

**Janine Kniess**

**Representando Sincronização Multimídia Através  
do Modelo Reflexivo Tempo Real RTR**

Florianópolis – SC  
2000

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA  
COMPUTAÇÃO**

**Janine Kniess**

**Representando Sincronização Multimídia Através  
do Modelo Reflexivo Tempo Real RTR**

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

**Olinto José Varela Furtado**

Florianópolis, setembro de 2000

# Representando Sincronização Multimídia Através do Modelo Reflexivo Tempo Real RTR

Janine Kniess

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração (Sistemas de Computação) e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.



---

Olinto José Varela Furtado, Dr  
(Orientador)



---

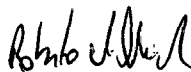
Fernando A. Ostuni Gauthier, Dr  
(Coordenador do Curso)

Banca Examinadora:



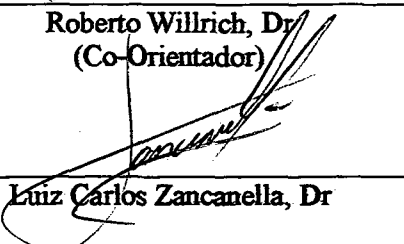
---

Olinto José Varela Furtado, Dr  
(Presidente)



---

Roberto Willrich, Dr  
(Co-Orientador)



---

Luiz Carlos Zancanella, Dr



---

Rômulo Silva Oliveira, Dr

**“Aquele que triunfa não deve esquecer que  
uma vez, em qualquer ocasião,  
alguém lhe deu um auxílio ou uma idéia  
que o encaminhou na direção certa; deve  
lembrar-se também de que tem para  
com a vida o dever de auxiliar outra pessoa  
menos afortunada, da mesma maneira  
que recebeu auxílio.”  
( Napoleon Hill)**

**A meus pais, Daniel Kniess  
e Silézia de Sá Kniess,  
por seu carinho, apoio e  
dedicação em todos  
os momentos de minha vida.**

## **AGRADECIMENTOS**

Ao Prof. Olinto José Varela Furtado, pela orientação, dedicação e empenho, sem os quais este trabalho não teria se realizado.

Agradeço ao Prof. Roberto Willrich pelas sugestões valiosas durante o desenvolvimento deste trabalho.

Aos colegas do CPGCC, pela amizade e apoio durante o desenrolar do curso.

Agradeço aos meus familiares, pela paciência, por suas palavras de incentivo, companheirismo em todos os momentos, inclusive nos mais difíceis.

A Deus, pela oportunidade concedida.

## RESUMO

Sistemas multimídia diferem dos sistemas convencionais por permitirem a interação do usuário com o sistema em grande escala, objetivando reforçar uma idéia, ou exemplificar uma situação. Em função disso, modelos multimídia que induzam a uma técnica de descrição de documentos multimídia vem sendo propostos. Dentre tais técnicas de descrição, destaca-se por sua importância as técnicas para representação da sincronização multimídia. A maioria das técnicas de representação de sincronização multimídia prevêem somente a apresentação simultânea das mídias de forma independente, sem considerar aspectos de sincronização entre elas. Assim sendo cada vez mais são necessários modelos e linguagens que flexibilizem a representação da sincronização multimídia.

Esta dissertação propõe validar o Modelo RTR que explora as potencialidades dos paradigmas de orientação a objetos e reflexão computacional, no sentido de averiguar sua adequação na especificação da sincronização multimídia, visando contribuir para a solução de vários problemas encontrados atualmente nessa classe de especificação.

A potencialidade do modelo é demonstrada através de um exemplo envolvendo situações típicas de um cenário multimídia real. Além disso, uma extensão do modelo para especificação de sincronização multimídia é descrita e exemplificada. Adicionalmente, esta dissertação também apresenta um estudo abrangente sobre os requisitos para um modelo multimídia e sobre os principais modelos e linguagens reflexivas.

## ABSTRACT

**M**ultimedia systems differ from the conventional ones because they allow the interaction of the user with the system in a great scale, reinforcing an idea or exemplifying a situation. Thus, multimedia models are proposed which induce to a technique of description of multimedia documents. Among those techniques of description are to be emphasized, because of their importance, the ones used for representing multimedia synchronization. Most of the techniques used for representing the multimedia synchronization only foresee the simultaneous presentation of the media in an independent way, without considering aspects of their synchronization. Therefore, we more and more need models and languages which turn the representation of the multimedia synchronization more flexible.

This dissertation is intended to evaluate the RTR model which explores the potentialities of the paradigms of orientation toward objects and the computational reflection for programming the applications in real time, in order to investigate their adequacy for the specification of the multimedia synchronization, trying to contribute to the solution of several problems we face at the present moment in that class of specification.

The potentiality of the model is demonstrated through an example that involves typical situations of a real multimedia scenery. Furthermore, an extension of the model for specification of multimedia synchronization is described and exemplified. Additionally, this dissertation also presents a comprehensive study about the requisites for a multimedia model and the main reflexive models and languages.



## SUMÁRIO

<b>Capítulo 1 – Introdução.....</b>	<b>1</b>
1.1 – Motivação.....	1
<b>Capítulo 2 – Multimídia – Visão Geral.....</b>	<b>5</b>
2.1 – Introdução.....	5
2.2 – Abordagem Conceitual.....	5
2.2.1 – Multimídia.....	5
2.2.2 – Classificação dos Tipos de Mídias.....	6
2.3 – Documentos Multimídia e Hipermídia.....	7
2.3.1 – Introdução.....	7
2.3.2 – Documento Hipertexto.....	7
2.3.3 – Documento Multimídia.....	8
2.3.4 – Documento Hipermídia.....	9
2.3.5 – Autoria de Documentos Multimídia.....	10
2.3.5.1 – Criação de Documentos Multimídia.....	10
2.4 – Requisitos para um Modelo Multimídia.....	12
2.4.1- Introdução.....	12
2.4.2 – Estrutura de Conteúdo.....	13
2.4.3 – Estrutura Conceitual.....	14
2.4.3.1 – Componentes e Grupos de Componentes.....	14
2.4.3.2 – Os Caminhos de percurso e as Estruturas de Informação.....	15
2.4.3.3 – Composição Temporal do Documento.....	16
2.4.3.4 – Relações Condicionais.....	16
2.4.3.5 – As Relações Temporais.....	16
2.4.4 – Estrutura de Apresentação.....	19
2.4.4.1 – Especificação das Características de Apresentação.....	19
2.4.4.2 – Interações.....	20
2.5 – Sincronização Multimídia.....	21
2.5.1 – Introdução.....	21
2.5.2 – Especificação de Sincronização Multimídia.....	22
2.6 – Abordagem para Especificação de Documentos Multimídia.....	25
2.6.1 – Introdução.....	25
2.6.2 – Linguagem Scripting.....	25
2.6.3 – Linha Temporal.....	26
2.6.4 – Composição Hierárquica.....	27
2.6.5 – Modelos Baseados em Ícones.....	28
2.6.6 – Modelos Baseados em Cartões ou Páginas.....	29
2.6.7 – Redes de Petri.....	29
2.6.8 – Abordagem Baseada em Informação.....	31
2.6.9 – Modelo de Composição Via Pontos de Referência.....	32
2.6.10 -Abordagem Baseada em Reflexão Computacional.....	33
2.7 – Conclusões.....	33

<b>Capítulo 3 – Reflexão Computacional.....</b>	<b>35</b>
3.1 – Introdução.....	35
3.2 – Arquitetura Reflexiva.....	37
3.3 – Reflexão Computacional e Tempo Real.....	40
3.4 – Programas e Sistemas Reflexivos.....	41
3.4.1 – Introdução.....	41
3.4.2 – Sistemas Reflexivos.....	41
3.4.3 – Programas Reflexivos.....	42
3.5 – Linguagens e Modelos Reflexivos – Trabalhos Existentes .....	43
3.5.1 – Linguagens Reflexivas.....	43
3.5.1.1 – Meta-Classes de Smalltalk.....	44
3.5.1.2 – Meta-Objetos de Open C++.....	45
3.5.1.3 – Protocolo Metajava.....	47
3.5.1.4 – Sistema MetaXa.....	49
3.5.1.5 – Protocolo Java Reflexivo.....	51
3.5.1.6 – OpenJava.....	53
3.5.2 – Modelos Reflexivos.....	54
3.5.2.1 – Real-Time Meta Object Protocol (RT-MOP).....	54
3.5.2.2 – Modelo R <sup>2</sup> .....	56
3.5.2.3 – Modelo Reflexivo Tempo Real RTR.....	59
3.6 – Conclusões.....	62
<b>Capítulo 4 – Modelo RTR – Expressando e Implementando Restrições de Sincronização Multimídia.....</b>	<b>64</b>
4.1 – Introdução.....	64
4.2 – Relações de Sincronização Multimídia.....	64
4.2.1 – Representando os Operadores Baseados em Intervalos Segundo o Modelo RTR.....	68
4.2.1.1 – Operador Before.....	69
4.2.1.2 – Operador Beforeendof .....	70
4.2.1.3 – Operador Cobegin.....	71
4.2.1.4 – Operador Coend.....	71
4.2.1.5 – Operador Delayed.....	72
4.2.1.6 – Operador Startin.....	73
4.2.1.7 – Operador Endin.....	74
4.2.1.8 – Operador Cross.....	74
4.2.1.9 – Operador Overlaps.....	75
4.3 – Exemplo de Especificação da Sincronização Multimídia Utilizando o Modelo RTR.....	76
4.3.1 - Considerações Gerais Sobre o Exemplo Apresentado.....	82
4.4 – Conclusões.....	83
<b>Capítulo 5 – Proposta Alternativa para Especificação do Métodos que Representam o Comportamento dos Operadores de Intervalos.....</b>	<b>85</b>

<b>5.1 – Introdução.....</b>	<b>85</b>
<b>5.2 – Apresentação da Extensão Proposta.....</b>	<b>86</b>
<b>5.3 – Especificação da Sincronização de uma Aplicação Multimídia Utilizando a Extensão Proposta para o Modelo RTR.....</b>	<b>92</b>
<b>5.3.1 – Definição da Aplicação Multimídia.....</b>	<b>93</b>
<b>5.3.2 – Especificação da Sincronização da Aplicação Multimídia.....</b>	<b>96</b>
<b>5.4 – Conclusões.....</b>	<b>99</b>
<b>Capítulo 6 – Conclusões e Perspectivas.....</b>	<b>101</b>
<b>Referências Bibliográficas.....</b>	<b>104</b>
<b>Apêndice A – Pseudo - Código dos Métodos que Verificam a Consistência e Sincronização para os Operadores de Intervalos.....</b>	<b>111</b>

## ÍNDICE DE FIGURAS

<b>FIGURA 2.1</b> – Documentos Multimídia [HAR94].....	8
<b>FIGURA 2.2</b> – Passos na Construção de Documentos Multimídia.....	12
<b>FIGURA 2.3</b> – Tipos de Links e Estruturas para Navegação em um Documento.....	15
<b>FIGURA 2.4</b> – Relações Temporais Básicas entre Intervalos.....	17
<b>FIGURA 2.5</b> – Operadores do Modelo Baseado em Relações de Intervalo Estendido.....	17
<b>FIGURA 2.6</b> – Exemplo de Script.....	25
<b>FIGURA 2.7</b> – Exemplo de Linha Temporal.....	26
<b>FIGURA 2.8</b> – Composição Hierárquica.....	28
<b>FIGURA 2.9</b> – Composição Usando Redes de Petri.....	30
<b>FIGURA 2.10</b> – Sincronização Via Pontos de Referência.....	32
<b>FIGURA 3.11</b> – Arquitetura Reflexiva.....	38
<b>FIGURA 3.12</b> – Ligações de Objetos.....	52
<b>FIGURA 3.13</b> – Data Flow of OpenJava Compiler.....	53
<b>FIGURA 3.14</b> – Estrutura Geral do Modelo R <sup>2</sup> .....	57
<b>FIGURA 3.15</b> – Estrutura Geral do Modelo RTR.....	61
<b>FIGURA 4.16</b> – Representação Gráfica da Relação “Animation while(d1,d2) Audio”.....	66
<b>FIGURA 4.17</b> – Representação Gráfica da Relação “Animation before(d1) Audio”.....	69
<b>FIGURA 4.18</b> – Representação Gráfica da Relação “Animation beforeendof(d1)Audio”.....	70
<b>FIGURA 4.19</b> – Representação Gráfica da Relação “Animation Cobegin(d1) Audio”.....	71
<b>FIGURA 4.20</b> – Representação Gráfica da Relação “Animation Coend(d1) Audio”.....	71
<b>FIGURA 4.21</b> – Representação Gráfica da Relação “Animation Delayed(d1,d2) Audio”.....	72
<b>FIGURA 4.22</b> – Representação Gráfica da Relação “Animation Startin(d1,d2) Audio”.....	73
<b>FIGURA 4.23</b> – Representação Gráfica da Relação “Animation Endin(d1,d2) Audio”.....	74
<b>FIGURA 4.24</b> – Representação Gráfica da Relação “Animation Cross(d1,d2) Audio”.....	75
<b>FIGURA 4.25</b> – Representação Gráfica da Relação “Animation Overlaps(d1,d2)Audio”.....	75
<b>FIGURA 4.26</b> – Exemplo de Sincronização [Bla96].....	77
<b>FIGURA 4.27</b> – Representação da Sincronização Baseado no Modelo de Intervalos.....	77
<b>FIGURA 4.28</b> – Pseudo-Código do OBTR “Controlador”.....	79
<b>FIGURA 4.29</b> – Pseudo-Código do OBTR “Apresentador”.....	81
<b>FIGURA 5.30</b> – Pseudo-Código do Operador Before.....	87

<b>FIGURA 5.31 – Pseudo-Código do Método Sincroniza.....</b>	<b>89</b>
<b>FIGURA 5.32 – Representação Gráfica do Operador Before.....</b>	<b>90</b>
<b>FIGURA 5.33 – Pseudo-Código do Comportamento do Operador Before para Tempos Desconhecidos.....</b>	<b>92</b>
<b>FIGURA 5.34 – Apresentação do Cenário Cena1 do Módulo Turismo.....</b>	<b>94</b>
<b>FIGURA 5.35 – Apresentação do Cenário Cena2 do Módulo Turismo.....</b>	<b>94</b>
<b>FIGURA 5.36 – Apresentação do Cenário Cena3 do Módulo Turismo.....</b>	<b>95</b>
<b>FIGURA 5.37 – Apresentação do Cenário Cena1 do Módulo Síntese Econômica.....</b>	<b>95</b>
<b>FIGURA 5.38 – Apresentação do Cenário Cena2 do Módulo Síntese Econômica.....</b>	<b>96</b>
<b>FIGURA 5.39 – Especificação da Sincronização Baseada na Nova Proposta para o Modelo RTR.....</b>	<b>99</b>

## ABREVIACOES E SIGLAS

<b>CPU</b>	<b>Central De Processamento De Unidade</b>
<b>HTML</b>	<b>Hipertext Markup Language</b>
<b>ISO</b>	<b>International Organization for Standardization</b>
<b>MOE</b>	<b>Meta-Objeto Escalonador</b>
<b>MOG</b>	<b>Meta-Objeto Gerenciador</b>
<b>MOP</b>	<b>Protocolo de MetaObjeto</b>
<b>MOR</b>	<b>Meta-Objeto Relgio</b>
<b>MET</b>	<b>Mximo Tempo de Execuo</b>
<b>Qos</b>	<b>Qualidade de Servio</b>
<b>RTR</b>	<b>Reflexivo Tempo Real</b>
<b>STR</b>	<b>Sistemas Tempo Real</b>

### 1.1 – Motivação

Sistemas multimídia são sistemas formados a partir da união de diversas mídias, com intuito de fixar ou transmitir mais facilmente uma informação. O atributo interação também é considerado como absolutamente necessário para definir um sistema como multimídia [FER95].

Em decorrência da evolução tecnológica, a demanda por Sistemas Multimídia tem aumentado rapidamente nos últimos anos, tornando a tecnologia multimídia extremamente importante e necessária. Pode-se tomar como exemplo aplicações de treinamento e educação, pontos de vendas (exposição de produtos), simulação de processos/máquinas, automatização de sistemas bancários, dentre outras. Todas estas aplicações possuem uma ampla faixa de requisitos funcionais e operacionais a serem contemplados, particularmente destacamos o requisito sincronização, que será objeto de estudo desta dissertação.

Em sistemas multimídia, a necessidade de sincronização entre mídias pode se fazer necessária como por exemplo, legendas sincronizadas com áudio e imagens de uma animação. Dessa forma, é indispensável que estas aplicações apresentem mecanismos que dêem suporte a esta sincronização, fazendo com que estas mídias iniciem e terminem simultaneamente [PAU96].

Para que se possa representar todos os requisitos de sincronização de uma aplicação, é necessário um estilo de especificação adequado. Visando a especificação de aplicações multimídia complexas com garantia de consistência temporal, BLAKOWSKI [BLA96] identifica alguns requisitos importantes que um estilo deve satisfazer: todos

os tipos de relação de sincronização devem ser facilmente descritos; a integração entre objetos de mídia dependentes e independentes do tempo deve ser suportada; níveis hierárquicos de sincronização devem ser suportados para permitir a manipulação de cenários multimídia grandes e complexos. Na literatura são encontrados vários métodos para a especificação de sincronização multimídia. Esses métodos classificam-se nas seguintes categorias: especificação baseada em intervalos, especificação baseada em eixos, especificação baseada em fluxo e especificação baseada em eventos.

Em [BLA96] fica claro que esses métodos não satisfazem completamente todos os requisitos anteriores. Adicionalmente, questões como flexibilidade (capacidade de adaptação de uma aplicação multimídia a diferentes plataformas), reutilização e facilidade/simplicidade de especificação/implementação, também são requisitos importantes para a classe de aplicações multimídia.

Como muitos desses requisitos não são contemplados satisfatoriamente pelas abordagens/técnicas existentes, a busca de soluções alternativas continua sendo uma área ativa de pesquisas.

Neste sentido, este trabalho de dissertação propõe-se a investigar a adequação do Modelo Reflexivo para Tempo Real (RTR) [FUR97], como modelo alternativo para especificação da questão da sincronização multimídia.

O modelo RTR que alia os paradigmas de orientação a objetos e reflexão computacional, tem como objetivo estabelecer uma filosofia para o desenvolvimento de aplicações voltadas para o domínio tempo real, procurando reduzir problemas como o gerenciamento de complexidade e a falta de flexibilidade na representação de aspectos temporais.

Por ser uma extensão do modelo de objetos convencional, o modelo RTR herda seus mecanismos de estruturação e conseqüentemente sua potencialidade relativa ao entendimento, reuso, extensão e manutenção de sistemas [FUR97]. A reflexão computacional, por sua vez, possibilita a uma aplicação o controle sobre seu próprio comportamento, através da separação entre suas atividades funcionais e de



gerenciamento, contribuindo diretamente para uma maior organização interna do sistema e para o aumento da modularidade e flexibilidade.

Em função de suas características, o modelo mostra-se bastante atrativo à programação de aplicações tempo real que sigam a abordagem *soft*. Dentre estas aplicações, encontram-se os sistemas multimídia [FRA95], incluindo mecanismos para especificação da sincronização multimídia.

Adicionalmente, será apresentada uma extensão proposta para o modelo RTR, visando flexibilizar a especificação da sincronização multimídia, abstraindo do usuário aspectos relativos a verificação da consistência da sincronização, e semântica dos operadores de intervalos [WAH94], nos cenários multimídia.

Os resultados alcançados neste trabalho de dissertação visam contribuir para a avaliação do modelo RTR, explorando suas potencialidades na especificação da sincronização em aplicações multimídia.

Esta dissertação está organizada em 6 capítulos. Este primeiro capítulo apresentou a motivação básica para o desenvolvimento da dissertação, enfatizando a importância dos Sistemas Multimídia, e os principais problemas encontrados na especificação dos requisitos de sincronização em aplicações multimídia. Ainda neste capítulo, foram estabelecidos os objetivos e a composição da dissertação.

O capítulo 2 apresenta os principais aspectos relativos a multimídia, iniciando-se com uma abordagem conceitual sobre multimídia, destacando as características sobre autoria de documentos multimídia e os requisitos inerentes à criação de documentos multimídia. Em seguida, diversos requisitos para um modelo multimídia são apresentados, com ênfase nas contribuições e limitações destas propostas. Além disso, este capítulo abordará a questão sincronização multimídia, destacando os métodos para especificação de sincronização multimídia. Concluindo o capítulo, são apresentadas as várias abordagens de especificação da sincronização baseadas nos métodos descritos.

O capítulo 3 descreve o paradigma da reflexão computacional. Inicialmente a arquitetura reflexiva é caracterizada. Adicionalmente, os requisitos, as características,

limitações e contribuições da reflexão computacional em relação a programação convencional são apresentados. Complementando o capítulo, algumas linguagens e modelos reflexivos são demonstrados.

O capítulo 4 descreve a expressividade e potencialidade do modelo RTR, na especificação da sincronização multimídia. Segundo a filosofia do modelo RTR, as relações de sincronização multimídia, podem ser representadas através de restrições temporais básicas presentes no modelo. Na seqüência, a especificação de um cenário multimídia é apresentada visando avaliar a capacidade do modelo RTR na representação e controle da sincronização multimídia.

No capítulo 5, uma extensão proposta para o modelo RTR é apresentada. Inicialmente a dinâmica de funcionamento da extensão proposta é demonstrada, com ênfase para os métodos desenvolvidos afim de representar a semântica dos operadores de intervalos, e a verificação da consistência da sincronização. Adicionalmente, a especificação da sincronização de um cenário multimídia é descrita com base na filosofia da extensão proposta para o modelo RTR.

No capítulo 6 são apresentados as conclusões do trabalho, destacando-se as contribuições, as vantagens e as limitações da abordagem proposta. Finalizando o capítulo são apresentadas as perspectivas relativas a continuidade do trabalho.

## Multimídia – Visão Geral

---

### 2.1 – Introdução

Este capítulo explora diferentes questões relacionadas a multimídia, definindo conceitos e técnicas que nortearam o desenvolvimento deste trabalho. *A priori*, apresentaremos uma abordagem conceitual sobre multimídia, apresentando características sobre autoria de documentos multimídia envolvendo requisitos necessários e obstáculos envolvidos na criação de documentos multimídia. Em seguida, diversos requisitos para um modelo multimídia são apresentados, sendo identificado as contribuições e limitações destas propostas. Ainda neste capítulo, é feita uma breve descrição sobre sincronização multimídia, com ênfase nos métodos para especificação de sincronização multimídia. Na seqüência, após considerarmos a expressividade e limitações dos métodos para especificação de sincronização, são apresentadas e exemplificadas as várias abordagens de especificação da sincronização baseadas nos mecanismos descritos.

### 2.2 – Abordagem Conceitual

Pouco adianta um sistema ser tecnicamente eficiente se ele não se utiliza de uma interface igualmente eficiente. A multimídia busca oferecer uma interface simples e convidativa permitindo explorar ao máximo todos os recursos do sistema [PER95].

#### 2.2.1 - Multimídia

Multimídia é um dos termos mais usados nesta década. Este termo está no cruzamento das cinco maiores indústrias: informática, telecomunicações, publicidade,

consumidores de dispositivos de áudio e vídeo, indústria de televisão e cinema [FLU95].

É a união de várias mídias (meios) com o objetivo de fixar e/ou transmitir mais facilmente uma informação [FER95]. A palavra multimídia é composta de duas partes: o prefixo **multi** e o radical **mídia** [FLU95]:

- **Multi**: originário da palavra latina *multus* que significa “numerosos”. O uso deste prefixo não é recente e muitas palavras de origem latina empregam este radical, como *multiformis* ( que tem várias formas) ou *multicolor* (várias cores).
- **Mídia**: plural da palavra latina *medium* que significa meio, centro. Ele é derivado do adjetivo *medius*, que está no centro. No contexto de multimídia, este radical refere-se ao tipo de informação ou tipo de portador de informação, como dados alfanuméricos, imagens, áudio, vídeo, etc.

### 2.2.2 – Classificação dos Tipos de Mídia

Existem várias definições para sistemas multimídia, como por exemplo: “*são aqueles que integram vários tipos de mídia, como texto, imagens, sons e o vídeo*”. Esta definição certamente não é suficientemente precisa para a definição de multimídia. Nestes termos qualquer sistema integrando dois ou mais tipos de informações poderia ser classificado como sendo multimídia. Por exemplo, como qualquer arquivo *Word* integra basicamente três tipos de mídia, texto, gráficos e imagens, poderia ser classificado como sendo uma aplicação multimídia.

