificamente no caso do reuso de cenários ou grupos, é necessário apenas redefinir as transições que partem deles. A figura 4.5 ilustra um exemplo: em uma especificação A é definido um cenário cujo comportamento interno prevê transições para Grupo1 e Cena2. Ao ser reusado por uma especificação B, estes não mais existem. Deste modo, as transições foram redefinidas para Cena3 e Cena4, respectivamente, respeitando o contexto da nova aplicação.

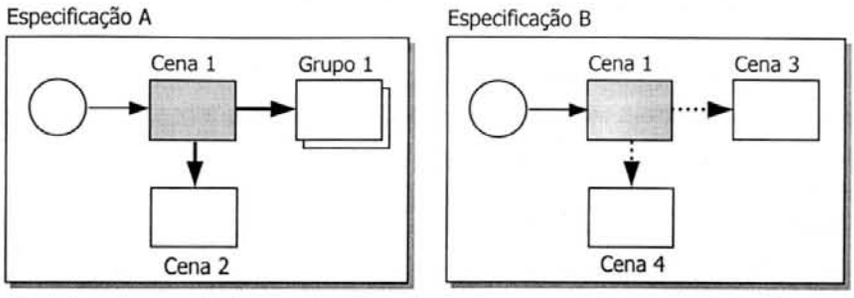


FIGURA 4.5 - Reutilização de cenários e grupos

4.3 Sincronização Temporal

A sincronização temporal de uma aplicação, conforme mencionado no início deste capítulo, diz respeito à ordenação da apresentação de seus componentes no tempo. Este trabalho propõe a definição da sincronização em cada cenário individualmente, contribuindo para a diminuição da complexidade no processo de especificação. Também já comentado anteriormente, propõ-se que as relações temporais entre os elementos que fazem parte de um cenário sejam estabelecidas a partir de um grafo. Graças à possibilidade modularizar o comportamento temporal da aplicação em diversos cenários, é possível evitar que a complexidade das redes se torne muito elevada. Os tópicos, a seguir, apresentam como a sincronização entre elementos multimídia pode ser definida.

4.3.1 Apresentação seqüencial e paralela

A apresentação de componentes multimídia pode ocorrer seqüencialmente ou simultaneamente. Na apresentação seqüencial, o início da apresentação do componente depende do final da apresentação de outro. O paralelismo, por sua vez, prevê que componentes passam a ser apresentadas a partir de um mesmo instante no tempo. Na figura 4.1 aparecem os dois tipos de sincronização básica. Em Cena1, à esquerda, é definida a apresentação de um texto (Intro) seguido da apresentação de uma imagem (Máquinas). Em Cena4, por sua vez, tem-se a apresentação simultânea de um vídeo (Vídeo1) e um botão (Botão).

A semântica associada aos arcos que conectam os componentes na visão temporal é diferente da associada aos arcos no nível de estruturação lógica. A presença de dois arcos partindo de um mesmo ícone requerem que, ao final da apresentação do componente multimídia que ele representa, todos os ícones destino associados devem ser executados. Na figura 4.6, por exemplo, ambos B e C são apresentados tão logo encerre a execução de A .

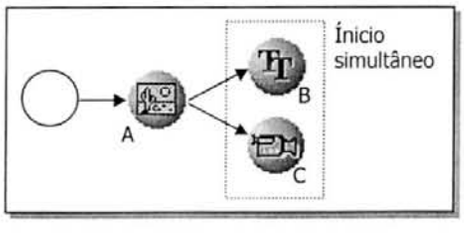


FIGURA 4.6 - Apresentação seqüencial e simultânea

4.3.2 Duração de eventos e delay

A cada componente multimídia está associado um tempo mínimo e um tempo máximo de apresentação. No caso de uma imagem ou de um texto estes valores são equivalentes por se tratarem de componentes independentes do tempo. Observe na figura 4.7 que o tempo de apresentação da imagem *Slide1* é dez segundos. Quando se depara com componentes como áudio e vídeo, no entanto, a tendência é determinar um tempo mínimo e máximo de apresentação, uma vez que dificilmente elas serão apresentadas à taxa nominal em decorrência de problemas como tráfego de rede. Neste caso, a representação destes tempos é dada por um intervalo.

Em algumas cirscunstâncias é possível omitir a duração associada a um componente. A seguir são apresentadas as situações quando isto é possível:

- É possível omitir a duração da apresentação de um componente multimídia quando nenhum arco parte dele ligando-o a outros componentes. Quando isto ocorre, eles são apresentados até que haja a troca de cenário. Na figura 4.6, por exemplo, os ícones B e C não precisam ter obrigatoriamente uma duração associada, pois neste cenário não existem ícones cuja apresentação dependa deles. Já na figura 4.7, a falta da definição de duração em qualquer um dos ícones acarretaria em um cenário temporalmente incoerente.
- Se um componente multimídia independente do tempo está sendo apresentado simultaneamente a um componente dependente do tempo, o primeiro pode omitir sua duração, como pode ser visto na figura 4.10.
- Na representação da interação do usuário. Neste modelo, a interação é representada como uma construção cujo tempo de apresentação é indeterminado, variando entre os valores mínimo e máximo associados. Quando o limiar máximo é atingido, o cenário prossegue com sua apresentação. O modelo permite especificar um botão sem duração determinada. Isto pode ser feito em qualquer circunstância, ao contrário dos outros componentes; neste caso, a evolução do cenário ocorrerá somente após a interação.

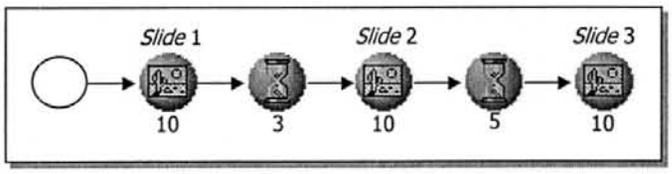


FIGURA 4.7 - Utilização de *delay*

Não menos importante é a possibilidade de modelar um atraso entre a apresentação de dois componentes consecutivos. Para tal, pode-se utilizar uma construção especial, que não possui um objeto multimídia associado, mas que representa a ocorrência de um *delay*. A figura 4.7 ilustra três *slides* (*Slide1*, *Slide2* e *Slide3*) sendo apresentados seqüencialmente, sendo que entre o primeiro e o segundo é modelado um atraso de 3 segundos e entre o segundo e o terceiro, um atraso de 5 segundos.

4.3.3 Mecanismo de transição entre cenários

O ícone de transição viabiliza a navegação entre cenários. Sua execução implica na imediata suspensão da apresentação de todas os componentes associados ao cenário corrente e início da apresentação do novo cenário. Na figura 4.8, Cena1 modela um

botão (Próxima Cena) sendo apresentado simultaneamente a um vídeo (Intro). Caso o botão seja pressionado ou tenham passado dez segundos, o vídeo é encerrado e inicia-se a apresentação de Cena2. Transições podem ser realizadas também para o final da aplicação. A figura 4.8 ilustra a definição de uma transição em Cena3. Esta transição é realizada para Final2, indicando que após a apresentação desta cena a aplicação é encerrada.

A possibilidade de especificar a estrutura lógica em vários níveis impacta a definição das transições. Exemplificando, a transição em Cena3 poderia ser alternativamente realizada para Final, presente no grupo inicial (Aplicação A). O comportamento hierárquico para definir as transições possíveis ocorre da mesma forma com cenários. Neste caso, todas as cenas ligadas diretamente a Cena3 ou a Grupo1 (gerador de Cena3) poderiam ser habilitadas pela transição em questão.

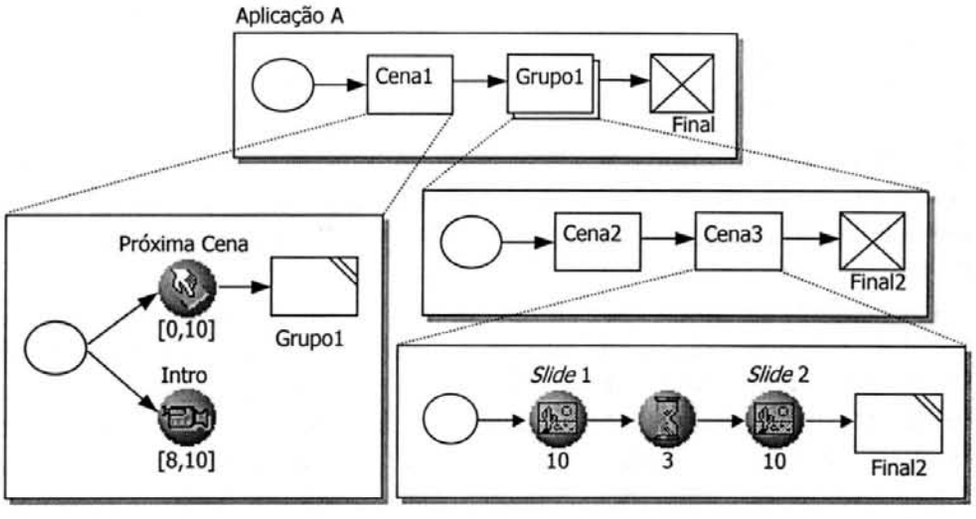


FIGURA 4.8 - Transição de cenas

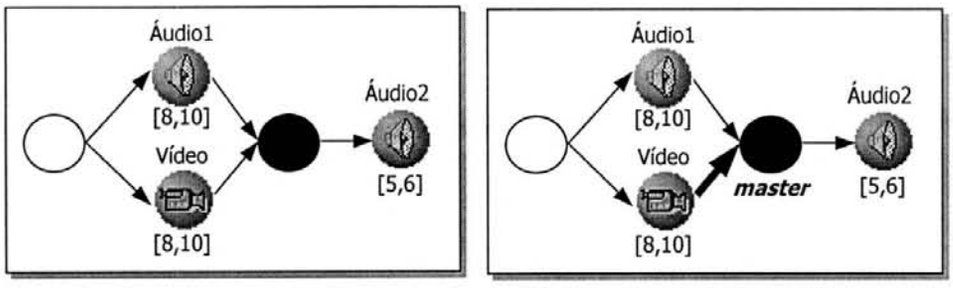
A figura mostra ainda a necessidade em haver coerência entre as ligações estabelecidas no nível de estruturação lógica e as transições definidas na visão temporal de cada cenário. No exemplo apresentado, a visão temporal de Cena1 só pode apresentar transições para Grupo1. Nada impede, no entanto, que em Cena1 sejam definidas mais do que uma transição para Grupo1.

O ícone de transição foge à semântica convencional das redes de Petri. Na realidade, é o mecanismo que permite integrar o modelo de sincronização aos dispositivos de estruturação lógica da aplicação. Mais do que isso, procura solucionar algumas deficiências do modelo HTPSN. Uma destas deficiências, por exemplo, é a impossibilidade de definir uma composição *Capítulo A*, que pode, sob a intervenção do leitor pelo disparo de um evento de seleção, acionar outra composição *Capítulo B*, que não está contida em A. Para tanto, componentes dos *capítulos A* e *B* deveriam estar definidos em uma mesma composição, perdendo a estruturação lógica [SOA97].

4.3.4 Pontos de sincronização

Os pontos de sincronização permitem associar o início da apresentação de um ou mais componentes multimídia a diferentes políticas relacionadas à finalização da apresentação de outros componentes que convergem para estes pontos. A figura 4.9a ilustra um cenário onde são apresentados em paralelo um vídeo (Vídeo) e um áudio

(Áudio1), seguidos de outro áudio (Áudio2). Esta representação implica na obrigatoriedade de ambos terminarem sua apresentação ou expirar seu tempo de duração para que Audio 2 passe a ser executado.



(a) Ponto de sincronização

(b) Política *master* de disparo do ponto de sincronização

FIGURA 4.9 - Pontos de sincronização e políticas de disparo

Para permitir um maior poder de especificação, adotou-se algumas políticas amplamente comentadas na literatura, que permitem a associação de comportamentos diferentes aos pontos de sincronização [SEN94]. Por simplificação, apenas três delas estão presentes no modelo:

- *Master*: significa que um ponto de sincronização é disparado quando a apresentação de um componente multimídia mestre é encerrada, interrompendo todos os outros. Esta seria a política a ser utilizada no exemplo da figura 4.9a acima no caso de se desejar que o final da apresentação do vídeo (mestre) interrompesse a execução de Audio 1, iniciando Audio2 (vide figura 4.9b). O componente mestre é identificado pela presença do caracter *m* ou pela palavra *master* próximo a ela.

- *Earliest*: o disparo de um ponto de sincronização ocorre quando a apresentação do primeiro componente for encerrada, ocorrendo então a interrupção dos demais. Esta política é graficamente representada pela presença do caracter *e* ou pela palavra *earliest* próximo ao ponto de sincronização.

- *Latest*: A ausência de indicação próxima ao componente ou ao ponto de sincronização significa que todos os componentes que antecedem este ponto serão executados (ou finalizarão pelo esgotamento de seu tempo máximo de apresentação) antes de ele ser disparado. Esta política é também conhecida por *latest* (figura 4.9a).

4.3.5 Instantes de sincronização

Em MUSE, um conjunto de ações pode ser sincronizada unicamente com relação ao início ou fim de um conjunto de ações. A sincronização entre componentes em outros instantes que não sejam o início e fim de suas apresentações requer a divisão destes componentes em partes, dando origem a um conjunto de segmentos, cada um deles representando um intervalo na sua apresentação. A granularidade desta divisão está associada ao grau de precisão desejado para a sincronização. A figura 4.10 ilustra a sincronização de duas legendas com um vídeo, onde este é dividido em quatro segmentos denotados respectivamente pela apresentação do quadro 1 ao 29, do 30 ao 44, do 45 ao 89 e, por fim, o quadro 90. A primeira legenda é apresentada simultaneamente ao segundo segmento do vídeo e a segunda legenda ao mesmo tempo que o terceiro.

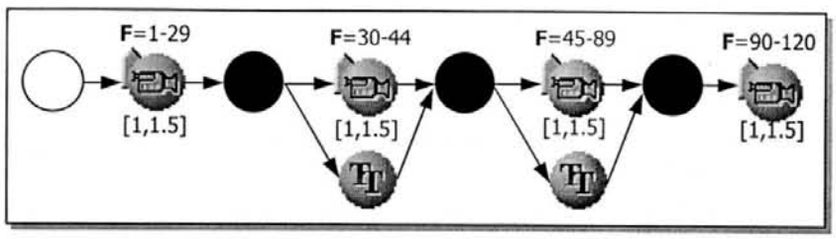


FIGURA 4.10 - Sincronização de um vídeo com legendas

4.3.6 Ícones de composição

Na definição de cenários, existe ainda a possibilidade de se utilizar ícones de composição. Eles completam a funcionalidade do modelo permitindo que parte do comportamento do cenário seja encapsulado em um sub-grafo. Duas são as constribuições de sua utilização:

- reduzir a complexidade dos cenários. Esta é uma característica importante principalmente quando existem componentes multimídia segmentados na definição do cenário. Neste caso, é possível diminuir o número de ícones na representação do cenário (ficam distribuídos em mais de uma janela), facilitando sua compreensão.
- a utilização de um ícone de composição associado a um ponto de sincronização permite representar, por exemplo, que um sub-grafo inteiro possa ser desabilitado (independente do ponto em que esteja executando) no disparo da transição. Esta funcionalidade aumenta sensivelmente o grau de expressão do modelo.

É importante salientar que os ícones de composição não foram implementados no ambiente MUSE.

4.4 Sincronização espacial

A sincronização espacial permite ao autor visualizar o posicionamento dos componentes visíveis de um cenário. Não é possível realizar a sincronização espacial considerando um tempo determinado transcorrido após o início de sua execução. Isto se deve ao fato de que em cada uma das execuções da aplicação, devido às variações temporais admissíveis, os componentes podem ser apresentados em instantes diferentes. Por esta razão, a sincronização espacial é realizada sempre com relação a apresentação de um componente. A disposição espacial dos componentes do cenário Cena1 (figura 4.1) durante a apresentação de Intro, por exemplo, permitirá organizar apenas o próprio componente Intro. Se na definição deste cenário houvesse outros componentes com apresentação simultânea a Intro, estes também seriam apresentados nesta visão. A visão espacial do mesmo cenário durante a apresentação de Máquinas, por sua vez, apresentará apenas o componente Máquinas.

4.5 Um exemplo de utilizaçã o do modelo

O exemplo ilustrado na figura 4.11 modela uma aplicação apresentada em [BLA96], onde inicialmente um vídeo (VD) e um áudio (AU) são apresentados simultaneamente. A seguir, é apresentada a repetição de uma interação gravada do usuário (RI), uma seqüência de três *slides* (P1-P3) e uma animação (ANI) que é parcialmente comentada por uma seqüência de áudio (Áudio2). Após o seu início, uma questão de múltipla escolha é apresentada ao usuário (Interação). Caso o usuário faça a seleção, uma imagem final (P4) é apresentada. Naturalmente, esta é apenas uma das diversas formas de representar essa aplicação. A facilidade no entendimento da mesma é obtida principalmente pelo bom senso do usuário no momento da especificação. Na figura a seguir, as durações associadas aos ícones são fictícias; o exemplo original é apresentado sem a definição das mesmas.

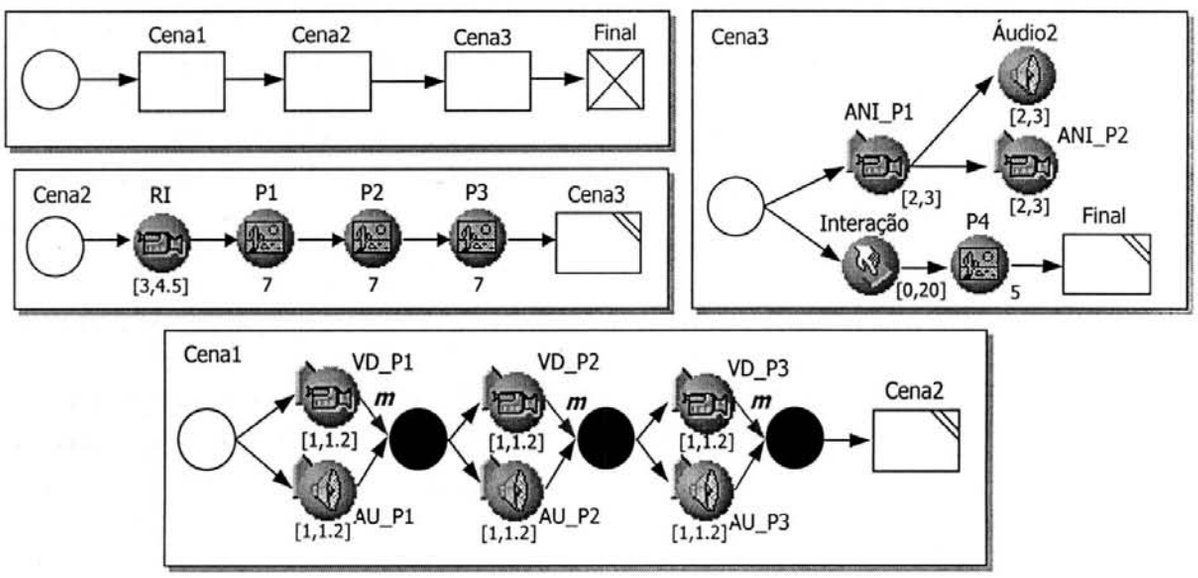


FIGURA 4.11 - Representação de um exemplo completo

5 Tradução das Especificações para E-LOTOS

A formalização das especificações é fundamental no processo de especificação e validação das mesmas. O modelo de autoria proposto, em decorrência de sua alta flexibilidade e expressividade, permite a definição de cenários incoerentes sob o ponto de vista temporal. O processo de análise permite detectar, por exemplo, conflitos na utilização de recursos ou até mesmo identificar a viabilidade ou não de se chegar ao final da aplicação, considerando todos os caminhos (navegação) possíveis.

O processo especificação-verificação, conforme mencionado na introdução deste trabalho, é realizado com base no suporte de uma técnica de descrição formal. As aplicações criadas pelos autores são mapeadas para uma TDF, analisadas através de técnicas de simulação e verificação, e os resultados obtidos devem ser apresentados ao usuário, permitindo a ele realizar as modificações necessárias para que a aplicação atenda aos requisitos de correção lógica e temporal. Dentro do contexto do projeto DAMD, optou-se pela TDF E-LOTOS (*Enhancements to LOTOS*) [ISO97] para representar formalmente as aplicações multimídia interativas.

E-LOTOS é uma versão aprimorada de LOTOS, proposta pela ISO, que se encontra em fase de padronização (*draft proposal*). Suas principais inovações são:

- introdução de uma noção de tempo quantitativo, tornando possível a definição do tempo do qual as ações ou comportamentos poderão ocorrer;
- definição de tipos, utilizando estruturas *record* e *union*, de forma explícita, tornando a declaração mais simples e menor;
- as portas (*gates*) são tipificadas de forma explícita, na declaração de processos;
- possibilidade de modularizar uma especificação, através de definições de módulos e interfaces, sendo que um módulo pode conter definições de tipos, funções e processos; e
- definição de novos operadores, como o operador *case*, o qual permite a definição explícita de seleções.

A seguir são ilustrados os operadores da linguagem utilizados para modelar as aplicações multimídia interativas. Após, apresenta-se a forma de representação e tradução destas aplicações em E-LOTOS.

5.1 Características e operadores da TDF E-LOTOS

E-LOTOS é uma linguagem em dois níveis: o nível inferior, representado pela linguagem base (*Base Language*) e o nível superior que trata da linguagem modular (*Modular Language*). A linguagem base concentra toda a base formal de um grande conjunto de operações primitivas e dos dados; a linguagem modular, por sua vez, foi criada com o objetivo de melhorar a forma limitada de módulos presente na linguagem LOTOS. A idéia é fazer com que os módulos utilizados na parte de dados sejam os mesmos utilizados na parte comportamental, ou seja, será permitido definir, dentro de um módulo, tipos de dados, funções e processos.

O número de operadores existentes em E-LOTOS é elevado. Verificou-se, contudo, que apenas parte deles é necessária para representar aplicações multimídia interativas. Nesta seção serão apresentados apenas os operadores utilizados para representar esta classe de aplicações. Os exemplos foram extraídos de [MAR97].

5.1.1 Declaração de tipos de dados

Declarar um tipo pode ser simplesmente utilizar um novo identificador para um tipo pré-existente. Deste modo,

```
type <identificador de tipo> is
  <expressão de tipo>
endtype
```

define um novo tipo de dado, como no seguinte exemplo:

```
type point is
  (x⇒float,y⇒float)
endtype
```

No exemplo anterior, *point* é um novo identificador de tipo. Ele define as coordenadas de um ponto em um plano através de um registro (*record*) de duas variáveis do tipo real (*float*). Este tipo faz parte dos tipos primitivos de dados utilizados na definição de tipos do usuário (*bool*, *int*, *float*, *string*).

Uma expressão de tipo também pode ser um simples identificador de tipo, onde há a renomeação pura de um tipo de dado. Existem ainda os tipos vazio (*none*) e universal (*any*). O tipo vazio não possui nenhum valor e é utilizado para indicar a funcionalidade de processos que nunca terminam. Por outro lado, o universal é utilizado principalmente com portas, indicando que a comunicação pode ser realizada com dados de qualquer tipo.

Registros são estruturas que agrupam um conjunto de campos, que podem ser relacionados a diferentes tipos de dados. No exemplo anterior, pode-se observar que a expressão de tipos era um registro que continha um conjunto de dois outros registros de expressões do tipo *float*. Existe também o registro especial *etc*, que é um supertipo para qualquer outro registro. Assim:

```
(name⇒string,etc)
```

é um registro que possui pelo menos um campo (*name*), mas pode ser estendido para outros tipos, como os apresentados a seguir:

```
(name⇒string,age⇒int) ou (name⇒string,age⇒int,etc)
```

que poderá ser estendido novamente.

Ainda em relação à declaração de tipos, existe uma segunda forma que permite definir novos tipos, listando todos os construtores envolvidos:

```
type <identificador de tipo> is
  <identificador de construtor>[(<registro de expressão de tipo>)]
  (|<identificador de construtor>[(<registro de expressão de tipo>)])*
endtype
```

Exemplo:

```
type pdu is
    send(packet,bit) | ack(bit) | error
endtype
```

Agora, o tipo *pdu* é definido como sendo um tipo *union*, possuindo os construtores *send* e *ack*, onde *packet* e *bit* são dois tipos previamente definidos.

Assim, pode-se utilizar um construtor para organizar certos tipos de dados utilizados, como o envio de dados constituídos de um pacote de informações e um bit de controle, ou apenas para representar uma situação (*error*), sem a associação de nenhum tipo de dado.

É possível representar também, através dos construtores, uma recursividade:

```
type IntegerList is
  null | elem(int,IntegerList)
endtype
```

A linguagem base não dispõe de um mecanismo para a definição de tipos parametrizados (tarefa reservada à linguagem modular). O componente de tipos de dados da linguagem LOTOS é bem limitada, permitindo encapsular somente tipos de dados e operações. A linguagem E-LOTOS permite definir os tipos de dados de uma forma mais amigável, sendo possível também a definição de tipos de dados concretos, descrevendo como os valores de dados são representados e como alguns procedimentos associados operam sobre eles. Os tipos de dados abstratos são representados a nível da linguagem modular.

5.1.2 Declaração de processos

Um processo pode ser declarado de duas formas:

```
process <identificador de processo> [[[<identificador de porta>[:<expressão de tipo>]
  (,<identificador de porta>[:<expressão de tipo>])*]]]
  [(<variáveis locais>)]:exit(<expressão de tipo>)
  [raises[<identificador de exceção>[:<expressão de tipo>]
    (,<identificador de exceção>[:<expressão de tipo>])*]]
  is B
endproc
```

ou

```
process <identificador de processo> [[[<identificador de porta>[:<expressão de tipo>]
  (,<identificador de porta>[:<expressão de tipo>])*]]]
  ([in <variáveis locais>][out <variáveis locais>])
  [(<variáveis locais>)]:exit(<expressão de tipo>)
  [raises[<identificador de exceção>[:<expressão de tipo>]
    (,<identificador de exceção>[:<expressão de tipo>])*]]
  is B
endproc
```

A declaração de processos possui uma lista de parâmetros (de entrada e saída), tipo do resultado (indicado através de uma notação *exit*), e uma lista de parâmetros que representam as exceções tipificadas. Além disto, os processos podem ter um comportamento tempo-real e se comunicar através de portas (*gates*). Exemplo:

```
process Buffer [enter:int, out:int]:exit(none) is
  loop forever
    var x:int
  in
    enter?x;out!x
  endloop
endproc
```

O processo acima representa um *buffer* com capacidade unitária, onde um número inteiro é armazenado, através da porta *enter*, podendo ser recuperado a qualquer

momento pela porta *out*. Nota-se que neste caso na há nenhum retorno de resultado (*none*), e o processo nunca termina.

5.1.3 Expressões de comportamento

Dado um sistema, a sua especificação formal na linguagem E-LOTOS representa o seu comportamento, ou seja, a ordem temporal em que os eventos do sistema devem ocorrer. Para tal, a linguagem E-LOTOS possui um conjunto de expressões de comportamento e operadores. A seguir estão relacionadas as expressões de comportamento relevantes à especificação de aplicações multimídia interativas.

a) Ações observáveis

A estrutura geral de uma ação especificada em E-LOTOS é a seguinte:

<porta>[<parâmetro(s) da porta>][@<variável ou constante de tempo>]
[[<expressão condicional relacionada à habilitação da porta>]]
[**start**(,constante que representa o tempo de habilitação da porta>)]

ou simplesmente

G[P_1][@P_2][[E]][**start**(N)]

Neste caso, a expressão E é uma expressão booleana que considera no seu contexto os parâmetros P_1 e P_2, onde estes parâmetros são ditos de recebimento e/ou encaixe de dados, e os parâmetros de P_2 são relativos ao instante de ocorrência da ação. Em outras palavras, a ocorrência de uma ação associada à porta G implica no encaixe e/ou recebimento dos valores contidos em P_1 e P_2, além da avaliação da expressão E como sendo verdadeira após N unidades de tempo a partir da habilitação da ação. A não satisfação de alguma destas condições implica na não ocorrência da ação, gerando assim o comportamento *stop*.

b) Atraso

A expressão *wait(<expressão de dados>)* permite a introdução de um atraso de d unidades de tempo, resultante da avaliação da expressão de dados, ou de um atraso não-determinístico, terminado logo em seguida. Exemplo:

wait(3);i;**exit**

O comportamento acima representa o atraso de uma ação interna do sistema, de 3 unidades de tempo. A passagem do tempo irá decrementando o valor em *wait* até que ele seja *wait(0)*, parando de atrasar o processo seguinte.

c) Terminação

Uma terminação é uma ação de controle interno à álgebra de E-LOTOS, mas pode ser declarada explicitamente. Exemplo:

req_disrupt ; end ; **exit**

Após a ocorrência das ações *req_disrupt* e *end*, o processo é terminado.

d) Composição Seqüencial

A composição seqüencial

<expressão de comportamento>;<expressão de comportamento>

é semelhante ao operador de habilitação da linguagem LOTOS (*enable*), pois combina duas expressões de comportamento. No entanto, ela não gera uma ação interna no momento da habilitação da segunda expressão. Além disto, engloba o operador de prefixação da linguagem LOTOS, já que ações são comportamentos. Assim:

wait(3);?t:=3;I;P[...];**exit**

é uma seqüência, onde ocorrerá primeiramente um atraso, seguido de uma atribuição, de uma ação interna, do comportamento *P*, e da expressão *exit*. Portanto, o que inicia um comportamento será sempre a ação de terminação do comportamento anterior.

e) Composição paralela

A composição paralela entre comportamentos pode ser feita através de três formas:

<expressão de comportamento> | | <expressão de comportamento>

conhecida como sincronização completa,

<expressão de comportamento>
|[[<identificador de porta>(,<identificador de porta>)*]]|
<expressão de comportamento>

representando a concorrência e, por último, a forma

<expressão de comportamento> | | | < expressão de comportamento>

que é o entrelaçamento.

A sincronização completa estabelece que o conjunto de ações a serem sincronizadas será o conjunto de todas as ações possíveis. Exemplo:

a;b;B_1 | | a;b;B_2

Neste caso, as ações *a* e *b* ocorrerão nesta ordem, dentro dos comportamentos envolvidos, transformando-se no comportamento $B_1 \,||\, B_2$.

As ações internas e as exceções são as únicas não-sincronizáveis. Elas ocorrem de forma entrelaçada e a passagem do tempo ocorre para os dois comportamentos ao mesmo tempo.

A concorrência se difere da sincronização completa apenas por definir o conjunto de ações a serem sincronizadas. Já o entrelaçamento não permite a sincronização de nenhuma ação observável, e pode ser entendido como sendo uma concorrência, onde o conjunto de ações a serem sincronizadas é vazio.

Com isto, levando em consideração a questão do tempo, pode-se sincronizar ações do tipo $a@?t_1[E_1]$ e $a@?t_2\ [E_2]$, que serão sincronizadas somente quando as expressões E_1 e E_2 forem verdadeiras, simultaneamente. Outros tipos de sincronismo seriam entre as ações $a@!t_1$ e $a@?t_2$,que ocorrerão somente se $t_1=t_2$, e entre as ações $a(start(t_1))$ e $a(start(t_2))$, que ocorrerão em um tempo d tal que $d \geq t1$ e $d \geq t2$.

A linguagem E-LOTOS também possui um operador de paralelismo genérico que permite uma sincronização explícita de vários comportamentos, representado por

```
par [<ident. de porta>#<expressão de valor>
(,<ident. de porta>#<expressão de valor>)* ]
[[<ident. de porta>(,<ident. de porta>)*]]→<expressão de comportamento>
(|| [[<ident. de porta>(,<ident. de porta>)*]]→<expressão de comportamento>)*
endpar
```

ou ainda por

```
par <parâmetros de sincronização> in
<expressão de valor> ||| <expressão de comportamento>
endpar
```

Exemplo:

```
par G1#n1,...,Gp#np in
  [Γ1]→B1
  || ...
  || [Γn]→Bn
endpar
```

Assim, se o comportamento B_i puder executar uma ação associada à porta G, e se esta porta pertencer ao conjunto de portas G_i (chamado de lista de sincronização), então esta ação deverá ser sincronizada com as ações de outros comportamentos, feita da seguinte forma:

- se a porta G pertencer a lista $G_1\#n_1$,...,$G_p\#n_p$ e possuir um grau n $(G\#_n)$, então o comportamento B_i tem que ser sincronizado com $n-1$ outros comportamentos que contenham a porta G em suas listas de sincronização;
- se a porta G não pertencer a lista $G_1\#n_1$, ..., $G_p\#n_p$, então o comportamento B_i deverá ser sincronizado com todos os comportamentos que contenham a porta G em suas listas de sincronização.
- Por outro lado, se a porta G não pertencer ao conjunto G_i, então o comportamento B_i pode entrelaçar a ação com os outros comportamentos.

f) Escolha

O operador de escolha não determinista entre dois processos B_1 e B_2 é representado por $B_1[]B_2$. O processo se comporta como B_1 ou B_2, com a escolha sendo feita no instante da realização da primeira ação (porta, exceção ou ação interna) de um dos processos. Exemplo:

```
a;P1[...][]wait(3);b;P2[...]
```

Neste caso, a ocorrência da ação a resolve a escolha, onde $P_1[...]$ será o comportamento que sucede está ação. No entanto, após 3 unidade de tempo, a ação b poderá ocorrer; somente a passagem do tempo (*wait(3)*) não resolverá o processo de escolha.

g) Preempção

A expressão $B_1[>B_2$, conhecida também como desabilitação, representa a possibilidade de um comportamento B_1 ser desabilitado por um comportamento B_2. Se B_1 terminar, não existirá mais possibilidade de B_2 evoluir, mas durante a evolução de B_1 se ocorrer a primeira ação (porta, exceção ou ação interna) de B_2, o comportamento do processo composto evolui para o de B_2.

h) Laço

A forma mais comum de representar a expressão *loop* é a seguinte:

```
loop forever
  [var<variáveis locais>]
  [init<expressão de comportamento>]
  in
    <expressão de comportamento>
endloop
```

Com essa estrutura tem-se uma outra forma de se resolver problemas que antes só eram possíveis mediante a recursividade. Esse tipo de estrutura aproxima de E-LOTOS das linguagens imperativas.

i) Ocultação

O operador *hide* permite ocultar as portas listadas, tornando-as internas quando de suas realizações. Todas as portas ocultadas tornam-se urgentes. A diferença em relação à sintaxa de LOTOS está nas portas, que agora são tipadas.

```
hide<identificador de portas>[<expressão de tipo de registro>], ... in
  <expressão de comportamento>
endhide
```

Exemplo:

```
hide G in
  wait(1);G;B1 | | wait(2);G;B2
endhide
```

Neste exemplo simples, a ação representada a partir da porta G será substituída pela ação interna i. Portanto, ele possui a mesma semântica que

```
wait(2);I;hide G in B1 | | B2 endhide
```

j) Variáveis Locais

A expressão

```
local
  var<variáveis locais>
[init<expressão de comportamento>]
in<expressão de comportamento>
endloc
```

é utilizada para restringir o escopo de variáveis, declarando variáveis locais para um determinado comportamento. Exemplo:

```
local
        var less: intlist, gtr: intlist
init
        less:= smaller(xs,x);
        gtr:= greater (xs,x)
in
        B
endloc
```

O campo *init* é reservado para a inicialização das variáveis locais, caso seja necessário. Assim, as variáveis *less* e *gtr* serão válidas somente para o comportamento *B*.

5.1.4 Declaração de módulos

Um módulo é uma seqüência de declarações de tipos, construtores, valores constantes e processos, utilizando-se de todos os recursos da linguagem base. Assim, um módulo pode ser declarado da seguinte forma:

module<identificador de módulo> **is** <declarações> **endmod**

onde as declarações são feitas através das declarações da linguagem base, enriquecidas com uma forma de declaração de valores constantes, que é apenas uma função.

value <identificador de módulo>[:<expressão de tipo>] **is** <expressão de dados> **endval**

5.2 Representação em E-LOTOS de aplicações multimídia interativas

Baseado nas características e funcionalidades da TDF escolhida e considerando o modelo de autoria oferecido ao usuário, estudou-se como representar as aplicações multimídia interativas de maneira sistemática em E-LOTOS. Na realidade, dois fatores foram norteadores neste estudo: por um lado, se a representação das aplicações fosse simplificada em demasia, seria impossível realizar as etapas de verificação e validação de forma consistente. Por outro lado, se fosse adotada uma forma de representação muito extensa, o processo de mapeamento das aplicações descritas no modelo de autoria para E-LOTOS se tornaria muito complexo, talvez até inviável de ser implementado.

A solução encontrada, de certo modo, aproveita a estrutura hierárquica provida pelo modelo de autoria ao considerar suas quatro entidades essenciais: aplicação, grupo, cena e componente multimídia. Todas estas entidades são descritas como *processos* que evoluem de acordo com as relações de sincronização previamente estabelecidas entre elas. No caso de um grupo, por exemplo, o final da apresentação de um cenário dá início a apresentação de um outro. O mesmo ocorre quando o elemento considerado é um componente multimídia; o final da apresentação de um vídeo acarreta na apresentação de um outro elemento como áudio, vídeo ou texto. A hierarquia se reflete na composição destes elementos; o processo que representa a aplicação instancia o grupo inicial. Este, por sua vez, pode instanciar vários cenários. Os cenários, por fim, instanciam componentes multimídia e restrições. Este trabalho

baseou-se em uma uma abordagem apresentada em [COU96]. Detalhes sobre a forma de representação das aplicações são apresentados nos itens seguintes.

5.2.1 Estruturação do processo raiz e declaração dos dados

A estruturação da aplicação se dá a partir da instanciação do processo raiz. A figura 5.1 apresenta o esqueleto, em E-LOTOS, da aplicação apresentada anteriormente na figura 4.11. Grande parte dos exemplos apresentados neste capítulo são originários desta figura. Inicialmente são importados três módulos (1) cujas funções serão vistas nos itens a seguir. Na definição dos pontos de ligação da aplicação com o ambiente externo (2) aparecem:

- as portas de início e fim da aplicação (i_Aplicação, f_Aplicação); a aplicação é iniciada pelo evento i_Aplicação proveniente do ambiente; ao ser encerrada, ela sincroniza com ele no evento f_Aplicação;
- as interações do usuário (Interação); todas as interações do usuário são modeladas como eventos provenientes do ambiente. Por esta razão, para cada interação presente na especificação existe uma porta representando este elo de ligação;
- a porta Dado, utilizada para representar a execução dos objetos multimídia da aplicação.

```
specification Aplicação
  import processos, classes, midias                                    (1)
  [i_Aplicação,f_Aplicação,Interação, Dado:class]                      (2)
  behavior
  local var
    RI:StreamClass,d1:Time,d2:Time,                                    (3)
    P1:BitmapClass,dP1:Time,
    P2:BitmapClass,dP2:Time,
    P3:BitmapClass,dP3:Time
    ...
  init
    ...
    ?P1:=((Teste,6),                                                   (4)
          content⇒(content_reference⇒http://www.inf.ufrgs.br/test.gif),
          original_box_size⇒(1000,750),
          original_position⇒(50,50))
    ...
  in
    GrupoInicial[i_Aplicação,f_Aplicação,Interação,Dado]               (5)
      (...,RI,d1,d2,P1,dP1,P2,dP2,P3,dP3,...)
end spec

module processos is                                                    (6)
...
endmod
```

FIGURA 5.1 - Esqueleto da estrutura principal de uma aplicação em E-LOTOS

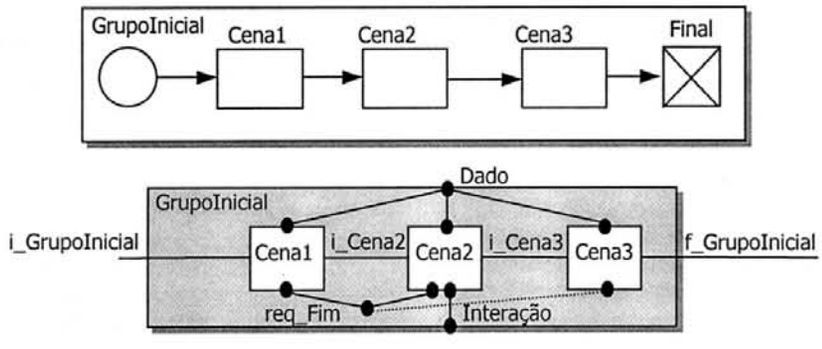
Em seguida, tem-se a definição do comportamento da aplicação, iniciada pela declaração das variáveis utilizadas na mesma (3). Neste trabalho, variáveis são todos os componentes multimídia como áudio, texto, vídeo, botão e imagem. A representação destes elementos encontra-se em uma biblioteca (módulo) denominada *classes* que define tipos de dados para os componentes multimídia suportados pelo ambiente. Assim, tem-se tipos como *BitmapClass*, *StreamClass*, *SwitchButtonClass* cuja definição está baseada em suas respectivas classes MHEG-5. A definição completa desta biblioteca está no anexo A-1 deste trabalho. Uma vez definidas as variáveis, tem-se a iniciação das mesmas (4), quando lhe são atribuídos seus valores

iniciais, provenientes do ambiente de autoria. Na figura 5.1, P1, que é uma imagem, está sendo iniciado com sua localização, bem como com suas dimensões e localização em relação ao dispositivo de saída. A figura não apresenta o início das outras variáveis com o intuito de reduzir o tamanho da especificação e, desta forma, facilitar seu entendimento.

Por fim, o processo inicial (GrupoInicial) é instanciado (5) sendo que todas as variáveis e eventos externos (interação do usuário) são repassados a este processo. Na parte inferior da figura 5.1 pode-se visualizar o módulo processos, que conterá a definição dos grupos, cenários, componentes multimídia e restrições que compõem a aplicação.