Segundo [FLU95], a definição de sistemas multimídia mais aceita é a seguinte:

**SISTEMA MULTIMÍDIA É UM SISTEMA CAPAZ DE MANIPULAR AO MENOS UM TIPO DE MÍDIA DISCRETA E UM TIPO DE MÍDIA CONTÍNUA, AS DUAS NUMA FORMA DIGITAL.**

## 2.3 – Documentos Multimídia e Hiperarquia

### 2.3.1 – Introdução

Enquanto documentos multimídia podem ser vistos como uma estrutura descrevendo a coordenação e o estilo de apresentação de uma coleção de componentes constituídos de mídias estáticas e dinâmicas, documentos hiperarquia podem ser vistos como uma combinação de documentos hipertexto e multimídia.

Nesta seção será apresentada uma descrição detalhada da estrutura e das funcionalidades dos diversos tipos de documentos multimídia existentes na literatura. Adicionalmente, serão apresentadas as características pertinentes à autoria de documentos multimídia.

### 2.3.2 – Documento Hipertexto

Documento eletrônico que permite percorrer o texto de forma não-linear ou ainda não seqüencial, “navegando” entre as diversas informações (textos) [FER95]. Os dados são armazenados em uma rede de nós conectados por ligações ou links.

- **Nós:** contém as unidades de informações compostas por texto e outras informações gráficas. São os textos que podem ser interligados.
- **Links:** definem as relações lógicas (ou semânticas) entre os nós; isto é, definem relações entre conceitos e idéias. A noção de âncora, permite a especificação de uma parte da informação que será parte ou destino de um link.

Qualquer palavra em um hipertexto pode ser palavra-chave. Essas palavras funcionam como “botões” que quando acionadas levam a outros textos. Estes outros textos, por sua vez podem conter outras palavras-chave, que levam a outros textos e assim sucessivamente [FER95]. Como textos e imagens são informações estáticas (as informações não evoluem no tempo), a noção de tempo não é utilizada quando da

especificação de documentos hipertextos clássicos. No hipertexto, o tempo de apresentação é determinado pelo leitor do documento.

Em um CD-Rom onde a capacidade de armazenamento é muito grande, o hipertexto é valioso. Este auxilia a busca de informações e oferece uma maneira inovadora de organizar o texto [MAR97], como por exemplo, enciclopédias (por possuírem um índice) e dicionários (por apresentarem um esquema de procura linear).

### 2.3.3 – Documento Multimídia

Um documento multimídia pode ser visto como uma estrutura descrevendo a coordenação e o estilo de apresentação de uma coleção de componentes constituídos de mídias estáticas e dinâmicas. A Fig. 2.1 apresenta um pequeno documento multimídia. Um exemplo de documento multimídia real pode ser a apresentação dos pontos turísticos de uma cidade, constituído de textos, imagens e vídeos de cada ponto turístico, apresentados em uma ordem seqüencial e possivelmente com alguns mecanismos de interação.

Quando o autor cria um documento multimídia, ele define a **orquestração** da apresentação dos componentes a partir da definição dos instantes de início e fim das

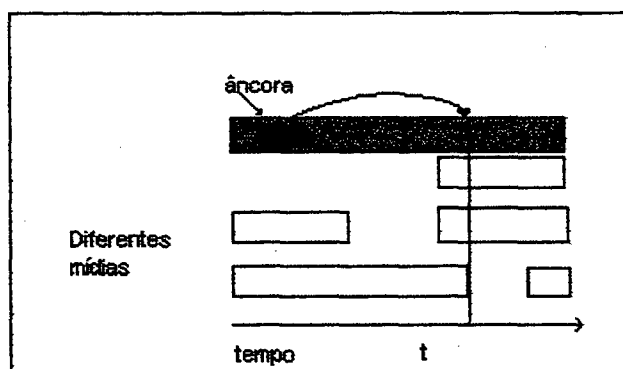


FIGURA 2.1 - Documento Multimídia [Har94]

apresentações e das relações condicionais e temporais entre e no interior dos componentes. Esta descrição da ordem temporal relativa de apresentação dos componentes é chamada **cenário multimídia**.

Documentos podem ser **interativos**. Tipicamente, existem dois métodos de interação a partir do qual o leitor pode controlar a apresentação [HAR94]:

- **um método de controle de apresentação**, que permite o ajuste da referência temporal utilizada pela apresentação. Com este mecanismo, o leitor pode interromper, repetir, fazer avanços ou retorno rápidos na apresentação do documento, isto a partir de uma interface similar aos controles de um videocassete.
- **um método de interação similar a um link hipertexto rudimentar**, definido por uma âncora e um link. Seguindo o link, a apresentação salta para outra parte do documento, como ilustrado na Fig. 2.1. Quando o leitor clica sobre a âncora, a apresentação salta para o ponto t.

### 2.3.4 – Documento Hipermídia

Um documento hipermídia é uma combinação de documento hipertexto e multimídia. Esta classe de aplicação representa uma evolução natural do hipertexto, na qual as noções ou conceitos de nós hipertexto podem agora ser expressos por diferentes tipos de mídia. A inclusão de dados multimídia aumenta o poder de expressão da informação contida em uma aplicação e torna a apresentação mais atrativa e realista.

Como nos documentos hipertextos, as âncoras permitem a especificação de uma parte da apresentação de uma mídia que será fonte ou destino de um *link hipermídia*. Por exemplo, uma âncora pode ser uma seqüência de quadros de um vídeo ou uma região de uma imagem.

Os links hipermídia podem ser temporizados. Este tipo de link é habilitado durante um determinado intervalo temporal e ele pode ser ativado automaticamente em função das suas restrições temporais. Por exemplo, um link temporizado pode levar a uma página de ajuda caso o leitor não ative nenhum outro link em 30 segundos. Este tipo de link introduz a noção de sistemas hipermídia ativos.

A inclusão de dados multimídia em estruturas hipertexto introduz um aspecto temporal na especificação de aplicações hipermídia. Dessa forma é necessário que os

modelos de especificação hipermídia forneçam mecanismos que permitam a integração temporal de uma variedade de mídias contínuas e discretas. Além disso, ele deve fornecer mecanismos de descrição das possíveis interações com o leitor do documento, isto é ele deve permitir a descrição dos links hipertextos (com um caráter temporal), e outros métodos de interação.

### **2.3.5 – Autoria de Documentos Multimídia**

Ambientes de desenvolvimento multimídia facilitam e automatizam a autoria (criação) de documentos multimídia. Há uma grande variedade de tais ambientes, também chamados de sistemas de autoria. Estes sistemas de autoria são projetados para fornecer ferramentas de criação e organização de uma variedade de mídias como textos, gráficos, imagens, animações áudios e vídeos a fim de produzir documentos multimídia. Os usuários deste tipo de software, os autores de documentos, são profissionais que desenvolvem apresentações educacionais e de marketing e artistas gráficos que fazem decisões acerca do *layout* gráfico e estilo de interação que os usuários finais vêem e ouvem [BUF94].

Sistemas de autoria geralmente fornecem meios para gerar certos tipos de interações comuns através do uso de templates reutilizáveis ou módulos de software, que podem ser personalizados pelo projetista para um objetivo particular. Como exemplo, muitos programas de treinamento usam um formato de questão múltipla escolha. Existem também pacotes de software de autoria dedicados a produção de aplicações de realidade virtual. Estes softwares de modelagem tridimensional permitem ao autor criar formas, luminosidade, e regras de comportamento para o ambiente virtual.

#### **2.3.5.1 – Criação de Documentos Multimídia**

A autoria de documentos multimídia é um processo complexo. Ilustraremos os passos envolvidos e olharemos os obstáculos mais comuns.

Embora diferentes documentos multimídia possam impor diferentes requisitos, há um conjunto de passos relativamente consistentes que o autor deve seguir quando ele



desenvolve um documento. Estes passos, apresentados na Fig. 2.2, guiam o autor da inspiração inicial ao documento acabado. Como apresentado na Fig. 2.2, o processo de autoria pode ser dividido em quatro estágios:

- **análise e projeto preliminares**, em que os requisitos para o documento, seu conteúdo e suas interfaces são especificados;
- **aquisição de material**, em que os materiais que formarão o documento são coletados, criados ou digitalizados;
- **composição do documento**, em que é realizada a composição lógica (através de links), temporal e espacial dos componentes do documento;
- **avaliação e liberação**, em que o documento é testado, refinado e, finalmente, distribuído para sua audiência.

Observe na Fig. 2.2 que há realimentações importantes no processo de autoria (por exemplo, na composição do documento pode-se verificar a necessidade da aquisição de novos materiais). Ainda nesta figura, estes quatro estágios não são sempre realizados em seqüência. Certos estágios podem ser realizados em paralelo (por exemplo, a aquisição de material pode ocorrer simultaneamente com a composição do documento). Além disso, a criação do documento pode ser feita via prototipagem, onde as etapas de aquisição de material, composição e avaliação podem ser repetidas até a conclusão do documento. Mas a etapa de análise e projeto preliminar deve ser a mais completa possível, afim de aumentar as chances de se alcançar as metas definidas.

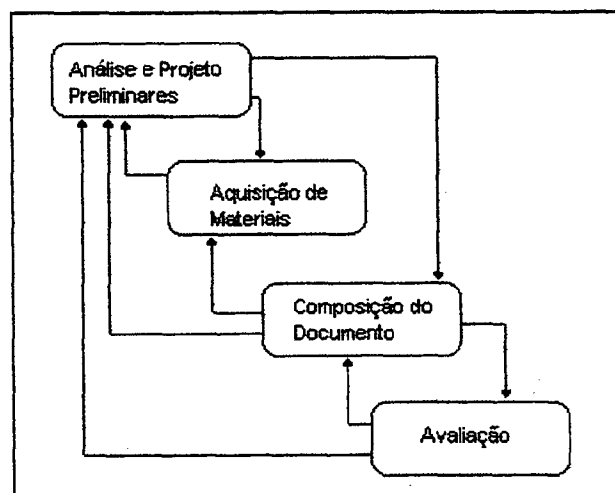


FIGURA 2.2 - Passos na Construção de Documentos Multimídia

A seção seguinte, abordará os principais requisitos apresentados na literatura, para construção de um modelo multimídia.

## 2.4 – Requisitos para um Modelo Multimídia

### 2.4.1 – Introdução

Ferramentas de autoria de documentos multimídia são baseados em um modelo multimídia que induz uma técnica de descrição de documentos multimídia. Baseado na ISO ODA (*Office Document Architecture*) [ISO8613] várias abordagens dividem a descrição de documentos em três partes:

- **estrutura de conteúdo**, descreve as informações que constituem os componentes.
- **a estrutura de apresentação**, que descreve como, onde e quando os diferentes componentes serão apresentados.
- **estrutura conceitual**, que especifica a organização semântica do documento seus componentes, suas relações lógicas e temporais e em que instante estes componentes serão apresentados;

Segundo [FLU95], a partição da descrição de documentos nestas três estruturas oferece várias vantagens. A separação entre a estrutura lógica e de apresentação, por exemplo, permite a reutilização da estrutura lógica quando da apresentação do documento em dispositivos diferentes, onde somente uma modificação simples ou adaptação da estrutura de apresentação deve ser realizada; e a separação entre a estrutura de apresentação e do conteúdo é necessária pois os documentos podem utilizar os mesmos dados em diferentes contextos [SCH94] ou por diferentes documentos.

De forma independente a essas três estruturas, um modelo para essas aplicações deve permitir também a especificação dos possíveis métodos de interação com o usuário. Dessa forma, um modelo deve permitir a definição de âncoras, de links hipermídia e de outros métodos de interação de documentos multimídia [SAM98].

### 2.4.2 - Estrutura de Conteúdo

Uma das primeiras etapas da realização de um documento multimídia é a geração ou captura dos materiais (textos, imagens, áudios, vídeos, etc.) que vão ser utilizados para compor o documento. Por exemplo, o autor pode usar uma câmera de vídeo para registrar uma seqüência de vídeo, criar uma informação gráfica a partir de um editor. A estas informações daremos o nome de **dados primitivos**.

A estrutura de conteúdo é responsável pela descrição desses dados primitivos. Por exemplo, ela pode ser constituída por um conjunto de dados primitivos e de suas descrições. O par constituído de dados primitivos e da descrição de seus dados é denominado **objeto multimídia**.

Ao nível da estrutura de conteúdo, uma aplicação multimídia deve permitir a especificação das informações de acesso e de manipulação dos dados primitivos e dos valores originais das características espaciais, sonoras e temporais de apresentação (por exemplo, a velocidade e tamanho originais da apresentação de uma seqüência de vídeo).

### 2.4.3 - Estrutura Conceitual

A estrutura conceitual especifica os componentes e os grupos de componentes de uma aplicação, e a composição lógica e temporal desses componentes [WIL96].

#### 2.4.3.1 - Componentes e Grupos de Componentes

A estrutura conceitual contém a definição dos componentes semânticos da aplicação, permitindo a divisão de um documento, por exemplo, capítulos e parágrafos, ou uma série de seqüências de vídeo e de títulos.

A tarefa de especificação de documentos multimídia se torna delicada e complexa com o aumento de tamanho do documento. A estrutura conceitual é utilizada para a construção de apresentações complexas a partir de pequenos grupos de informação. Assim, a descrição do documento pode ser dividida em seções, cada seção pode ser criada de forma independente das outras seções [HAR95]: as apresentações que expressam um conjunto ou uma idéia ou um conceito podem ser agrupadas e este agrupamento pode ser reutilizado em qualquer parte do documento. Permite-se, desta forma a definição das relações lógicas e temporais entre esses grupos e a definição de restrições temporais associadas a um grupo de apresentações. Concluindo, a estrutura conceitual introduz o benefício da modularidade, da encapsulação e mecanismos de abstração.

#### 2.4.3.2 - Os Caminhos de Percurso e as Estruturas de Informação

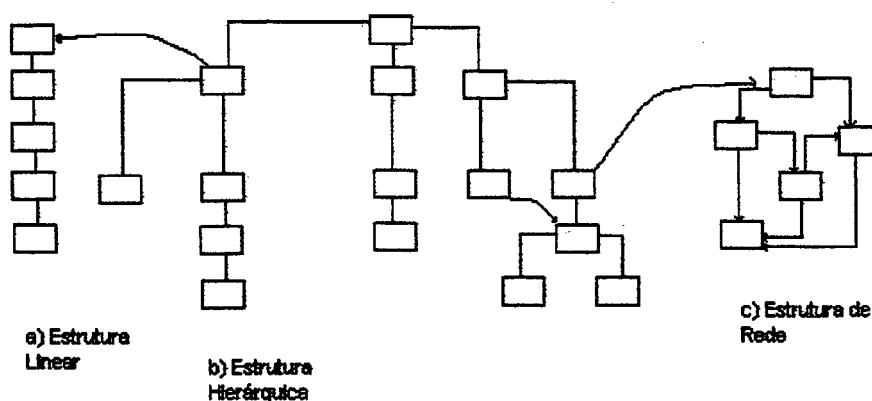
Estrutura conceitual define os caminhos de percurso do documento. Estes caminhos de percursos são definidos pelos links. Fundamentalmente, existem dois tipos de links [SHA93]:

- **link estrutural**, que fornece uma estrutura básica de um documento. Quando o usuário segue esses links, a estrutura base é preservada como foi definida pelo autor do documento. Na Fig. 2.3, os traços que ligam os componentes (representados por quadrados) são links estruturais.

- **link hiper-estrutural**, que permite a definição das relações que não seguem a estrutura base de um documento. Estes links podem ser divididos em links associativos e referenciais [GIN95]. Os links **associativos** conectam conceitos associados e os links **referenciais** conectam informações adicionais a um conceito. Na Fig. 2.3, as flechas representam os links hiper-estruturais do documento.

Segundo [GIN95], são definidos três tipos básicos de estruturas:

- **estruturas Lineares** (Fig. 2.3a): ela é utilizada nos documentos do tipo "visita guiada", onde o documento fornece um procedimento de ajuda com uma lista de ações que devem ser executadas em uma ordem particular. Por exemplo, as estruturas lineares estão presentes no material de treinamento, pois um instrutor normalmente ensina os módulos de uma maneira seqüencial. O autor pode incluir questões em vários pontos, caso o leitor der uma resposta errada, informações adicionais podem ser apresentadas via links referenciais.
- **estrutura Hierárquica** (Fig. 2.3b): ela está presente nos documentos comparáveis aos livros (organizados em capítulos e seções), e o autor pode utilizar, por exemplo, links referenciais para um glossário ou um seção de referências.
- **estrutura em Rede** (Fig. 2.3c): contém ligações associativas. Estes links são de natureza semântica e pragmática, e são verdadeiramente não seqüenciais. A estrutura em rede é mais adequada para a organização de informação na forma de enciclopédia.



**FIGURA 2.3 - Tipos de Links e Estruturas para Navegação em um Documento**

### 2.4.3.3 - Composição Temporal do Documento

A estrutura conceitual define também a estrutura temporal de uma aplicação, que consiste na descrição dos instantes de partida e parada das apresentações dos componentes e de suas relações condicionais (ou causais) e temporais. Essas relações são estabelecidas por eventos definidos no interior de diferentes apresentações. Existem dois tipos de eventos que podem ocorrer durante uma apresentação [WIL96]:

- **eventos síncronos (previsíveis):** são os eventos cujas posições no tempo são determinadas previamente (por exemplo, início da apresentação de uma seqüência de áudio ou vídeo). A posição destes eventos pode ser determinada somente sob condições ideais (sem considerar os atrasos imprevisíveis, como sobrecarga na rede).
- **eventos assíncronos (imprevisíveis):** são os eventos cujas posições no tempo não podem ser determinada previamente. Como o instante em que a aplicação chega a um determinado estado ou a interação do usuário.

### 2.4.3.4- Relações Condicionais

Uma relação condicional é definida como uma condição associada a um conjunto de componentes, e ações que serão aplicadas a um conjunto de componentes quando esta condição é satisfeita. Por exemplo, “após o término da apresentação de A, se o link L está ativado, então apresentar B”, é uma relação condicional [WIL96].

### 2.4.3.5 - As Relações Temporais

Para a descrição das relações temporais, um modelo de composição requer um modelo temporal [WIL96]. Com relação a sua unidade elementar, duas classes de modelos temporais podem ser identificadas [WAH94]:

- **modelo temporal baseado em pontos:** são os modelos cuja unidade temporal são os eventos. Existem três relações temporais que podem ser definidas entre dois eventos: um evento pode ocorrer antes, durante e após outro evento.

- modelo temporal baseado em intervalo:** são os modelos cuja unidade temporal são os intervalos. [HAM72] e [ALL83] definem 13 relações temporais possíveis entre dois intervalos demonstrados na Fig. 2.4: *antes*, *precede*, *sobreposto*, *durante*, *começo*, *fim* e *igual*, e mais as relações inversas (com exceção da relação igual).

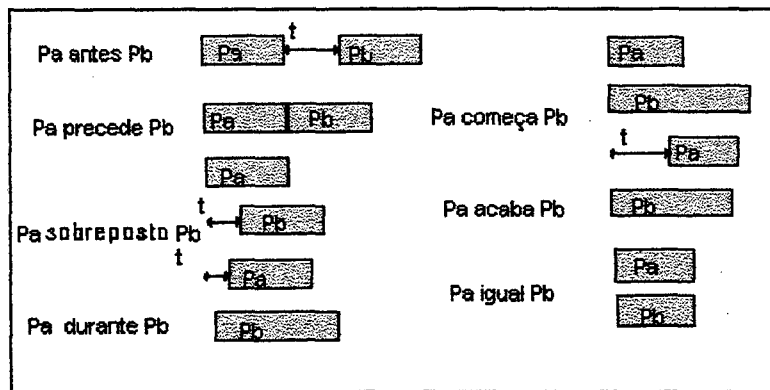


FIGURA 2.4 - Relações Temporais Básicas entre Intervalos

Por sua vez, o modelo baseado em intervalos estendido [WAH94] consiste de 29 relações de intervalos (definido a partir de disjunções das 13 relações básicas) que são consideradas relevantes para apresentação multimídia. Estas relações podem ser representadas de forma simplificada, pelos 10 operadores apresentados na Fig. 2.5.

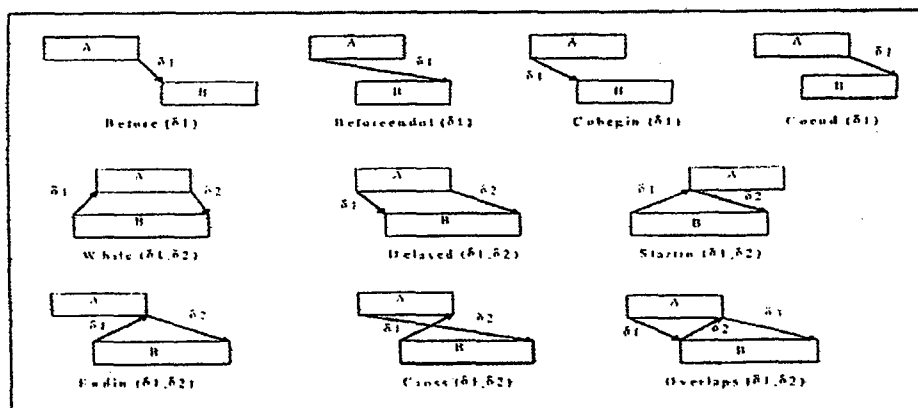


FIGURA 2.5 – Operadores do Modelo Baseado em Relações de Intervalo Estendido

WAHL [WAH94] demonstra que os modelos baseados em intervalos são mais apropriados às aplicações multimídia pois:

- **abordagens baseadas em intervalos** abrangem um número maior de relações que as abordagens baseadas sobre pontos e;
- **abordagens baseadas em intervalos** oferecem uma abstração de mais alto nível para a descrição das relações temporais multimídia, pois a duração de apresentação dos componentes é visível com os intervalos temporais (as apresentações devem ter duração diferente de zero e infinito).

Estas relações temporais definem sincronizações que podem ser definidas entre eventos definidos dentro de uma apresentação ou entre eventos definidos em apresentações distintas:

- **sincronização intramídia**: são as relações temporais entre os eventos ou intervalos definidos no interior de uma mídia contínua. Por exemplo, definidos entre os quadros de uma seqüência de vídeo.
- **sincronização intermídia**: são as relações temporais entre os eventos ou intervalos definidos em diferentes apresentações (mídias).

Em geral, as sincronizações intramídia são dependências temporais naturais que são definidas implicitamente quando da produção dos dados primitivos (na estrutura do conteúdo). O esquema de sincronização intramídia pode ser alterado pelo autor do documento quando da definição da estrutura de apresentação. Por exemplo, a sincronização entre dois quadros de uma seqüência de vídeo pode ser definida pela velocidade de apresentação desta seqüência. As sincronizações intermídia são geralmente dependências temporais artificiais especificadas explicitamente pelo autor da estrutura conceitual de um documento. Assim, em geral as sincronizações intermídia são descritas pela estrutura conceitual do documento, e as sincronizações intramídia fazem parte da estrutura de apresentação.

As relações temporais de um documento podem ser expressas de forma estática, que consiste em organizar explicitamente as relações temporais entre os componentes (**orquestração**); ou de forma dinâmica, em que as relações temporais denominadas relações ao vivo são definidas [FLU95]. Alguns exemplo são: a sincronização entre um vídeo e um áudio em tempo-real, ou as sincronizações de um trabalho cooperativo.



Na seção 2.5 será abordado aspectos específicos à sincronização em aplicações multimídia, e descrito diversos métodos para a especificação de sincronização multimídia.

#### **2.4.4 - Estrutura de Apresentação**

A estrutura de apresentação de um documento descreve as características espaciais, sonoras e temporais de cada apresentação que compõem o documento. A este nível, o autor deve especificar as características de apresentação de cada componente e a composição espacial destas apresentações em um dado instante. A especificação das características de apresentação inclui a descrição da maneira como o componente será visto (descrição das características sonoras) pelo leitor do documento. A composição espacial descreve também as relações espaciais entre as apresentações.

##### **2.4.4 .1- Especificação das Características de Apresentação**

A estrutura conceitual define os componentes do documento e suas relações. A partir disto, o autor deve definir o conteúdo dos componentes (isto é, associar um objeto multimídia ao componente) e particularizar e/ou definir suas características de apresentação. Por exemplo, o autor pode trocar as características originais de apresentação dos objetos multimídia (como o volume sonoro, velocidade de apresentação, etc.), ou ele pode definir novas características como a posição espacial de apresentação de uma imagem. Afim de modelar uma apresentação, um modelo multimídia deve permitir a especificação das seguintes informações:

- **características temporais de apresentação das informações dinâmicas**, como a velocidade, posição de início e de fim de um vídeo e o número de repetições;
- **características espaciais de apresentação de informações visuais**, como o tamanho, a posição e o estilo de apresentação;
- **características das apresentações sonoras**, como o volume de apresentação;
- **dispositivos de saída**, chamados aqui de canais, nas quais as informações serão apresentadas e vista pelo leitor (por exemplo, uma janela, ou um canal de áudio).