5.2.2 Representação de grupos

Grupos são formados por cenários, outros grupos e até mesmo componentes multimídia, os chamados componentes compartilhados. Sua representação é feita com base na instanciação de processos que identificam seus elementos constituintes, considerando o relacionamento existente entre eles. Na figura 5.2 é apresentada a definição do grupo raiz (GrupoInicial), instanciado na figura anterior. Pode-se observar a existência de três partes bem distintas: a primeira apresenta o grupo de acordo com o modelo de autoria, a segunda um diagrama destacando os eventos e pontos relevantes de sua representação em E-LOTOS e a terceira parte reproduz a representação simplificada do grupo utilizando a sintaxe textual da TDF. As próximas figuras apresentadas nesta seção seguem este mesmo padrão.




```
process GrupoInicial[i_GrupoInicial,f_GrupoInicial,Interação,Dado]
  (...,RI:StreamClass,d1:Time,d2:Time,P1:BitmapClass,                                          (1)
   dP1:Time, P2:BitmapClass,dP2:Time,P3:BitmapClass,
   dP3:Time,...):exit is
  hide i_Cena,i_Cena2,i_Cena3, req_Fim in                                                      (2)
    i_GrupoInicial;                                                                            (3)
    par i_Cena2#2,i_Cena3#2                                                                    (4)
      [i_Cena2]→Cena1[i_Cena,i_Cena2,Dado,req_Fim](...)                                        (4)
      [i_Cena2,i_Cena3]→Cena2[i_Cena2,i_Cena3,Interação,Dado,req_Fim](RI,d1RI,d2RI,P1,dP1,P2,dP2,P3,dP3)  (5)
      [i_Cena3]→Cena3[i_Cena3,f_GrupoInicial,Dado,req_Fim](...)                                (6)
    endpar
    [>req_Fim;exit                                                                             (7)
  endhide
endproc
```

FIGURA 5.2 - Modelagem do GrupoInicial em E-LOTOS

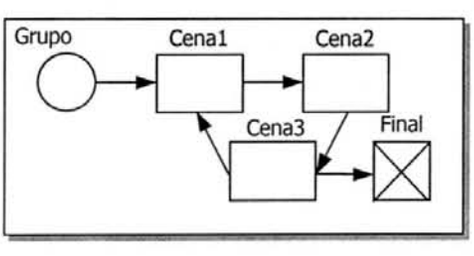
Da figura apresentada acima pode-se extrair vários aspectos importantes, considerados no processo de modelagem de grupos e cenários. Por ser o grupo raiz da

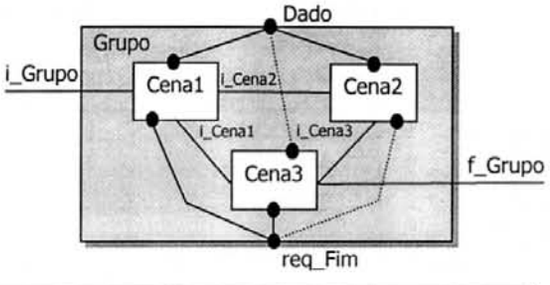
especificação, o processo GrupoInicial recebe todos os componentes multimídia (variáveis) presentes na aplicação (1). Vale lembrar que elas já foram iniciadas com os valores reais antes da instanciação do processo GrupoInicial (vide figura 5.1) e que serão distribuídas aos processos (grupos e cenários) aos quais elas fazem parte. Por simplificação, apenas os dados relativos a Cena2 foram passados ao processo correspondente (5).

Prosseguindo, pode-se observar a utilização do operador de ocultação (2). Alguns eventos como o início de Cena2 (i_Cena2) e Cena3 (i_Cena3) não são visíveis externamente ao processo. Sua utilidade é permitir a sincronização entre os grupos e cenários envolvidos, modelada com a utilização do operador de sincronização *par* (4). Como já mencionado, os eventos relevantes nesta sincronização são i_Cena2 e i_Cena3. Na definição do operador está estabelecido que a ocorrência deste evento deve sincronizar sempre duas cenas entre si (grau da sincronização). Deste modo, por exemplo, o início de Cena2 está associado ao evento de final de Cena1 (i_Cena2) (4 e 5). Da mesma forma, o início de Cena3 está sincronizado com o final de Cena2 (i_Cena3) (5 e 6).

Antes que o operador *par* seja executado, contudo, é necessária a ocorrência do evento que dá início à apresentação do grupo (i_GrupoInicial) (3). A necessidade deste evento será apresentada detalhadamente a seguir. Por enquanto, destaca-se a presença do operador de desabilitação na modelagem do grupo (7). Como pode ser observado, o evento req_Fim abrange todos os processos do mesmo; ele é utilizado para modelar a finalização da aplicação. Ao ser gerado (por Cena3), o grupo e os cenários associados são terminados com *exit*. Como se trata do grupo raiz, a aplicação como um todo é terminada.

O grupo no exemplo anterior modela uma estrutura linear. A figura 5.3 ilustra um grupo hipotético composto de três cenários (Cena1, 2 e 3) estruturados em forma de rede. Num primeiro instante, pode-se observar que não se trata do grupo raiz de uma aplicação; isto fica evidenciado na definição da porta req_Fim que, agora, fica visível ao ambiente (1).



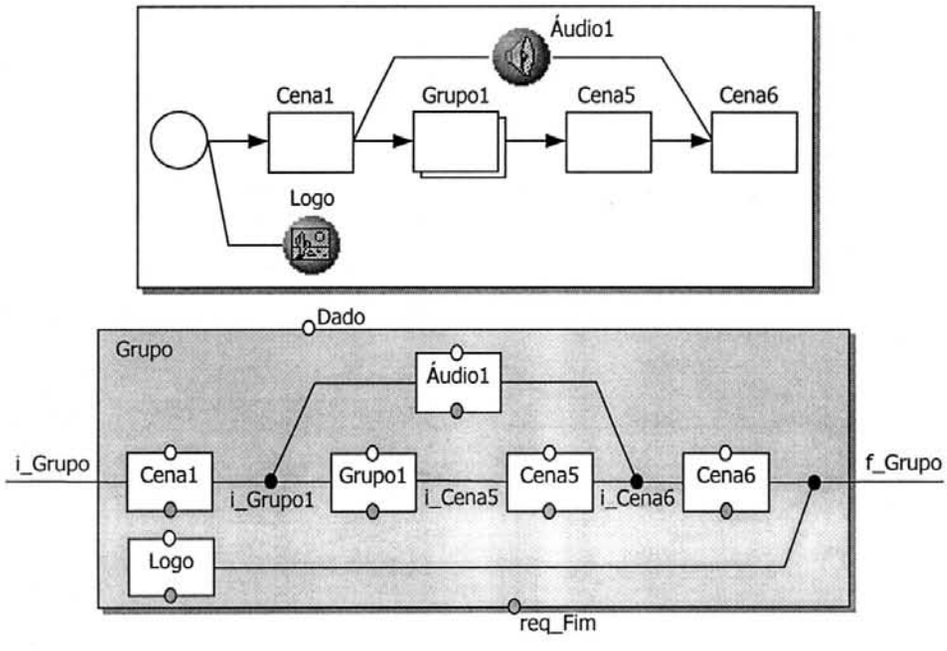



```
process Grupo[i_Grupo,f_Grupo,Dado,req_Fim]:exit is                          (1)
  ...
  hide i_Cena,i_Cena1,i_Cena2,i_Cena3 in
    i_Grupo;
    par i_Cena1#2,i_Cena2#2,i_Cena3#2
      [i_Cena1,i_Cena2 ]→Cena1[i_Cena,i_Cena1,i_Cena2,Dado, req_Fim](...)     (2)
      [i_Cena2,i_Cena3 ]→Cena2[i_Cena2,i_Cena3,Dado, req_Fim](...)
      [i_Cena3,i_Cena1,i_Cena]→Cena3[i_Cena3,f_Grupo,i_Cena1,Dado, req_Fim](...)  (3)
    endpar
    [> req_Fim;exit
  endhide
endproc
```

FIGURA 5.3 - Representação de grupos com estrutura de rede

Mais importante que isto, no entanto, é observar que Cena1 pode ser ativada por dois eventos distintos: i_Grupo, quando o grupo é iniciado pela primeira vez ou i_Cena1, gerado por Cena3 (2). O efeito da existência de dois pontos de entrada é que, na definição do cenário, ele será iniciado por *i_Cena [] i_Cena1*, indicando que evoluirá quando qualquer um dos eventos ocorrer. Do mesmo modo, Cena3 apresenta dois pontos de saída: o primeiro (f_Grupo) encerra a execução do grupo e o segundo (i_Cena1), como já foi mencionado, habilita a reapresentação da Cena1. Neste caso, é impossível estabelecer *a priori* como a aplicação se comportará pois ainda não se conhece a definição das cenas; possivelmente, a decisão entre a repetição da apresentação de um cenário ou o encerramento da aplicação está associado ao pressionamento de botões e o comportamento da aplicação, assim, varia a cada execução.

Antes de apresentar a forma de representação dos cenários, deve-se abordar a modelagem de componentes multimídia compartilhados. Muito pouco é modificado com a adição destes componentes. A figura 5.4 apresenta um grupo onde Áudio1 é executado durante a apresentação de Grupo1 e Cena5, e Logo é apresentado durante toda a aplicação. No diagrama que apresenta as relações entre os processos foram suprimidas as linhas que ligam as portas Dado e req_Fim para facilitar sua compreensão.



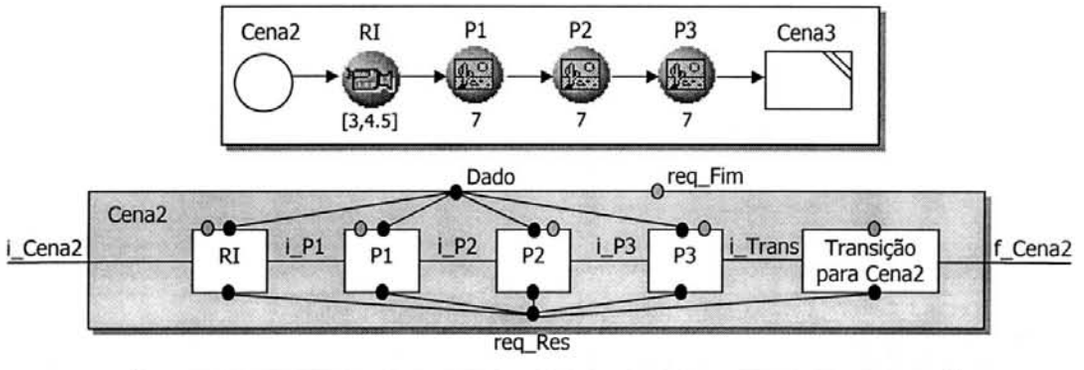

```
process Grupo[i_Grupo,f_Grupo,Dado,req_Fim]:exit is
  ...
  hide i_Logo,i_Cena,i_Grupo1,i_Cena5,i_Cena6 in
    i_Grupo;
    par i_Grupo1#3,f_Grupo#2,i_Cena5#2,i_Cena6#3                    (1)
      [i_Grupo1]→Cena1[i_Cena,i_Grupo1,Dado,req_Fim](...)
      [i_Grupo1,i_Cena5 ]→Grupo1[i_Grupo1,i_Cena5,Dado,req_Fim](...)
      [i_Cena5,i_Cena6]→Cena5[i_Cena5,i_Cena6,Dado,req_Fim](...)
      [i_Cena6, f_Grupo] →Cena6[i_Cena6,fGrupo,Dado,req_Fim](...)
      [i_Grupo1, i_Cena6] →Áudio1[i_Grupo1,i_Cena6,Dado,req_Fim](...)  (2)
      [i_Logo,f_Grupo] →Logo[i_Logo,f_Grupo,Dado,req_Fim](...)        (3)
    endpar
    [> req_Fim;exit
  endhide
endproc
```

FIGURA 5.4 - Representação de componentes compartilhados

Uma das novidades existentes na modelagem de componentes compartilhados diz respeito ao grau da sincronização (1). É possível observar, por exemplo, que Cena1, Grupo1 e Áudio1 sincronizam no evento i_Grupo1 da seguinte forma: o final de Cena1 habilita simultaneamente a execução de Grupo1 e Áudio1. O mesmo ocorre com Áudio1, Cena5 e Cena6 em relação ao evento i_Cena6: Cena6 só pode ser executada após a finalização de Áudio1 e Cena5 (2). Por fim, o grupo só é encerrado (f_Grupo) após o término da apresentação de Cena6 e Logo (3).

5.2.3 Representação de cenários

A modelagem de cenários difere em vários pontos da modelagem dos grupos. Uma das diferenças é que não se instanciam mais processos de cenários e grupos, mas de componentes multimídia e restrições. Outra diferença importante é a utilização do operador *loop* na sua representação. Ele aparece na definição de todos os cenários, permitindo modelar que sua apresentação ocorra mais de uma vez durante a execução da aplicação (estrutura em rede). A figura 5.5 apresenta o cenário Cena2, instanciado anterioremente na figura 5.2. Como eles são reiniciados após cada apresentação (*loop forever*), sua execução nunca termina (2). Assim, o operador de desabilitação (evento req_Fim) também aparece na modelagem dos cenários. Na ocorrência deste evento, o cenário é encerrado com *exit*. A sincronização entre os componentes é realizada da mesma forma como ocorre nos grupos.




```
process Cena2 [i_Cena2,f_Cena2,Dado,req_Fim]
  (RI:StreamClass,d1:Time,d2:Time,P1:BitmapClass,dP1:Time,
   P2:BitmapClass,dP2:Time,P3:BitmapClass,dP3:Time)
  hide i_Cena,i_P1,i_P2,i_P3,req_Res in                          (1)
    loop forever in                                              (2)
      i_Cena2;
      par i_P1#2,i_P2#2,i_P3#2, i_Trans#2
        [i_P1]→RI [i_Cena, i_P1,Dado,req_Fim,req_Res] (RI,d1,d2)
        [i_P1,i_P2]→P1[i_P1,i_P2,Dado,req_Fim,req_Res](P1,dP1))
        [i_P2,i_P3]→P2[i_P2, i_P3,Dado,req_Fim,req_Res](P2,dP2))
        [i_P3,i_Trans]→P3[i_P3,i_Trans ,Dado,req_Fim,req_Res](P3,dP3))
        [i_Trans]→Transição[i_Trans,f_Cena2,req_Fim,req_Res]()
      endpar
    endloop
    [> req_Fim;exit
  endhide
endproc
```

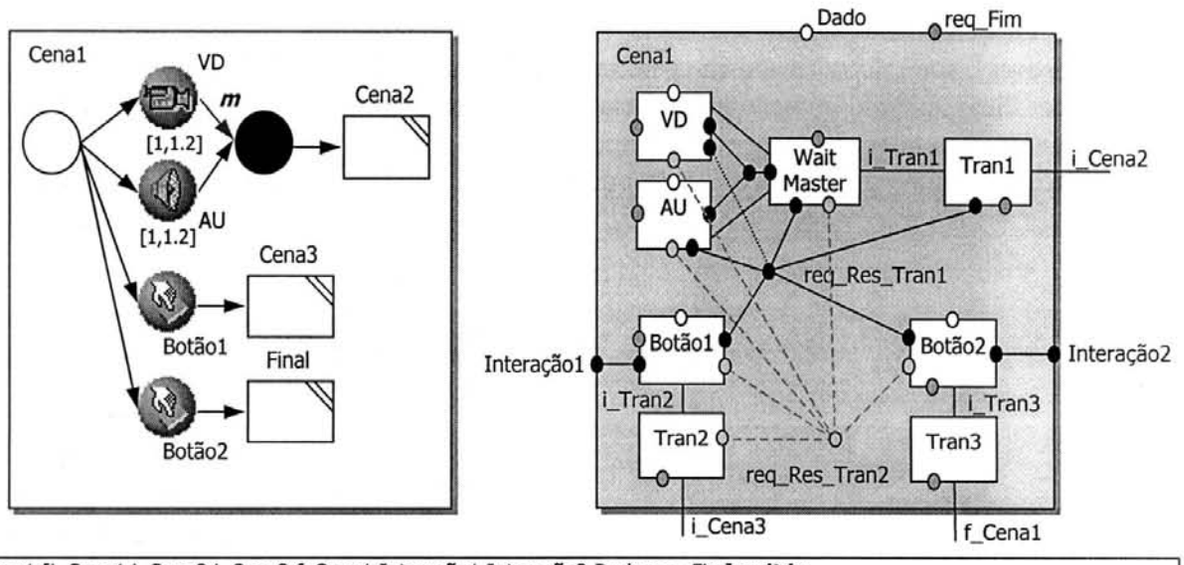
FIGURA 5.5 - Representação de cenários

Na representação de cenários aparece também o evento req_Res. Antes de ocorrer uma transição para outro cenário, é necessário reiniciar o estado de todos os

seus componentes. Eles passam a aguardar pela solicitação de reapresentação do cenário, quando então estarão aptos a evoluir novamente. Deste modo, o código que modela uma transição para outro cenário apresenta apenas três eventos: i_Trans, que denota a ocorrência da transição, req_Res, que provoca o reinício dos componentes e f_Cena, indicando o final da apresentação do cenário (vide figura 5.6). Como a transição também é um processo que está sempre em execução, ela é desabilitada pela ocorrência do evento req_Fim.

```
process Transição [i_Trans,f_Cena,req_Fim,req_Res]:exit is
  loop forever in
    i_Trans;req_Res;f_Cena
  endloop
  [> req_Fim;exit
endproc
```

FIGURA 5.6 - Código de uma transição simples




```
process Cena1 [i_Cena1,i_Cena2,i_Cena3,f_Cena1,Interação1,Interação2,Dado,req_Fim]:exit is
  (...)
  hide i_Cena,i_VD,i_AU,i_Botão1,i_Botão2,ctl_Master,req_Res_Tran1,req_Res_Tran2 in
    loop forever in
      i_Cena1;
      par f_VD#2,f_AU#2,ctl_Master#3,i_Tran1#2,i_Tran2#2
        [f_VD,ctl_Master]→VD[i_VD,f_VD,Dado,req_Fim,ctl_Master,req_Res_Tran1,req_Res_Tran2] (...)
        [f_AU,ctl_Master]→AU[i_AU,f_AU,Dado,req_Fim,ctlMaster, ,req_Res_Tran1,req_Res_Tran2](...)
        [f_VD,f_AU,ctl_Master,i_Tran1]→WaitMaster[f_VD,f_AU,i_Tran1,req_Fim,ctl_Master,req_Res_Tran1,req_Res_Tran2](...)
        [i_Tran1]→Transição1[i_Trans1,i_Cena2,req_Fim,req_Res_Tran1](...)
        [i_Tran2]→Botão1[i_Botão1,i_Tran2,Interação1,Dado,req_Fim,req_Res_Tran1,req_Res_Tran2](...)
        [i_Tran2]→Transição2[i_Trans2,i_Cena3,req_Fim,req_Res_Tran2](...)
        [i_Tran3]→Botão2[i_Botão2,i_Tran3,Interação2,Dado,req_Fim,req_Res_Tran1,req_Res_Tran2](...)
        [i_Tran3]→Transição3[i_Trans3,req_Fim](...)
      endpar
    endloop
    [> req_Fim;exit
  endhide
endproc
```

FIGURA 5.7 - Cenário com mais de um ponto de saída e com restrições

A figura 5.7 apresenta outras situações peculiares à representação dos cenários. A primeira delas diz respeito a existência de mais de uma transição (para

outros cenários) no mesmo cenário; quando isto ocorre, existe um evento req_Res para cada uma delas. Na figura, por exemplo, existem as transições para Cena2 e Cena3. No instante em que uma delas é executada, todos os componentes e restrições são reiniciados. Caso ocorra uma transição para Cena2, eles serão reiniciados pela ocorrência do evento req_Res_Tran1. Por outro lado, caso a transição seja para Cena3, eles serão reiniciados pelo evento req_Res_Tran2.

Nesta figura aparece também uma transição para o final da aplicação. Neste caso, sua representação é diferente da apresentada anteriormente. Agora, o evento req_Fim é gerado e todos os componentes multimídia, cenários e grupos da aplicação são encerrados (vide figura 5.8FIGURA 5.8).

```
process Transição [i_Trans,f_Cena,req_Fim,req_Res]:exit is
    i_Trans,req_Fim;f_Cena;exit
endproc
```

FIGURA 5.8 - Código de uma transição simples

É importante abordar também as restrições. Elas necessitam de uma porta para controlar a apresentação dos componentes que os antecedem (ctl_master); quando a transição está pronta para ser disparada (requisitos foram satisfeitos), este evento é utilizado para desabilitar a apresentação dos componentes que estão associados à transição; na verdade, eles não são encerradas, apenas reiniciados do mesmo modo que acontece na ocorrência dos eventos req_Res.

5.2.4 Representação de componentes multimídia e restrições

Como foi comentado a pouco, os cenários instanciam processos que representam a apresentação de mídias e transições. No primeiro caso, esta ação se repete pois, por exemplo, imagem e texto apresentam comportamentos semelhantes. Esta repetição é ainda maior quando se tratam de dois objetos multimídia da mesma categoria; a maneira de modelá-los é sempre a mesma, mudando apenas parâmetros relativos à sua apresentação. Em [HIR95] estes comportamentos de objetos simples são apresentados como objetos básicos ou monolíticos.

Estes objetos são definidos pela ocorrência de eventos síncronos (início e fim) e assíncronos (interação do usuário). Diversas combinações destes eventos podem ser formuladas, mas apenas oito são apontados como relevantes na definição de cenários multimídia interativos. Neste trabalho, são utilizadas três destas combinações (vide tabela 5.1). O quarto objeto apresentado nesta tabela (pSsSe - *Synchronous start Synchronous end*) não é apresentado em [HIR95]. Ele permite modelar componentes multimídia dependentes do tempo como vídeo e áudio, que podem ter duração mínima e máxima. Na definição dos processos foi utilizada a porta Dado para representar a apresentação do componente multimídia. Estes processos compõem a já citada biblioteca *mídias*.