Apresentações alternativas podem ser definidas afim de repor uma apresentação principal se ela não puder ser apresentada em um certo sistema (permitindo a criação de documentos adaptáveis aos recursos disponíveis), se existirem problemas de acesso, ou restrições temporais não satisfeitas.

#### 2.4.4.2- Interações

Em aplicações interativas (por exemplo, multimídia interativa) o usuário deve dispor de um conjunto de mecanismos que permitam o controle da apresentação. Existem basicamente quatro métodos de interação [HAR95]:

- **navegação:** este método permite a especificação de um conjunto de escolhas dado ao usuário a fim de que ele possa selecionar um contexto entre vários. Ela é geralmente definida através da criação de um *link* ou *script* que liga as âncoras origens as âncoras destinos;
- **controle de apresentação:** este método de interação é freqüentemente encontrado em documentos multimídia, onde o leitor pode parar, recomeçar, avançar ou retroceder a apresentação do componente multimídia;
- **controle do ambiente:** este método permite a particularização do ambiente de apresentação do documento. Por exemplo, o leitor pode desativar o canal de áudio ou ainda pode alterar o tamanho de uma janela;
- **interação da aplicação:** nos métodos anteriores, o autor cria o documento e o leitor apenas interage com ele. Existem aplicações que requerem mecanismos específicos, por exemplo, nas aplicações de tele-ensino, o modelo deve permitir a especificação da noção de acompanhamento dos alunos (por exemplo, a partir de um campo de entrada de dados). Outro método de interação é a pesquisa por palavras-chave. Este tipo de mecanismo de interação é suportado por ferramentas especializadas.

Um modelo multimídia ideal deveria permitir a especificação de todos estes tipos de interação. Além disso, afim de simplificar a tarefa de estruturação conceitual de um documento multimídia, um modelo deveria fornecer uma abordagem uniforme de

representação dos componentes, das relações lógicas e temporais, e dos mecanismos de interação.

## 2.5 – Sincronização Multimídia

### 2.5.1 – Introdução

Mídias baseadas no tempo, como vídeo e o áudio, requerem algum tipo de sincronização. Um telejornal por exemplo, pode ser utilizado para ilustrar a importância da temporização na multimídia. Vários tipos de mídias, produzidas de maneiras diferentes chegam na tela da televisão com algumas considerações críticas de tempo. O *storyboard* de um telejornal normalmente faz distinção entre voz, vídeo, gráficos, imagens e o próprio apresentador. A importância da voz estar sincronizada com os movimentos da boca do apresentador é a mais simples e aparente sincronização requerida [RAD95].

A maioria das técnicas de representação de sincronização das diversas mídias na multimídia prevê somente a apresentação simultânea das mídias de forma independente, sem considerar aspectos de sincronização entre elas [RAD95]. Entretanto, em sistemas multimídia, a necessidade de sincronização entre mídias pode se fazer necessária (como por exemplo, legendas sincronizadas com áudio e imagens de uma animação). Dessa forma, é indispensável que estas aplicações apresentem mecanismos que dêem suporte a esta sincronização, fazendo com que estas mídias iniciem e terminem simultaneamente [PAU96].

O avanço dos sistemas multimídia são caracterizados pela geração integrada e controlada por computador de armazenamento, comunicação, manipulação e apresentação de mídias dependentes e independentes de tempo. A principal questão com relação a esse aspecto, é como fornecer integração, a representação digital de alguns dados e a sincronização entre vários tipos de mídias e dados [BLA96]. É por isso que um modelo multimídia deve permitir a especificação de **métodos de tolerância de sincronização** [WYN95]. Assim, o autor pode expressar quais compromissos de

sincronização são aceitáveis e os meios de tratar as exceções quando da violação desses compromissos.

### 2.5.2 – Especificação de Sincronização Multimídia

Devido a complexidade da sincronização para vários objetos em uma apresentação multimídia, incluindo interação com o usuário, são necessários métodos de especificação sofisticados, os quais devem satisfazer as seguintes exigências [BLA96]:

- todos os tipos de relação de sincronização devem ser facilmente descritos;
- a integração de objetos de mídia dependentes e independente do tempo deve ser suportada;
- o método deve permitir a definição de QoS (*Quality of Service*);
- níveis hierárquicos de sincronização devem ser suportados para permitir a manipulação de cenários multimídia grandes e complexos.

Na literatura são encontrados vários métodos para a especificação de sincronização multimídia. Esses métodos classificam-se segundo [BLA96] nas seguintes categorias:

- **especificação baseada em fluxo de controle:** na especificação baseada em fluxo de controle, o fluxo das *threads* de apresentação concorrentes, é sincronizado em pontos pré-definidos da apresentação. Esta especificação subdivide-se em:
  - a) **Especificação hierárquica:** a descrição de sincronização hierárquica [AFN89], [SHE90], tem como base dois principais operadores de sincronização: sincronização serial e sincronização paralela. Na especificação de sincronização hierárquica, objetos multimídia são considerados como uma árvore constituída de nós, que denotam a apresentação serial ou paralela da saída da sub-árvore. Estruturas hierárquicas podem ser sincronizadas em seu início ou fim. Trocando o nível de sincronização por um nível inferior, introduz-se vários pontos de sincronização adicionais para cada ponto de sincronização anterior. Obtém-se

assim uma granularidade mais fina de dados, e portanto maior precisão na sincronização.

- b) **Pontos de referência:** no caso de sincronização via pontos de referência [BLA96], um objeto de mídia dependente do tempo é considerado como uma seqüência de unidades de dados lógicos. O tempo inicial e final da apresentação de um objeto de mídia, em adição ao tempo inicial das sub-unidades de objetos de mídia dependentes do tempo, são chamados pontos de referência. A sincronização entre objetos é definida pela conexão dos pontos de referência dos objetos de mídia. A apresentação das sub-unidades que participam do mesmo ponto de sincronização, devem ser iniciada ou finalizada quando o ponto for alcançado. Este método de sincronização, especifica relações temporais entre objetos sem referência explícita ao tempo.
- c) **Redes de Petri:** permitem todos os tipos de especificação de sincronização. O principal problema relacionado a esse método de especificação, são as complexas especificações e a insuficiente abstração do conteúdo dos objetos de mídia [BLA96].
- **especificação baseada em intervalos:** na especificação de sincronização baseada em intervalo de tempo, a duração de um objeto de mídia é chamado intervalo. A vantagem da especificação baseada em intervalos é que manipula facilmente unidades lógicas de mídias, tanto quanto a interação do usuário. É possível especificar relações temporais não-determinísticas adicionais pela definição de intervalos para duração e atrasos. É um modelo flexível que permite a especificação de apresentações com variações na apresentação em tempo de execução [BLA96].
  - **especificação baseada em eixos:** também chamado de modelo de tempo contínuo, onde objetos de mídia são associados a um eixo de tempo, que representa uma abstração do tempo. A remoção de um objeto não afeta a sincronização de outros. Este modelo tem dificuldades para descrever sincronização baseada em eventos ou estruturas de dados complexas [BLA96].
  - **especificação baseada em eventos:** no caso de sincronização baseada em eventos, a apresentação das ações são iniciadas pelos eventos de sincronização como por exemplo em, *HyTime* e *HyperODA*. Segundo [BLA96] ações típicas de uma

apresentação são: iniciar uma apresentação, parar uma apresentação e preparar uma apresentação. Os eventos que iniciam as ações de uma apresentação, podem ser externos (por exemplo, gerado por um temporizador) ou interno gerado por um objeto de mídia dependente do tempo que alcança uma LDU (Unidade de Dado Lógico) específica. Este tipo de especificação é facilmente estendido para novos tipos de sincronização. O maior inconveniente é que este tipo de especificação torna difícil a manipulação de cenários realísticos [Bla96].

- **especificação baseada em Scripts:** Um *script* neste contexto, é uma descrição textual de um cenário de sincronização multimídia [IBM,90], [TSI91]. Scripts são muito poderosos, porque eles representam ambientes complexos de programação. Uma desvantagem, é que este método é mais procedural que declarativo. Os métodos declarativos, oferecem maior flexibilidade de manipulação pelo usuário [BLA96].

Na seção 2.6, apresentaremos as abordagens existentes com base nos mecanismos de especificação descritos. Adicionalmente, será demonstrada as potencialidades da técnica da Reflexão computacional, dentre as quais citamos, abstração, reutilização e modularização, como um mecanismo atrativo para especificação da sincronização multimídia.

## 2.6 - Abordagens para Especificação de Documentos

### Multimídia

#### 2.6.1 - Introdução

Existem várias ferramentas de autoria. Uma lista com links para as diversas ferramentas de autoria comerciais e de domínio público pode ser encontradas em [FAQ97]. Um número de diferentes paradigmas básicos são suportados pelos sistemas de autoria multimídia. Mostraremos nesse capítulo alguns paradigmas ou abordagens para especificação de documentos multimídia.