TABELA 5.1 - Objetos monolíticos utilizados para representar objetos multimídia

Objeto	Código E-LOTOS	Descrição
	Synchronous start Synchronous end	
	**process** pSsSe[Início,Fim,Dado:class] (Mídia:class,d:time)**:exit is** Início;Dado(!Midia);**wait**(d);Fim@t[t=0];**exit** **endproc**	Utilizado para modelar componentes independentes do tempo (imagem, texto) com duração previamente definida.
	Synchronous start Asynchronous maximum end	
	**process** pSsAme[Início,Fim,Usuário,Dado:class] (Mídia:class,d1,d2:time)**:exit is** Início; Dado(!Mídia);**wait**(d1) ; (Usuário@t[t<=d2][]**wait**(d2);**exit**);Fim@t[t=0];**exit** **endproc**	Utilizado para modelar a interação do usuário; mesmo que a interação não ocorra no intervalo [d1,d2] o processo é terminado após o tempo máximo (d2).
	Synchronous start Asynchronous end	
	**process** pSsAe[Início,Fim,Usuário,Dado:class] (Mídia:class,d:time)**:exit is** Início;Dado(!Mídia);**wait**(d);Usuário;Fim@t[t=0]**;exit** **endproc**	Utilizado para modelar a interação do usuário, sem um tempo máximo de espera definido; o processo só termina após a interação.
	Synchronous start Synchronous maximum end	
	**process** pSsSme[Início,Fim,Dado:class] (Mídia:class,d1,d2:time)**:exit is** Início;Dado(!Mídia);**wait**(d);end@t[t≤d2]**;exit** **endproc**	Utilizado para modelar componentes dependentes do tempo como vídeo e aúdio que possuem uma duração mínima e máxima.

O comportamento repetitivo existente em relação aos componentes também ocorre com as restrições. O modelo de autoria e conseqüentemente o próprio ambiente, descrito na próxima seção, permitem o estabelecimento de três tipos de restrições temporais: Wait Master, Wait Latest e Wait Earliest. A representação E-LOTOS destas restrições pode ser visualizada na tabela 5.2. As restrições, ao contrário dos componentes, não são implementadas em bibliotecas, pois o comportamento das mesmas é diferente dependendo do número de componentes que convergem para a restrição.

TABELA 5.2 - Restrições descritas em E-LOTOS

WaitMaster

```
process WaitMaster[f_A,f_B,f_M,f_Restrição,req_Fim,ctl_Master,req_Res]:exit is
  loop forever in
    (f_A|||f_B)[>f_M;ctl_Master;f_Restrição@t[t=0]
    [>req_Res
  endloop
  [>req_Fim;exit
endproc
```

WaitLatest

```
process WaitLatest[f_A,f_B,f_C,f_Restrição,req_Fim,req_Res]:exit is
  loop forever in
    (f_A|||f_B|||f_C);f_Restrição @t[t=0]
    [>req_Res
  endloop
  [>req_Fim;exit
endproc
```

WaitEarliest

```
process WaitEarliest[f_A,f_B,f_C, f_Restrição,req_Fim,req_Res]:exit is
  loop forever in
    (f_A[]f_B[]f_C);f_Restrição@t[t=0]
    [>req_Res
  endloop
  [>req_Fim;exit
endproc
```

A figura 5.9 ilustra a representação do componente P2, presente na definição do cenário da figura 5.3. Também na definição do processo de componentes multimídia aparece o evento req_Fim. Assim como os grupos e cenários, elas estão

sempre em execução (1); a justificativa para isto é a possibilidade de haver laços na definição do cenário. Neste caso, em uma única execução do cenário, os componentes podem ser executados mais de uma vez.

```
process P2 [i_P2,f_P2, Dado,req_Fim,req_Res]:exit is
  (P2:BitmapClass,dP2:Time)
  loop forever in                                        (1)
    pSsSe[i_P2, f_P2, Dado] (P2,dP2)
    [>req_Res
  endloop
  [> req_Fim;exit
endproc
```

FIGURA 5.9 - Representação de objetos multimídia simples

O componente P2 acima pertence a um cenário com apenas uma transição; é importante visualizar também como é a representação dos componentes que pertencem a um cenário com mais de uma transição. Esta ilustração é apresentada na figura 5.10 que apresenta Botão1, um componente que pertence a um cenário com duas transições (vide figura 5.7). É possível observar que o processo é reiniciado após a ocorrência de qualquer um dos eventos req_Res_Tran1 ou req_Res_Tran2 (1), correspondentes às duas transições.

```
process Botão1 [i_Botão, f_Botão, Interação1,Dado,req_Fim,req_Res_Tran1,req_Res_Tran2]:exit is
  (...)
  loop forever in
    pSsAe[i_Botão, f_Botão, Interação1,Dado] (...)
    [>req_Res_Tran1[]req_Res_Tran2                       (1)
  endloop
  [> req_Fim;exit
endproc
```

FIGURA 5.10 - Objetos multimídia em cenários com mais de uma transição

Por fim, basta ilustrar o processo de um componente multimídia que participa de um ponto de sincronização, como VD e AU na figura 5.7. Sua modelagem considera o evento *ctl_master* (1), que controla a apresentação dos componentes envolvidos nesta sincronização. Ele é adicionado aos eventos que reiniciam o processo (vide figura 5.11).

```
process VD [i_VD,f_VD,Dado,req_Fim,ctl_Master,req_Res_Tran1,req_Res_Tran2]:exit is
  (...)
  loop forever in
    pSsAe[i_P2, f_P2, Interação1,Dado] (...)
    [>req_Res_Tran1[]req_Res_Tran2[]ctl_Master           (1)
  endloop
  [> req_Fim;exit
endproc
```

FIGURA 5.11 - Objetos multimídia envolvidos com pontos de sincronização

6 O Ambiente MUSE

O ambiente desenvolvido suporta completamente o modelo de autoria proposto no capítulo 4. Ao autor é oferecida uma interface orientada a objetos, objetivando agregar facilidade de uso a poder de expressão. Na realidade, os sistemas de autoria baseados em objetos são os primeiros a obter sucesso na tentativa de reunir estes dois conceitos. Nas linguagens de programação e nos sistemas de autoria convencionais eles estabelecem entre si uma relação inversa onde, na maioria das vezes, sistemas com um elevado poder de expressão são de difícil utilização (linguagens de programação, por exemplo) e vice-versa. A figura 6.1 reproduz um gráfico apresentado em [CHA97] ilustrando o que foi comentado. Os conceitos de orientação a objetos embutidos no ambiente serão apresentados nos tópicos a seguir.

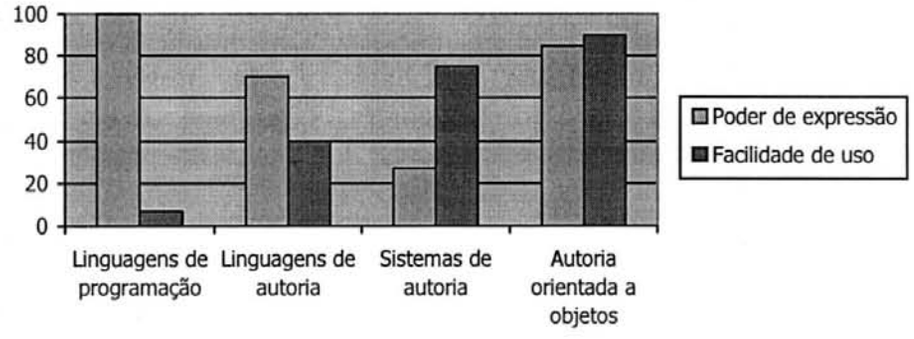


FIGURA 6.1 - Poder de expressão *vs.* facilidade de uso em ambientes de autoria

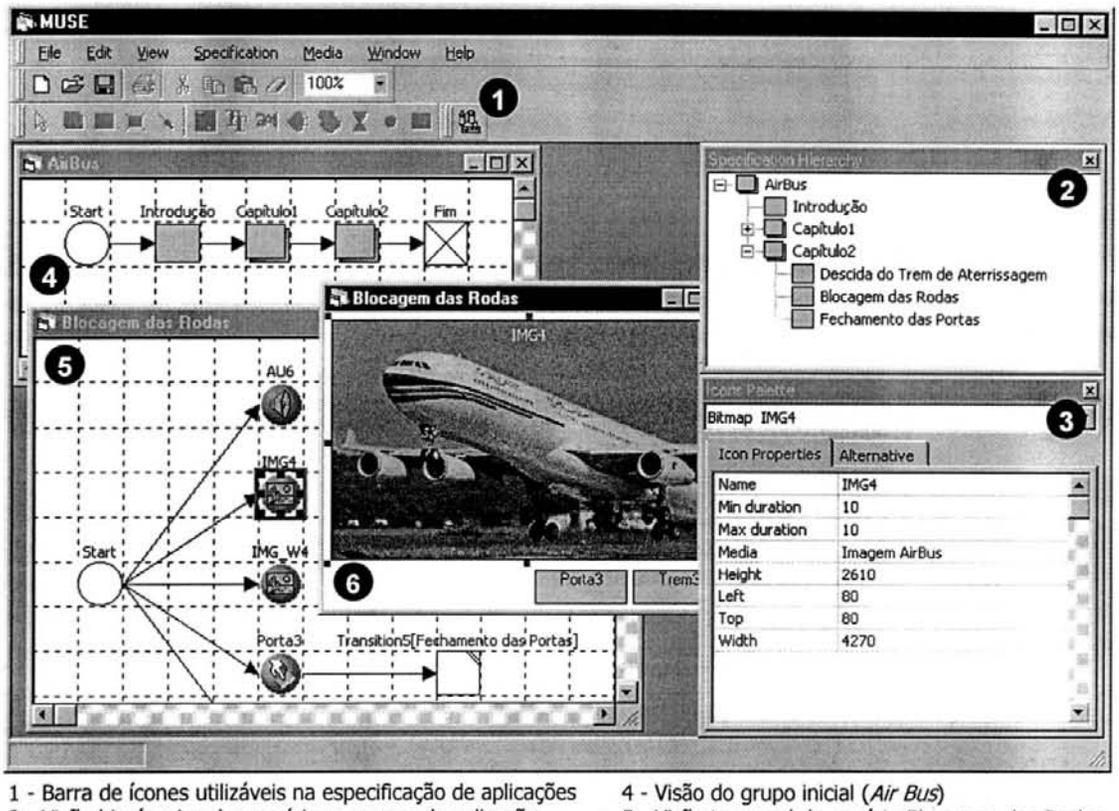
6.1 Funcionalidades básicas

As funcionalidades de MUSE são categorizadas em torno de duas unidades: o repositório de componentes multimídia e a área de especificação propriamente dita. A primeira delas provê mecanismos para adicionar e gerenciar componentes multimídia utilizados na aplicação. A segunda, por sua vez, é composta por diversas janelas que permitem ao autor definir o comportamento da mesma, considerando sua estrutura lógica, bem como as relações temporais e espaciais estabelecidas entre os elementos que dela fazem parte.

A figura 6.2 apresenta a interface do ambiente. Na barra superior (1) pode-se visualizar as opções disponíveis ao usuário para especificar sua aplicação. Pode-se verificar ainda a presença das janelas *Specification Hierarchy* (2) e *Icons Palette* (3) que permitem visualizar, respectivamente, a estrutura lógica da aplicação e as propriedades do ícone selecionado. Na mesma figura, pode-se observar parte da aplicação denominada *AirBus*. Em (4) é apresentado o grupo inicial da aplicação (*AirBus*). A visão temporal do cenário Blocagem das Rodas, integrante do grupo Capítulo2 (2), é apresentada em (5). A visão espacial deste mesmo cenário pode ser visualizada em (6).

No parágrafo anterior destacou-se os componentes essenciais de MUSE. Sob o ponto de vista prático, a criação de aplicações neste ambiente ocorre basicamente pela manipulação de ícones. Ao iniciar uma nova especificação um grupo vazio (raiz) é apresentado; nele, o autor pode inserir ícones de grupo ou cenário, por exemplo,

simplesmente selecionando estes elementos na barra superior (1) e pressionado o botão esquerdo do *mouse* sobre o grupo. A visão hierárquica (2) é atualizada automaticamente quando ícones de grupo ou de cenários são adicionados ou retirados da especificação. Ao selecionar-se um ícone, suas respectivas propriedades são apresentadas na *Icons Palette* (3), possibilitando que os valores associados a elas sejam modificados de acordo com o desejo do autor. Vale ressaltar, ainda, que a navegação entre grupos e cenários pode ser realizada tanto pelo duplo clique no ícone indicativo de um grupo ou cenário, como diretamente a partir da visão hierárquica. A seguir, são detalhadas as funcionalidades do ambiente.



1 - Barra de ícones utilizáveis na especificação de aplicações
2 - Visão hierárquica dos cenários e grupos da aplicação
3 - Propriedades de apresentação do ícone selecionado (IMG4)
4 - Visão do grupo inicial (*Air Bus*)
5 - Visão temporal do cenário Blocagem das Rodas
6 - Visão espacial do cenário Blocagem das Rodas

FIGURA 6.2 - Interface gráfica do ambiente de autoria

6.1.1 Repositório de componentes multimídia

Como já foi mencionado, o repositório provê mecanismos para colecionar objetos multimídia utilizados na aplicação. A qualquer momento o usuário pode adicionar componentes ao repositório. Isto é feito a partir da localização de um componente local ou da criação de uma referência a um componente remoto. A figura 6.3 apresenta a janela *New Media* que permite adicionar novos componentes ao repositório. URL indica o endereço onde o componente pode ser encontrado. *Name* permite ao autor associar um nome fantasia ao componente adicionado permitindo a ele identificá-lo facilmente durante o processo de especificação. A tabela 6.1 apresenta as categorias de componentes multimídia suportados pelo ambiente.

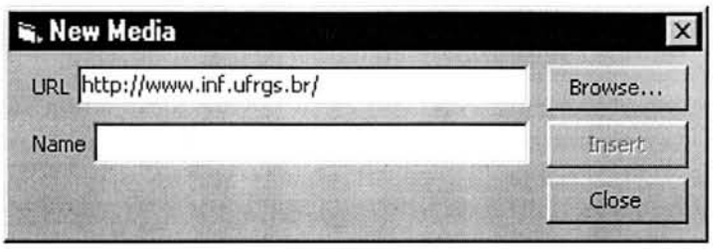


FIGURA 6.3 - Adição de componentes ao repositório

TABELA 6.1 - Componentes multimídia suportados pelo ambiente

Categoria do componente	Formatos suportados
Imagem	bmp, gif, jpg, pcx
Texto	txt
Vídeo	avi
Áudio	au, wav

Tão logo um componente tenha sido adicionado ao repositório ele pode ser gerenciado e acessado pelo autor através da janela *Medias Palette*, apresentada na figura 6.4. Para tal, o nome fantasia é utilizado.

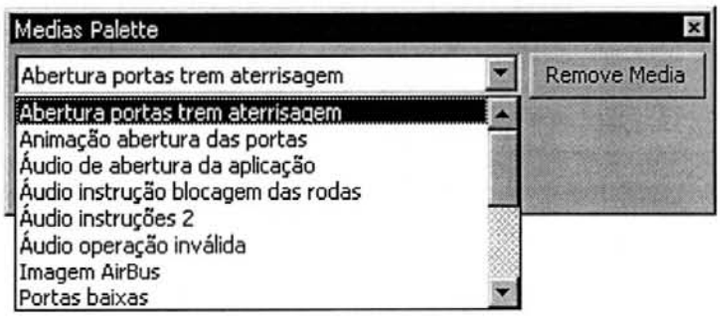


FIGURA 6.4 - Gerenciamento dos componentes da aplicação

Uma vez realizada a seleção de um componente, são apresentadas suas respectivas propriedades de conteúdo como sua localização e categoria, por exemplo (vide figura 6.5). Através do botão *Remove Media* é possível descartar componentes da aplicação. Quando isto ocorre, se ele estiver sendo referenciado na especificação, esta associação é automaticamente desfeita.

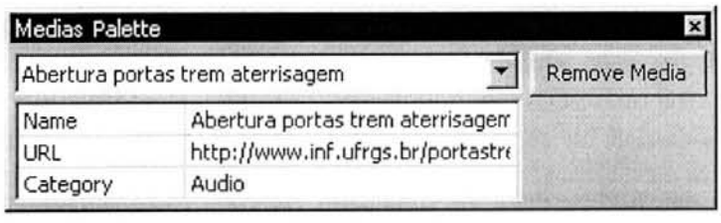


FIGURA 6.5 - Propriedades de conteúdo do componente selecionado

6.1.2 Visão hierárquica da aplicação

A hierarquia da especificação provê ao usuário a visão geral da aplicação apresentando todos os cenários e grupos na forma de uma árvore, provendo uma visão clara do relacionamento entre os mesmos. Na figura 6.6 pode ser observada a vista hierárquica da aplicação apresentada anteriormente na figura 6.2. No nível mais abstrato ela foi organizada em um cenário (Introdução) e dois grupos (Capítulo1 e 2). Cada um dos grupos, por sua vez, foi organizado com a utilização de outros cenários.

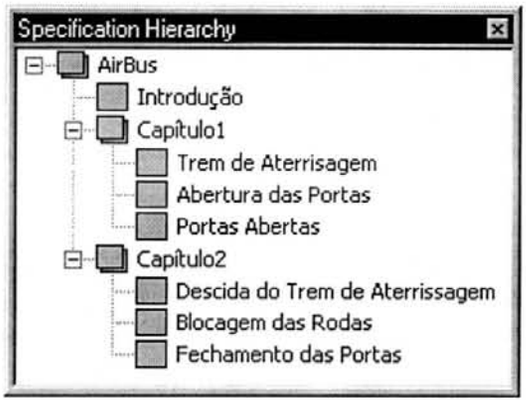


FIGURA 6.6 - Visão hierárquica da aplicação

Com o aumento do tamanho das especificações, é importante que sejam disponibilizados mecanismos que permitam ao autor visualizá-las por partes, sob pena de perder o controle sobre a estrutura das mesmas. O essencial é que ele seja capaz de interagir com os cenários e grupos que de certa forma interferem diretamente no que ele está especificando em um determinado momento. No ambiente, esta funcionalidade é oferecida através dos símbolos gráficos mais (+) e menos (-) presentes ao lado esquerdo de cada grupo. Ao serem pressionados, eles expandem ou escondem a estrutura interna do respectivo grupo. Na figura 6.7, por exemplo, o usuário visualiza apenas as cenas presentes em Capítulo2. As cenas de Capítulo1 não são importantes durante a especificação do último grupo citado e, desta forma, são momentaneamente omitidas da visão hierárquica.

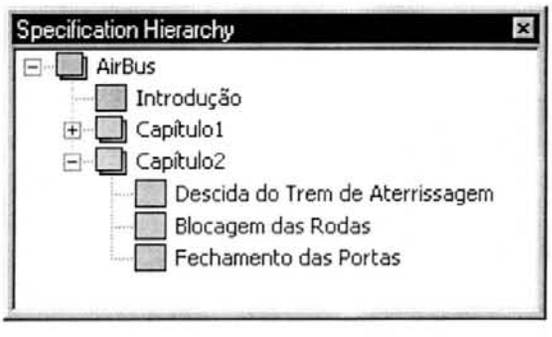


FIGURA 6.7 - Visão modular baseada no contexto da especificação

Além de mecanismos para visualizar a estrutura lógica da aplicação de forma modular, o processo de especificação requer acesso rápido a grupos e cenários, objetivando agilizar seu desenvolvimento. Em MUSE, isto é realizado com a

utilização de menus sensíveis ao contexto; ao pressionar o botão direito do *mouse* sobre um grupo ou cenário, o usuário pode solicitar sua visualização, permitindo a ele operar com o mesmo. A figura 6.8 ilustra a requisição para que o grupo *AirBus* seja apresentado. No caso de se pressionar com o mouse sobre um cenário, as opções no menu sensível ao contexto são *Scene Temporal View* e *Scene Spatial View*. Vale ressaltar, no entanto, que a presença destes menus não é restrita à visão hierárquica; ao contrário, eles aparecem em todas as vistas existentes no ambiente.

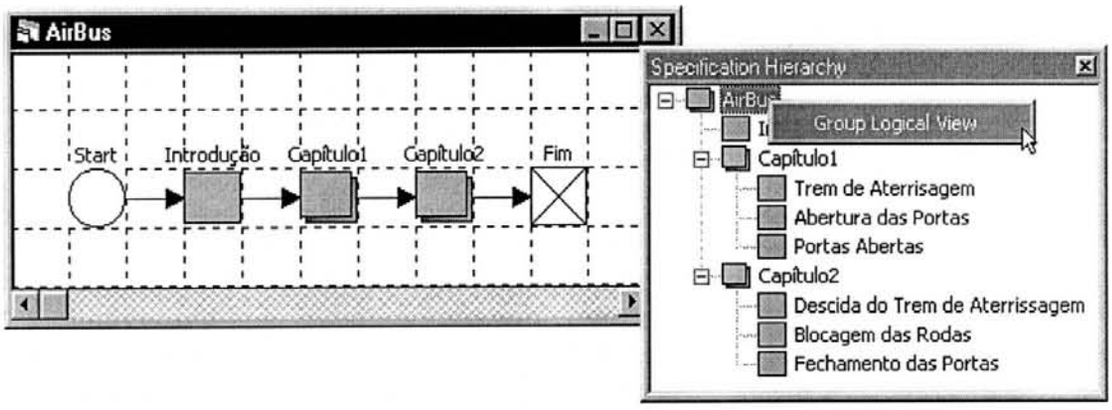


FIGURA 6.8 - Acesso agilizado aos grupos e cenários da aplicação

6.1.3 Palheta de ícones

A palheta de ícones viabiliza a visualização e edição das propriedades de apresentação de cada um dos ícones presentes nos diversos grupos e cenários da aplicação. Este elemento de interface é essencial no processo de autoria e agrega o conceito de orientação a objetos comentado anteriormente. Cada um dos ícones é encarado como um objeto, cujas propriedades são manipulados na janela *Icons Palette*.