#### 2.6.2 - Linguagem Scripting

O paradigma Scripting, ou baseado em linguagens, é o método de autoria no estilo da programação tradicional apresentado na Fig. 2.6. Neste caso nós temos uma linguagem de programação que especifica elementos multimídia (por nome de arquivo), sequenciamento, hotspots, sincronização, etc. Linguagens Scripting são interpretadas, ao invés de compiladas, a diferença de velocidade de execução em comparação a outros métodos de autoria é mínima. Esta forma de concepção de documentos é adotada por [GIB91], [HER94], [KLA90] e [VAZ93].

```
set win=main_win
set cursor=wait
clear win
pu background "pastel.pic"
"put text "heading1.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 2.6 - Exemplo de Script

Estes modelos têm um poder de expressão muito grande, mas a especificação da composição de um documento multimídia na forma textual é difícil de produzir e

modificar ([ACK94], [HUD93]). Além disso, a composição temporal dos componentes é difícil de identificar.

Os modelos gráficos, vistos mais adiante, têm a vantagem de ilustrar de maneira gráfica a semântica das relações temporais. Este tipo de modelo simplifica a especificação das restrições temporais e reduz o tempo de criação do documento. Como deficiência, modelos gráficos têm um poder de expressão geralmente menor que os modelos orientados à linguagem. Para os sistemas de autoria, há portanto um dilema acerca de como balancear a facilidade de uso com poder e flexibilidade.

Por outro lado, a construção de um software extremamente fácil de aprender e utilizar, restringe a ação de autores experimentados ou limita as possibilidades interativas com o usuário final. Fornecer grande flexibilidade e poder ao software, pode torná-lo de difícil manipulação. Uma das soluções tem sido combinar ferramentas de construção simples, similar aos programas de desenhos dirigidos por menus com linguagens Scripting. Scripting é a maneira de associar um *script*, um conjunto de comandos escritos numa forma semelhante a programas de computador, com um elemento interativo numa tela, tal como um botão. Alguns exemplos de linguagens scripting são Apple HyperTalk para HyperCard, Lingo Macromedia para Director e Asymetrix OpenScript para Toolbook.

### 2.6.3 - Linha Temporal

A linha temporal permite o alinhamento das apresentações em um eixo temporal. A Fig. 2.7 apresenta um exemplo de linha temporal. Ela indica que um áudio será apresentado de 4 até 16 unidades de tempo, uma imagem será apresentada de 8 até 16 unidades de tempo, e finalmente um vídeo será apresentado de 16 até 28 unidades de tempo.

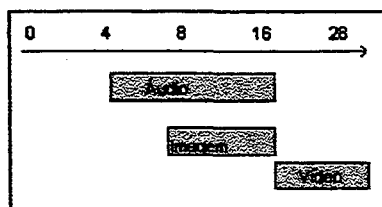


FIGURA 2.7 - Exemplo de Linha Temporal



Esta técnica oferece como principal vantagem uma grande simplicidade de expressão dos esquemas de sincronização. Além disso, o autor tem uma visão muito clara das informações que serão apresentadas e em que momento. Infelizmente ela apresenta várias limitações:

- permite somente a especificação do alinhamento temporal ideal das apresentações, na medida que ela define pontos de partida e fim ideais das apresentações dos componentes do documento. Não é possível especificar métodos de tolerância da sincronização ou ações relativas a perdas de sincronismo;
- requer o conhecimento exato da duração das apresentações. Geralmente, o autor deve manipular os segmentos de informação de maneira manual afim de obter o comportamento temporal desejado. Isto através da edição dos dados pela mudança da velocidade de apresentação;
- como esta abordagem não fornece mecanismos de estruturação, nem a representação das relações lógicas, ela não permite a definição da estrutura conceitual completa de documentos multimídia interativos.

Alguns modelos multimídia utilizam versões estendidas da abordagem linha temporal, permitindo a definição de relações lógicas e temporais entre apresentações (por exemplo, a abordagem proposta por [HIR95]). Em todo caso, os modelos resultantes destas extensões tornam-se geralmente complexos e perdem a vantagem principal do modelo linha temporal de base que é a simplicidade de compreensão do comportamento temporal do documento.

O sistema MAestro [DRA91], QuickTime [APP91], Macromedia Director (Macintosh e Windows), Animation Works Interactive (Windows), MediaBlitz (Windows), Producer (Macintosh e Windows), e a norma HyTime [NEW91] adotam este paradigma.

#### **2.6.4 - Composição Hierárquica**

Na composição hierárquica de documentos multimídia, os componentes de um documento são organizados na forma de uma árvore, como ilustrado na Fig. 2.8. Nesta

árvore, os nós internos representam relações temporais entre as sub-árvores de saída. Estas relações temporais são definidas por operadores de sincronização. Os principais operadores de sincronização são os operadores série e paralelo. Estes operadores definem que um conjunto de ações serão executados em série ou paralelo. Estas ações podem ser atômicas ou compostas [BLA92]: uma ação atômica manipula a apresentação de uma informação, a interação com o usuário ou um atraso; as ações compostas são combinações de operadores de sincronização e ações atômicas.

Uma das vantagens desta abordagem é a possibilidade de agrupar itens em “mini-apresentações” que podem ser manipuladas como um todo [HAR95]. Esta abordagem apresenta também algumas limitações:

- não permite a representação de relações temporais fora das fronteiras hierárquicas. Assim, os links hiper-estruturais não podem ser especificados;
- relações lógicas e temporais não são representadas de uma maneira natural. Ao contrário da abordagem linha temporal, o autor não tem a visão conjunta das informações apresentadas e suas datas de apresentação;
- um conjunto de ações pode ser sincronizado apenas com relação ao início ou fim de um conjunto de ações. Isto significa que, por exemplo a apresentação de legendas em uma certa parte de um vídeo requer o corte da seqüência de imagens em vários componentes consecutivos.

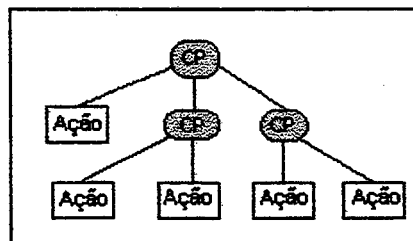


FIGURA 2.8 - Composição Hierárquica

### 2.6.5 - Modelos Baseados em Ícones

A criação de um documento multimídia pela utilização de uma abordagem baseada em ícones é similar a sua programação (por exemplo, utilizando a linguagem

C), mas com a ajuda de uma interface gráfica. Esta interface fornece ícones de alto nível, afim de visualizar e manipular a estrutura do documento.

Um conjunto de ícones é arranjado em um grafo que especifica interações e caminhos de controle de apresentação de um documento multimídia. Em geral, as funcionalidades associadas a cada ícone podem ser modificadas utilizando menus e editores associados.

Esta abordagem é de utilização simples para pequenas aplicações. Mas para aplicações complexas, a compreensão e a manipulação são dificultadas.

A abordagem de composição de documentos multimídia baseada em ícones é adotada, por exemplo, pelos softwares AimTech IconAuthor [AIM93], Eyes M/M [EYE92], Authorware Professional [AUT89], mTropolis e HSC Interactive.

### **2.6.6 - Modelos Baseados em Cartões ou Páginas**

Nessa abordagem, os elementos são organizados em páginas de um livro ou em uma pilha de cartões. Ferramentas de autoria baseadas neste paradigma permitem que o autor ligue as páginas ou cartões formando uma estrutura de páginas ou cartões.

Sistemas de autoria baseados em cartões ou páginas fornecem um paradigma simples para organizar elementos multimídia. Estes tipos de ferramentas de autoria permitem que o autor organize os elementos em seqüência ou agrupamentos lógicos tal como capítulos e páginas de um livro ou cartões em um catálogo de cartões [APP94].

Como exemplo de ferramentas de autoria adotando este paradigma temos: HyperCard (Macintosh), SuperCard (Macintosh), ToolBook (Windows) e VisualBasic (Windows).

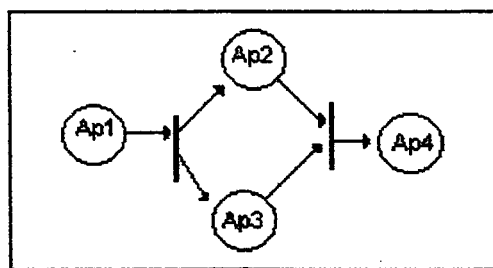
### **2.6.7 - Redes De Petri**

Redes de Petri são ferramentas gráficas e matemáticas aplicáveis na modelagem de muitos sistemas, principalmente aqueles que necessitam representar algum tipo de

concorrência. As redes de Petri apresentam uma estrutura similar a fluxogramas, podendo ser utilizadas para visualização de sistemas de comunicação [MUR89]. Redes de Petri é uma técnica de descrição formal muito utilizada na engenharia de protocolos, automação industrial e muitas outras áreas. Permitindo tanto a especificação formal de um sistema quanto a aplicação de técnicas de análise sobre o sistema afim de verificar certas propriedades [MUR89].

Redes de Petri também se prestam para a modelagem de documentos multimídia, isto devido a sua representação gráfica, a facilidade de modelagem dos esquemas de sincronização, e a possibilidade de análise de propriedades importantes do comportamento lógico e temporal do documento. Existem vários modelos multimídia baseados em redes de Petri. Dentre eles, OCPN [LIT90], TSPN [DIA93], RTSM [YAN96], Trellis [STO90], HTSPN [SÉN95], I-HTSPN [WIL96a], Java I-HTSPN [WIL97], MORENA [BOT95]. Nestes modelos, geralmente um lugar da rede de Petri representa uma apresentação, as transições e arcos definem relações lógicas e temporais.

A Fig. 2.9 ilustra a utilização de redes de Petri para especificar um cenário multimídia. Esta especificação indica que inicialmente AP1 será apresentada; terminada esta apresentação, AP2 e AP3 serão apresentadas em paralelo; tão logo estas duas apresentações terminem, AP4 será apresentada.



**FIGURA 2.9 - Composição Usando Redes de Petri**

Esta abordagem apresenta algumas desvantagens:

- de maneira similar a composição hierárquica, na maior parte dos modelos baseados em redes de Petri, um conjunto de ações podem ser sincronizadas com relação ao início ou fim de um conjunto de ações;

- esta abordagem não contém os conceitos apropriados para a representação das sincronizações condicionais mais complexas nem para os métodos de interação de aplicação;
- em [HUD93], os autores afirmam que uma outra limitação desta abordagem é sua complexidade: este formalismo tende a ser muito potente e de muito baixo nível para sua utilização direta pelo autor do documento.

A grande vantagem do uso de um modelo de autoria baseado em redes de Petri é o caráter formal desta técnica de descrição. A especificação de documentos multimídia utilizando modelos informais ou semi-formais podem ser fontes de ambigüidade e geralmente não permite a utilização de ferramentas de análise avançadas. Ao contrário do anterior, modelos multimídia baseados em métodos formais, como aqueles baseados em redes de Petri, permite a construção de uma semântica precisa e não ambígua de um documento e permitem a utilização de técnicas de análise sofisticadas, como a simulação, validação, verificação do comportamento do documento.

Nem todos os modelos ditos baseados em redes de Petri apresentados na literatura podem ser considerados modelos formais. Assim, alguns destes modelos não permitem a aplicação de técnicas de análise para validar o comportamento do documento multimídia.

### **2.6.8 - Abordagem Baseada em Informação**

Nesta abordagem, também chamada de centrada na informação, o conteúdo é obtido de informações estruturadas e armazenadas em uma base de dados. A estruturação envolve a divisão da informação em nós e identificadores chaves e em seguida a ligação destes conceitos [GIN95]. Como exemplo podemos citar o WWW(World Wide Web). O autor primeiro cria o texto e outras mídias, então ele estrutura estes usando um editor HTML, (*Hypertext Markup Language*). Ligações para outros documentos são embutidas durante o processo de Markup. Usuários podem ver este documento usando um sistema de apresentação (Browser) tal como NSCA Mosaic, Netscape Navigator ou Internet Explorer.

Multimedia Viewer da Microsoft é outro exemplo de um sistema combinado de autoria e apresentação na abordagem centrada na informação.

### 2.6.9 - Modelo de Composição Via Pontos de Referência

Na abordagem de composição via pontos de referência, as apresentações são vistas como seqüências de sub-unidades discretas [BLA92]. A posição de uma sub-unidade (por exemplo, um quadro de um vídeo ou uma amostragem de áudio) em um objeto é chamado de **ponto de referência**. As mídias discretas (por exemplo, texto) apresentam somente dois pontos de referência: início e fim de apresentação. A sincronização é definida por conexões entre pontos de referência como demonstrado na Fig. 2.10.

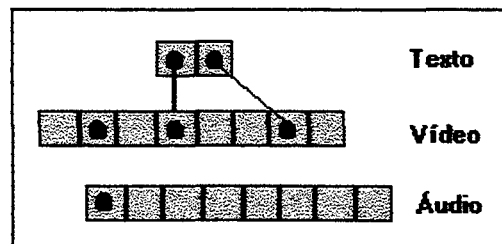


FIGURA 2.10 – Sincronização Via Pontos de Referência

Uma vantagem do conceito de ponto de referência é a facilidade de mudança da velocidade de apresentação [HAI96], porque não existe referência explícita ao tempo. As limitações desta abordagem são as seguintes:

- as sincronizações são definidas por um conjunto de conexões entre pontos de referência. Quando da apresentação de um documento, as irregularidades de apresentação são ignoradas. Não existe a possibilidade de tolerância a este nível;
- a utilização única de uma abordagem de composição via pontos de referência não permite a especificação de retardos, nem a definição de sincronizações condicionais outras que as conexões entre os pontos de referência;
- esta abordagem não é apropriada para a descrição da estrutura conceitual dos documentos multimídia. Isto pois ela não propõe nenhum mecanismo de estruturação e não há distinção entre o componente e seu conteúdo.

### 2.6.10 – Abordagem Baseada em Reflexão Computacional

Nesta subseção, apresentaremos a técnica da Reflexão computacional como uma abordagem com expressiva potencialidade na especificação da sincronização em aplicações multimídia.

A técnica de reflexão computacional vem sendo largamente utilizada com intuito de obter-se um maior grau de modularidade em aplicações, imprimindo aos sistemas computacionais maior capacidade de gerenciamento, maior legibilidade e facilidade de manutenção. Alguns exemplos da aplicação de técnicas reflexivas são encontrados nas áreas de tolerância a faltas [FAB95], concorrência [YO88] e tempo real [HON94] (dentre as quais, destacam-se as aplicações multimídia).

A utilização da técnica reflexiva na programação de sistemas multimídia traz grandes benefícios, pois permite separar os aspectos funcionais dos aspectos não-funcionais, onde as funcionalidades básicas da aplicação são expressas através de objetos-base, e o controle do comportamento desses objetos (restrições, escalonamento, concorrência e sincronização) são expressos através de meta-objetos.

Como exemplo de ferramenta adotando este paradigma citamos o Modelo RTR (Reflexivo Tempo Real) [FUR97], haja visto que será objeto de estudo neste trabalho.

## 2.7 – Conclusões

Neste capítulo, foram apresentadas algumas definições básicas a respeito de sistemas multimídia. Alguns modelos de documentos foram analisados, procurando-se salientar dos mesmos as potencialidades e limitações. Os documentos abordados foram, documentos hipertexto, documentos multimídia e documentos hipermídia. A seguir, foram apresentados os principais requisitos encontrados na descrição de documentos multimídia. Além disso, a questão sincronização e os mecanismos para especificação da sincronização também foram discutidos. Adicionalmente, também foram descritas algumas das abordagens para especificação de aplicações multimídia existentes, sendo que a reflexão computacional foi introduzida neste trabalho como uma promissora

abordagem para especificação de documentos multimídia, podendo contribuir significativamente na solução de muitos dos problemas atualmente encontrados na programação de aplicações multimídia devido a presença de características peculiares não disponíveis nas demais abordagens citadas.

No próximo capítulo serão comentadas várias questões genéricas relativas a reflexão computacional, sendo apresentado uma breve descrição dos modelos e linguagens reflexivas, enfatizando suas características básicas e limitações.



### 3.1 – Introdução

A reflexão computacional define uma nova arquitetura de software. A arquitetura reflexiva é composta por um meta-nível, onde se encontram as estruturas de dados e as ações a serem realizadas sobre o sistema objeto, localizado no nível base. Assim, em uma arquitetura reflexiva, um sistema computacional possui dois componentes interrelacionados: um subsistema objeto, que faz computação sobre um domínio externo ao sistema, e um subsistema reflexivo, que faz computações sobre o sistema objeto.

Em linguagens orientadas a objetos, a reflexão computacional é implementada através de meta-classes, que contém informações sobre a estrutura de classes da aplicação ou de meta-objetos, que contém informações sobre a implementação e a interpretação de objetos da aplicação.

Nos últimos dez anos, e, com maior intensidade recentemente, a reflexão computacional tem atraído a atenção de pesquisadores como forma de extensão de linguagens de programação e pela sua utilidade na implementação de diversos mecanismos, como por exemplo, mecanismos de programação distribuída de forma simplificada e transparente à aplicação. A arquitetura reflexiva, ao estabelecer a separação de domínios, permite a construção de arquiteturas reusáveis de software. Podem ser reutilizados os componentes do programa de nível-base e os componentes do programa de meta-nível [LIS97].

A separação de domínios é o aspecto mais atraente da arquitetura reflexiva, não apenas por incentivar a reutilização, mas principalmente por permitir ao programador da aplicação concentrar-se na solução do problema de programação específico do

domínio da aplicação. A reflexão tem sido adotada para expressar propriedades não funcionais do sistema, como confiabilidade e segurança, de forma independente do domínio da aplicação.

O conceito de reflexão computacional no modelo de objetos foi introduzido em 1987 por Patti Maes [MAE87]. No conceito de Maes, reflexão computacional é a atividade executada por um sistema computacional quando faz computações sobre (e possivelmente afetando) suas próprias computações. Por sistema computacional entende-se um sistema baseado em computador, com o objetivo de realizar computações e fornecer informações em um determinado domínio de aplicação. Neste conceito, a reflexão é definida como uma forma de introspeção, na qual o sistema tenta tirar conclusões sobre as suas próprias computações, as quais podem ser afetadas posteriormente. Em resumo, por reflexão computacional segundo LISBÔA (1997), entende-se: “Reflexão computacional é toda a atividade de um sistema computacional realizada sobre si mesmo, e de forma separada das computações em curso, com o objetivo de resolver seus próprios problemas e obter informações sobre suas computações”.

Dotar um sistema computacional de tal capacidade significa possibilitar a realização de computações flexíveis, capazes de alterar e adaptar dinamicamente componentes do sistema. Uma definição neste sentido é dada por Stell [STE94]: a reflexão computacional é a capacidade de um sistema computacional de interromper o processo de execução (por exemplo, quando ocorre um erro), realizar computações ou fazer deduções no meta-nível e retornar ao nível de execução traduzindo o impacto das decisões, para então retomar o processo de execução.

Nesta última definição é introduzido o conceito de níveis de computação, aproximando-se do significado físico de reflexão, visto que a ocorrência de um evento (por exemplo, um erro) no processo de execução reflete-se em um nível superior (meta-nível) de computação.

Como vemos em [LIS97], a reflexão computacional possui um significado mais diretamente relacionado com o modelo de objetos, embora seja encontrada também no

modelo de programação funcional e programação em lógica. No modelo de objetos, refere-se à obtenção de dados sobre as propriedades de classes e de objetos de uma aplicação, e a possibilidade de modificar esses dados, modificando-se assim as propriedades das classes e dos objetos da aplicação.

A importância da adoção de reflexão computacional no modelo de objetos é assim percebida por:

- **BRIAN FOOTE [FOO93]**: “O casamento de reflexão com programação e técnicas de projetos orientadas a objetos traz promessas de mudanças dramáticas na forma como nós pensamos, organizamos, implementamos e usamos<sup>1</sup> sistemas e linguagens de programação.”
- **MADSEN [MAD95]**: “Por vários anos, a reflexão tem sido um tópico de pesquisa, mas tem tido pouco impacto na principais linguagens orientadas a objetos”. O autor destaca a reflexão entre os tópicos que merecem mais atenção no modelo de objetos por sua possibilidade de contribuição para uma melhoria qualitativa da programação.

Neste capítulo apresentaremos o paradigma da Reflexão computacional, destacando sua arquitetura, seus requisitos, suas características, suas limitações e suas vantagens e desvantagens, com relação à programação convencional. Por último apresentaremos algumas linguagens e modelos com características reflexivas.

## 3.2 - Arquitetura Reflexiva

Quando uma aplicação é executada, ela realiza computações sobre dados de seu próprio domínio com o objetivo de atender os requisitos especificados para aquela particular aplicação; quando a computação é reflexiva, ela realiza computações sobre dados do seu próprio sistema, tendo por objetivo resolver problemas do sistema em si

---

<sup>1</sup> Grifo do autor

ou atuar sobre a aplicação. A auto-representação de um sistema permite a alteração e adaptação de determinados aspectos de sua estrutura e comportamento [LIS97].

Por domínio de um sistema computacional entende-se todo conteúdo de informação expresso por objetos e operações, relacionado com uma determinada área. Para efeito das definições a seguir, designa-se de domínio externo do sistema ou domínio da aplicação o conteúdo de informação relacionado com a aplicação, e por domínio interno do sistema ou auto-domínio o conteúdo de informação sobre o próprio sistema, incluindo a representação interna do domínio da aplicação [LIS97].

Segundo [LIS97], reflexão computacional define uma arquitetura em níveis, denominada arquitetura reflexiva, composta por um meta-nível, onde se encontram as estruturas de dados e as ações a serem realizadas sobre o sistema objeto, localizado no nível base. A Fig. 3.11 esquematiza a arquitetura reflexiva, mostrando que as ações no meta-nível são executadas sobre dados que representam informações sobre o programa de nível base, enquanto que o programa de nível base executa ações sobre os seus próprios dados, com a finalidade de atender aos usuários de seus serviços.

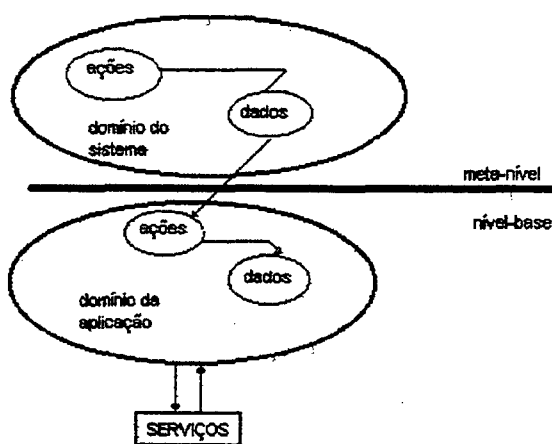


FIGURA 3.11 - Arquitetura Reflexiva

Assim, em uma arquitetura reflexiva, um sistema computacional possui dois componentes interrelacionados: um subsistema objeto, que faz computações sobre um domínio externo ao sistema, e um subsistema reflexivo, que faz computações sobre o sistema objeto. Os dados do nível base são usados no meta-nível para a realização de computações reflexivas que podem interferir nas computações de nível base.

O modo de atuação exige uma conexão entre o componente reflexivo e o sistema objeto. Esta conexão exige que o componente reflexivo (meta-nível) possua informações sobre os componentes do sistema objeto e sobre as situações (por exemplo, erros, eventos) que exigem uma atuação do meta-nível, entre outras. Estas informações variam de acordo com os objetivos da reflexão computacional: o componente reflexivo pode atuar sobre a estrutura ou sobre o comportamento do sistema objeto [LIS97].

No caso de Reflexão estrutural, obtêm-se informações dos elementos envolvidos, as quais poderão ser utilizados em procedimentos de controle, ou outro tipo de computação, como por exemplo, informações sobre classes ascendentes, descendentes, alteração e atuação sobre classes. Em Reflexão comportamental de objetos, também conhecida como Reflexão computacional, a função básica do meta-objeto é explicitar como o objeto reage diante de uma mensagem, possibilitando a intervenção no estado da computação. Como mostrado no exemplo apresentado em [LIS97], da atuação sobre o comportamento do sistema objeto descrito abaixo:

Supondo que um objeto *x* receba uma mensagem *x.ExibirValor*, endereçada ao método *ExibirValor*, com o seguinte comportamento: um valor *y* é exibido no vídeo da estação de trabalho onde o objeto *x* está sendo executado. O seu meta-objeto pode alterar esse comportamento, fazendo que o valor de *y* seja exibido em diversas estações de trabalho, além daquela onde o objeto *x* encontra-se em execução.

No modelo de objetos, a computação reflexiva pode ser feita sobre classes ou instâncias de classes. No primeiro caso, o meta-nível é composto por meta-classes, as quais contém informações sobre os aspectos estruturais de componentes do nível base tais como descrição de variáveis e de métodos que irão compor todas as suas instâncias.

Esta estrutura pode ser alterada e, em decorrência disso, todas as instâncias também serão alteradas. No segundo caso, o meta-nível é composto por meta-objetos, que contém informações sobre os aspectos de comportamento dos objetos de nível base, como por exemplo, como um determinado objeto trata uma mensagem [LIS97].

### 3.3 – Reflexão Computacional e Tempo Real

Embora pouco explorada no domínio tempo real, a abordagem de reflexão computacional é vista como uma abordagem promissora no que se refere a estruturação de sistemas tempo real complexos [STA96], apta a contribuir nas questões de flexibilidade e gerenciamento da complexidade dos sistemas tempo real atuais e futuros.

O uso da reflexão no domínio tempo real, permite:

- adicionar ou modificar construções temporais (via meta-objetos) específicos de um domínio (ou de uma aplicação);
- definir um comportamento alternativo para o caso de exceções temporais (não satisfação de *deadlines*, por exemplo);
- substituir/alterar algoritmo de escalonamento, adequando-o a aplicação em questão;
- mudar o comportamento do programa, em função de informações obtidas em tempo de execução, como por exemplo disponibilidade de tempo, carga do sistema e atributos de QoS (*Quality of Service*);
- definir e/ou ajustar o tempo de execução das atividades do sistema, em função da experiência adquirida com a evolução deste;
- possibilitar a realização de análise de escalonabilidade dinâmica, no meta-nível da aplicação;
- incrementar a portabilidade dos sistemas favorecendo sua utilização em ambientes abertos.

Entretanto, apesar do potencial da abordagem reflexiva, seu uso para tempo real é questionável com relação a dois importantes aspectos: **desempenho** e **previsibilidade**. Com relação ao desempenho, admite-se um custo adicional ao processamento reflexivo.

Com relação a previsibilidade, o problema não está na reflexão em si, mas no fato de que ela habilita a produção de sistemas tempo real muito mais flexíveis [MIT97], para as quais a análise de pior caso ainda é uma questão de pesquisa em aberto; contudo embora a questão de previsibilidade (especialmente na presença de reflexão estrutural) possa dificultar (ou mesmo impedir) o uso de reflexão na programação de STR *hard*, sua utilização na programação de STR *soft* (onde as exigências de previsibilidade são menos severas) é perfeitamente viável e promissora.

## 3.4 – Programas e Sistemas Reflexivos

### 3.4.1 - Introdução

A arquitetura reflexiva pode ser adotada para estruturar sistemas ou programas.

A arquitetura reflexiva estabelece conexões entre níveis de execução ou programas distintos, como por exemplo, entre um programa de depuração e um programa de aplicação. Exemplos de computações reflexiva de sistemas são [LIS97]: manter estatísticas sobre desempenho de um programa ou sistema, coletar informações para fins de depuração, realizar monitoração da execução de um sistema, possibilitar a reconfiguração de programas através da carga dinâmica de componentes, etc.

### 3.4.2 – Sistemas Reflexivos

Sistemas reflexivos são intrinsecamente relacionados através de seu ambiente operacional, visto que necessitam informações ‘privilegiadas’ a respeito do componente da reflexão. Em ambientes integrados, as informações estão, de alguma forma, presentes em algum componente do ambiente, bastando torná-las disponíveis [LIS97].

Quando o objetivo é utilizar a reflexão computacional para desenvolver ferramentas como depuradores, que necessitam inspecionar o código de um programa sem compromisso de eficiência, a utilização de linguagens de programação do tipo ‘*script*’ se apresenta como uma solução adequada. Linguagens ‘*script*’ são linguagens

de uso específico em um ambiente, normalmente projetadas com base em uma linguagem de programação de uso geral, mas com expressividade limitada, visto que se destinam primordialmente ao desenvolvimento de programas para uso apenas em seu ambiente operacional. São linguagens interpretadas, com forte vinculação a outros componentes do ambiente, como o sistema operacional e interfaces. Esta vinculação baseia-se em informações internas a respeito do sistema, independente de aplicações do usuário, e, portanto caracterizando a qualidade reflexiva destas linguagens [LIS97].

Quando o objetivo é utilizar a reflexão computacional para implementar o mesmo conceito através de diferentes estratégias (por exemplo, diversos protocolos de comunicação), visando flexibilidade e adaptabilidade de programas em execução a um ambiente, a reflexão computacional pode ser adotada a nível de sistema operacional. Os sistemas operacionais reflexivos propõem soluções baseadas no conceito de associação dinâmica de objetos/meta-objetos, sendo estes últimos responsáveis pela implementação de estratégias; mudando a associação entre objetos/meta-objetos, obtém-se a variedade de estratégias. Esta é a proposta do sistema operacional Apertos[YOK92], para oferecer serviços de distribuição adequados às necessidades de sistemas categorizados como ‘*mobile*’.

### 3.4.3 – Programas Reflexivos

Para incorporar o conceito de reflexão computacional ao processo de criação de programas, é necessário adotar um novo modelo de abstração: a separação de domínios. Um dos domínios compreende as funcionalidades da aplicação e o outro domínio é a própria aplicação.

Um *programa reflexivo* é estruturado na forma de dois programas: um programa de nível base e um programa de meta-nível. KICZALES [KIC92] assim explica a diferença entre eles: “O programa de nível base expressa o comportamento desejado pelo cliente, em termos das funcionalidades fornecidas pelo ambiente de suporte. O programa de meta-nível pode adequar aspectos particulares de implementação do ambiente de suporte de forma que melhor atenda às necessidades do programa de nível base.”



Para executar uma computação não reflexiva, o programador precisa somente conhecer a interface do objeto, isto é, quais são os métodos e as mensagens que o objeto responde e qual o serviço que cada um deles fornece ao objeto cliente. E mais ainda: sob o ponto de vista de segurança das aplicações, os objetos são intocáveis pelos seus clientes; somente o próprio objeto pode alterar o seu estado, como resultado de suas computações internas [LIS97].

Para ser uma computação reflexiva, o método a ser executado realiza algum serviço para o próprio sistema, como por exemplo, atuar como intermediário entre um objeto cliente e seu servidor para assegurar que o serviço solicitado seja fornecido[LIS97].

Para realização dessas computações, é necessário ter acesso a informações que uma linguagem de programação geralmente não torna acessível ao programador comum. Os argumentos de uma computação reflexiva são informações internas do sistema, como por exemplo: quais são as interfaces de um objeto, qual é a classe ancestral do objeto, quais são as instâncias de uma determinada classe, que mensagens são trocadas entre objetos da aplicação e quais são os argumentos. Lembrando que um objeto encapsula seus dados e suas operações, sendo visível apenas pela sua interface externa, para inspecionar a sua implementação é necessário quebrar a barreira do encapsulamento feito pelo compilador. Portanto, como primeiro princípio de programação reflexiva, pode-se estabelecer que a reflexão computacional deve fornecer uma interface para o compilador da linguagem de programação utilizada na aplicação [LIS97].

Na seção seguinte descreveremos algumas das principais propostas de linguagens e modelos reflexivos existentes, destacando características principais, aplicações, contribuições e limitações.

## **3.5 – Linguagens e Modelos Reflexivos - Trabalhos Existentes**

### **3.5.1 – Linguagens Reflexivas**

As características reflexivas são implementadas de diferentes modos, dependendo da linguagem de nível base e seu enfoque particular do modelo de objetos. Os mecanismos de reflexão, da mesma forma que outros mecanismos de linguagens de programação, exigem uma associação (*'binding'*) entre componentes de um programa.

A idéia básica da programação em meta-nível é tornar disponível partes da implementação da linguagem, para permitir a sua reprogramação. Para suportar o conceito de reflexão computacional, uma linguagem de programação necessita de mecanismos que: (a) permitam a conexão entre o nível base e o meta-nível; (b) tornem disponíveis, no meta-nível, informações sobre os componentes do nível base.

Através da reificação, é possível ver componentes do programa e o estado da computação como dados internos, que podem ser modificados de forma a alterar dinamicamente o programa de nível base. Portanto, o programa de meta-nível pode ser visto como uma interface de alto-nível (meta-interface) ao programa de nível base, habilitando o usuário a fazer adaptações do programa para atender necessidades particulares.

Podemos destacar as seguintes linguagens de programação baseadas em reflexão computacional:

### 3.5.1.1 – Meta-Classes de Smalltalk

A linguagem SmallTalk [GOL83] possui, por concepção, características reflexivas, sendo particularmente adequada para a construção de aplicações baseadas no conceito de reflexão computacional. Sendo uma linguagem na qual todos os componentes são objetos, SmallTalk oferece uma organização típica do modelo: os objetos encapsulam seu próprio estado e a meta-classe correspondente descreve as suas interfaces relacionadas com a implementação de todas as classes.

Entre as facilidades encontradas para a utilização de reflexão computacional, destacam-se [GOL83]:

- todas as entidades da linguagem são consideradas como objetos, inclusive a entidade que contém descrições de objetos (classes). Sendo uma classe considerada um objeto, é possível fazer operações sobre a classe, inclusive alterando a estrutura dos objetos da classe. Assim, computação reflexiva estrutural de classes atua sobre as instâncias da classe;
- todas as classes de SmallTalk são instâncias de uma classe **Object**, que é a raiz da hierarquia de classes do sistema SmallTalk. A partir da classe **Object** são derivadas as demais classes e cada classe tem a sua representação descrita em uma meta-classe. Cada classe é uma instância de sua própria meta-classe. Por exemplo, uma classe denominada **Point** é uma instância da meta-classe **Point class**; isto significa que todos os objetos instanciados a partir da classe **Point** são descritos pela meta-classe **Point class**;
- todas as meta-classes são instâncias da classe **MetaClass**. Portanto, a meta-classe **PointClass** é uma subclasse de **Object class**, e também uma instância de **MetaClass**. As facilidades reflexivas são implementadas em meta-classes especiais da hierarquia, de forma distribuída.

O ambiente SmallTalk fornece a representação dos principais componentes das classes, como informações a respeito de hierarquia e representação das variáveis da classe, que podem ser acessados e modificados por outros objetos em tempo de execução. A liberdade de, por exemplo, alterar a hierarquia de classes ou a representação de uma variável, torna este modelo bastante dinâmico e extremamente frágil sob o aspecto de robustez de programas.

### 3.5.1.2 – Meta-Objetos de Open C++

A linguagem C++ [STR91] não oferece, por concepção, facilidades reflexivas. Para possibilitar acesso, em tempo de execução, às informações a respeito das classes e objetos do programa é necessário modificar a implementação da linguagem ou fazer extensões baseadas em pré-processadores.

O pré-processador OpenC++ - Versão 1.2 [CHI93a] adota um modelo de reflexão computacional no qual chamadas de métodos e acessos a variáveis de instância são

tornados disponíveis ao programador, em tempo de execução. O protocolo possui as seguintes características:

- a) **Meta-objeto:** objetos selecionados do programa podem ser instanciados como objetos reflexivos, associados a um meta-objeto; outros objetos da mesma classe podem ser instanciados como objetos não reflexivos;
- b) **Relação:** a associação é única entre objetos e meta-objetos, isto é, um meta-objeto não pode ser compartilhado por mais de um objeto;
- c) **Controle:** um meta-objeto pode controlar mensagens dirigidas a seus referentes (chamadas de funções-membro) e acesso à variáveis de instância (variáveis-membro);
- d) **Todas as classes** usadas para instanciar os meta-objetos descendem da classe `MetaObj`.

Diretivas especiais são adicionadas ao pré-processador para serem utilizadas na criação de objetos, métodos e variáveis reflexivas. Para criar um objeto reflexivo deve-se declarar sua classe da seguinte forma:

```
// MOP reflect class X: M;
```

A diretiva `//MOP` indica que uma instância da classe `X` pode ser refletida e que seu meta-objeto será uma instância da classe `M`. Quando o compilador lê esta linha de código, ele cria automaticamente, uma classe denominada `refl_X`. A classe `refl_X` é uma subclasse de `X`.

Se o usuário deseja criar um objeto reflexivo, deve criar uma instância da classe `refl_X`; caso deseje criar um objeto comum, deve instanciar a classe `X`.

A computação no meta nível é realizada por redefinição dos métodos adquiridos da classe `MetaObj` e pelos métodos particulares (definidos ou herdados) da classe do meta-objeto em questão. Os métodos oferecidos pela classe `MetaObj` se dividem em três categorias:

- i – **Métodos que implementam chamadas e acessos à variáveis:** são executados quando um método reflexivo é chamado ou uma variável reflexiva é acessada.

**Exemplo:** O método `Meta_MethodCall()` é executado quando um método reflexivo é chamado. Sua redefinição na classe do meta-objeto é usada para determinar todas as computações a serem realizadas no meta-nível por ocasião da interceptação da mensagem.

ii- Métodos usados pelo meta-objeto para executar operações sobre o objeto de nível base: incluem métodos usados para executar um método de nível base, consultar/atribuir valor de variável reflexiva.

**Exemplo:** O método `Meta_HandleMethodCall()` é usado para executar um método de nível base, cuja chamada foi interceptada.

iii- Métodos diversos usados para operações internas de reflexão, como construtores/destruidores especiais e informações sobre nomes de classes, métodos e identificadores de variáveis de objetos reflexivos.

**Exemplo:** O método `Meta_GetMethodName()` fornece o nome do método reflexivo cuja identificação é fornecida como argumento.

Em resumo, o protocolo de OpenC++ - Versão 1.2 oferece um ambiente de programação reflexiva que possibilita que certos aspectos de um programa C++ sejam redefinidos no meta-nível, através de interfaces bem definidas. Sendo implementado através de um pré-processador, suas características reflexivas são estabelecidas de forma estática.

Na sua versão mais recente, o protocolo de OpenC++- Versão 2.0 [CHI96] é compilado, originando meta-classes que são tratadas pelo programa como objetos, de forma similar à SmallTalk. Os principais conceitos envolvidos são:

- **class object:** representação de uma classe no meta-nível;
- **metaclass:** uma classe cuja instâncias são meta-objetos;
- **metaclass Class:** a meta-classe padrão.

### 3.5.1.3 – Protocolo MetaJava

Este protocolo, proposto por Golm [GOM97], implementa reflexão estrutural e comportamental em Java através da extensão da máquina virtual Java (JVM). O modelo abstrato da Máquina Virtual Java (JVM) determina que todas as classes e interfaces são objetos do tipo `class`. Objetos do tipo `Class` são automaticamente criados quando novas classes são carregadas para serem executadas, e contém informações sobre o nome da classe, a super-classe, a lista de interfaces implementadas e o carregador de classes utilizado. Estes objetos do tipo `Class` são apenas descritores de classes, não sendo permitido alterar a estrutura de classes [SUN97].

MetaJava permite a associação de meta-objetos a objetos, classes e referências, e utiliza eventos síncronos para realizar a transferência da computação do programa de nível base para o programa de meta-nível. Se um meta-objeto é associado a um objeto, todos os eventos originados no objeto referente são transmitidos ao meta-objeto correspondente; quando associado a uma classe, todos os eventos originados nas instâncias da classe referente são transmitidos ao meta-objeto; e, quando associado a uma referência, somente os eventos gerados por operações que usam essa referência são transmitidos ao meta-objeto correspondente.

Múltiplos meta-objetos podem ser associados a um componente de nível base, formando uma hierarquia (torre) de meta-objetos, na qual a ordem é importante: o meta-objeto de mais recente associação é executado em primeiro lugar.

A extensão MetaJava foi implementada através da modificação da máquina virtual Java, em dois níveis: o nível inferior fornece a interface de meta-nível para a máquina virtual Java e o nível superior implementa mecanismos de transferência de controle para o meta-nível.

A interface de meta-nível (MLI) fornece métodos que fazem a reificação da máquina virtual Java, disponibilizando estes dados ao programa de meta-nível. Estes métodos são agrupados nas seguintes categorias [GOM97b]:

- **acesso a dados de instância:** fornecem acesso aos nomes, tipos e estrutura das variáveis de um objeto;
- **suporte para modificação de código:** faz a reificação do código de um método, fornecendo a seqüência de bytecodes correspondente;
- **associação entre o nível base e o meta-nível:** associa um meta-objeto a um objeto de nível base e fornece informação sobre o meta-objeto associado a um objeto;
- **modificação de constantes:** permite alterar as constantes de uma classe;
- **suporte para código nativo:** permite instalar e executar código nativo, diretamente pelo processador;
- **execução:** permite executar métodos arbitrários;
- **criação de instâncias:** cria novas instâncias de uma dada classe.

MetaJava é bastante flexível, permitindo associações de metaobjetos a componentes de nível base representados por classes, objetos e referências, e fornecendo ao programa de meta-nível informações que devem ser cuidadosamente utilizadas. A reflexão estrutural viola o encapsulamento de classes e a torre de meta-objetos pode provocar conflitos semânticos.

#### 3.5.1.4 – Sistema MetaXa

O sistema MetaXa pode ser considerado uma evolução do sistema MetaJava. O sistema MetaXa foi projetado para ser um sistema reflexivo, que além de permitir reflexão estrutural, também fornecesse de algum modo reflexão comportamental.

Na maioria dos sistemas reflexivos há três tipos de códigos:

- **código de nível-base,** que resolve o problema de aplicação;
- **código de nível-base,** que adapta o código de nível-base para diferentes ambientes;
- **código de configuração,** que conecta base e meta.

O objetivo de MetaXa é separar estas três categorias de código, conservando tipos seguros e eficientes.

O propósito para reflexão em MetaXa não é baseado em linguagem, mas baseado em sistemas. Então não houve uma extensão de linguagem porém foi construída uma própria extensão da Máquina Virtual Java (JVM).

Eventos são os métodos de MetaXa para controle de transferência de um nível-base para o nível-meta. Há eventos para vários mecanismos da JVM, como invocações de métodos, acessos à variáveis, operações de monitor, criação de objeto, ou carga de classe. O manipulador de eventos do meta objeto pega um objeto evento que contém a informação necessária para executar a operação requisitada. Eventos ocorrem sincronamente, isto significa, que a computação base é suspensa até que os eventos sejam tratados.

Um metaobjeto tem acesso a máquina virtual por uma interface-de-meta-nível (MLI). A MLI contém métodos para reflexão estrutural, além de alguns métodos auxiliares para, por exemplo, criar objetos cujo estado (suas variáveis de instância) é completamente administrado pelo metaobjeto. Nenhuma memória é reservada para tal objeto pela JVM. Este mecanismo de criação é útil para *array* esparsos ou para objetos que servem como *stubs* ou *proxies*.

As vantagens de MetaXa podem ser resumidas como segue:

- Há uma relação *n-para-n* entre classes e metaobjetos de instâncias de classe. Vários sistemas reflexivos conectam a classe do objeto de nível-base e a classe do metaobjeto, em uma fase inicial de desenvolvimento. Em tais sistemas não é possível usar metaobjetos diferentes para objetos de nível-base da mesma classe. O sistema MetaXa permite modificação da relação de metaobjeto em tempo de execução;
- Metaobjetos tem controle completo através de argumentos e passagem de valor de retorno. Contanto que satisfaça as regras de tipo estático, quaisquer dados podem ser passados como parâmetro ou valor de retorno.

Por outro lado, MetaXa tem algumas desvantagens. A maior desvantagem é a perda do suporte da linguagem para programação reflexiva. Durante a configuração do metaobjeto e comunicação pelo MLI, o programador têm que se referir a certas entidades do modelo de programação. Estas poderiam ser entidades primeira de classe,



como objetos, mas também outras entidades (como métodos ou *slots*) que só podem ser referenciadas indiretamente usando *strings*. Se o compilador pudesse conferir estas referências, vários erros de programação poderiam ser descobertos em uma fase inicial do desenvolvimento.

### 3.5.1.5 – Protocolo Java Reflexivo

Numa linguagem de programação completamente reflexiva todas as propriedades de uma linguagem são acessadas e mudadas pelos programadores (em um modo controlado). Porém, tornar Java completamente reflexivo poderia ser uma tarefa muito complexa, e não pode ser feito sem apoio de sua máquina virtual e compilador.

Para manter-se simples e seguro, o protocolo Java Reflexivo fornece propriedades limitadas de reflexão, as quais são simples mas poderosas o bastante para habilitar substituição de um sistema alternativo a nível componente. O protocolo de reflexão Java Reflexivo descrito por WU [WU97] somente oferece reflexão de métodos. Quando um método é chamado, o seu meta-objeto correspondente intercepta esta chamada e passa a fazer computações no meta-nível. Deste modo, os programadores podem fazer invocações de métodos se comportarem conforme suas necessidades particulares pela implementação de metaobjetos.

De forma similar a Open C++, Java Reflexivo utiliza um pré-processador para implementar a reflexão. O pré-processador cria uma classe reflexiva como uma subclasse da classe da aplicação. A classe reflexiva possui uma instância de uma classe *metaobject*, que implementa a interceptação de chamadas aos métodos do objeto referente. O objeto referente é instanciado a partir da classe reflexiva.

Java Reflexivo possibilita aos programadores mudar o comportamento dos métodos invocados, pela especificação do que deve ser feito antes ou depois da execução do método normal.

A questão chave de Java Reflexivo é encontrar um modo de interceptar uma invocação de métodos, sem exigir qualquer mudança na linguagem Java, em seu compilador, ou em sua máquina virtual. A questão está sendo resolvida pelo uso de classes.

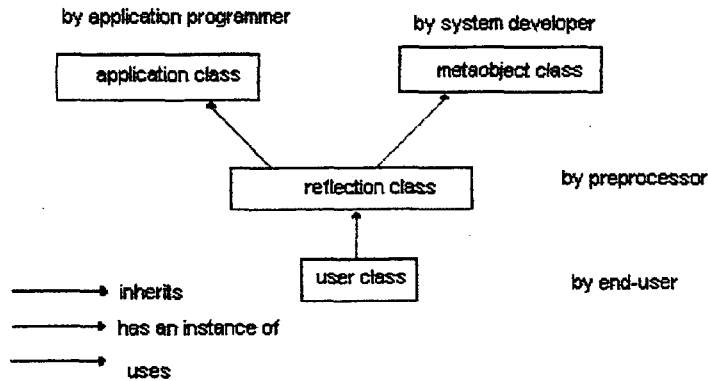


Figura 3.12: Ligação de Objetos

A idéia geral é implementar uma classe reflexiva como uma subclasse de uma classe da aplicação. Cada operação de uma classe de aplicação é sobrescrita na classe reflexiva de modo que uma invocação seja redirecionada para um metaobjeto, como descrito na Fig. 3.12. Uma classe reflexiva tem metaobjeto variáveis que são referência para uma instância de um metaobjeto de classe.

Deste modo, se programadores criam objetos ao invés de classes reflexivas instanciadas da classe de aplicação original, a invocação de um método objeto irá ser interceptada e negociar com o metaobjeto.

Java Reflexivo tem fornecido a flexibilidade exigida para implementar aplicações flexíveis e abertas. Faltam porém algumas questões como por exemplo, desenvolver conceitos que permitam usar flexibilidade de modo confortável, porém controlado. Por exemplo, se um objeto da aplicação ligado a um metaobjeto que forneça algum mecanismo de controle de concorrência, tiver seu metaobjeto mudado dinamicamente, e o novo metaobjeto não fornecer um controle de concorrência correto, o sistema entrará em um estado inconsistente. Dessa forma, alguns mecanismos podem ser necessários para evitar mau uso da flexibilidade permitida por Java Reflexivo.

### 3.5.1.6 – OpenJava

OpenJava é uma extensão da linguagem Java. As características estendidas de OpenJava são especificadas por um programa de meta-nível em tempo de compilação. Por distinção, programas escritos em OpenJava são chamados de programas de nível-base. Se nenhum programa de meta-nível é determinado, OpenJava é idêntico a Java.

O programa de meta-nível estende OpenJava pela interface chamada OpenJava MOP (Protocolo de Metaobjeto). O Compilador de OpenJava consiste em três fases: preprocessador, tradutor de fonte-para-fonte de OpenJava para Java, e o *back-end* compilador de Java. O MOP de OpenJava é uma interface para controlar o tradutor na segunda fase. Permite especificar como uma característica estendida de OpenJava é traduzida em código de Java.

Uma característica estendida de OpenJava é fornecida como um software

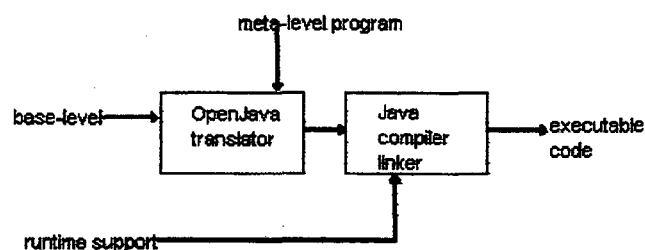


FIGURA 3.13 - Data Flow of OpenJava Compiler

adicional para o compilador. O software adicional não só consiste de um programa de meta-nível mas de um código suporte para tempo de execução. O código suporte para tempo de execução fornece classes usadas pelo programa de nível-base traduzido em Java. O programa de nível-base em OpenJava é traduzido primeiro em Java pelo programa de nível-meta e é dinamicamente linkado com o código suporte em tempo de execução.

MOP (Protocolo de Metaobjeto) Programado em OpenJava segue os três passos:

- decide com o que o programa de nível-base deveria se parecer;

- Compreende o que deveria ser traduzido e que código suporte é necessário em tempo de execução;
- escreve um programa de meta-nível para executar a tradução e também escreve o código suporte em tempo de execução.

Uma desvantagem de OpenJava é que somente permite aos meta-programadores estender a sintaxe da linguagem de modo limitado. A parte mais fácil da extensão da sintaxe de OpenJava é na adição de novas modificações para declaração de classe ou declarações de membros.

### 3.5.2 – Modelos Reflexivos

Alguns modelos baseados em reflexão computacional, têm sido propostos a fim de direcionar a implementação de ferramentas para o tratamento de aplicações baseadas em objetos e em geral com necessidade de satisfação de requisitos tempo real. Um modelo de programação para tempo real é uma abstração capaz de representar tanto o sistema computacional de controle a ser produzido, quanto as entidades do mundo real (o ambiente da aplicação) por ele controladas. Algumas características como, modularização, reusabilidade, separação dos aspectos funcionais da aplicação dos aspectos de controle presentes na reflexão computacional vem incentivando a construção de modelos que seguem esse paradigma [LIS97].

Apresentaremos uma descrição dos principais modelos de programação tempo real reflexivos baseados em objetos, destacando suas características básicas, vantagens e limitações.

#### 3.5.2.1 – Real-Time Meta-Object Protocol (RT-MOP)

Em [MIT97] é apresentada uma proposta inicial de um MOP (*Meta-Object Protocol*) de tempo real baseado em grupos de escalonamento, o qual tem como objetivo disciplinar e controlar mudanças em tempo de execução e servir como um mecanismo de estruturação de sistemas. Neste sentido o RT-MOP proposto pode ser

visto como sendo um modelo de programação tempo real baseado em reflexão computacional.

Segundo o modelo proposto, um sistema tempo real deve ser estruturado na forma de objetos de aplicação (objetos-base) e meta objetos, sendo que cada objeto da aplicação possui um meta-objeto associado, o qual é responsável pelo controle e pela modificação da estrutura e do comportamento do objeto de aplicação correspondente. Conjuntos de objetos de aplicação (e seus respectivos meta-objetos) são estruturados em grupos de escalonamento, sendo que cada grupo possui um meta-objeto “*scheduler*” responsável pelo escalonamento das tarefas de seus objetos constituintes segundo uma política de escalonamento própria; além disso, cada grupo possui também um gerente responsável pelo gerenciamento da entrada e saída de elementos no grupo (*membership*) em tempo de execução, sendo que o acesso a um grupo é controlado para manter garantias de escalonabilidade em tempo de execução. Adicionalmente, os meta-grupos de escalonamento também são reflexivos e podem ter seu comportamento alterado (mudança da política de escalonamento, por exemplo) através da invocação de operações do meta-meta-grupo.

No modelo proposto, nem todos os objetos da aplicação necessitam ser reflexivos, mas em qualquer caso as informações temporais necessárias ao escalonamento devem estar presentes na interface destes objetos; entretanto, na referência disponível [MIT97], nada é adiantado sobre a forma pela qual tais instruções são associadas aos objetos (ou a seus métodos...), e de que forma objetos não reflexivos serão considerados no escalonamento realizado pelo escalonador do grupo.

O modelo de concorrência proposto, adota o conceito de objetos ativos formados pelo encapsulamento de tarefas (*threads*) dentro dos objetos; tal modelo é fortemente influenciado pelo modelo de concorrência usado em TAO [MIT96]. Cada objeto ativo pode possuir várias *threads*, as quais são criadas quando o objeto é criado e são executadas de acordo com o escalonador do grupo de escalonamento ao qual o objeto pertence. Este modelo suporta tanto concorrência externa quanto concorrência interna, as quais são controladas através do uso de métodos sincronizados, conforme proposto em [MIT96].

A questão de como os grupos serão distribuídos em uma rede ainda é uma questão em aberto; entretanto os autores consideram o grupo como sendo a unidade adequada para distribuição, e propõem que grupos de escalonamento dentro de um sistema possam residir em diferentes nodos da rede, não permitindo que os grupos possuam distribuição interna (visando tornar o escalonamento tempo real mais fácil).

Embora a proposta seja preliminar e muitos aspectos ainda estão por ser definidos (na referência considerada [MIT97], apenas hipóteses são consideradas), o uso de grupos de escalonamento é sem dúvida uma boa alternativa para prover a flexibilidade, de forma controlada, que muitos sistemas tempo real atuais necessitam; além disso por ser reflexivo, o modelo proposto herda as vantagens do uso do paradigma de reflexão computacional para aplicações tempo real. Contudo, a realização prática deste modelo depende da definição de uma estrutura reflexiva concreta no âmbito de alguma linguagem de programação.

### 3.5.2.2 – Modelo “R<sup>2</sup>”

Proposto por Honda e Tokoro [HON94], “R<sup>2</sup>” (*Real-time Reflective*) é um modelo de computação concorrente baseado em objetos e reflexão computacional, segundo o qual um sistema computacional é estruturado como uma coleção de objetos básicos controlados por meta-objetos. Tais objetos comunicam-se através de passagem de mensagem (como no modelo convencional) e executam concorrentemente; entretanto, apenas um método (um fragmento de um *script*) pode executar por vez, visto que os objetos são “*single-threaded*”.

O modelo proposto é voltado para sistemas de tempo real *soft*, visto que não existe garantia com relação a satisfação de restrições temporais; o que o modelo oferece é a garantia de que a violação de qualquer restrição temporal será detectado e que as ações alternativas especificadas serão realizadas.

Além dos objetos básicos, o modelo R<sup>2</sup> introduz quatro tipos de meta-objetos: meta-objetos padrão, meta-objetos de tempo real, meta-objeto alarm-clock e meta-objeto escalonador. A interação entre os diversos tipos de objetos e meta-objetos que

compõe o modelo  $R^2$  é, esquematicamente, mostrada na Fig. 3.14 [HON94]. As funcionalidades destes objetos podem ser, resumidamente, assim descritas:

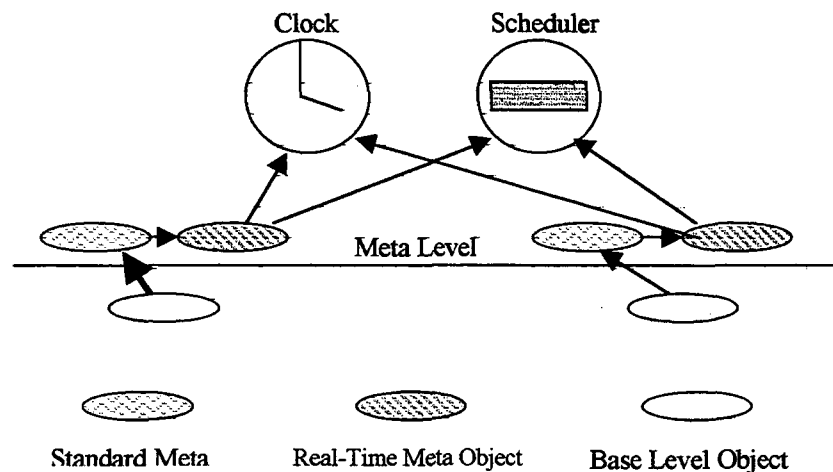


FIGURA 3.14 – Estrutura Geral do Modelo  $R^2$

- **Objetos Básicos** – Seu *script* (correspondendo a uma coleção de métodos) implementa as funcionalidades da aplicação.
- **Meta-Objeto Padrão** – Responsável pelo controle de concorrência, pela manipulação de meta-mensagens relativas ao seu objeto básico e pelo suporte ao meta-objeto de tempo real correspondente.
- **Meta-Objeto Tempo Real** – Processa as especificações temporais das tarefas interagindo com os meta-objetos alarm-clock e escalonador, visando prover as propriedades de melhor esforço e menor prejuízo. Também é responsável pela detecção de eventuais violações das restrições temporais especificadas e pela realização das ações alternativas apropriadas.
- **Meta-Objeto Alarm-Clock** – É uma abstração da função *clock* provida pelo sistema operacional/hardware subjacente, cujas funções básicas são fornecer o valor corrente do *clock* do sistema e notificar a passagem de um determinado intervalo de tempo.
- **Meta-Objeto Escalonador** – Determina a ordem de execução das tarefas (fragmentos de código do *script*), visando satisfazer as restrições temporais especificadas e melhorar a utilização da CPU. Na arquitetura proposta existe flexibilidade para escolha de um algoritmo de escalonamento que seja mais adequado à aplicação em questão.

As restrições temporais básicas consideradas no modelo são o tempo de *start* e o *deadline* de uma tarefa (uma seção de um *script*), as quais são associadas a segmentos de códigos dos objetos-base e reportadas para seu meta-objeto de tempo real que ficará responsável pelo processamento das restrições temporais especificadas. Além disso, o modelo proposto habilita a introdução de novas construções temporais, visando melhor se enquadrar às especificações temporais relativas a uma determinada aplicação.

O modelo  $R^2$  foi implementado como uma extensão da linguagem ABCL/R2, a qual é uma linguagem que incorpora os conceitos de concorrência de ABCL/1 [YON86] e o conceito de reflexão individual de ABCL/R. Nesta implementação, os objetos e meta-objetos que compõem a arquitetura  $R^2$  foram implementados com objetos ABCL/R2.

O modelo  $R^2$  introduz uma nova e interessante forma de desenvolvimento de sistemas tempo real usando a idéia de reflexão computacional baseada em meta-objetos. As principais vantagens decorrentes desta filosofia residem na flexibilidade e na expressividade que podem ser obtidas tanto pela definição de novas construções temporais quanto pela utilização de diferentes algoritmos de escalonamento (introduzidos via meta-objetos), sem afetar a programação das funcionalidades da aplicação e do suporte subjacente.

Apesar de ser claramente endereçado para sistemas tempo real *soft*, o modelo proposto empenha-se no sentido de realizar as propriedades de melhor esforço e menor prejuízo, buscando maximizar a taxa de satisfação das restrições temporais através do oferecimento de suporte para a definição de construções temporais e de algoritmos de escalonamento mais adequados a uma determinada classe de aplicações.

Um aspecto discutível do modelo  $R^2$  é o fato de que as restrições temporais são associadas a segmentos de código e não a um “*script*”; desta forma, as restrições temporais especificadas não influenciarão o escalonamento dos “*script's*”, mas somente o escalonamento de segmentos de código dos mesmos, reduzindo a efetividade do escalonamento realizado a nível de aplicação.



Adicionalmente, o esquema utilizado na expressão de restrições temporais dificulta a definição e o uso de restrições temporais do tipo *time-trigger* (como periodicidade, por exemplo), além de ferir a uniformidade do modelo de objetos e dificultar o reuso e a manutenção dos objetos-base.

Outro aspecto discutível no modelo  $R^2$ , é a não consideração do controle de sincronização condicional a nível de meta-objetos, uma vez que este aspecto pode afetar o escalonamento das tarefas do sistema. além disso, o modelo  $R^2$  é fortemente influenciado pelo paradigma de programação de ABCL/R (programação por continuação), de forma que a implementação de seu comportamento em outras linguagens, implicaria na mudança de sua filosofia e estruturação básicas.

### 3.5.2.3 – Modelo Reflexivo Tempo Real - RTR

O Modelo Reflexivo Tempo Real RTR [FUR97], é um modelo de programação orientado à objetos e reflexivo destinado a programação de aplicações tempo real, com destaque àquelas que seguem o paradigma do melhor esforço. Dentre as principais vantagens do modelo destacamos, flexibilidade, capacidade para gerenciamento da complexidade e sua independência de suporte e de ambiente operacional relativa ao controle de restrições temporais das tarefas de uma aplicação e ao escalonamento tempo real destas tarefas.

A principal limitação do modelo é a falta de previsibilidade em função da flexibilidade oferecida; devido a essa restrição, o modelo não é adequado para programação de aplicações tempo real *hard*.

Por outro lado, o Modelo RTR apresenta características atrativas para modelagem e programação de aplicações tempo real *soft*, incluindo aplicações tempo real abertas [STA96].

O Modelo RTR (Reflexivo Tempo Real) é um modelo de programação para aplicações tempo real que se caracteriza como sendo uma extensão concorrente, reflexiva e tempo real do modelo de objetos convencional.

Por ser uma extensão do modelo de objetos convencional, o modelo proposto herda seus mecanismos de estruturação e conseqüentemente sua potencialidade relativa ao entendimento, reuso, extensão e manutenção de sistemas. Conseqüentemente, por ser orientado a objetos, o modelo proposto possibilita um tratamento das questões de concorrência e distribuição inerentes a muitos dos STR atuais, além de facilitar o uso de reflexão computacional.

Reflexão computacional é introduzida no modelo RTR através da abordagem de meta-objetos [MAE87], que estabelece a separação dos aspectos funcionais da aplicação (implementados através de objetos-base) do controle de seu comportamento (implementado através de meta-objetos), flexibilizando o desenvolvimento de STR, na medida em que permite que as políticas de controle sejam modificadas e/ou substituídas (inclusive dinamicamente), sem que os objetos base da aplicação e o suporte de execução necessitem ser modificados; assim sendo, a evolução do sistema bem como sua independência de ambiente operacional ficam facilitadas. No Modelo RTR, todos os aspectos temporais (restrições, exceções e escalonamento) e mais os aspectos de concorrência e sincronização são tratados de forma reflexiva, possibilitando o uso de diferentes mecanismos e políticas no controle do comportamento dos aspectos refletidos. Adicionalmente, a separação explícita entre questões funcionais e não funcionais, contribui para o entendimento do sistema, favorece o reuso e a manutenção de objetos-base e meta-objetos e incrementa a produtividade do programador.

Estruturalmente o modelo RTR é composto por objetos-base de tempo real, meta-objetos gerenciadores (um para cada objeto-base tempo real existente), um meta-objeto escalonador e um meta-objeto relógio, os quais interagem através de passagem de mensagens (ativações de métodos) síncronas e assíncronas, visando a realização das funcionalidades da aplicação de acordo com as restrições temporais associadas aos métodos dos objetos-base.

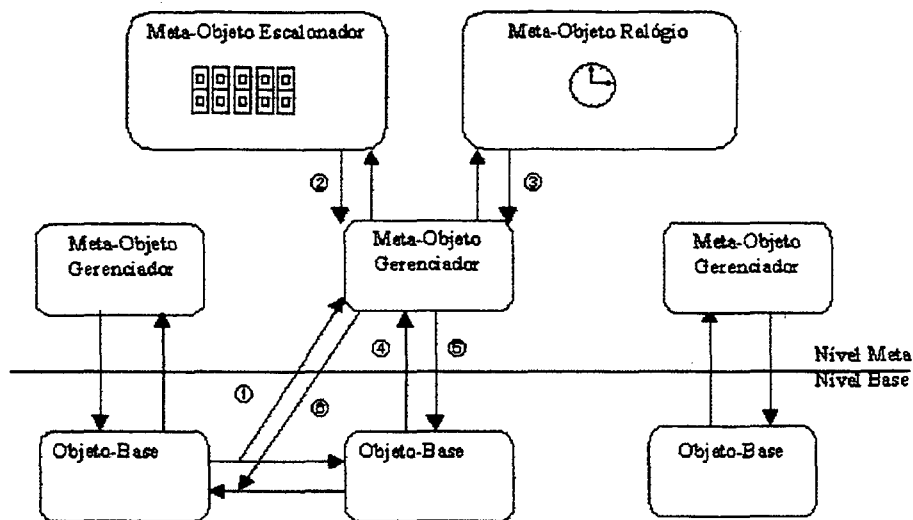


FIGURA 3.15 - Estrutura Geral do Modelo RTR

A Fig. 3.15 apresenta a dinâmica de funcionamento do modelo RTR, podendo ser assim descrita: A ativação de um determinado objeto-base é desviada para seu meta-objeto gerenciador correspondente (ação 1 da Fig. 3.15). O meta-objeto gerenciador interage com o meta-objeto escalonador (ação 2) e com o meta-objeto relógio (ação 3) para processar as restrições temporais associadas ao método solicitado e verificará as restrições de concorrência e sincronização; caso estas restrições não sejam violadas, ativará o método solicitado no objeto-base (ação 4) retornando em seguida, via meta-objeto gerenciador para o objeto que deu origem a chamada (ações 5 e 6).

Conforme descrito em [FUR97], o modelo RTR está estruturado com os seguintes componentes básicos:

- **Objetos-Base de Tempo Real (OBTR)** implementam a funcionalidade da aplicação e, em adição ao modelo de objetos convencional, podem ter restrições temporais e manipuladores de exceções temporais associados à declaração e ativação dos métodos.
- **Meta-Objetos Gerenciadores (MOG)** gerenciam o comportamento dos objetos-base, sendo responsáveis pelo gerenciamento das requisições para ativação dos métodos de seus objetos-base correspondentes, pelo controle de concorrência na execução de métodos, pelo gerenciamento das restrições de sincronização e pelo

processamento das restrições temporais e das exceções temporais associadas aos métodos dos objetos-base.

- **Meta-Objeto Escalonador (MOE)** tem como função básica receber, ordenar e liberar os pedidos de escalonamento (provenientes dos MOG) segundo uma determinada política de escalonamento, visando fornecer os parâmetros necessários ao escalonamento realizado pelo suporte subjacente.
- **Meta-Objeto Relógio (MOR)** tem como função controlar a passagem do tempo (visando a detecção de violação das restrições temporais) e realizar ativações programadas para um tempo futuro (ativações “*time-trigger*”).

### 3.6 – Conclusões

O presente capítulo apresentou as potencialidades e limitações do paradigma da reflexão computacional para a programação atual. A seguir a dinâmica de funcionamento de uma arquitetura reflexiva foi apresentada, sendo identificadas suas características, contribuições e limitações em relação ao modelo de programação convencional. Adicionalmente, a questão da reflexão computacional no domínio tempo real foi discutida, abordando suas principais características e sua utilização na programação de STR *soft*. Além disso, os principais modelos e linguagens baseados neste paradigma foram descritos.

Com relação aos modelos e linguagens apresentados, foram identificadas suas principais características, contribuições, aplicações e limitações.

Com base no estudo realizado, constatamos que muitos dos modelos e linguagens apresentados, não exploram completamente o paradigma da reflexão computacional tais como, flexibilidade, capacidade de integração, de evolução e independência de ambiente operacional. Tais fatores, flexibilizam a programação das aplicações, dentre as quais, aquelas que envolvem sincronização multimídia.

Neste sentido, este trabalho propõe a utilização do Modelo RTR (Reflexivo Tempo Real) baseado no paradigma da reflexão computacional, como um modelo alternativo e promissor, que permita que aplicações envolvendo sincronização

multimídia possam ser estruturadas e programadas de forma sistemática, confiável e flexível. Além disso, conforme apresentado na seção 3.5.2.3 deste capítulo, o modelo RTR também tem como objetivo ser independente de linguagem de programação e de ambiente operacional, favorecer a extensão e a evolução dos sistemas, permitindo sua adaptação a mudanças internas (no sistema) e externas (no ambiente operacional), favorecer o reuso e a manutenção do software desenvolvido. Assim sendo, concluímos que o modelo RTR pode contribuir significativamente na solução de muitos dos problemas atualmente encontrados na programação de aplicações multimídia, principalmente na especificação dos requisitos de sincronização.

O capítulo seguinte tem como objetivo, averiguar a adequação do modelo RTR para modelagem e programação de aplicações multimídia.

## Modelo RTR – Expressando e Implementando Restrições de Sincronizações Multimídia

### 4.1 – Introdução

Este capítulo tem como objetivo avaliar o Modelo Reflexivo para Tempo Real RTR [FUR97], afim de verificar sua adequação para especificação/implementação na questão da sincronização em aplicações multimídia. Inicialmente será apresentado a semântica de funcionamento dos operadores presentes no modelo baseado em intervalos [WAH94], seguindo a paradigma da estrutura proposta pelo modelo RTR. Além disso, será demonstrado um exemplo realizado sobre um cenário multimídia que objetiva verificar as potencialidades do modelo na programação de aplicações multimídia sob o aspecto da sincronização.

### 4.2 – Relações de Sincronização Multimídia

O modelo RTR é um modelo de programação para aplicações tempo real, apresentando funcionalidades particularmente atrativas para aplicações tempo real *soft* (onde as exigências de previsibilidade são menos severas), nas quais se inclui a classe de aplicações multimídia [FUR97].

O modelo RTR será usado neste trabalho como forma alternativa para tratar a questão da especificação da sincronização, e o nosso propósito é mostrar como esta especificação pode ser realizada segundo a filosofia do modelo. Para tanto, mostraremos que aplicações multimídia como, tele-conferências, ensino à distância, instrutores automáticos, entretenimento (jogos), aplicações médicas, comércio eletrônico, etc.,

cujos aspectos temporais podem ser adequadamente tratados através de abordagens do tipo melhor esforço, podem ser representadas pelo modelo [FUR97].

O modelo RTR permite que as várias situações de sincronização encontradas em aplicações multimídia, adotando o modelo de especificação baseado em intervalos [WAH94], sejam representadas.

Segundo o modelo baseado em intervalos, podem ser representados 13 diferentes tipos de relações temporais [ALL83]. Por sua vez, o modelo baseado em intervalos estendido [WAH94] consiste de 29 relações de intervalos (definidas a partir de disjunções das 13 relações básicas) que são consideradas relevantes para apresentação multimídia. Estas relações podem ser representadas, de forma simplificada, pelos 10 operadores conforme apresentados na seção 2.4.3.5.

No modelo RTR estes operadores (e conseqüentemente as relações de sincronização que eles representam) podem ser representados através de restrições temporais básicas, tais como *Periodic* (deve executar uma vez a cada período de tempo previamente especificado), *Aperiodic* (deve executar dentro de seu respectivo *deadline*), *Start-at* (o método é executado uma única vez, mas com início de execução programado para um instante de tempo futuro), e *ActivationInterval* (especifica um intervalo de tempo no qual um método deverá ser executado) [FUR97]. Por exemplo, a relação de sincronização descrita em [BLA96] como:

#### Animation while (d1, d2) Audio

Pode ser expressa no modelo RTR pela associação da restrição temporal “*ActivationInterval*” aos métodos responsáveis pela apresentação das mídias envolvidas, como mostrado abaixo.