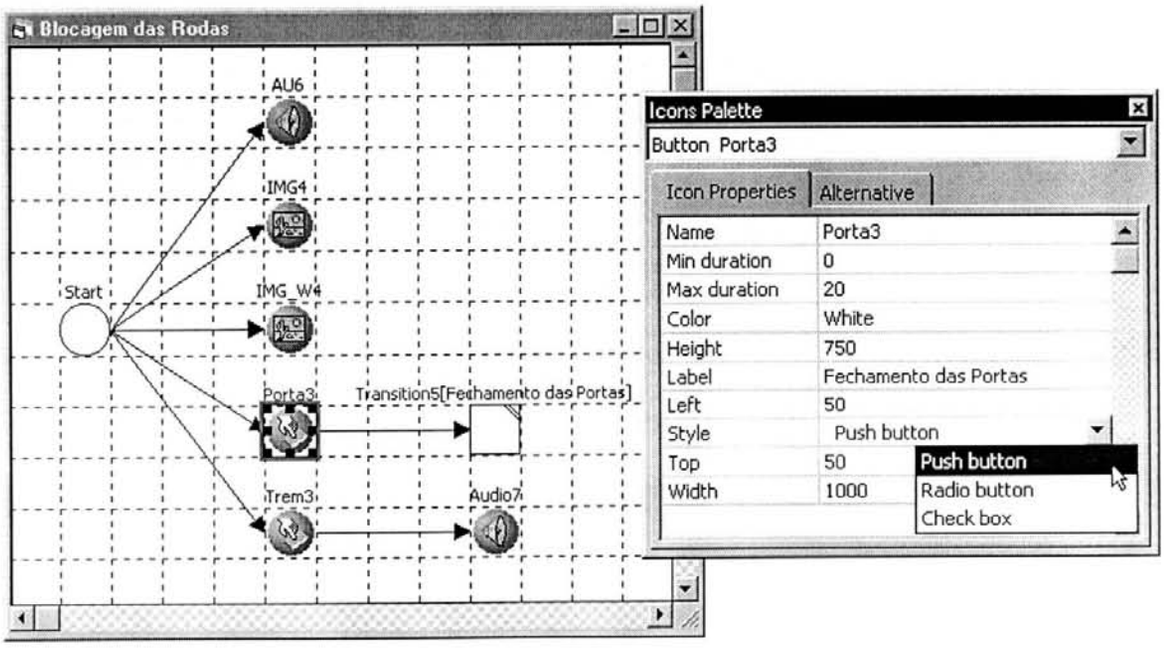


FIGURA 6.9 - Palheta de ícones

A figura 6.9 apresenta a visão temporal do cenário Blocagem das Rodas. Neste cenário, o ícone Porta3 está selecionado, e suas respectivas propriedades são apresentadas na palheta. Como Porta3 representa um botão, as propriedades relativas a este tipo de ícone são apresentadas. Entre estas características pode-se citar, por exemplo, *Label* que indica o texto que aparecerá no botão e *Style* que permite selecionar seu estilo (*Push button, Radio button* ou *Check box*). A tabela 6.2 apresenta as propriedades de cada um dos tipos de ícones suportados pelo ambiente, extraídas do padrão MHEG-5.

A propriedade *Media*, presente nos ícones Bitmap, Texto, Vídeo e Áudio, merece especial atenção. Ela permite associar um objeto multimídia presente no repositório a um ícone de apresentação. Na figura 6.10, por exemplo, o ícone IMG_W4 ainda não possui um componente multimídia associado; o autor pode optar entre Imagem AirBus e Portas Baixas, que são os componentes da categoria Bitmap presentes atualmente no repositório. No ambiente é possível observar claramente a distinção entre o componente multimídia propriamente dito e o ícone de apresentação. A vantagem de manter esta diferenciação é a possibilidade de referenciar o mesmo componente de diversos pontos da aplicação sem que haja a necessidade de manter várias cópias dele no repositório. As propriedades de apresentação (localização do objeto na tela, por exemplo) é que precisam ser definidas individualmente e, por esta razão, estão associadas ao ícone e não ao componente. Outra vantagem, talvez até mais importante, é que durante a execução da aplicação ela será buscada remotamente uma única vez.

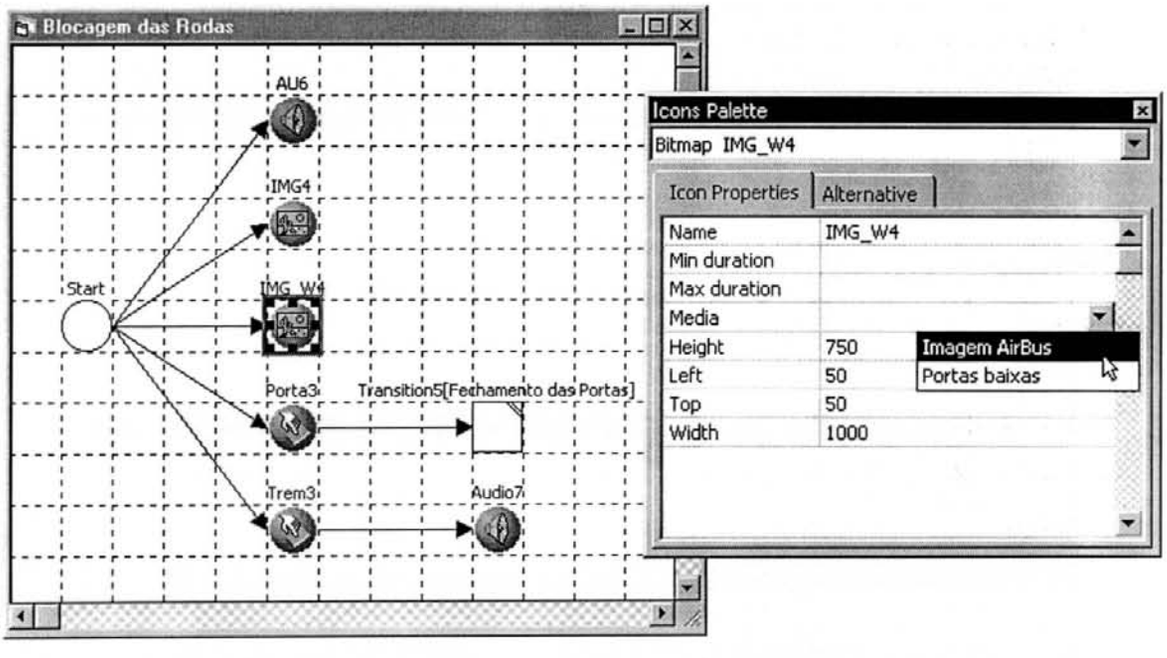


FIGURA 6.10 - Associação do componente ao ícone de apresentação

TABELA 6.2 - Propriedades dos ícones

Grupo, Cena, Fim	
Name	Nome fantasia do ícone
Ponto de Sincronização	
Name*	
Icone	Indica qual é o ícone mestre. Só é definido quando Style for *WaitMaster*
Style	Indica o tipo de sincronização. Pode ser: *WaitEarliest*, *WaitLatest* ou *Wait Master*
Transição	
Name*	
Tranition to	Aponta o cenário para o qual ocorrerá a transição
Bitmap	
Name*	
Min Duration	Duração mínima da apresentação do componente associado ao ícone
Max Duration	Duração máxima da apresentação do componente associado ao ícone
Media	Indica o componente associado ao ícone. Precisa ser da mesma categoria do ícone
Heigth	Altura do componente
Left	Posição em x do componente em relação ao dispositivo de saída
Top	Posição em y do componente em relação ao dispositivo de saída
Width	Largura do componente
Texto	
Name*	
Min Duration*	
Max Duration*	
Media*	
Background color	Cor de fundo (caixa que contém o texto)
Color	Cor da letra
Font	Tipo da fonte
Height*	
Justification	O texto pode ser justificado à esquerda, à direita ou ao centro
Left*	
Orientation	A orientação pode ser horizontal ou vertical
Top*	
Text wrapping	Indica se o texto pode ou não ser quebrado. Os valores possíveis são True ou False
Width*	
Vídeo	
Name*	
Min duration*	
Max duration*	
Media*	
First Frame	Indica o primeiro quadro do vídeo a ser apresentado
Height*	
Last frame	Indica o último quadro do vídeo a ser apresentado
Left*	
Termination	Indica o que ocorrerá com o vídeo após a execução (Freeze ou Terminate).
Top*	
Width*	
Áudio	
Name*	
Min duration*	
Max duration*	
Media*	
Volum	Volume a ser utilizado na execução do áudio
Delay	
Name*	
Duration	Tempo proposital a ser aguardado
Interação do usuário	
Name*	
Min duration*	
Max duration*	
Color*	
Height*	
Label	Texto a ser apresentado sobre o botão
Left*	
Style	Estilo do botão. Pode ser *Push button, Radio button* ou *Check box*
Top*	
Width*	

* O significado já foi apresentado na definição de ícones anteriores

Outra propriedade bastante relevante é *Transition to*, presente nos ícones de Transição. Ela permite indicar para onde a aplicação deve evoluir após a apresentação do cenário onde o ícone se encontra. A figura 6.11 apresenta a transição presente no cenário Portas Abertas. A única transição possível para este cenário é Capítulo2, pois em Capítulo1 não existe nenhuma ligação partindo do ícone Portas Abertas. A única ligação existente se encontra na definição do grupo inicial (*AirBus*) no qual Capítulo1 está ligado a Capítulo2.

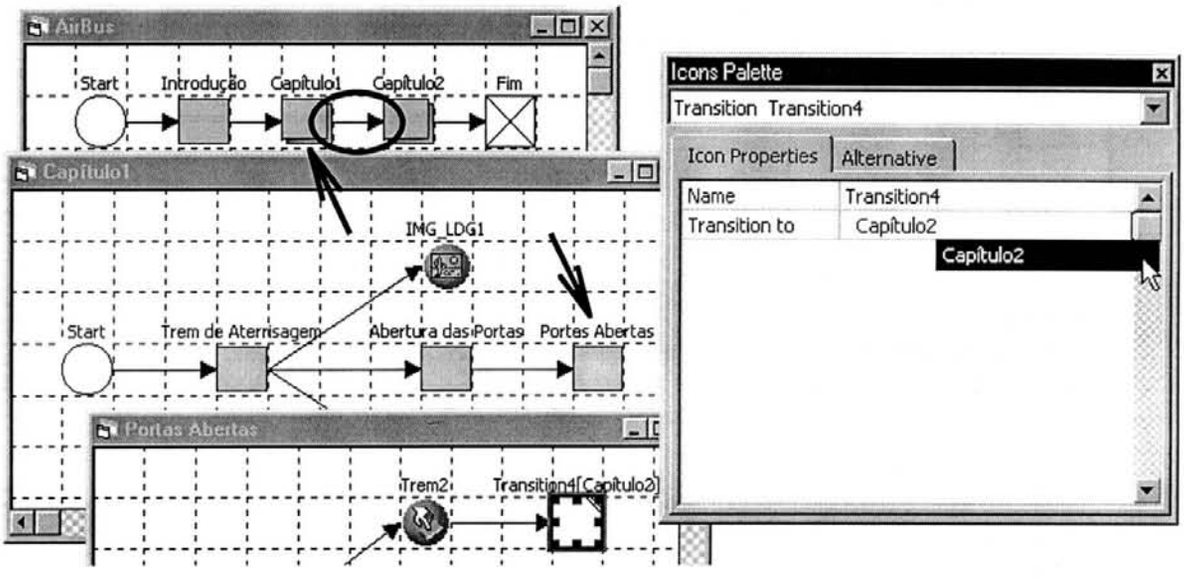


FIGURA 6.11 - Definição das transições

6.1.4 Visão espacial dos cenários

Pouco foi apresentado sobre a visão espacial dos cenários. Até agora deu-se ênfase à visão temporal que é a mais utilizada pelo autor no processo de desenvolvimento. É importante mencionar, no entanto, que cada cenário da aplicação possui uma vista espacial correspondente onde é possível relacionar os elementos entre si, considerando o dispositivo de saída em que serão apresentados. Evita-se, assim, que durante a execução eles apareçam uns sobre os outros ou em posições indesejadadas. A figura 6.12 ilustra a visão espacial do cenário Blocagem das Rodas. Como pode-se observar, é possível movimentar e redimensionar os elementos, sendo que as propriedades relativas à posição (*Top* e *Left*) e dimensão (*Width* e *Height*) são atualizadas automaticamente na palheta de propriedades.

Além disso, na mesma figura percebe-se que em alguns casos o componente real é apresentado, em outros não (representadas em cinza). A apresentação do componente real está associada às seguintes condições:

- existência de um componente associado ao ícone de apresentação,
- o componente deve estar presente localmente,
- o componente deve ser uma imagem ou texto.

Esta é uma restrição técnica do ambiente atual; em versões posteriores estas condições poderão vir a ser modificadas.

Componentes multimídia compartilhados aparecem na vista espacial dos cenários que elas englobam. Neste caso, se elas são movimentadas ou

redimensionadas na vista especial de um cenário, esta modificação se reflete em todos os outros cenários que a compartilham.

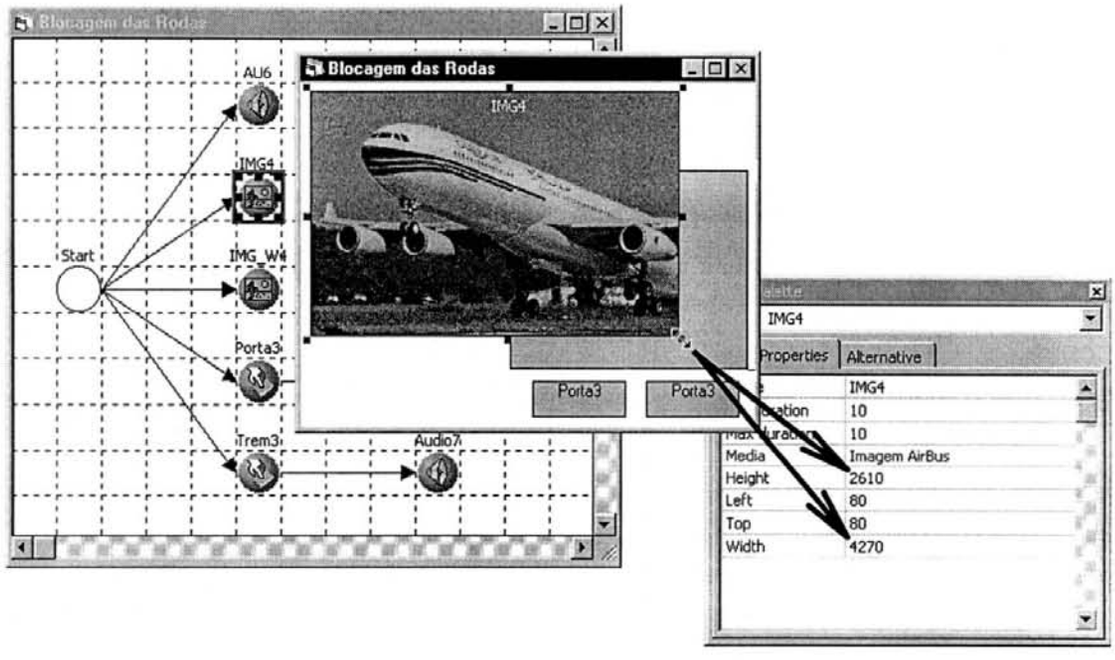


FIGURA 6.12 - Visão espacial do cenário Blocagem das Rodas

6.1.5 Segmentação de componentes multimídia dependentes do tempo

Componentes multimídia dependentes do tempo como vídeo e áudio podem ser divididos em segmentos menores, permitindo a sincronização de outros elementos com pontos bem específicos dos mesmos. O ambiente provê mecanismos que facilitam o processo de fragmentação destes componentes. A figura 6.13 apresenta a janela que disponibiliza esta funcionalidade. O processo de divisão pode ser feito de duas maneiras: a cada *n* quadros ou de acordo com pontos de referência. Abaixo, na divisão de AG_W, optou-se pela segunda maneira.

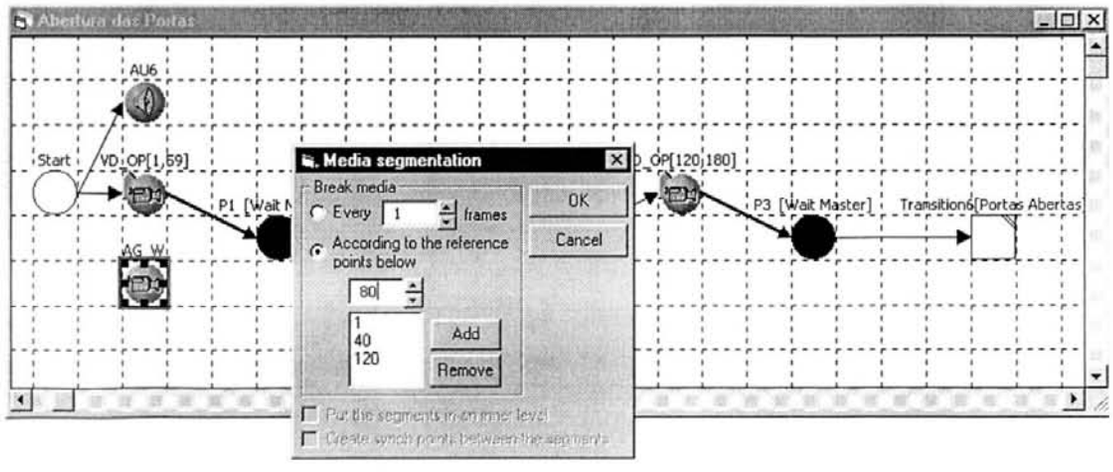


FIGURA 6.13 - Segmentação de componentes dependentes do tempo

De acordo com a figura, serão criados três ícones que representarão três partes distintas do vídeo (quadros 1 a 39, 40 a 79 e 80 a 120). Os tempos associados aos ícones resultantes são calculados com base no ícone original. No exemplo, o tempo associado a AG_W é [3,4]. Cada um dos três ícones terá [1,1.3] de duração, uma vez que o vídeo está sendo dividido em três partes iguais. O resultado da segmentação é apresentado na figura 6.14.

O processo inverso, ou seja, reunir os segmentos que representam uma componente multimídia dependente do tempo, também é disponibilizado em MUSE. Para tal, basta pressionar com o botão direito do *mouse* sobre qualquer um dos segmentos e escolher a opção *Join Media Intervals* (vide figura 6.14). Aqui, mais uma vez os tempos são recalculados e o componente volta a ser representado por um único ícone.

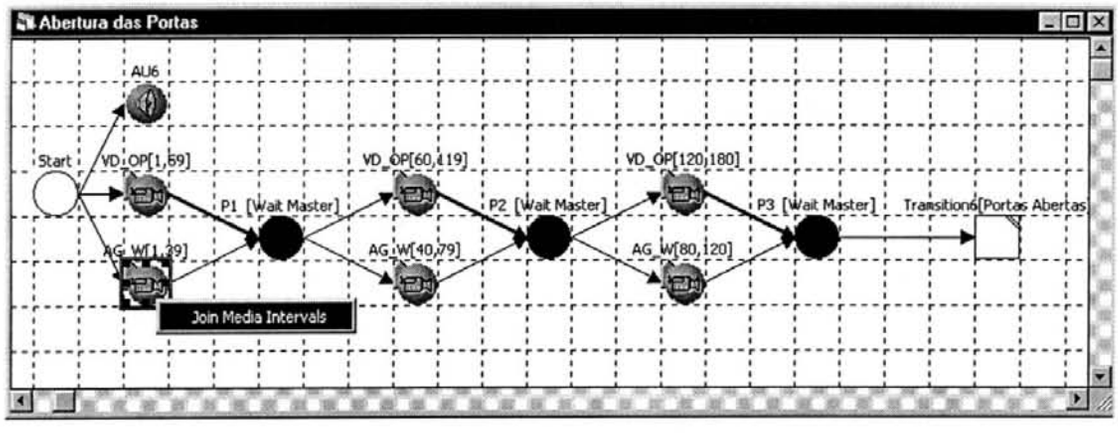


FIGURA 6.14 - Reagrupamento de segmentos

6.1.6 Reuso de grupos e cenários

A possibilidade de reusar grupos e cenários de aplicações definidas anteriormente também é um conceito de orientação a objetos adotado no ambiente. Para tal, eles são armazenados em arquivos individuais. Arquivos com a extensão .spe, .grp e .lot indicam, respectivamente arquivos de especificação, grupo e cenário. A figura 6.15 ilustra uma nova aplicação sendo especificada. Pode-se observar, ainda, o desejo de recuperar algum grupo da aplicação *AirBus*, apresentada parcialmente neste capítulo.