```
void Animation (...), ActivationInterval (StarTime, EndTime),  
AnimationException (  
  
begin ...end;
```

```
void Audio (...), ActivationInterval (StartTime, EndTime),
    AudioException ( )
begin...end;
```

A sincronização desejada, representada graficamente na Fig. 2, será alcançada pela ativação assíncrona (denotada pelo símbolo “@”) dos métodos `Animation( )` e `Audio( )` abaixo especificada, onde  $T_1$ ,  $T_2$ ,  $\delta_1$  e  $\delta_2$  são atributos temporais da relação de sincronização representada na Fig. 4.16.

```
@Animation (...), (T1, T2);
@Audio (...), (T1 + d1, T2-d2);
```

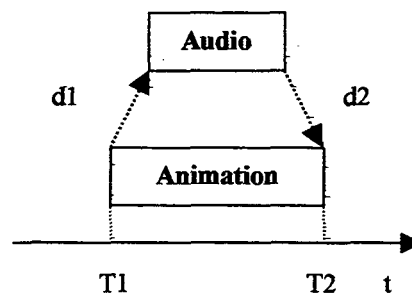


FIGURA 4.16 – Representação Gráfica da Relação “Animation while (d1,d2)Audio”

A sincronização das mídias será obtida através da interação entre **MOG**, **MOE** e **MOR**, segundo a semântica de execução das restrições temporais associadas aos métodos responsáveis pela apresentação dessas mídias. Neste caso, as ativações dos métodos `Animation()` e `Audio()` do objeto-base *Apresentador* serão desviados para o **MOG** apresentador que por sua vez ativará o método que implementa a restrição temporal *ActivationInterval*, o qual define (via interação com o **MOE**) o momento em que os métodos originalmente ativados deverão efetivamente ser executados, impondo assim o comportamento desejado (ou seja, a semântica de execução do operador de sincronização *while*).

Um dos problemas básicos do método de especificação baseado em intervalos, é que ele não permite que as relações temporais entre sub-unidades de uma mídia sejam representadas diretamente (a menos que ela seja explicitamente subdividida) [FUR97].



No exemplo apresentado é omitida a relação temporal existente entre os quadros da animação e entre os pacotes de áudio; usando o modelo RTR estas relações podem ser representadas através de métodos periódicos (onde cada período corresponde a um intervalo de apresentação) responsáveis pela apresentação (*display*) das sub-unidades que compõem cada mídia. Assim, a função básica dos métodos **Animation()** e **Audio()**, além de fixarem os intervalos de apresentação das mídias e a relação entre eles, é ativar os métodos periódicos **DisplayVideo()** e **DisplayAudio()** responsáveis pela apresentação dos quadros de imagem e pacotes de som que compõem as mídias consideradas. Neste caso a sincronização intramídia será obtida implicitamente através dos intervalos de ativação, preservando a abstração dos métodos.

Alternativamente à forma de especificação apresentada acima, a restrição de periodicidade que garante por si só a sincronização intramídia, pode ser redefinida para que a sincronização intermídia possa ser obtida; isto pode, por exemplo, ser obtido através da correlação temporal entre apresentações de sub-unidades de múltiplas mídias, tornando-se uma delas como sendo a mídia mestra. Por exemplo, no caso de sincronização de lábios (*lip-synchronization*), onde um atraso do vídeo com relação ao áudio e vice-versa é tolerável (dentro de limites preestabelecidos), o controle de sincronização pode ser efetivado tomando-se a mídia “*Audio*” como mestra e verificando se a apresentação de cada quadro de “*Video*” está sendo apresentado dentro das restrições toleradas. Este controle pode ser embutido em uma restrição *periodic* convencional, ou por questão de clareza e flexibilidade, pode ser implementado em um novo tipo de restrição temporal que englobe o controle de periodicidade com o controle da sincronização intermídia, como exemplificado abaixo [FUR97].

```
...
void DisplayAudio (...), Periodic (P=30, MET=3, Fim),
    AudioException()
```

```
begin...end;
```

```
void DisplayVideo (...), AudioSynchronization (P=40, MET=5, Varição=5
    VBA=15, VAA=150, Fim),
    VideoException()
```

**begin...end;**

Segundo este exemplo, a mídia áudio deve ser apresentada a uma taxa de 1 pacote a cada 30 ms e não suporta nenhum *jitter*, enquanto que a mídia vídeo deve ser apresentada a uma taxa de 1 quadro a cada 40ms suportando um *jitter* de +/- 5ms, o que implica que um quadro será apresentado entre 35 e 45 ms após o quadro anterior. Além disso, o exemplo estabelece que o vídeo pode preceder o áudio associado em até 15 ms (atributo “VBA –Video Before Áudio”) e que o vídeo não pode estar atrasado com relação ao áudio correspondente em mais de 150ms (atributo “VAA”-Video After Áudio).

As restrições temporais utilizadas no exemplo apresentado, estabelecem que a apresentação de cada mídia iniciará imediatamente após a ativação dos métodos envolvidos (“DisplayVídeo()” e DisplayAudio()) e se estenderá até o tempo especificado através do atributo “Fim”; alternativamente, o início da apresentação poderia ser explicitamente especificado através de um atributo adicional “Inicio”. Além disso, visando flexibilidade e reuso, os valores dos atributos temporais podem ser estabelecidos no momento da ativação dos métodos e não no momento da declaração como mostrado no exemplo utilizado.

Da mesma forma que o operador *while*, todos os demais operadores podem ser representados no modelo RTR através de restrições temporais básicas citadas anteriormente ou de uma combinação delas [FUR97].

A seção seguinte descreverá o comportamento dos demais operadores presentes no modelo baseado em intervalos seguindo o paradigma da estrutura proposta pelo modelo RTR

#### **4.2.1 - Representando os Operadores Baseados em Intervalos**

##### **Segundo o Modelo RTR**

Como exposto no item 4.2, os operadores do modelo de intervalos estendido usados na representação das relações de sincronização multimídia, podem ser implementados pelo modelo RTR através da associação de restrições temporais básicas

(*ActivationInterval*, *Periodic*, *Aperiodic* e *Start-at*) aos métodos responsáveis pela apresentação das mídias. A sincronização desejada será obtida pela ativação assíncrona (representada pelo símbolo @) destes métodos e pelo fornecimento de valores aos atributos das restrições temporais utilizadas [FUR97].

Nesta seção, representaremos todos os operadores considerando que os métodos responsáveis pela representação das mídias (métodos **Animation()** e **Audio()**) possuem um intervalo de ativação associado (restrição temporal *ActivationInterval*). Contudo, enfatizamos que esta é apenas uma das possibilidades existentes; dependendo da aplicação e do contexto no qual a sincronização deve ser obtida, diferentes restrições e/ou diferentes esquemas de ativação podem vir a ser utilizados [FUR97].

### Declaração dos métodos **Animation()** e **Audio()**

```

...
void Animation(...), ActivationInterval (Início, Fim)
{...}
void Audio(...), ActivationInterval (Início, Fim)
{...}
...

```

#### 4.2.1.1 - Operador Before

*Animation before (d1) Audio*

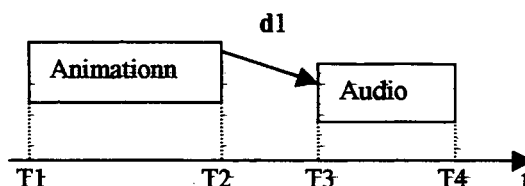


FIGURA 4.17 - Representação Gráfica da Relação “Animation before (d1) Audio”

```

...
@Animation (...), (T1,T2);
@Audio (...), (T2+d1, T4);
...

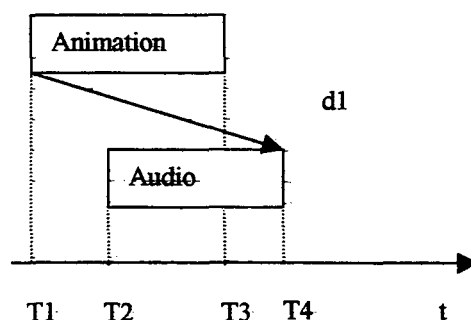
```

**Comportamento das mídias no operador de intervalo before:**

- No operador de intervalos *before* a mídia “Animation” deve iniciar sua apresentação antes da apresentação da mídia “Audio”;
- A mídia “Audio”, deverá iniciar sua apresentação após o término da mídia “Animation” adicionado a um intervalo de tempo ( $d1$ ms) estabelecido na ativação do método.

**4.2.1.2 - Operador Beforeendof**

*Animation beforeendof (d1) Audio*



**FIGURA 4.18 – Representação Gráfica da Relação “Animation beforeendof (d1) Audio”**

...

*@Animation (...), (T1,T3);*

*@Audio (...), (T2, T1+d1);*

...

**Comportamento das mídias no operador de intervalo beforeendof:**

- No operador de intervalo *beforeendof* a apresentação da mídia “Audio” deverá iniciar durante a apresentação da mídia “Animation”;
- O término da apresentação da mídia “Audio” deverá ocorrer no instante de tempo obtido por  $(T1+d1)$ , onde “T1” corresponde ao início da apresentação da mídia “Animation” e “d1” o intervalo de tempo entre estes eventos.

### 4.2.1.3 - Operador Cobegin

*Animation cobegin (d1) Audio*

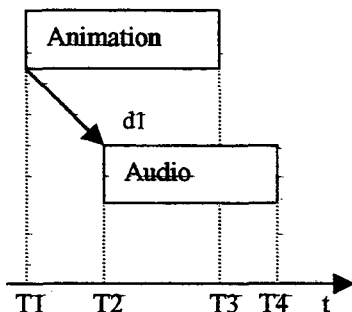


FIGURA 4.19 – Representação Gráfica da Relação “Animation cobegin (d1) Audio”

...

@Animation (...), (T1,T3);

@Audio (...), (T1 +d1,T4);

...

*Comportamento das mídias no operador de intervalo cobegin:*

- No operador de intervalo *cobegin* a apresentação da mídia “Audio” deverá iniciar durante a apresentação da mídia “Animation”, no intervalo de tempo (T1+d1), onde “T1” corresponde ao início da apresentação da mídia “Animation” e “d1” o intervalo de tempo permitido entre estes dois eventos.

### 4.2.1.4 - Operador Coend

*Animation coend (d1) Audio*

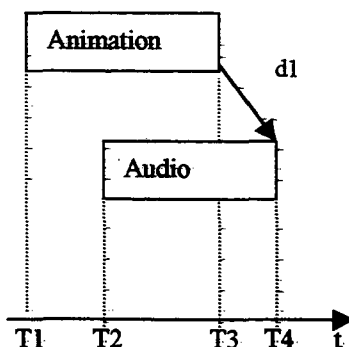


FIGURA 4.20 – Representação Gráfica da Relação “Animation coend (d1) Audio”

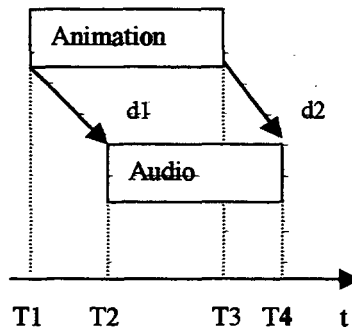
...  
 @Animation (...), (T1,T3);  
 @Audio (...), (T2,T3+d1);  
 ...

**Comportamento das mídias no operador de intervalo coend:**

- No operador de intervalo *coend* a mídia “Audio” deverá iniciar sua apresentação durante a apresentação da mídia “Animation” ;
- O “Audio” termina sua apresentação em (T3+d1), onde “T3” corresponde ao final da apresentação da mídia “Animation” e “d1” o intervalo de tempo permitido entre os dois eventos.

**4.2.1.5 - Operador Delayed**

*Animation delayed (d1,d2) Audio*



**FIGURA 4.21 – Representação Gráfica da Relação “Animation delayed (d1,d2) Audio”**

...  
 @Animation (...), (T1,T3);  
 @Audio (...), (T1 +d1,T3+d2);  
 ...

**Comportamento das mídias no operador de intervalo delayed:**

- No operador de intervalo *delayed* a apresentação da mídia “Animation” deverá iniciar em “T1” e terminar no instante de tempo definido pelo atributo “Fim” declarado na ativação dos métodos representado por “T3”;

- O início da apresentação da mídia “*Audio*” é dada por  $(T1+d1)$ , onde “*T1*” corresponde ao início da apresentação da mídia “*Animation*” e “*d1*” o intervalo de tempo permitido entre os dois eventos;
- A apresentação da mídia “*Audio*” será finalizada em  $(T3+d2)$ , sendo que “*T3*” corresponde ao final da apresentação da mídia “*Animation*” e “*d2*” o intervalo de tempo permitido entre estes dois eventos.

#### 4.2.1.6 - Operador Startin

*Audio startin (d1,d2) Animation*

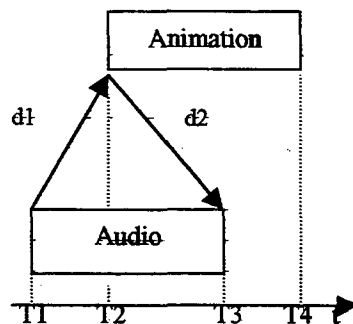


FIGURA 4.22 – Representação Gráfica da Relação “*Animation Startin (d1,d2) Audio*”

...  
 @Audio(...), (T1,T1+d1+d2);  
 @Animation (...), (T1 +d1,T4);

#### *Comportamento das mídias no operador de intervalo startin:*

- No operador de intervalo *startin* a mídia “*Audio*” deverá iniciar sua apresentação em “*T1*” e terminar em  $(T1+d1+d2)$  onde, “*T1*” corresponde ao instante de tempo representado como o início da apresentação da mídia “*Audio*” e “*d1*” o intervalo de tempo permitido entre os eventos representados pelo início da apresentação do “*Audio*” até o início da apresentação da mídia “*Animation*”, e “*d2*” o intervalo de tempo entre os eventos representados pelo início da apresentação da mídia “*Animation*” e final da apresentação da mídia “*Audio*”;

- A apresentação da mídia “*Animation*” é dada por  $(T1+d1)$ , e sua apresentação deverá ser finalizada em “ $T4$ ”.

#### 4.2.1.7 - Operador Endin

##### *Animation endin (d1,d2) Audio*

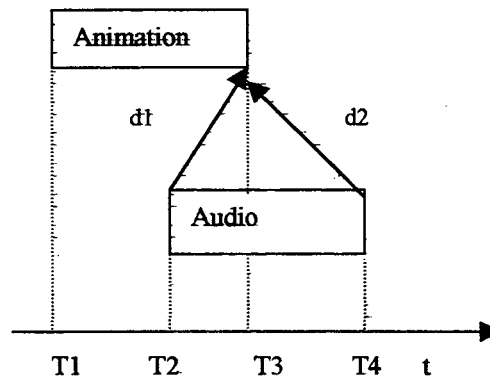


FIGURA 4.23 – Representação Gráfica da Relação “*Animation Endin (d1,d2) Audio*”

...  
 @Animation(...), (T1,T3);  
 @Audio (...), (T3 -d1,T3+d2);  
 ...

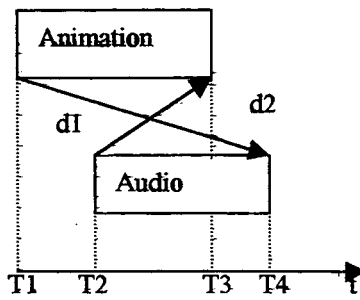
##### *Comportamento das mídias no operador de intervalo startin:*

- No operador de intervalo *endin* a mídia “*Animation*” deverá iniciar sua apresentação no intervalo de tempo “ $T1$ ”, e finalizada em  $T3$ ;
- A apresentação da mídia “*Audio*” inicia em  $(T3-d1)$ , e deverá ser finalizada em  $(T3+d2)$ , onde “ $d1$ ” e “ $d2$ ” são os intervalos de tempo permitidos entre a ocorrência destes eventos.

#### 4.2.1.8 - Operador Cross

##### *Animation cross (d1,d2) Audio*





**FIGURA 4.24 – Representação Gráfica da Relação “Animation cross (d1,d2) Audio”**

...  
**@Animation (...), (T1, T2+d1);**

**@Audio (...), (T2, T1+d2);**

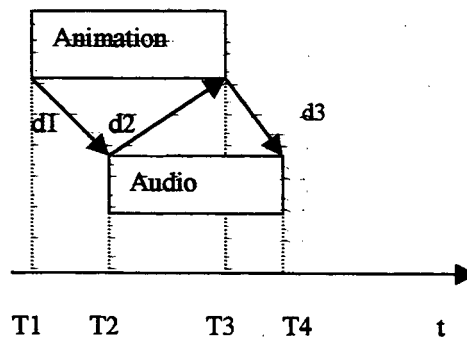
...

**Comportamento das mídias no operador de intervalo cross:**

- No operador de intervalo *cross* a mídia “Animation” deverá iniciar sua apresentação no instante de tempo “T1” representado na declaração (ou ativação) do método como o atributo “Início”, sendo que o seu término é obtido de (T2+d1);
- No operador de intervalo *cross* a mídia “Audio” deverá iniciar sua apresentação no intervalo de tempo “T2”, e sua apresentação será finalizada em (T1+d2).

**4.2.1.9 - Operador Overlaps**

**Animation overlaps (d1,d2) Audio**



**FIGURA 4.25 – Representação Gráfica da Relação “Animation overlaps (d1,d2) Audio”**

...

**@Animation (...), (T1, T1+d1+d2);**

*@Audio (...), (T1+d1, T1+d1+d2+d3);*

...

*Comportamento das mídias no operador de intervalo overlaps:*

- No operador de intervalo *overlaps* a mídia “Animation” deverá iniciar sua apresentação no instante de tempo “T1” definido na declaração (ou ativação) do método, sendo que o final de sua apresentação é dado por  $(T1+d1+d2+d3)$ ;
- A mídia “Audio” deverá iniciar sua apresentação em  $(T1+d1)$ , e seu término é dado por  $(T1+d1+d2)$ .

Na próxima seção será apresentado a especificação da sincronização de um cenário multimídia utilizando a abordagem proposta pelo modelo RTR, destacando as vantagens e desvantagens inerentes a aplicabilidade do modelo nessa classe de aplicação.

### 4.3 – Exemplo de Especificação da Sincronização Multimídia Utilizando o Modelo RTR

Nesta seção apresentaremos a especificação de um cenário multimídia utilizando o modelo RTR, afim de avaliar sua expressividade e capacidade de representar e controlar a sincronização em cenários multimídia. O exemplo segundo [BLA96], consiste de uma seqüência de áudio/vídeo (Áudio1, Vídeo) sincronizados (*lip synchronization*), seguida pelo *replay* de uma interação de usuário gravada, uma seqüência de Slides (S1-S3) e uma animação, que é parcialmente comentada usando uma seqüência de áudio (Áudio2). Iniciando a apresentação da animação, uma questão de múltipla escolha é apresentada para o usuário (Interação). Se o usuário tiver feito a seleção o slide S4 é mostrado.

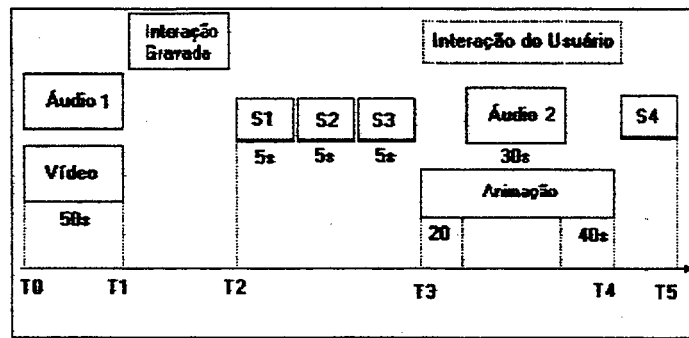


FIGURA 4.26 – Exemplo de Sincronização [BLA96].

Segundo [BLA96], a representação da sincronização baseada no modelo de intervalos pode ser especificada como segue:

```

Audio1 while (0,0) Video
Audio1 before (0) Recorded Interaction
RecordedInteraction before (0) S1
S1 before (0) S2
S2 before (0) S3
S3 before (0) Interaction
S3 before (0) Animation
Animation while (20,40) Audio2
Interaction before (0) S4
    
```

FIGURA 4.27 – Representação da Sincronização Baseado no Modelo de Intervalos.

As relações de sincronização especificadas na Fig. 4.27, podem ser representadas conforme descrito em [FUR97], através de restrições temporais convencionais associadas aos métodos dos objetos-base da aplicação. Apresentaremos a seguir uma série de considerações relativas a solução proposta.