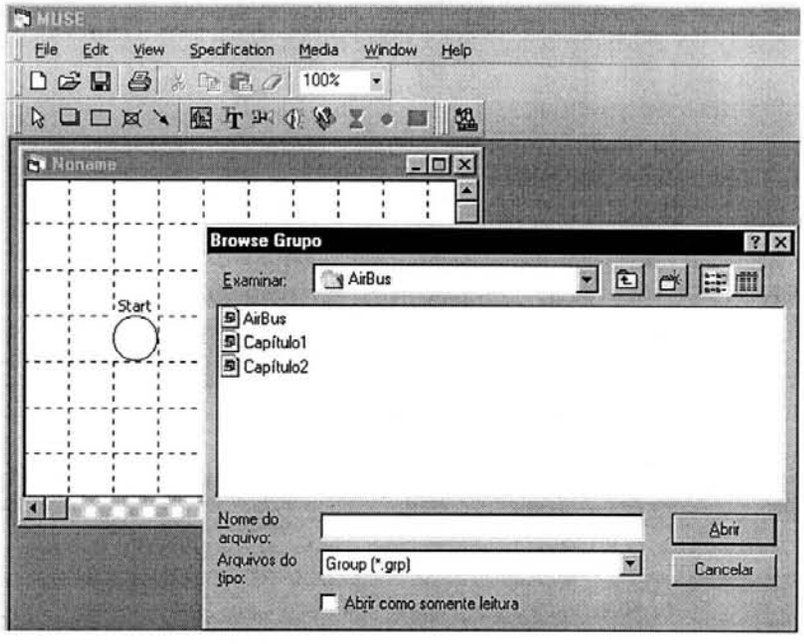


FIGURA 6.15 - Reuso de grupos e cenários

A figura 6.16 apresenta a aplicação após a recuperação do grupo Capítulo1. Na visão hierárquica apresentada na mesma figura fica evidente que, ao reusar um grupo, todos os cenários pertencentes a ele são também reaproveitados.

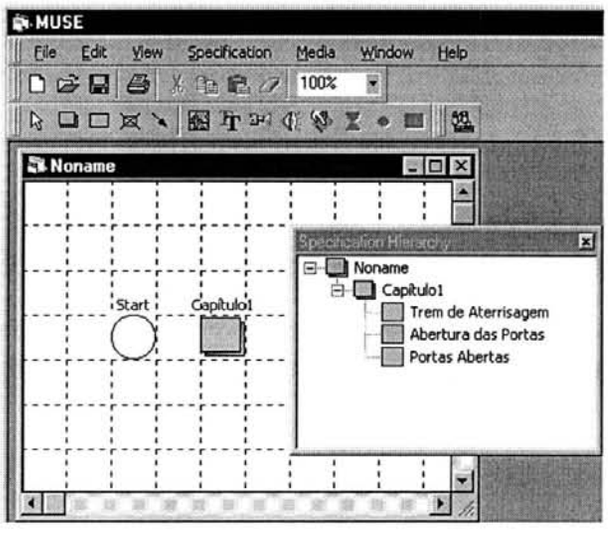


FIGURA 6.16 - Reutilização do grupo Capítulo1

Ao reutilizar um grupo ou cenário, algumas adaptações podem ser necessárias. Estas adaptações estão relacionadas aos componentes e às transições. No primeiro caso, um erro ocorre quando, ao tentar reusar um grupo ou cenário, existir um componente multimídia com o mesmo nome fantasia de algum componente já presente no repositório da aplicação corrente. Na ocorrência deste conflito, uma mensagem como a apresentada na figura 6.16 aparece, indicando que o componente não será carregado. Para solucioná-lo, o autor pode adicionar um novo componente ao

repositório com um nome fantasia diferente dos já existentes e referenciar o componente que o ícone não lido estava referenciando.

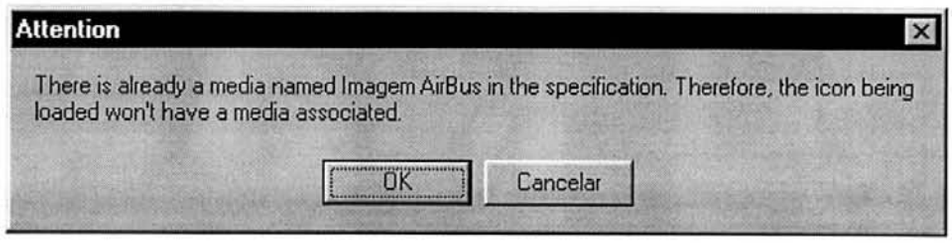


FIGURA 6.17 - Componente com mesmo nome já presente no repositório

A segunda adaptação necessária, como já mencionado, refere-se às transições. Exemplificando, os cenários associados ao grupo Capítulo1 da aplicação *AirBus* apresentavam originalmente transições para grupos e cenários daquela aplicação. Como na nova aplicação estes grupos não existem, faz-se necessário redefini-las. Da mesma forma que ocorre com os componentes, durante a recuperação de um grupo ou cenário, todas as transições impossíveis de se materializar na nova aplicação são descartadas e uma mensagem é apresentada ao usuário (vide figura 6.18)

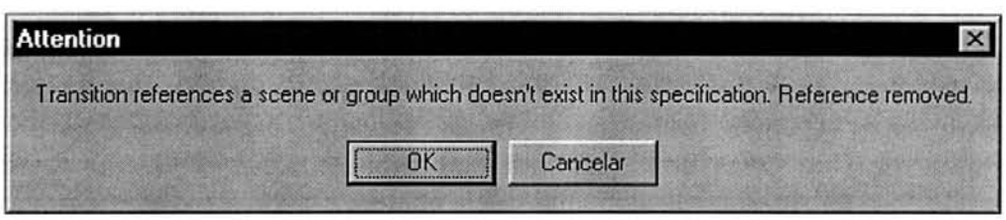


FIGURA 6.18 - Mensagem de transição inexistente

Na figura 6.19 pode-se observar o resultado da reutilização do grupo Capítulo1. A transição que anteriormente apontava para Capítulo2 (vide figura 6.11) foi desfeita. Como o grupo Capítulo1 ainda não foi ligado a nenhum outro grupo ou cenário, não existem opções de transição possíveis.

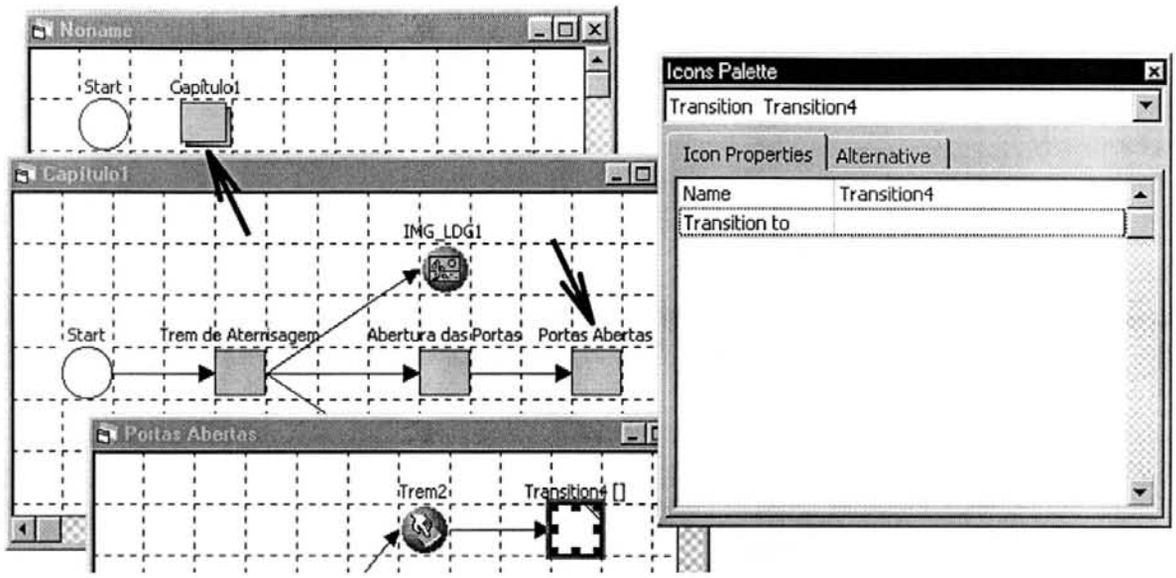


FIGURA 6.19 - Atualização automática das transições possíveis

6.1.7 Geração automática de código E-LOTOS

A geração automática de código E-LOTOS é obtida através da opção especial de salvamento *Save As E-LOTOS.* A geração só é possível, contudo, quando todos as propriedades essenciais tiverem valores definidos e as transições estiverem determinadas. Assim, ao requisitar a geração de código, o pré-compilador é ativado para verificar se estas condições mínimas são atendidas. Todos os problemas encontrados são apresentados em uma janela conforme pode ser visualizado na figura 6.20. O código só é gerado quando nenhum problema for detectado pelo pré-compilador.

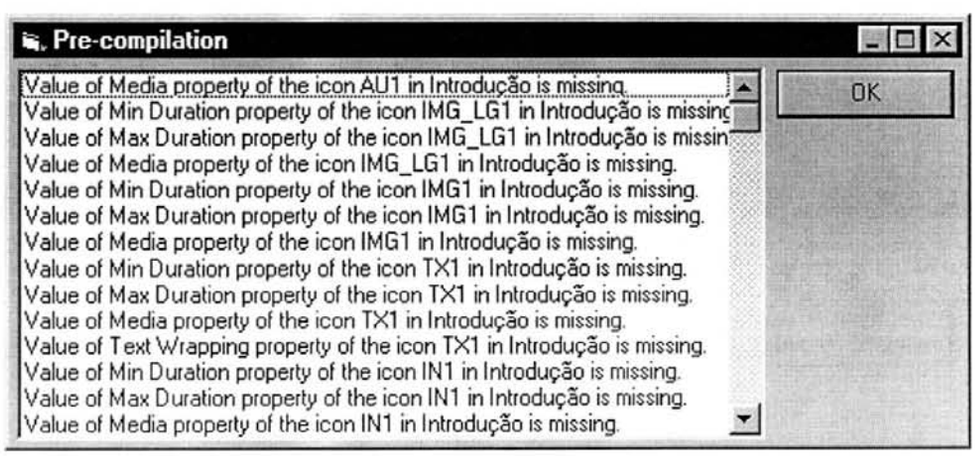


FIGURA 6.20 - Resultados da pré-compilação

6.2 A implementação

O ambiente MUSE foi desenvolvido na plataforma PC. Esta decisão foi tomada pelo fato de ela ser amplamente difundida no mercado, abranger um grande número de usuários e fornecer subsídios que permitem o desenvolvimento de

ferramentas sofisticadas e de fácil utilização. A implementação foi realizada com a utilização da ferramenta Visual Basic 5.0, segundo o paradigma baseado em objetos suportado pela linguagem. Utilizou-se uma série de controles ActiveX (bibliotecas) que foram integrados ao processo de desenvolvimento como forma de agilizar a implementação e permitir que os esforços fossem concentrados na solução como um todo e não em detalhes isolados.

A seguir são apresentadas as principais classes de janela do ambiente. Após, aborda-se a estrutura de classes que mantêm a consistência das especificações. Por fim, são apresentados diagramas dos procedimentos mais importantes realizados no ambiente.

6.2.1 Classes de janela

A figura 6.20 apresenta a estrutura geral do ambiente onde é possível observar as janelas de edição disponibilizadas ao autor. Estas são implementadas em classes individuais, denominadas classes de janela ou *forms*. As mais importantes de MUSE são:

- *fmJanelaPrincipal*: representa a janela principal do ambiente. Funciona como se fosse um módulo global do ambiente; por convenção, as variáveis que devem ser "visíveis" a todos os módulos são declaradas nesta classe. Apresenta a peculiaridade de ser uma *MDI (Multiple Document Interface)* pai.
- *fmJanelaFilha*: classe onde é definida a estrutura lógica dos grupos e o comportamento temporal dos cenários. Associado a esta janela está o controle *Interact* (itAreaEspecificacao), que provê funcionalidades genéricas para a manipulação de diagramas como a adição e remoção de símbolos gráficos, bem como operações de arraste e ligação de objetos.
- *fmVisãoEspacial*: esta classe apresenta funcionalidades para a sincronização espacial dos elementos que constituem cada cenário. A ela está associado o controle *ddWindow*, que provê mecanismos para adicionar, mover e redimensionar "caixas". Tanto fmVisãoEspacial quanto fmJanelaFilha são implementadas como MDI filhas.

Ainda na figura 6.20 é possível observar as classes que implementam as janelas de apoio do ambiente:

- *fmPalhetaMidias*: corresponde à palheta de propriedades dos componentes;
- *fmHierarquiaEspecificacao*: implementa a janela *Specification Hierarchy*;
- *fmPalhetaIcones*: janela onde são manipuladas as propriedades dos ícones.

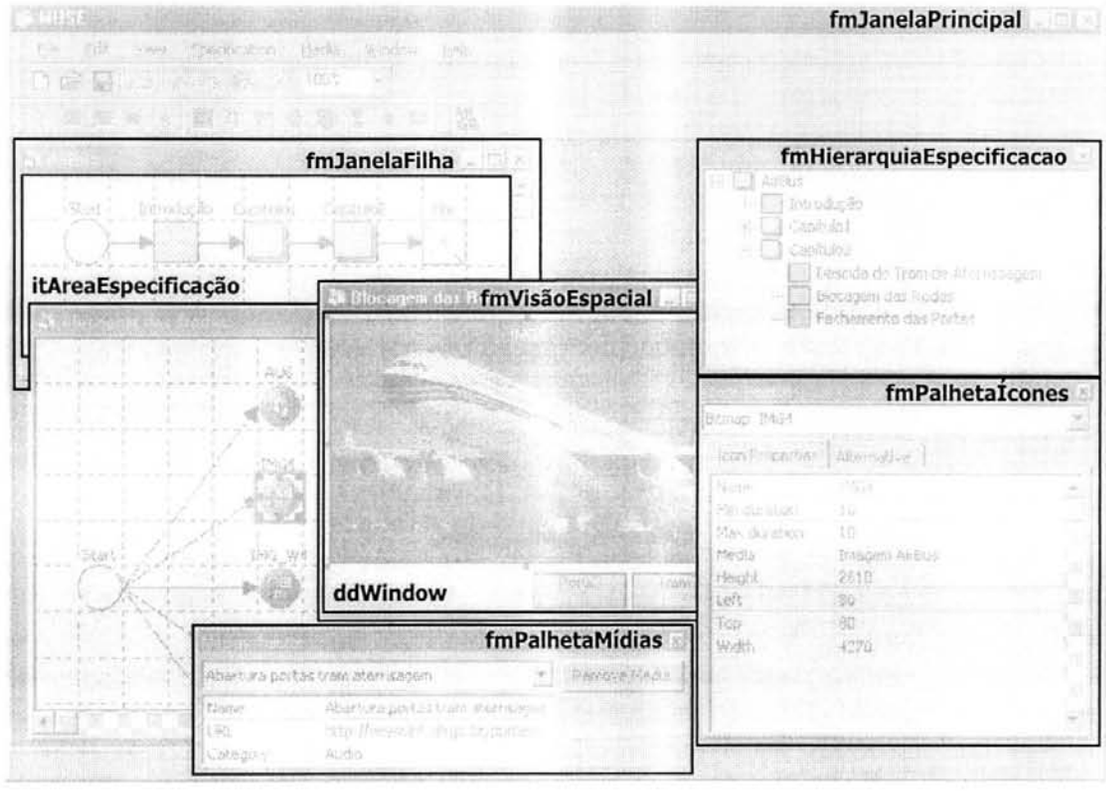


FIGURA 6.21 - Classes de janela do ambiente

6.2.2 Classes para a consistênc ia da especificação

Independente das classes de janela, o ambiente conta também com um conjunto de classes que mantém a consistência das especificações. Na verdade, qualquer modificação em alguma das janelas mencionadas anteriormente provoca uma imediata atualização em objetos deste conjunto [GRA97]. Esta estrutura, contudo, não permite a realização de modificações que estejam fora dos padrões propostos pelo modelo de autoria.

A primeira classe deste conjunto é clsEspecificação, composta por duas coleções: uma de componentes multimídia e outra de cenários (vide figura 6.22). Antes de abordá-las, contudo, alerta-se que apenas as propriedades mais relevantes destas e das próximas classes serão apresentadas, em virtude da complexidade do ambiente. A coleção de componentes multimídia reflete o conteúdo presente no repositório. A coleção de cenários, por sua vez, não apresenta apenas cenários, mas também os grupos presentes na especificação. Ao iniciar uma nova especificação, a coleção de componentes multimídia encontra-se vazia; ao contrário, a coleção de cenários apresenta um elemento (GrupoInicial).

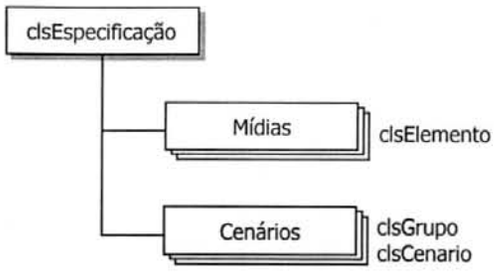


FIGURA 6.22 - A classe clsEspecificação

A coleção Mídias manipula elementos da classe clsElemento. A estrutura básica desta classe é apresentada na figura 6.23. Quando o autor solicita a adição de um novo componente ao repositório uma nova instância da classe clsElemento é alocada e adicionada à coleção.

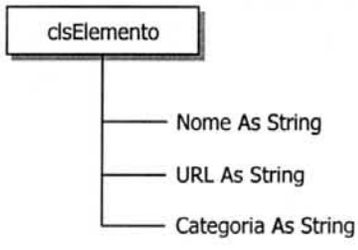


FIGURA 6.23 - A classe clsElemento

A coleção Cenários, por outro lado, apresenta uma estrutura bem mais complexa ao manipular elementos da classe clsGrupo (figura 6.24) e clsCenário (figura 6.25). Uma classe clsGrupo é instanciada e adicionada à coleção Cenários quando o autor insere um ícone de grupo à especificação. Da mesma forma, uma nova instância da classe clsCenário é alocada e adicionada a esta coleção quando um ícone de cenário é acrescentado à especificação.

A figura 6.24 ilustra a classe clsGrupo. Uma propriedade relevante é JanelaAssociada, que referencia a janela (classe fmJanelaFilha) do respectivo grupo. Também importante é a coleção Elementos. Nela encontram-se as entidades que fazem parte do grupo (ícone de cenário, de imagem, entre outros). Na realidade esta coleção aparece também na definição da classe clsCenário (figura 6.25). A diferença entre elas são os elementos que podem ser adicionados a cada uma. Por exemplo, na coleção da classe clsCenário não é possível existir uma instância de clsIconeGrupo, pois este ícone não pode aparecer em um cenário.

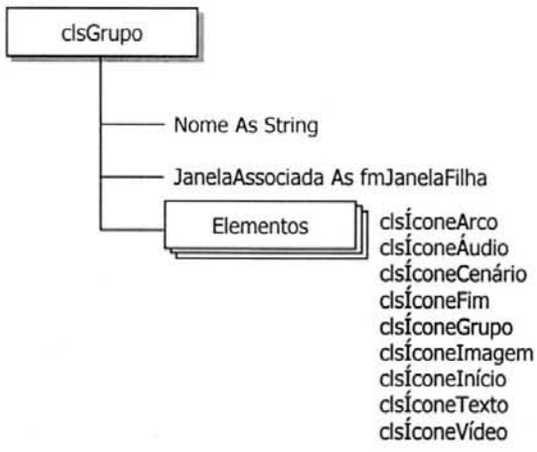


FIGURA 6.24 - A classe clsGrupo

É possível observar muitas outras semelhanças entre estas duas classes. Em ambas aparecem as propriedades Nome e JanelaAssociada. Apenas em clsCenário, no entanto, existe uma referência à visão espacial (classe fmVisãoEspacial); vale lembrar que a estrutura lógica (grupos) não apresenta esta visão.

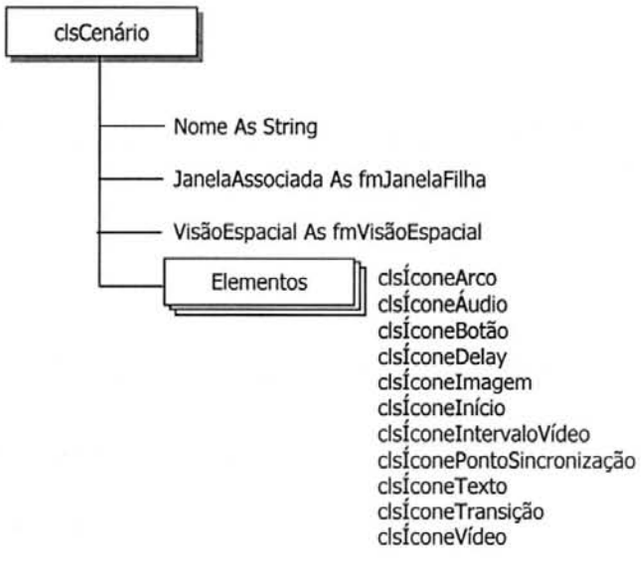


FIGURA 6.25 - A classe clsCenário

As classes clsIcone implementam ícones suportados no ambiente. Objetos destas classes são instanciados quando um novo ícone é adicionado aos diagramas. clsÍconeÁudio, por exemplo, provê as funcionalidades relativas ao ícone de áudio. Algumas propriedades desta classe são Nome, MidiaAssociada, MinDuration, MaxDuration e Volum, que coincidem com as apresentadas para este ícone na tabela 6.2 (se enquadram em *outras propriedades* na figura 6.26). De maneira geral, esta coincidência ocorre com todas as classes clsIcone. Além das coincidentes, no entanto, todas estas classes apresentam as propriedades xPos e yPos, que indicam onde os ícones devem aparecer quando a janela for aberta (vide figura 6.26).

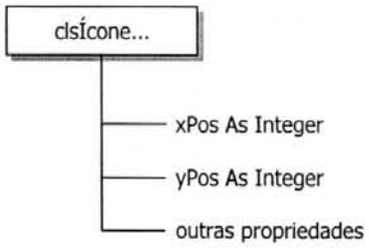


FIGURA 6.26 - As classes clsÍcone

Por fim, tem-se a classe clsArco que implementa as ligações entre ícones. As principais propriedades da mesma são os identificadores dos ícones origem e destino (vide figura 6.27).