```
OBTR class Controlador
begin
...
void Video (...), ActivationInterval(Tinicio, Tfim);
    VideoException( )
begin
...
    DisplayVideo (...), (25, TFin, 80, 10, 150);
...
end;
void AudioI(...), ActivationInterval(Tinicio, Tfim);
    AudioIException( )
begin
...
    DisplayAudioI (...), (30, TFin);
...
end;
void RecordedInteraction (...), ActivationInterval (Tinicio, Tfim);
    RecordedInteractionException( )
begin
...
end;
void Slide(...), Periodic (P, Tinicio, Tfim, MET=15);
    SlideException ( )
begin
...
    // Display um Slide a cada periodo
...
end;
void Animacao (...), ActivationInterval (Tinicio, Tfim);
    AnimationException ( )
begin
...
    DisplayAnimacao (...), (10, 15, TFin);
...
end;
void Botão (...), ActivationInterval (Tinicio, Tfim);
    BotaoException( )
```

```

begin
  ...
end;
void Audio2(...), ActivationInterval (Tinicio, Tfim),
    AudioException ( )
begin
  ...
  DisplayAudio2(...), (10, 40);
  ...
end;
void slide4 (...), ActivationInterval(Tinicio, Tfim)
    Slide4Exception ( )
begin
  ...
end;
void Main ( )
begin
  ...
  @ Video (...), (T0,T1);
  @ Audio1 (...), (T0, T1);
  @ RecordedInteraction (...), (T1, T2);
  @ Slide (Slide1, Slide2, Slide3), (P,T2,T3);
  @ Botao (T3,T4);
  @ Animação(...), (T3,T4);
  @ Audio2 (...), (T3+d1, T4-d2);
  if (Botao==true)
  {
    // Apresenta Slide4
    @ Slide4 (...), (T4-T3, T5);
  }
  else
  break;
  }
  ...
end;
end;

```

FIGURA 4.28– Pseudo-Código do OBTR “Controlador”

- *Considerações sobre o OBTR “Controlador”*

1 – A relação “*while (T0,T1)*” entre as mídias “*Video*” e “*Audio1*” é representada pela restrição *ActivationInterval*, onde atribuindo-se valores idênticos aos atributos “*Tinicio*” e “*Tfim*” (respectivamente T0 e T1), resulta no comportamento especificado pela relação “*while(0,0)*”.

2 – A relação “*Audio1 before (d) RecordedInteraction*” pode ser expressa pela associação da restrição *ActivationInterval()*, usando-se o “*Tinicio*” de *RecordedInteraction* como sendo o “*Tfinal*” do *Audio1*.

3 – As relações “*S1 before(d) S2*”, “*S2 before (d)S3*” entre os slides S1, S2, S3, é representado pela restrição *Periodic()* que especifica um período de tempo no qual um slide deverá apresentado, sendo que o atributo “*MET*”, especifica o máximo tempo de execução do método.

4 – A relação “*Animation While (T3, T4) Audio2*” é representada pela associação da restrição *ActivationInterval()* aos métodos “*Animation ()* e *Audio()*”, especificando um intervalo no qual o método deverá ser executado representada pelos atributos (T3 e T4) respectivamente.

5 – Iniciando a apresentação da animação uma questão de múltipla escolha é apresentada ao usuário (Botão). Se o usuário tiver feito a seleção (botao==true), o Slide S4 é mostrado, ou a apresentação será encerrada. A relação “*Botao before (d) Slide4*” é representada pela associação da restrição *ActivationInterval()* ao método “*Slide4 ()*”, onde os atributos “*Tinicio*” e “*Tfim*” representam um intervalo de tempo no qual a apresentação do slide deve ser solicitada, representados por (T4-T3 e T5) respectivamente. A exceção “*Slide4Exception ()*” será ativada caso o valor atribuído ao atributo “*Tfim*” seja violado.

---

<sup>1</sup> A denominação do método “*Interaction before (d) Slide4*”, foi alterada para “*Botao before (d) Slide4*”, com intuito de facilitar a interpretação.

```

OBTR class Apresentador
begin
...
void DisplayVideo (...), AudioVideoSynchronization (P, Tfim, var, VBA, VAA, MET=50),
                                VideoException ( )
begin...end;
void DisplayAudio1 (...), Periodic (P,Tfim, MET=50),
                                Audio1Exception( )
begin...end;
void DisplayAnimacao (...), Periodic (P, Tfim, MET=40),
                                AnimacaoException ( )
begin...end;
void DisplayAudio2 (...), Periodic (P, Tfim, MET=40),
                                Audio2Exception ( )
begin...end;
...
end;

```

FIGURA 4.29 – Pseudo-Código do OBTR “Apresentador”

- **Considerações sobre o OBTR “Apresentador”**

1 – Com o objetivo de obter o comportamento temporal intramídia na *relação “Audio While (T0,T1) Animation”* que consiste de um cenário com sincronização de lábios (*lip-synchronization*), a restrição *AudioVideoSynchronization* é associada ao método “DisplayVideo()” especificando que, a mídia “Video” deve ser apresentada a taxa de 1 quadro a cada 25 ms através do atributo período “P”, suportando um *jitter* de ±80 ms entre a apresentação dos quadros representado pelo atributo “var”. Adicionalmente, o vídeo poderá preceder o áudio em até 10ms (atributo “VBA”-VideoBeforeAudio) e que o vídeo não pode estar atrasado com relação ao áudio correspondente em mais de 150 ms (atributo “VAA”-VÍdeoAfterAudio).

2 – A restrição *Periodic* associada ao método “DisplayAudio1( )” especifica que a mídia “Audio1” deve ser apresentada a taxa de 1 pacote (1/30) a cada 30 s o equivalente a 2666,67 amostras de áudio por ms e, não suporta nenhum *jitter*.

Conforme a dinâmica do Modelo RTR a sincronização das mídias presentes na aplicação será obtida através da interação entre MOG, MOE e MOR, segundo a semântica de execução das restrições temporais associadas aos métodos responsáveis pela apresentação dessas mídias. Neste caso, as ativações dos métodos “Video(), Audio1(), RecordedInteraction(), Slide(), Animacao() e Audio2()” do objeto-base *Controlador* serão desviadas para o MOG *Controlador*. Por sua vez, no momento que as ativações dos métodos são desviadas para seu MOG correspondente, este ativará o método que implementa a restrição temporal correspondente como por exemplo, os métodos “Video(), Audio1(), Animacao() e Audio2()”, ativarão a restrição *ActivationInterval*, a qual define (via interação com o MOE) o momento em que os métodos originalmente ativados deverão efetivamente serem executados, impondo assim o comportamento desejado. A interação com o usuário é programada explicitamente como parte da ativação das demais mídias. Adicionalmente o usuário poderá declarar novos tipos de restrições temporais, definindo porém, para cada novo tipo de restrição temporal introduzido a nível de objeto-base, uma implementação no meta-objeto gerenciador correspondente.

#### 4.3.1 – Considerações Gerais Sobre o Exemplo Apresentado

A especificação do cenário multimídia usado, permitiu verificar as funcionalidades do modelo RTR. Este exemplo permitiu demonstrar a simplicidade e a flexibilidade do modelo no controle da sincronização multimídia através de restrições temporais pré-definidas, tais como *Periodic*, *Aperiodic* e *ActivationInterval*. Esta flexibilidade também se estende aos métodos manipuladores de exceções, onde o usuário poderá especificar métodos manipuladores de exceções a serem executados quando a apresentação de uma mídia não puder ser realizada dentro de seu *deadline*. Adicionalmente a técnica reflexiva utilizada traz grandes benefícios na programação de sistemas multimídia permitindo separar os aspectos funcionais dos aspectos não-funcionais, onde a solução do problema em si (com relação as suas funcionalidades básicas) seja expressa através de objetos-base, e que o controle do comportamento desses objetos (restrições, exceções, escalonamento, concorrência e sincronização) sejam expresso através de meta-objetos. Complementarmente, deve ser ressaltado que



os métodos ativados poderão executar paralelamente (*threads* distintas), ou em série (mesma *thread*), de acordo com a sincronização desejada para o cenário. Para o exemplo apresentado, os métodos “**Audio1()**, **Video1()**, **Botao()**, **Audio2()** e **Animacao()**, necessitarão executar em *threads* distintas, os demais métodos poderão executar na mesma *thread*, evitando possíveis *overheads* causados pela utilização de muitas *threads* distintas.

A facilidade do modelo proposto e a efetividade de seu comportamento com relação ao controle dos eventos síncronos (previsíveis), onde as posições no tempo são determinadas previamente como por exemplo (início da apresentação de uma seqüência de áudio ou vídeo), ficaram demonstradas através da especificação do cenário exemplo apresentado.

Entretanto, o uso do modelo RTR torna um pouco complicado a representação de eventos assíncronos (imprevisíveis), cujas posições no tempo não podem ser determinadas previamente, como por exemplo o instante em que a aplicação chega a um determinado estado ou uma interação com o usuário. Estes eventos ocorrem freqüentemente em apresentações de aplicações multimídia. Além disso, na solução proposta, as restrições temporais são associadas diretamente aos métodos responsáveis pela apresentação das mídias, sendo necessário a criação de métodos artificiais para cada situação distinta em que a mídia aparece, dificultando a composição de cenários onde a mesma mídia deve ser apresentada sob diferentes restrições de sincronização.

## 4.4 – Conclusões

**E**ste capítulo se propôs a apresentar uma descrição detalhada da potencialidade e expressividade do Modelo Reflexivo Tempo Real (RTR), na especificação de restrições de sincronização multimídia.

Nesta descrição, inicialmente foi demonstrado que o modelo RTR representa dos operadores de intervalos [WAH94], e as relações de sincronização que eles representam, através de restrições temporais básicas pré-definidas no modelo, ou

personalizadas especificamente para atender as necessidades específicas de uma aplicação. Essa característica demonstra a flexibilidade do modelo RTR no controle da sincronização multimídia. Adicionalmente neste capítulo, foi apresentado a especificação de um cenário multimídia utilizando o modelo RTR. A dinâmica de funcionamento e os requisitos de sincronização deste exemplo foram estudados detalhadamente. O exemplo apresentado demonstrou que o modelo RTR apresenta em sua estrutura características essenciais para atender as várias situações de sincronização encontradas em cenários multimídia. Além disso, na solução proposta, algumas limitações presentes no modelo RTR para especificação da sincronização multimídia foram abordadas. Com base nisso, no próximo capítulo, será apresentada uma proposta alternativa para especificação da sincronização em aplicações multimídia no modelo RTR, visando contribuir na solução proposta pelo modelo.

## Proposta Alternativa para Especificação dos Métodos que Representam o Comportamento dos Operadores de Intervalos

---

### 5.1 – Introdução

Com base nos problemas e limitações encontrados na especificação de sincronização multimídia utilizando o modelo RTR, concluiu-se pela necessidade de se estender o modelo para prover a semântica de qualquer um dos operadores através da criação de um único método genérico (denominaremos aqui de **Sincroniza**), contendo como parâmetros os tempos (**Tinicio**, **Tfim**) de cada mídia envolvida na apresentação. Entretanto, por questão de legibilidade e de fidelidade com os cenários a serem apresentados, cada operador será representado por um método específico, o qual ativará o método genérico **Sincroniza** após ter verificado de forma reflexiva, se os valores atribuídos aos tempos (**Tinicio** e **Tfim**) de cada mídia são consistentes com a sincronização desejada, abstraindo do usuário questões relativa a representação da semântica dos operadores de intervalos. Nesta nova solução, eventos assíncronos e interação com o usuário embora continuem necessitando de uma programação explícita, passarão a ter sua execução diretamente controlada pelo método **Sincroniza** que implementa a semântica da sincronização dos operadores de intervalo.

Este capítulo apresenta a dinâmica de funcionamento da extensão da proposta original apresentada em [FUR97], para representação das várias situações de sincronização encontradas na multimídia através do modelo RTR. Adicionalmente, é apresentado a especificação de um cenário multimídia com base na nova abordagem de especificação proposta.

## 5.2 – Apresentação da Extensão Proposta

Para ilustrar a extensão proposta para especificação do comportamento dos operadores baseados em intervalos apresentaremos a seguir, um exemplo de aplicação utilizando o operador *before*. Para representar um cenário onde por exemplo, uma animação irá preceder a apresentação de um áudio, a relação de sincronização entre estas mídias pode ser expressa como exemplificado abaixo:

### *Animation before (d1) Audio*

Os métodos “**Animation()**” e “**Audio()**” serão responsáveis pela apresentação das mídias envolvidas, as quais possuem um intervalo predefinido no qual deverão ser apresentados. A sincronização desejada entre as mídias, será alcançada pela ativação do método que implementa o comportamento do operador *before* o qual tem a função de verificar se os valores atribuídos aos atributos “**Tinicio**” e “**Tfim**” de cada mídia expressam a sincronização desejada, e em caso afirmativo, ativará o método **Sincroniza()** que implementa a semântica dos operadores de intervalos. Os métodos a serem executados (“**Animation()**” e “**Audio()**”), juntamente com a definição de seus intervalos, e o tratador de exceção são passados como parâmetros:

```
...
before (Animation, TiAnimacao, TfAnimacao, Audio, TiAudio,
TfAudio, TratadorExcecao);
---
```

onde, o atributo temporal “**d1**” da relação de sincronização apresentada é deduzido implicitamente à partir do  $tf_{Animacao}$  e  $ti_{Audio}$ .

Afim de obtermos o comportamento especificado no exemplo citado, segundo a filosofia do modelo RTR, os métodos que especificam os operadores de intervalos (responsáveis pela verificação da consistência da sincronização), e o método *Sincroniza* (que provê a semântica do operadores) deverão ser implementados como métodos da seção de restrições temporais de um MOG, (disponibilizadas como bibliotecas ao usuário), como exemplificados respectivamente nas Fig. 5.30 e 5.31.

Por questão de objetividade, os métodos “Animation()” e “Audio()”, serão representados por “A” e “B” respectivamente na especificação dos métodos apresentados a seguir:

```
void before (tiA,tfA,tiB, tfB){
    // Verifica consistência da sincronização
    If (tfA>tiA) && (tfA <=tiB) && (tfB>tiB) then{
        // Ativa método Sincroniza do MOG
        Sincroniza();
    }
    Else
    {
        // Os tempos (tinicio, tfim) das mídias não são consistentes com
        // a sincronização desejada.
    }
}
```

FIGURA 5.30- Pseudo-Código do Operador *before*

```
void Sincroniza (A, tiA, tfA, B, tiB, tfB, id-excecao) {  
    If current-time < tfA then {  
        // Método A ainda pode ser apresentado  
        // Solicita pedido de escalonamento ao MOE  
        id-MOE.Escalona(...)  
        // Ao ser liberado pelo MOE, verifica se o início do intervalo já foi  
        alcançado  
        If current-time < tiA then {  
            // Ainda não chegou o tempo de início do intervalo no qual A  
            // deve ser executado.  
            // Aguardar instante da apresentação de A  
            wait (TiA - current-time);  
        }  
        // O intervalo em que A deve ser apresentado foi alcançado  
        // Verifica se há tempo suficiente para apresentação de A  
        If (current-time + texecA) < TfA then {  
            // Programa relógio para levantar exceção se em tfA o método A  
            // ainda não tiver terminado sua execução  
            MOR.ProgramaAtivacao(tfA, id-Met, id-excecao);  
            // Cria nova thread para execução do método A do objeto-base  
            Thread metodoA = new Thread(this);  
            metodoA.start();  
            // Solicita ao MOE escalonamento do método B  
            id-MOE.Escalona(...);  
            // Verifica se já chegou o tempo de início do intervalo no qual B  
            // deve ser executado  
            If current-time < TiB then {  
                // Aguardar instante da apresentação de B  
                wait (TiB - current-time);  
            }  
        }  
    }  
}
```

```

// O intervalo em que B deve ser apresentado foi alcançado
    // Verifica se há tempo suficiente para apresentação de B
    If (current-time+texecB) < TfB then{
        // Programa relógio para levantar exceção se em tfB o método B
        // ainda não tiver terminado sua execução
        MOR.ProgramaAtivacao(tfB, id-Met, id-excecao);
        // Ativa método B no objeto-base
        objeto-base.B(...);
    }
    Else
    {
        // Método B não pôde ser executado dentro de seu intervalo
        mínimo
        // Ativa tratador de exceção
        excecao(id_excecao);
    }
    Else
    {
        // Método A não pôde ser executado dentro de seu intervalo mínimo
        // Ativa tratador de exceção
        excecao(id_excecao);
    }
}
Else {
    // Tempo corrente maior que o TfA
    // Sinaliza exceção para que a apresentação seja cancelada
    excecao(id-excecao)
}
}
}

```

FIGURA 5.31- Pseudo-Código do Método Sincroniza

Os demais operadores poderão ser representados no modelo RTR da mesma forma que o operador *before*; ou seja, verificarão a consistência entre os intervalos especificados, a sincronização desejada e usarão o método *Sincroniza* para garantir essa sincronização. No apêndice A é apresentada a especificação de cada um desses operadores.

▪ **Análise da solução proposta**

O método proposto “*Sincroniza*”, poderá ser utilizado na representação de qualquer um dos operadores de intervalos levando-se em consideração algumas premissas básicas:

- Os tempos de cada mídia (**tinício**, **tfim**), são conhecidos;
- A sincronização desejada será garantida se o intervalo entre as mídias for respeitado (o que equivale dizer que não importa o momento exato em que o método é executado, desde que sua execução inicie e termine dentro do intervalo especificado);
- Os deltas entre as mídias correspondem ao tempo mínimo após um determinado evento, por exemplo para o operador *before* representado graficamente na Fig. 5.32.

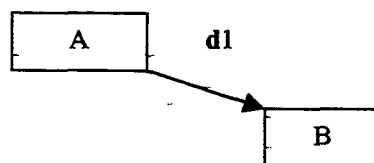


FIGURA 5.32 – Representação Gráfica do Operador *before*

- Considera que, B não pode iniciar antes de  $tf_A + d1$  (independentemente do tempo real em que a execução de A encerrar), e poderá ser executado em qualquer tempo entre  $tf_A + d1$  e  $tf_B$ , desde que sua execução esteja concluída em  $tf_B$ .

Os métodos que implementam os operadores de intervalos, e o método *Sincroniza*, fazem parte do comportamento do MOG e são ativados explicitamente a partir dos objetos-base da aplicação. Apesar dessa forma de especificação acarretar na perda da propriedade da abstração da reflexão computacional no que se refere a chamada dos métodos, o controle destes métodos continua sendo tratado de forma



reflexiva. Adicionalmente podemos concluir que, a dinâmica apresentada flexibiliza a combinação de um maior número de mídias e de diferentes operadores, sendo este fator essencial para especificação de cenários multimídia complexos. O exemplo abaixo ilustra a sincronização de 3 mídias utilizando o operador *before* :

```
before (A, tiA, tfA, (before(B, tiB, tfB, C, tiC, tfC, TratadorExcecao2),
TratadorExcecao1);
```

Diferentes operadores, podem ser combinados visando a obtenção de comportamentos diversificados, como por exemplo: *before(A, ti<sub>A</sub>, tf<sub>A</sub>, while(B, ti<sub>B</sub>, tf<sub>B</sub>, C, ti<sub>C</sub>, tf<sub>C</sub>, TratadorExcecao2), TratadorExcecao1).*

▪ *Outras considerações*

Além da forma de especificação apresentada, a qual considera que os tempos de execução das mídias envolvidas no cenário são conhecidos (e devem ser satisfeitos), podemos representar o comportamento dos operadores no modelo RTR, quando tais tempos são desconhecidos ou irrelevantes (situações em que o importante é a sincronização propriamente dita, e não o momento real no qual ocorrerá). Neste caso, cada operador de intervalo terá um método responsável por garantir a sincronização desejada. Para exemplificar esta situação, apresentaremos o comportamento do operador *before* para a relação de sincronização “*A before (d1) B*” na Fig. 5.33. Os demais operadores terão uma implementação similar.