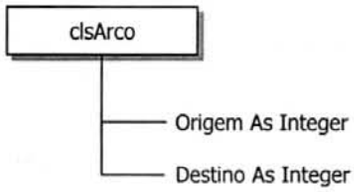


FIGURA 6.27 - A classe clsArco

Reunindo-se as classes apresentadas tem-se o diagrama ER da figura 6.28. Várias relações (particulares a alguns ícones) podem ser observadas nesta figura:

- objetos clsÍconeGrupo possuem referência aos objetos clsGrupo correspondentes;
- da mesma forma, objetos clsÍconeCenário referenciam objetos clsCenário;
- as entidades coloridas em cinza referenciam componentes multimídia (classe clsElemento);
- as transições referenciam um objeto que pode pertencer à classe clsÍconeGrupo, clsÍconeCenário ou clsÍconeFim;
- objetos da classe clsÍconePontoSincronização referenciam objetos da classe clsÍcone (caixa em cinza claro) quando implementam a política WaitMaster.

As relações entre objetos clsÍcone não são apresentadas nesta figura. Na realidade, as regras que estabelecem quem pode ser ligado a quem não estão implementadas na estrutura de dados e sim, num arquivo de inicialização utilizado pelo controle *Interact*.

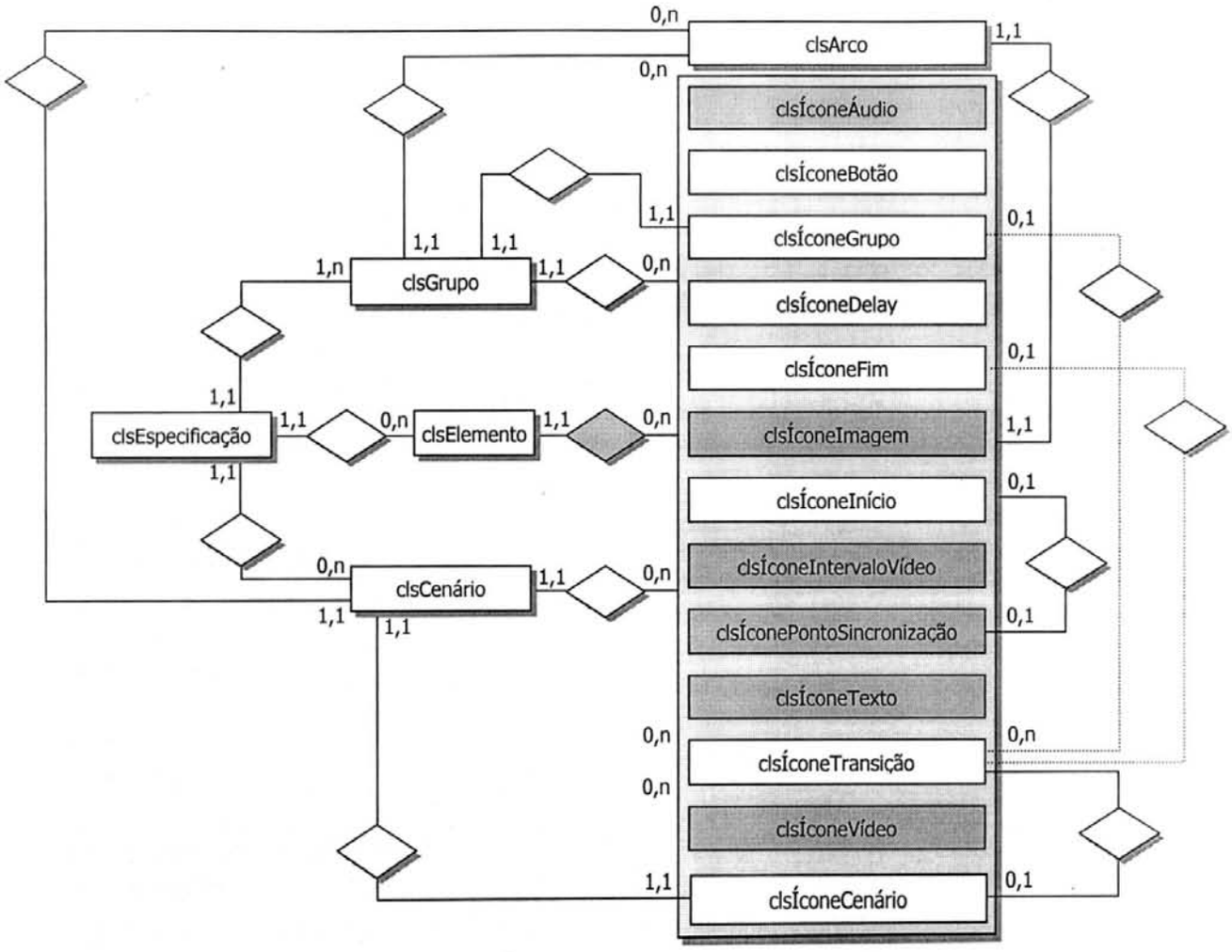


FIGURA 6.28 - Diagrama ER das classes do ambiente

7 Conclusões e Trabalhos Futuros

Este trabalho apresentou inicialmente um novo modelo para a especificação de aplicações multimídia interativas que leva em consideração restrições temporais de apresentação dos componentes multimídia que as compõem. Além disso, apresentou-se mecanismos de mapeamento deste modelo para a linguagem E-LOTOS. Por fim, ilustrou-se o ambiente desenvolvido. A maior contribuição deste trabalho é, portanto, a construção de um ambiente de especificação destinado ao usuário, oferecendo-lhe um alto poder de expressão e disponibilizando a representação formal de suas especificações com fins de análise da consistência lógica e temporal da aplicação.

O modelo proposto separa os conceitos de estruturação lógica e sincronização temporal. A estrutura lógica das aplicações possibilita aos projetistas organizar estas aplicações em capítulos, seções ou em qualquer outra unidade. Deste modo, modulariza-se a aplicação e define-se o comportamento temporal de cada módulo. Naturalmente, a complexidade das partes é bem menor do que o todo. Assim, a utilização de um modelo baseado em redes de Petri é adequado, já que redes pequenas estarão sendo manipuladas.

Com relação ao mapeamento das especificações para E-LOTOS, vale salientar a importância da solução proposta neste trabalho para a geração de código de forma sistemática. Possivelmente, o código gerado não seja o mais elegante. Importante, contudo, é que a forma de representação adotada, além de sistemática, oferece os subsídios necessários para a realização dos passos de verificação e validação, ao prover especificações bastante completas sob o ponto de vista dos eventos e ações tratadas (relevantes em aplicações multimídia interativas).

A implementação do ambiente permitiu comprovar os requisitos iniciais que o modelo deveria atender: facilidade de uso e elevado poder de expressão. Entre os fatores que contribuiram para isto destacam-se a interface sofisticada do ambiente, as funcionalidades para o reuso de grupos e cenários e o próprio modelo de autoria. Sobre os aspectos técnicos da implementação, é importante salientar a utilização dos controles. Graças à sua utilização o tempo de desenvolvimento do ambiente foi bastante reduzido sem perda de qualidade. Ao contrário, as bibliotecas utilizadas, por serem comercializadas, são de alta qualidade e apresentam um percentual ínfimo de erros. O grande desafio, no entanto, foi integrá-las ao que já havia sido desenvolvido; as funcionalidades oferecidas por elas nem sempre se encaixavam perfeitamente à aplicação sendo desenvolvida. Assim, várias adaptações tiveram que ser realizadas no código fonte existente para viabilizar sua utilização.

A continuidade deste trabalho prevê a criação de mecanismos que permitam ao usuário definir no próprio ambiente parâmetros de qualidade de serviço, que serão utilizados na busca dos componentes multimídia envolvidos na aplicação. Isso pode ser feito aumentando-se o conjunto de atributos de cada componente e explorando a funcionalidade de componentes alternativos. A inclusão de informações adicionais na especificação, naturalmente, reflete-se na saída gerada. Portanto, o segundo passo é a modificação do tradutor de forma que os novos dados sejam interpretados adequadamente. Os parâmetros de QoS da aplicação devem ser mapeados para os níveis inferiores de serviços da rede.

A alteração de QoS por parte do usuário também deverá ser prevista, já que consiste basicamente numa repetição do processo de mapeamento e negociação. É necessário, entretanto, definir uma interface que permita a um usuário leigo especificar a qualidade desejada. Em outras palavras, o ideal é que seja feito também um mapeamento de parâmetros do nível do usuário para parâmetros do nível da aplicação.

Vale ressaltar, por fim, que a utilização deste ambiente integrado às outras ferramentas sendo desenvolvidas no projeto possibilitará ao autor realizar todas as etapas que constituem o processo de desenvolvimento de aplicações multimídia interativas: especificação, verificação e apresentação. A facilidade do modelo de autoria apresentado ao usuário e a utilização de uma técnica de descrição formal para validar as aplicações por ele criadas tornam o ambiente atrativo, de fácil utilização sem perder a alta expressividade e, principalmente, capaz de apresentar resultados confiáveis sobre a coerência das aplicações criadas.

Anexo 1 A Biblioteca de Tipos Básicos

Este anexo apresenta o módulo que define os tipos de dados suportados pelo ambiente MUSE. Na definição em E-LOTOS das especificações, este módulo é importado; como ele não varia para cada especificação, não precisa ser sempre gerado. Além disto, é válido ressaltar que a hierarquia de tipos estabelecida abaixo reflete fielmente parte da definição da norma MHEG-5.

```
module classes is

  type Colour is
    absolute_colour (string)
  endtype

  type ButtonStyle is
    pushbutton
    | radiobutton
    | checkbox
  endtype

  type SwitchButtonClass is
    (PushButtonClass,
    buttonstyle ⇒ ButtonStyle)
  endtype

  type PushButtonClass is
    (Button, original_label ⇒ string)
  endtype

  type Button is
    (Visible,
    button_colour ⇒ Colour)
  endtype

  type Orientation is
    left
    | right
    | up
    | down
  endtype

  type Termination is
    freeze
    | disappear
  endtype

  type VideoClass is
    (Visible,
    termination ⇒ Termination)
  endtype

  type AudioClass is
    (Presentable,
    original_volume ⇒ int)
  endtype

  type LineOrientation is
    vertical
    | horizontal
  endtype
```

```
type JustificationEnum is
  start
  | end
  | centre
  | justified
endtype

type FontBody is
direct_font (string)
endtype

type TextClass is
  (Visible,
  original_font ⇒ FontBody,
  text_colour ⇒ Colour,
  background_colour ⇒ Colour,
  horizontal_justification ⇒ JustificationEnum,
  line_orientation ⇒ LineOrientation,
  text_wrapping ⇒ bool)
endtype

type BitmapClass is
  (Visible,
  tiling ⇒ bool,
  original_transparency ⇒ int)
endtype

type OriginalPosition is
  (x_position ⇒ int,
  y_position ⇒ int)
endtype

type OriginalBoxSize is
  (x_length ⇒ int,
  y_length ⇒ int)
endtype

type Visible is
  (Presentable,
  original_box_size ⇒ OriginalBoxSize,
  original_position ⇒ OriginalPosition)
endtype

type Presentable is
  Ingredient
endtype

type FontClass is
  Ingredient
endtype

type ReferencedContent is
  (content_reference ⇒ string)
endtype

type ContentBody is
  included_content (string)
  | referenced_content (ReferencedContent)
endtype

type Ingredient is
  (content ⇒ ContentBody)
endtype

endmodule
```

Anexo 2 Aplicação Exemplo em E-LOTOS

Este anexo apresenta a aplicação ilustrada na figura 4.11 em E-LOTOS. O código apresentado abaixo foi gerado automaticamente pelo ambiente MUSE.

```
specification Aplicação
    import processos, classes, mídias
    [i_GrupoInicial, f_GrupoInicial, u_Interação, Dado:class]
behavior
local var
    AU_P1: StreamClass, dminAU_P1: Time, dmaxAU_P1: Time,
    AU_P2: StreamClass, dminAU_P2: Time, dmaxAU_P2: Time,
    AU_P3: StreamClass, dminAU_P3: Time, dmaxAU_P3: Time,
    VD_P1: StreamClass, dminVD_P1: Time, dmaxVD_P1: Time,
    VD_P2: StreamClass, dminVD_P2: Time, dmaxVD_P2: Time,
    VD_P3: StreamClass, dminVD_P3: Time, dmaxVD_P3: Time,
    Video3: StreamClass, dminVideo3: Time, dmaxVideo3: Time,
    Bitmap1: BitmapClass, dBitmap1: Time,
    Bitmap2: BitmapClass, dBitmap2: Time,
    Bitmap3: BitmapClass, dBitmap3: Time,
    Audio2: StreamClass, dminAudio2: Time, dmaxAudio2: Time,
    Interação: SwitchButtonClass, dminInteração: Time, dmaxInteração: Time,
    P4: BitmapClass, dP4: Time,
    ANI_P1: StreamClass, dminANI_P1: Time, dmaxANI_P1: Time,
    ANI_P2: StreamClass, dminANI_P2: Time, dmaxANI_P2: Time
init
  ?AU_P1:= (((content=>(content_reference=>http://www.inf.ufrgs.br/audio.avi)),
       original_box_size=>(1000,750),
       original_position=>(50,50)),
       termination=>Freeze)
  ?dminAU_P1:= 0,9666666
  ?dmaxAU_P1:= 1,933333
  ?AU_P2:= (((content=>(content_reference=>http://www.inf.ufrgs.br/audio.avi)),
       original_box_size=>(1000,750),
       original_position=>(50,50)),
       termination=>Freeze)
  ?dminAU_P2:= 1
  ?dmaxAU_P2:= 2
  ?AU_P3:= (((content=>(content_reference=>http://www.inf.ufrgs.br/audio.avi)),
       original_box_size=>(1000,750),
       original_position=>(50,50)),
       termination=>Freeze)
  ?dminAU_P3:= 1,033333
  ?dmaxAU_P3:= 2,066667
  ?VD_P1:= (((content=>(content_reference=>http://www.inf.ufrgs.br/video.avi)),
       original_box_size=>(1000,750),
       original_position=>(50,50)),
       termination=>Freeze)
  ?dminVD_P1:= 0,9666666
  ?dmaxVD_P1:= 1,933333
  ?VD_P2:= (((content=>(content_reference=>http://www.inf.ufrgs.br/video.avi)),
       original_box_size=>(1000,750),
       original_position=>(50,50)),
       termination=>Freeze)
  ?dminVD_P2:= 1
  ?dmaxVD_P2:= 2
  ?VD_P3:= (((content=>(content_reference=>http://www.inf.ufrgs.br/video.avi)),
       original_box_size=>(1000,750),
       original_position=>(50,50)),
       termination=>Freeze)
  ?dminVD_P3:= 1,033333
  ?dmaxVD_P3:= 2,066667
  ?Video3:= (((content=>(content_reference=>http://www.inf.ufrgs.br/ri.avi)),
       original_box_size=>(1000,750),
       original_position=>(50,50)),
       termination=>Disappear)
  ?dminVideo3:= 3
```

```
    ?dmaxVideo3:= 4.5
    ?Bitmap1:= (((content=>(content_reference=>http://www.inf.ufrgs.br/picure1.gif)),
          original_box_size=>(1000,750),
          original_position=>(50,50)))
    ?dBitmap1:= 7
    ?Bitmap2:= (((content=>(content_reference=>http://www.inf.ufrgs.br/picture2.gif)),
          original_box_size=>(1000,750),
          original_position=>(50,50)))
    ?dBitmap2:= 7
    ?Bitmap3:= (((content=>(content_reference=>http://www.inf.ufrgs.br/picture3.gif)),
          original_box_size=>(1000,750),
          original_position=>(50,50)))
    ?dBitmap3:= 7
    ?Audio2:= ((content=>(content_reference=>http://www.inf.ufrgs.br/audio.au)),
          original_volum=>5)
    ?dminAudio2:= 2
    ?dmaxAudio2:= 3
    ?Interação:= ((((original_box_size=>(1000,750),
          original_position=>(50,50)),
          button_colour=>Grey),
          original_label=>Finalizar),
          button_style=>Push button)
    ?dminInteração:= 0
    ?dmaxInteração:= 20
    ?P4:= (((content=>(content_reference=>http://www.inf.ufrgs.br/picture4.gif)),
          original_box_size=>(1000,750),
          original_position=>(50,50)))
    ?dP4:= 5
    ?ANI_P1:= (((content=>(content_reference=>http://www.inf.ufrgs.br/animação.avi)),
          original_box_size=>(1000,750),
          original_position=>(50,50)),
          termination=>Freeze)
    ?dminANI_P1:= 1,933333
    ?dmaxANI_P1:= 2,9
    ?ANI_P2:= (((content=>(content_reference=>http://www.inf.ufrgs.br/animação.avi)),
          original_box_size=>(1000,750),
          original_position=>(50,50)),
          termination=>Freeze)
    ?dminANI_P2:= 2,066667
    ?dmaxANI_P2:= 3,1
in
    GrupoInicial[i_GrupoInicial, f_GrupoInicial, u_Interação, Dado]
          (AU_P1, dminAU_P1, dmaxAU_P1, AU_P2, dminAU_P2, dmaxAU_P2, AU_P3, dminAU_P3, dmaxAU_P3,
          VD_P1, dminVD_P1, dmaxVD_P1, VD_P2, dminVD_P2, dmaxVD_P2, VD_P3, dminVD_P3, dmaxVD_P3,
          Video3, dminVideo3, dmaxVideo3, Bitmap1, dBitmap1, Bitmap2, dBitmap2, Bitmap3, dBitmap3,
          Audio2, dminAudio2, dmaxAudio2, Interação, dminInteração, dmaxInteração, P4, dP4,
          ANI_P1, dminANI_P1, dmaxANI_P1, ANI_P2, dminANI_P2, dmaxANI_P2)
endspec

module processos is

process GrupoInicial[i_GrupoInicial, f_GrupoInicial, u_Interação, Dado]
    (AU_P1: StreamClass, dminAU_P1: Time, dmaxAU_P1: Time,
    AU_P2: StreamClass, dminAU_P2: Time, dmaxAU_P2: Time,
    AU_P3: StreamClass, dminAU_P3: Time, dmaxAU_P3: Time,
    VD_P1: StreamClass, dminVD_P1: Time, dmaxVD_P1: Time,
    VD_P2: StreamClass, dminVD_P2: Time, dmaxVD_P2: Time,
    VD_P3: StreamClass, dminVD_P3: Time, dmaxVD_P3: Time,
    Video3: StreamClass, dminVideo3: Time, dmaxVideo3: Time,
    Bitmap1: BitmapClass, dBitmap1: Time,
    Bitmap2: BitmapClass, dBitmap2: Time,
    Bitmap3: BitmapClass, dBitmap3: Time,
    Audio2: StreamClass, dminAudio2: Time, dmaxAudio2: Time,
    Interação: SwitchButtonClass, dminInteração: Time, dmaxInteração: Time,
    P4: BitmapClass, dP4: Time,
    ANI_P1: StreamClass, dminANI_P1: Time, dmaxANI_P1: Time,
    ANI_P2: StreamClass, dminANI_P2: Time, dmaxANI_P2: Time):exit is
```

```
    hide i_Cena, f_Cena1_i_Cena2, f_Cena2_i_Cena3, req_Fim in
        i_GrupoInicial;
        par f_Cena1_i_Cena2#2, f_Cena2_i_Cena3#2
            [f_Cena1_i_Cena2]->Cena1[i_Cena, f_Cena1_i_Cena2, Dado, req_Fim]
                (AU_P1, dminAU_P1, dmaxAU_P1, AU_P2, dminAU_P2, dmaxAU_P2, AU_P3, dminAU_P3,
                dmaxAU_P3,VD_P1, dminVD_P1, dmaxVD_P1, VD_P2, dminVD_P2, dmaxVD_P2, VD_P3,
                minVD_P3, dmaxVD_P3)
            [f_Cena1_i_Cena2, f_Cena2_i_Cena3]->Cena2[f_Cena1_i_Cena2, f_Cena2_i_Cena3, Dado, req_Fim]
                (Video3, dminVideo3, dmaxVideo3, Bitmap1, dBitmap1, Bitmap2, dBitmap2, Bitmap3, dBitmap3)
            [f_Cena2_i_Cena3]->Cena3[f_Cena2_i_Cena3, f_GrupoInicial, Dado, req_Fim]
                (Audio2, dminAudio2, dmaxAudio2, Interação, dminInteração, dmaxInteração, P4, dP4,
                ANI_P1, dminANI_P1, dmaxANI_P1, ANI_P2, dminANI_P2, dmaxANI_P2)
        endpar
        [>req_Fim;exit
    endhide
endproc

process Cena1[i_Cena1, f_Cena1, Dado, req_Fim]
    (AU_P1: StreamClass, dminAU_P1: Time, dmaxAU_P1: Time,
    AU_P2: StreamClass, dminAU_P2: Time, dmaxAU_P2: Time,
    AU_P3: StreamClass, dminAU_P3: Time, dmaxAU_P3: Time,
    VD_P1: StreamClass, dminVD_P1: Time, dmaxVD_P1: Time,
    VD_P2: StreamClass, dminVD_P2: Time, dmaxVD_P2: Time,
    VD_P3: StreamClass, dminVD_P3: Time, dmaxVD_P3: Time):exit is

    hide f_PS3, i_AU_P1, f_AU_P1, f_PS1, f_AU_P2, f_PS2, f_AU_P3, i_VD_P1, f_VD_P1, f_VD_P2, f_VD_P3,
        ctl_PS1, ctl_PS2, ctl_PS3, req_Res_Transition1 in
        loop forever in
            i_Cena1;
            par f_PS1#3, f_PS2#3, f_PS3#2, f_AU_P1#2, f_AU_P2#2, f_AU_P3#2, f_VD_P1#2, f_VD_P2#2,
                f_VD_P3#2, ctl_PS1#3, ctl_PS2#3, ctl_PS3#3
                [f_AU_P1, f_VD_P1, f_PS1, ctl_PS1]->PS1[f_AU_P1, f_VD_P1, f_PS1, Dado, req_Fim, ctl_PS1,
                    req_Res_Transition1]()
                [f_AU_P2, f_VD_P2, f_PS2, ctl_PS2]->PS2[f_AU_P2, f_VD_P2, f_PS2, Dado, req_Fim, ctl_PS2,
                    req_Res_Transition1]()
                [f_AU_P3, f_VD_P3, f_PS3, ctl_PS3]->PS3[f_AU_P3, f_VD_P3, f_PS3, Dado, req_Fim, ctl_PS3,
                    req_Res_Transition1]()
                [f_PS3]->Transition1[f_PS3, f_Cena1, Dado, req_Fim]()
                [f_AU_P1, ctl_PS1]->AU_P1[i_AU_P1, f_AU_P1, Dado, req_Fim, ctl_PS1, req_Res_Transition1]
                    (AU_P1 ,dminAU_P1 ,dmaxAU_P1)
                [f_PS1, f_AU_P2, ctl_PS2]->AU_P2[f_PS1, f_AU_P2, Dado, req_Fim, ctl_PS2, req_Res_Transition1]
                    (AU_P2 ,dminAU_P2 ,dmaxAU_P2)
                [f_PS2, f_AU_P3, ctl_PS3]->AU_P3[f_PS2, f_AU_P3, Dado, req_Fim, ctl_PS3, req_Res_Transition1]
                    (AU_P3 ,dminAU_P3 ,dmaxAU_P3)
                [f_VD_P1, ctl_PS1]->VD_P1[i_VD_P1, f_VD_P1, Dado, req_Fim, ctl_PS1, req_Res_Transition1]
                    (VD_P1 ,dminVD_P1 ,dmaxVD_P1)
                [f_PS1, f_VD_P2, ctl_PS2]->VD_P2[f_PS1, f_VD_P2, Dado, req_Fim, ctl_PS2, req_Res_Transition1]
                    (VD_P2 ,dminVD_P2 ,dmaxVD_P2)
                [f_PS2, f_VD_P3, ctl_PS3]->VD_P3[f_PS2, f_VD_P3, Dado, req_Fim, ctl_PS3, req_Res_Transition1]
                    (VD_P3 ,dminVD_P3 ,dmaxVD_P3)
            endpar
        endloop
        [>req_Fim;exit
    endhide
endproc

process PS1[f_AU_P1, f_VD_P1, f_Restrição, Dado, req_Fim, ctl_Master, req_Res_Transition1]()
    loop forever in
        (f_AU_P1)[> f_VD_P1; ctl_Master; f_Restrição@t[t=0]
        [>req_Res_Transition1
    endloop
    [>req_Fim;exit
endproc

process PS2[f_AU_P2, f_VD_P2, f_Restrição, Dado, req_Fim, ctl_Master, req_Res_Transition1]()
    loop forever in
        (f_AU_P2)[> f_VD_P2; ctl_Master; f_Restrição@t[t=0]
        [>req_Res_Transition1
    endloop
    [>req_Fim;exit
endproc
```

```
process PS3[f_AU_P3, f_VD_P3, f_Restrição, Dado, req_Fim, ctl_Master, req_Res_Transition1]()
    loop forever in
        (f_AU_P3)[> f_VD_P3; ctl_Master; f_Restrição@t[t=0]
        [>req_Res_Transition1
    endloop
    [>req_Fim;exit
endproc

process Transition1[i_Transition1, f_Cena, req_Fim, req_Res]()
    loop forever in
        i_Transition1; req_Res; f_Cena
    endloop
    [>req_Fim;exit
endproc

process AU_P1[i_AU_P1, f_AU_P1, Dado, req_Fim, ctl_PS1, req_Res_Transition1]
    (AU_P1:VideoClass, dminAU_P1:Time, dmaxAU_P1):exit is
    loop forever in
        pSsSme[i_AU_P1, f_AU_P1, Dado](AU_P1, dminAU_P1, dmaxAU_P1)
        [>req_Res_Transition1[]ctl_PS1
    endloop
    [>req_Fim;exit
endproc

process AU_P2[i_AU_P2, f_AU_P2, Dado, req_Fim, ctl_PS2, req_Res_Transition1]
   (AU_P2:VideoClass, dminAU_P2:Time, dmaxAU_P2):exit is
   loop forever in
      pSsSme[i_AU_P2, f_AU_P2, Dado](AU_P2, dminAU_P2, dmaxAU_P2)
      [>req_Res_Transition1[]ctl_PS2
   endloop
   [>req_Fim;exit
endproc

process AU_P3[i_AU_P3, f_AU_P3, Dado, req_Fim, ctl_PS3, req_Res_Transition1]
    (AU_P3:VideoClass, dminAU_P3:Time, dmaxAU_P3):exit is
    loop forever in
        pSsSme[i_AU_P3, f_AU_P3, Dado](AU_P3, dminAU_P3, dmaxAU_P3)
        [>req_Res_Transition1[]ctl_PS3
    endloop
    [>req_Fim;exit
endproc

process VD_P1[i_VD_P1, f_VD_P1, Dado, req_Fim, ctl_PS1, req_Res_Transition1]
    (VD_P1:VideoClass, dminVD_P1:Time, dmaxVD_P1):exit is
    loop forever in
        pSsSme[i_VD_P1, f_VD_P1, Dado](VD_P1, dminVD_P1, dmaxVD_P1)
        [>req_Res_Transition1[]ctl_PS1
    endloop
    [>req_Fim;exit
endproc

process VD_P2[i_VD_P2, f_VD_P2, Dado, req_Fim, ctl_PS2, req_Res_Transition1]
    (VD_P2:VideoClass, dminVD_P2:Time, dmaxVD_P2):exit is
    loop forever in
        pSsSme[i_VD_P2, f_VD_P2, Dado](VD_P2, dminVD_P2, dmaxVD_P2)
        [>req_Res_Transition1[]ctl_PS2
    endloop
    [>req_Fim;exit
endproc

process VD_P3[i_VD_P3, f_VD_P3, Dado, req_Fim, ctl_PS3, req_Res_Transition1]
    (VD_P3:VideoClass, dminVD_P3:Time, dmaxVD_P3):exit is
    loop forever in
        pSsSme[i_VD_P3, f_VD_P3, Dado](VD_P3, dminVD_P3, dmaxVD_P3)
        [>req_Res_Transition1[]ctl_PS3
    endloop
    [>req_Fim;exit
endproc
```

```
process Cena2[i_Cena2, f_Cena2, Dado, req_Fim]
    (Video3: StreamClass, dminVideo3: Time, dmaxVideo3: Time,
    Bitmap1: BitmapClass, dBitmap1: Time,
    Bitmap2: BitmapClass, dBitmap2: Time,
    Bitmap3: BitmapClass, dBitmap3: Time):exit is
    hide i_Video3, f_Video3, f_Bitmap1, f_Bitmap2, f_Bitmap3, req_Res_Transition3 in
        loop forever in
            i_Cena2;
            par f_Video3#2, f_Bitmap1#2, f_Bitmap2#2, f_Bitmap3#2
                [f_Video3]->Video3[i_Video3, f_Video3, Dado, req_Fim, req_Res_Transition3]
                    (Video3 ,dminVideo3 ,dmaxVideo3)
                [f_Video3, f_Bitmap1]->Bitmap1[f_Video3, f_Bitmap1, Dado, req_Fim, req_Res_Transition3]
                    (Bitmap1 ,dBitmap1)
                [f_Bitmap1, f_Bitmap2]->Bitmap2[f_Bitmap1, f_Bitmap2, Dado, req_Fim, req_Res_Transition3]
                    (Bitmap2 ,dBitmap2)
                [f_Bitmap2, f_Bitmap3]->Bitmap3[f_Bitmap2, f_Bitmap3, Dado, req_Fim, req_Res_Transition3]
                    (Bitmap3 ,dBitmap3)
                [f_Bitmap3]->Transition3[f_Bitmap3, f_Cena2, Dado, req_Fim]()
            endpar
        endloop
        [>req_Fim;exit
    endhide
endproc

process Video3[i_Video3, f_Video3, Dado, req_Fim, req_Res_Transition3]
    (Video3:VideoClass, dminVideo3:Time, dmaxVideo3):exit is
    loop forever in
        pSsSme[i_Video3, f_Video3, Dado](Video3, dminVideo3, dmaxVideo3)
        [>req_Res_Transition3
    endloop
    [>req_Fim;exit
endproc

process Bitmap1[i_Bitmap1, f_Bitmap1, Dado, req_Fim, req_Res_Transition3]
    (Bitmap1:BitmapClass, dBitmap1:Time):exit is
    loop forever in
        pSsSe[i_Bitmap1, f_Bitmap1, Dado](Bitmap1, dBitmap1)
        [>req_Res_Transition3
    endloop
    [>req_Fim;exit
endproc

process Bitmap2[i_Bitmap2, f_Bitmap2, Dado, req_Fim, req_Res_Transition3]
    (Bitmap2:BitmapClass, dBitmap2:Time):exit is
    loop forever in
        pSsSe[i_Bitmap2, f_Bitmap2, Dado](Bitmap2, dBitmap2)
        [>req_Res_Transition3
    endloop
    [>req_Fim;exit
endproc

process Bitmap3[i_Bitmap3, f_Bitmap3, Dado, req_Fim, req_Res_Transition3]
    (Bitmap3:BitmapClass, dBitmap3:Time):exit is
    loop forever in
        pSsSe[i_Bitmap3, f_Bitmap3, Dado](Bitmap3, dBitmap3)
        [>req_Res_Transition3
    endloop
    [>req_Fim;exit
endproc

process Transition3[i_Transition3, f_Cena, req_Fim, req_Res]()
    loop forever in
        i_Transition3; req_Res; f_Cena
    endloop
    [>req_Fim;exit
endproc
```

```
process Cena3[i_Cena3, f_Cena3, u_Interação, Dado, req_Fim]
    (Audio2: StreamClass, dminAudio2: Time, dmaxAudio2: Time,
    Interação: SwitchButtonClass, dminInteração: Time, dmaxInteração: Time,
    P4: BitmapClass, dP4: Time,
    ANI_P1: StreamClass, dminANI_P1: Time, dmaxANI_P1: Time,
    ANI_P2: StreamClass, dminANI_P2: Time, dmaxANI_P2: Time):exit is
    hide i_Interação, f_Interação, i_ANI_P1, f_ANI_P1, f_P4, f_Audio2, f_ANI_P2, req_Res_Transition4 in
        loop forever in
            i_Cena3;
            par f_Interação#2, f_P4#2, f_ANI_P1#3
                [f_ANI_P1]->Audio2[f_ANI_P1, f_Audio2, Dado, req_Fim, req_Res_Transition4]
                    (Audio2 ,dminAudio2 ,dmaxAudio2)
                [f_Interação]->Interação[i_Interação, f_Interação, u_Interação, Dado, req_Fim,
                    req_Res_Transition4](Interação ,dminInteração ,dmaxInteração)
                [f_Interação, f_P4]->P4[f_Interação, f_P4, Dado, req_Fim, req_Res_Transition4](P4 ,dP4)
                [f_ANI_P1]->ANI_P1[i_ANI_P1, f_ANI_P1, Dado, req_Fim, req_Res_Transition4]
                    (ANI_P1 ,dminANI_P1 ,dmaxANI_P1)
                [f_ANI_P1]->ANI_P2[f_ANI_P1, f_ANI_P2, Dado, req_Fim, req_Res_Transition4]
                    (ANI_P2 ,dminANI_P2 ,dmaxANI_P2)
                [f_P4]->Transition4[f_P4, f_Cena3, Dado, req_Fim]()
            endpar
        endloop
        [>req_Fim;exit
    endhide
endproc

process Audio2[i_Audio2, f_Audio2, Dado, req_Fim, req_Res_Transition4]
    (Audio2:AudioClass, dminAudio2:Time, dmaxAudio2):exit is
    loop forever in
        pSsSme[i_Audio2, f_Audio2, Dado](Audio2, dminAudio2, dmaxAudio2)
        [>req_Res_Transition4
    endloop
    [>req_Fim;exit
endproc

process Interação[i_Interação, f_Interação, u_Interação, Dado, req_Fim, req_Res_Transition4]
    (Interação:SwitchButtonClass, dminInteração:Time, dmaxInteração):exit is
    loop forever in
        pSsAme[i_Interação, f_Interação, u_Interação, Dado](Interação, dminInteração, dmaxInteração)
        [>req_Res_Transition4
    endloop
    [>req_Fim;exit
endproc

process P4[i_P4, f_P4, Dado, req_Fim, req_Res_Transition4]
    (P4:BitmapClass, dP4:Time):exit is
    loop forever in
        pSsSe[i_P4, f_P4, Dado](P4, dP4)
        [>req_Res_Transition4
    endloop
    [>req_Fim;exit
endproc

process ANI_P1[i_ANI_P1, f_ANI_P1, Dado, req_Fim, req_Res_Transition4]
    (ANI_P1:VideoClass, dminANI_P1:Time, dmaxANI_P1):exit is
    loop forever in
        pSsSme[i_ANI_P1, f_ANI_P1, Dado](ANI_P1, dminANI_P1, dmaxANI_P1)
        [>req_Res_Transition4
    endloop
    [>req_Fim;exit
endproc

process ANI_P2[i_ANI_P2, f_ANI_P2, Dado, req_Fim, req_Res_Transition4]
    (ANI_P2:VideoClass, dminANI_P2:Time, dmaxANI_P2):exit is
    loop forever in
        pSsSme[i_ANI_P2, f_ANI_P2, Dado](ANI_P2, dminANI_P2, dmaxANI_P2)
        [>req_Res_Transition4
    endloop
    [>req_Fim;exit
endproc
```

```
process Transition4[i_Transition4, f_Cena, req_Fim]()
    i_Transition4; req_Fim; f_Cena
endproc

endmod
```

Bibliografia

[BLA96] BLAKOWSKI, G.; STEINMETZ, R. A Media Synchronization Survey: Reference Model, Specification, and Case Studies. **IEEE Journal on Selected Areas in Communications**, New York, v.14, n.1, p.5-35, Jan. 1996.

[BUF94] BUFORD, J. F. K. **Multimedia Systems**. New York: ACM Press, 1994 (SIGGRAPH Series).

[BUL95] BULTERMAN, D. C. A. and HARDMAN, L. Multimedia Authoring Tools: State of the Art and Research Challenges. In: LEEUWEN, Jan Van (Ed.). **Computer Science Today**: Recent Trends and Developments. Berlin: Springer-Verlag, 1995. p. 575-591. (Lecture Notes in Computer Science, v.1000).

[CHA97] CHAPMAN, B. **Enhancing Interactivity and Productivity Through Object-Oriented Authoring:** An Instructional Designer's Perspective. Disponível por WWW em http://www.allencomm.com/. (04 de maio de 1997).

[COS96] COSTA, F. R. et al. Editor e Browser Gráfico para Sincronização Temporal e Espacial de Objetos Multimídia/Hipermídia. In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO GRÁFICA E PROCESSAMENTO DE IMAGENS, 9., 1996, Caxambu, MG. **Anais...** [S.l.: s.n], 1996.

[COU96] COURTIAT, J. P.; OLIVEIRA R.C. Proving Temporal Consistency in a New Multimedia Synchronization Model. **ACM Multimedia**, Boston, 1996.

[DIA93] DIAZ M.; SÉNAC P. Time Streams Petri Nets, a Model for Multimedia Streams Synchronization. In: INTERNATIONAL CONFERENCE ON MULTI-MEDIA MODELLING,1., 1993, Singapore. **Proceedings...** [S.l: s.n], 1993. v. 1, p. 257-273.

[FLU95] FLUCKIGER, F. **Understanding Networked Multimedia:** Applications and Technology. Great Britain: Prentice Hall, 1995.

[GAS98a] GASPARY, L.P.; ALMEIDA, M.J.B. Modeling and Validation Support for Interactive Networked Multimedia Applications. In: JOINT WORKSHOP ON PARALLEL AND DISTRIBUTED REAL-TIME SYSTEMS, (WPDRTS), 1998, Orlando. **Proceedings...** [S.l: s.n], 1998.

[GAS98b] GASPARY, L.P.; ALMEIDA, M.J.B. Ferramenta Grafica para Autoria e Concepcao Formal de Aplicacoes Multimídia Interativas. In: JORNADAS IBEROAMERICANAS DE INGENIERIA DE REQUISITOS Y AMBIENTES DE SOFTWARE, (IDEAS), 1998, Torres. **Anais...** Porto Alegre, 1998. v. 2, p. 224-235.

[GAS98c] GASPARY, L.P.; ALMEIDA, M.J.B. Authoring and E-LOTOS Conception of Interactive Networked Multimedia Applications in MUSE Environment. In: CONFERENCE ON PERFORMANCE OF INFORMATION AND COMMUNICATIONS SYSTEMS, (IFIP, TC6, WG6.3 flagship conference PICS), 1998, Lund, Sweden, 1998. **Proceedings...** [S.l: s.n], 1998.

[GAS98d] GASPARY, L.P.; ALMEIDA, M.J.B. MUSE - An Interactive Networked Multimedia Applications Specification Environment with E-LOTOS Translator. In: CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, (CAISE), 10., 1998, Pisa, Italy. **Proceedings...** Germany, 1998. (Lecture Notes in Computer Science, v.1413).

[GIN95] GINIGE, A.; LOWE D. B. **Hypermedia Authoring**. Sydney: University of Technology, 1995.

[GRA97] GRAVILLE, L.Z.; GASPARY, L. P.; ALMEIDA, M. J. OST - An Object-Oriented Computer Networks System Specification Tool. In: INTELIGENT NETWORKS AND INTELLIGENCE IN NETWORKS, (2IN), 1997, Paris, France. **Proceedings...** [S.l: s.n], 1997.

[HIR95] HIRZALLA, N.; FALCHUK, B.; KARMOUCH, A. A Temporal Model for Interactive Multimedia Scenarios. **IEEE Multimedia**, New York, p. 24-31, Fall 1995.

[ISO96] ISO. **Information Technology Coding of Multimedia and Hypermedia Information, Part 5: Support for Base-Level Interactive Applications, MHEG-5 IS Document Pre-release 5,** IEC DIS 13522-5, [S.l], 1996.

[ISO97] ISO. **Enhancements to LOTOS. Revised Working Drafts on Enhancements to LOTOS (V4),** IEC JTC1/SC21/WG7, [S.l], 1997.

[KOE92] KOEGEL, J. F.; RUTLEDGE, J. L.; HEINES, J. Visual Programming Abstractions for Interactive Multimedia Presentation Authoring. In: IEEE INTERNATIONAL WORKSHOP ON VISUAL LANGUAGES, 1992. **Proceedings...** [S.l: s.n], 1992.

[MAR97] MARTINS, R. F. **E-LOTOS - Aprimoramentos da Linguagem LOTOS**. Nota Interna - Projeto DAMD. Florianópolis: LCMI, 1997.

[SEN94] SÉNAC, P.; DIAZ, M; de SAQUI-SANNES, P. Toward a formal specification of multimedia synchronization scenarios. **Ann. Telécommun**. n. 49, [S.l], p. 297-314, 1994.

[SEN95] SÉNAC, P.; WILLRICH, R.; de SAQUI-SANNES, P. Hierarchical Time Stream Petri Nets: A Model for Hypermedia Systems. In: APPLICATION AND THEORY OF PETRI NETS, 1995. **Proceedings...** [S.l: s.n], 1995.

[SOA97] SOARES, L. F.;RODRIGUES, R. Autoria e Formatação Estruturada de Documentos Hipermídia com Restrições Temporais. In: WORKSHOP SOBRE SISTEMAS MULTIMÍDIA E HIPERMÍDIA, 3., 1997, São Carlos, SP. **Anais...** SP: UFSCar, 1997.

[WAH94] WAHL, T.; ROTHERMEL, K. Representing Time in Multimedia Systems. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA COMPUTING AND SYSTEMS, 1994, Boston, USA. **Proceedings...** [S.l: s.n], 1994.

[WIL96] WILLRICH, R.; de SAQUI-SANNES, P. Concepção Formal de Aplicações Multimídia Java. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 15., 1996, São Carlos, SP. **Anais...** SP: UFSCar, 1996.

Informática  UFRGS

MUSE: Um Ambiente para Modelagem de Aplicações Multimídia Interativas com Tradutor para E-LOTOS.

por

Luciano Paschoal Gaspary

Dissertação apresentada aos Senhores:

Prof. Dr. Wanderley Lopes de Souza (UFSCar)

Prof. Dr. Carlos Alberto Heuser

Prof. Dra. Liane Rockenbach Tarouco

Vista e permitida a impressão.
Porto Alegre, 31/7/98.

Profa. Dra. Maria Janilce B. de Almeida
Orientadora

Profa. Carla Maria Dal Sasso Freitas
Coordenadora do Curso de Pós-Graduação
em Ciência da Computação
Instituto de Informática - UFRGS