```
void before (parâmetros){
    //Solicita escalonamento do método A ao MOE
    id.MOE.Escalona()
    // Ativa método A no objeto-base
    id-obj_base.A();
    // Salvar tempo final de A
    tfA ← current-time;
    //Solicita escalonamento do método B ao MOE
    id-MOE.Escalona(...)
```

```
If (current-time - tfA) < d1 then {  
    // Aguarda a passagem de d1  
    wait(tfA+d1);  
}  
  
// Ativar método B no objeto-base  
id-obj_base_B();  
}
```

FIGURA 5.33- Pseudo-Código do Comportamento do Operador *before* para Tempos Desconhecidos

Com base nos exemplos apresentados, conclui-se que as características pertinentes a nova proposta, aumentam a expressividade do modelo, uma vez que:

- Uma determinada mídia pode ser combinada com quaisquer outras, através de quaisquer operadores de intervalos;
- A composição de cenários multimídia complexos fica facilitada (especialmente naqueles com mais de duas mídias correlacionadas);

Adicionalmente, nada impede que usuários familiarizados com o modelo e com a semântica dos operadores de intervalos desenvolvam aplicações que tirem vantagem dessas duas soluções (a original e a extensão proposta).

Na próxima seção apresentaremos um cenário multimídia com base na extensão proposta, que envolve sincronização síncrona e assíncrona entre mídias, afim de ilustrar a adequação da extensão proposta para especificação de cenários multimídia complexos.

### **5.3 – Especificação da Sincronização de uma Aplicação Multimídia Utilizando a Extensão Proposta para o Modelo RTR**

Nesta seção será definida uma aplicação multimídia e apresentada sua especificação utilizando a técnica de sincronização da extensão proposta para o modelo RTR.

### 5.3.1 = Definição da Aplicação Multimídia

A aplicação consiste de um CD-ROM multimídia, que tem como objetivo apresentar as características de uma cidade. O CD-ROM está organizado em módulos onde, cada módulo possui um documento multimídia correspondente contendo informações diversas a respeito de um cidade. Para exemplificar o uso do modelo RTR nesta aplicação, usaremos especificamente os módulos “Turismo” e “Síntese Econômica”.

Ao iniciar a apresentação da aplicação do módulo “Turismo”, a primeira cena (Cena1) é automaticamente apresentada. O usuário poderá interagir neste cenário avançando para a cena seguinte (Cena2), selecionar a apresentação do próximo módulo “Síntese Econômica”, ou ainda finalizar a apresentação. Durante a apresentação da Cena2, o usuário poderá somente finalizar a apresentação. Ao término da apresentação da Cena2, é apresentada a Cena3. A qualquer momento da apresentação da Cena3, o usuário poderá escolher pela apresentação do módulo 2 “Síntese Econômica”, ou finalizar a apresentação. No instante em que o usuário selecionar a apresentação do módulo 2 “Síntese Econômica”, nos cenários anteriores (que dispunham dessa opção), a Cena1 referente a este módulo é apresentada, sendo seguida pela apresentação da Cena2. Os botões de interação componentes de cada cenário estão disponíveis durante toda a apresentação. Todas as interações têm um comportamento do tipo *Master* em relação as mídias do cenário, isto é, interrompem a sua apresentação quando da interação do usuário.

Visando melhor entendimento da aplicação em questão, apresentaremos a seguir a descrição das relações de sincronização presentes em cada cenário.

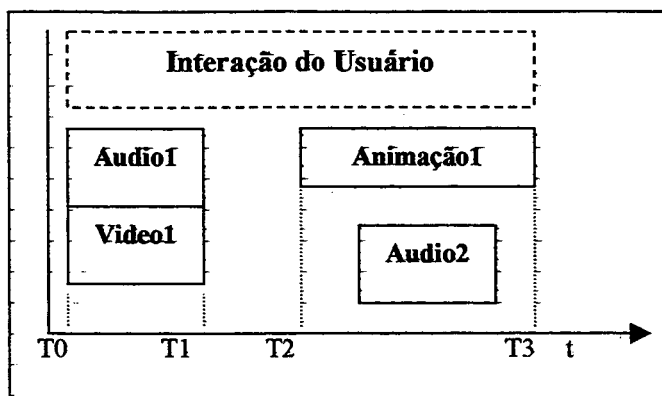


FIGURA 5.34 – Apresentação do Cenário Cena 1 do Módulo Turismo

A Fig. 5.34, apresenta um cenário que consiste de uma seqüência de áudio/vídeo (Audio1, Video1) sincronizados, uma animação que é comentada usando uma seqüência de áudio (Audio2). O usuário poderá interagir neste cenário avançando para a próxima cena (Cena2), finalizando a aplicação, ou escolhendo pela apresentação do modulo 2 “Síntese Econômica.

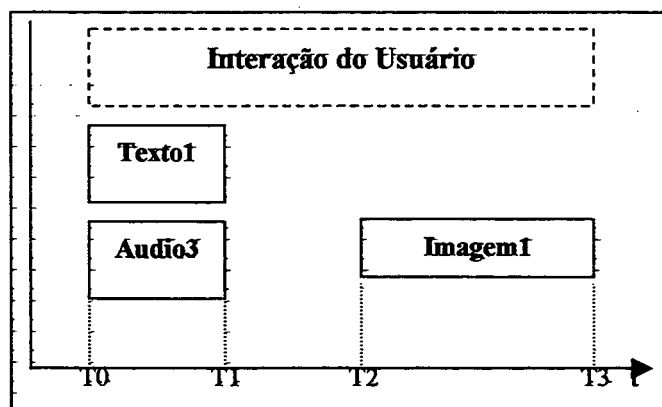


FIGURA 5.35 – Apresentação do Cenário Cena 2 do Módulo Turismo

A Cena2 do módulo Turismo é apresentada na Fig. 5.35. Este cenário é composto por um texto (Texto1) parcialmente comentado por uma seqüência de áudio (Audio3), seguido de uma imagem (Imagem1). Se o usuário interagir na aplicação a apresentação será encerrada, se não houver interação, a Cena3 é apresentada.

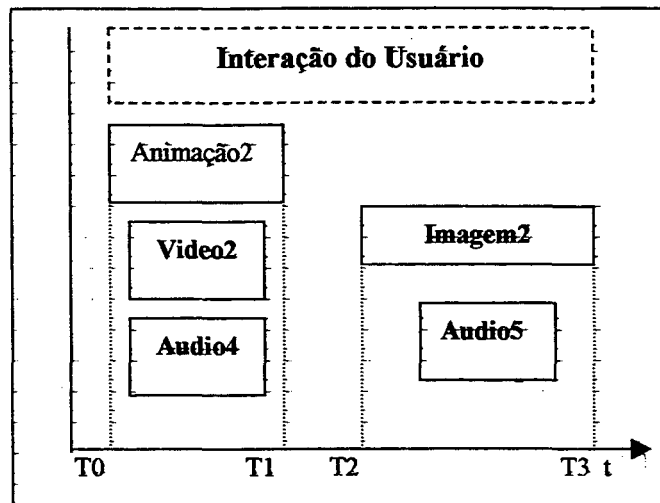


FIGURA 5.36 – Apresentação do Cenário Cena 3 do Módulo Turismo

A Cena3 apresentada na Fig. 5.36 é composta por áudio/vídeo (Audio4 e Video2) apresentados simultaneamente com uma animação (Animacao2), seguido de uma imagem (Imagem2) parcialmente comentado por um áudio (Audio5). A qualquer momento o usuário poderá escolher pela apresentação do modulo 2 “Síntese Econômica”, ou finalizar a apresentação.

As Fig. 5.37 e 5.38 ilustradas a seguir representam a sincronização dos cenários que compõem o módulo “Síntese Econômica”.

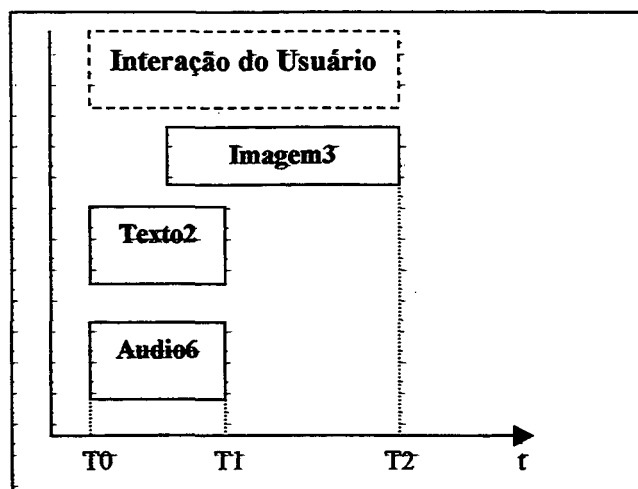


FIGURA 5.37 – Apresentação do cenário Cena 1 do módulo Síntese Econômica

A Cena1 demonstrada na Fig. 5.37 é composta por Texto/Áudio (Texto2 e Audio6) apresentados simultaneamente, seguido de uma imagem (Imagem3). Ao término da apresentação da imagem a Cena2 é apresentada, somente a opção para finalizar a apresentação estará disponível ao usuário.

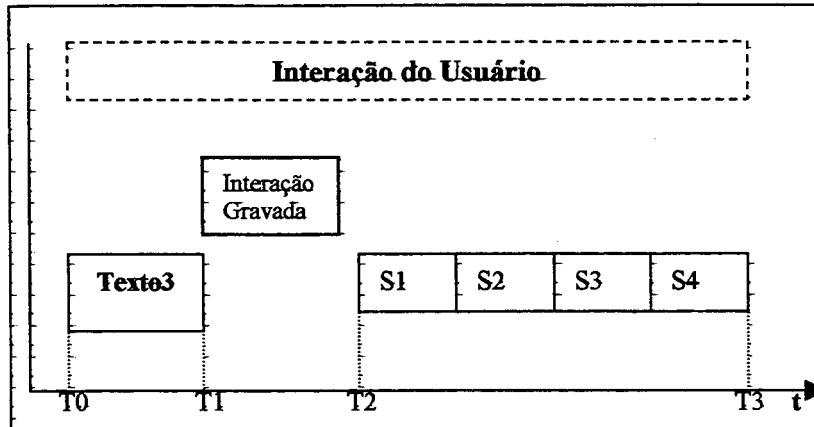


FIGURA 5.38 – Apresentação do Cenário Cena 2 do Módulo Síntese Econômica

O cenário (Cena2) ilustrado na Fig. 5.38 apresenta um texto (Texto3), seguido pelo *replay* de uma interação de usuário gravada e de uma seqüência de slides (S1-S4). Ao final do Slide4 a apresentação será encerrada. Adicionalmente, o usuário poderá interagir com o cenário e encerrar a apresentação.

### 5.3.2 – Especificação da Sincronização da Aplicação Multimídia

Afim de obtermos a sincronização desejada para a aplicação multimídia exemplificada, segundo a extensão proposta para o modelo RTR, especificaremos os cenários da aplicação como mostrado abaixo na Fig. 5.39.

```

OBTR Class Aplicacao{
    void Audio (id-audio,...)
    {
        ...
    }
    void Animacao(id-animacao,...)
    {
        ...
    }
    void Video(id-video,...)
    {

```

```

...
}

void Texto(id-texto,...)
{
...
}

void InteracaoGravada(id-igravada,...)
{
...
}

void slide(id-slide,...)
{
...
}

void Interacao(id-interacao, ...)
{
    // Este método especifica as possíveis interação do usuário nos cenários

    // Verifica se o botão Avançar foi pressionado
    IF (botao1==true)
    {
        // Interrompe apresentação do cenário atual
        // Apresenta Cena2Turismo, seguida da Cena3
        before(ApresentaCena2Turismo, ti, tf, ApresentaCena3Turismo, ti, tf,
        TratadorExcecao1);
    }
    Else
    // Verifica se o usuário selecionou o próximo módulo
    If (botao3==true)
    {
        Interrompe apresentação do cenário atual
        before(Apresenta Cena1SinteseEconomica, ti,
        tf, ApresentaCena2SinteseEconomica,ti,tf,
        TrafadorExcecao2);
    }
    Else

// Verifica se o usuário selecionou o botão finalizar
    If (botao2==true)
    {
        break;
    }
}

void Cena1Turismo(..)

// Sincroniza Cena1 Turismo
{
    before(while(audio(audio1), ti_audio, tf_audio, video(video1), ti_video, tf_video,
    TratadorExcecao2),

```

```

while(animacao(animacao1), ti_animacao, tf_animacao, audio(audio2), ti_audio, tf_audio,
TratadorExcecao3), TratadorExcecao1);
}
void Cena2Turismo(...)
// Sincroniza Cena2 Turismo
{
before(while(texto(texto1), ti_texto, tf_texto, audio(audio3), ti_audio, tf_audio, TratadorExcecao2),
imagem(imagem1), ti_imagem, tf_imagem, TratadorExcecao1);
}
void Cena3Turismo (...)
//Sincroniza Cena3 Turismo
{
before(while(animacao(animacao2), ti_animacao, tf_animacao(while(video(video2), ti_video,
tf_video, audio(audio4), ti_audio, tf_audio, TratadorExcecao3), TratadorExcecao2),
while(imagem(imagem2),
ti_imagem, tf_imagem, audio(audio5), ti_audio, tf_audio, TratadorExcecao4), TratadorExcecao1);
}
void Cena1SinteseEconomica(...)
// Sincroniza Cena1 Síntese Econômica
{
startin(while(texto(texto2), ti_texto, tf_texto, audio(audio6), ti_audio, tf_audio,
TratadorExcecao2),
imagem(imagem3), ti_imagem, tf_imagem, TratadorExcecao1);
}
}
void Cena2SinteseEconomica(...)
//Sincroniza Cena2 Síntese Econômica
{
before(texto(texto3), ti_texto, tf_texto, before(interacaogravada(igravada), ti_interacaogravada,
tf_interacaogravada, before(slide(S1), ti_slide, tf_slide, TratadorExcecao3), TratadorExcecao2),
before(slide(S2), ti_slide, tf_slide, before(slide(S3), ti_slide, tf_slide, slide(S4), ti_slide,
tf_slide, TratadorExcecao4, TratadorExcecao5), TratadorExcecao1);
}
}
void ApresentaCena1Turismo (...)
// Cena1Turismo é apresentada
{
// Display botões da Cena1Turismo
while(Interacao, ti_interacao, tf_interacao, Cena1Turismo, ti_Cena1Turismo, tf_Cena1Turismo,
TratadorExcecao1);
}
void ApresentaCena2Turismo (...)
// Cena2Turismo é apresentada
{
// Display botoes da Cena2Turismo
while(Interacao, ti_interacao, tf_interacao, Cena2Turismo, ti_Cena2Turismo, tf_Cena2Turismo,
TratadorExcecao1);
}
void ApresentaCena3Turismo (...)
// Cena3Turismo é apresentada
{
// Display botoes da Cena3Turismo
while(Interacao, ti_interacao, tf_interacao, Cena3Turismo, ti_Cena3Turismo, tf_Cena3Turismo,

```



```

    TratadorExcecao1);
}
void Apresenta Cena1SinteseEconomica (...)
// Cena1SinteseEconomica é apresentada
{
// Display botoes da Cena1SinteseEconomica
while(Interacao, ti_interacao, tf_interacao, Cena1SinteseEconomica, ti_Cena1SinteseEconomica,
tf_Cena1SinteseEconomica, TratadorExcecao1);
} void Apresenta Cena2SinteseEconomica (...)
// Cena2SinteseEconomica é apresentada
{
// Display botoes da Cena2SinteseEconomica
while(Interacao, ti_interacao, tf_interacao, Cena2SinteseEconomica, ti_Cena2SinteseEconomica,
tf_Cena1SinteseEconomica, TratadorExcecao1);
}
void Main()
{
...
// Cria janela de apresentação
ApresentaCena1Turismo (...);
// Fecha janela de apresentação
}
}

```

FIGURA 5.39 – Especificação da Sincronização Baseado na Nova Proposta para o Modelo RTR

O exemplo de aplicação apresentado, demonstrou como a utilização da reflexão computacional flexibiliza a programação de aplicações multimídia, permitindo abstrair dos programadores aspectos não funcionais da aplicação, como a verificação da consistência da sincronização e semântica dos operadores, sendo estes tratado a nível meta pelos métodos que implementam tal funcionalidade. Adicionalmente, na extensão proposta para o modelo RTR, eventos assíncronos como interação do usuário ficam facilitados visto que, a interação deverá ser sincronizada como as demais mídias do cenário, podendo ser controlada similarmente pelos mesmos métodos para prover a sincronização.

## 5.4 – Conclusões

Neste capítulo, apresentamos uma extensão do Modelo RTR, proposto por FURTADO [FUR97], para especificação da sincronização em aplicações multimídia. Inicialmente a dinâmica de funcionamento do método “Sincroniza” desenvolvido para prover a semântica do operadores, e dos métodos que especificam os

operadores de intervalos ( responsáveis pela verificação da consistência da sincronização), foram apresentadas. Em seguida, a representação da especificação da sincronização quando os tempos de execução das mídias envolvidas nos cenários são desconhecidos ou irrelevantes, foi demonstrada e exemplificada com base na extensão proposta para o modelo RTR. Adicionalmente, também foi descrita a especificação da sincronização de uma aplicação multimídia segundo o paradigma da extensão proposta.

O controle da sincronização multimídia através da extensão proposta, possibilitou o aumento da expressividade e da potencialidade do modelo RTR na representação dessa classe de aplicação. Em particular, a utilização de métodos específicos que representem o comportamento dos operadores de intervalos, abstraindo a semântica de funcionamento dos operadores do programador da aplicação, facilitam a especificação dos requisitos de sincronização presentes na multimídia.

Além disso, dentre as vantagens da extensão proposta, destacamos sua flexibilidade no controle dos eventos assíncronos, tais como a interação do usuário, haja visto que esses eventos são sincronizados como as demais mídias do cenário, podendo ser controlados pelos métodos desenvolvidos visando fornecer a sincronização.

## Conclusões e Perspectivas

Neste trabalho foi proposto validar um modelo de programação para aplicações tempo real, denominado Modelo RTR (Reflexivo Tempo Real), visando identificar suas principais características e dinâmica de funcionamento na especificação da sincronização em aplicações multimídia. Com este intuito, um cenário multimídia foi especificado, buscando-se nesta etapa, identificar as potencialidades do modelo RTR a fim de satisfazer os requisitos necessários para sincronização de aplicações multimídia. Por fim, uma extensão do modelo RTR foi proposta buscando prover maior flexibilidade na especificação da sincronização dos operadores de intervalos, segundo o modelo proposto por [WAH94], aos programadores de aplicações multimídia.

Conforme apresentado no capítulo 2, a maioria das técnicas de representação de sincronização multimídia existentes prevê somente a apresentação simultânea das mídias de forma independente, sem considerar aspectos de sincronização entre elas [RAD95]. Usando o modelo RTR as relações temporais entre sub-unidades de uma mídia, podem ser representadas através do uso de restrições temporais básicas, como por exemplo para esta representação, pode-se utilizar uma restrição de periodicidade (*periodic*), onde cada período corresponde a um intervalo de apresentação, responsáveis pela apresentação (*display*) das sub-unidades que compõem cada mídia.

De maneira geral, em comparação aos outros modelos de especificação citados neste trabalho, o modelo RTR destaca-se nos seguintes aspectos:

- demonstra simplicidade e flexibilidade no controle de sincronização multimídia através de restrições temporais pré-definidas, tais como *Periodic*, *Aperiodic*, *ActivationInterval*. Além disso, o programador poderá declarar novos tipos de restrições temporais, definindo porém, para cada novo tipo de restrição temporal

introduzido a nível de objeto-base, uma implementação no meta-objeto gerenciador correspondente;

- métodos manipuladores de exceção poderão ser especificados, a fim de serem executados quando a apresentação de uma mídia não puder ser realizada dentro de seu *deadline* como por exemplo;
- por adotar uma arquitetura reflexiva, questões referentes ao controle da aplicação (concorrência, restrições, exceções, escalonamento e sincronização), são expressos através de meta-objetos, sendo as funcionalidades básicas da aplicação tratadas através de objetos-base, flexibilizando a programação da especificação da sincronização em aplicações multimídia;

Com base em algumas limitações encontradas no modelo RTR para especificação da sincronização em aplicações multimídia, concluímos pela proposição de uma extensão para abordar alguns aspectos relativos à sincronização multimídia não disponíveis na proposta original do modelo RTR.

Dentre as principais vantagens da extensão proposta para o Modelo RTR, destacamos sua flexibilidade, expressividade e sua efetividade na representação de quaisquer um dos operadores de intervalos, além disso, a combinação de um maior número de mídias e diferentes operadores, requisito essencial na especificação de cenários multimídia complexos, ficaram demonstrados. Essas vantagens são decorrentes dos seguintes aspectos:

- o método genérico (*Sincroniza*) desenvolvido, é capaz de prover a semântica de qualquer um dos operadores de intervalos, interagindo com o MOE e MOR, e analisando os intervalos de execução das mídias envolvidas;
- métodos que especificam os operadores de intervalos, verificando se os valores de **Tinicio** e **Tfim** fornecidos, são consistentes com a sincronização desejada, caso a premissa seja verdadeira o método *Sincroniza* é ativado a fim de prover a sincronização.
- As interações do usuário tem seu comportamento controlado pelos mesmos métodos (**sincroniza** e **operadores de intervalos**), haja visto que podem ser sincronizadas como as demais mídias do cenário.

Adicionalmente, além da representação da sincronização quando os tempos de execução das mídias envolvidas no cenário são conhecidos, foram demonstrados métodos que especificam os operadores de intervalos, quando tais tempos são desconhecidos (situações em que o importante é a sincronização propriamente dita, e não o momento em que ocorrerá). Neste caso, cada operador terá um método responsável por garantir a sincronização desejada.

Por outro lado, como limitação do modelo proposto destacamos que o uso do modelo proposto presume um conhecimento prévio do modelo RTR por parte dos programadores de aplicações multimídia.

Enfim, destacamos que os resultados obtidos foram bastante satisfatórios, uma vez que ficou demonstrado que a dinâmica de funcionamento do Modelo RTR pode contribuir na especificação da sincronização de aplicações multimídia, abrangendo vários aspectos relevantes a essa classe de aplicação. Além disso, os resultados obtidos neste trabalho de dissertação deram origem a duas publicações [KNI 00a] e [KNI 00b], sendo a primeira no principal evento da área de multimídia no Brasil. (, SbMídia'2000, VI Simpósio Brasileiro de Sistemas Multimídia e Hiperemídia)

Com base nos resultados obtidos, acreditamos que o trabalho deve ser continuado e destacamos às seguintes perspectivas de continuidade:

- Desenvolvimento a implementação de um protótipo de uma aplicação multimídia visando testar os mecanismos apresentados na extensão proposta para o Modelo RTR;
- Experimentação/análise de diferentes algoritmos de escalonamento, utilizando uma das linguagens reflexivas apresentadas no capítulo 3;
- Especificação/implementação de um ambiente gráfico (visual) para programação de cenários multimídia, segundo a filosofia da extensão do Modelo RTR proposta nesta dissertação.

**REFERÊNCIAS BIBLIOGRÁFICAS**

- [ACK94] ACKERMANN, P. **Direct Manipulation of Temporal Structures in a Multimedia Application Framework**, ACM International Conference on Multimedia, São Francisco, California, 1994.
- [AFN89] AFNOR Expert group, **Multimedia Synchronization; Definitions and Model, Input Contribution on Time Variant Aspects and Synchronization in ODA-Extensions**, ISO IEC JTC 1/SC 18/WG3, Feb. 1989.
- [AIM93] AimTech Corporation. **IconAuthor User Manual for OSF/Motif**. 1993.
- [ALL83] ALLEN, J.F. **Maintaining Knowledge about Temporal Intervals**, communications ACM, november 1993, vol 26, n.11, pp 832-834.
- [APP91] APPLE Computer Inc. **QuickTime Developer's GUIDE**. Developer technical publications, 1991.
- [APP94] **Multimedia Demystified – A Guide to the World of Multimedia from Apple Computer**, Inc. Random House/NewMedia Series, 1994.
- [AUT89] INC. **Authoware Professional Manual**. Redwood City, CA, 1989.
- [BLA92] BLAKOWSKI, G; HÜBEL, Jeans. J; MÜHLHAÜSER, M. **Tool Support for the Synchronization and Presentation of Distributed Multimedia**. Computer Communication, december 1992. vol. 15 n° 10. pp. 611-618.
- [BLA96] BLAKOWSKI, G; STEINMETS, R. **A Media Synchronization Survey: Reference Model, Specification, and Case Studies**. IEEE Journal in Selected Areas in Communications, vol. 14, n° 1, pp. 5-35, January 1996.
- [BOT95] BOTAFOGO, R; Mossé D. **The MORENA Model for Hypermedia Authoring and Browsing**. In proc. Of the IEEE Int. Conf. On Multimedia Computing and Systems, Washington, DC, pp. 42-49, May 1995.
- [BUF94] BUFORD, J.F. **Multimedia Systems**, ACM Press. Siggraph Series, New York, 1994.
- [CHI93a] CHIBA, S. **Open C++ Programmer's Guide Version 1.2**, Technical Report 93-3, Department of Information Science, University of Tokio, 1993.

- [CHI95] CHIBA, S. **A Metaobject Protocol for C++**, in OOPSLA'95 Proceedings, 1995.
- [CHI96] CHIBA, S. **Open C++ programmer's Guide Version 2.0**. Palo Alto: PARC Xerox, 1996.
- [DIA93] DIAZ, M.; SÉNAC, P. **Time Streams Petri Nets, a Model for Multimedia Streams Synchronization**. In proc. Of the First International Conference on MultiMedia Modeling, november 1993, vol 1, pp. 257-273.
- [DRA91] DRAPEU, G.D; H. GREENGLED. **MAestro – A Distributed Multimedia Authoring Enviroment**. In proc. of the USENIX Summer'91 Conference Nashville TN, 1991, pp. 315-328.
- [EYE92] EYES 2.0. **Reference Manual**. Center for Productivity Enhancemment. University of Massachusetts, Lowell. 1992.
- [FAB95] FABRE, I.; NICOMETTE, V.; PÉRENNOU, T et al. **Implementing Fault Tolerant Applications using Reflective Object-Oriented Programming**. Proceedings of the 25 th IEEE International Symposium on Fault-Tolerant Computing, Pasadena (CA), June 1995.
- [FER95] FERREIRA, J. D. **Multimídia para Programadores e Analistas**. IBPI, Instituto Brasileiro de Pesquisa em Informática, Rio de Janeiro: Infobok, 1995, pp. 7.
- [FAQ97] **MULTIMEDIA Authoring Systems FAQ**. Version 2.13. <http://www.tiac.net/users/jalisglar/MMASFAQ.HTML>. June 1997.
- [FOO93] FOOT, B. **Architectural Balkanization in the Post-Linguistic Era**. In: OOPSLA'93 Workshop On Reflection And Meta-Level Architectures, 1993. pp. 1-9
- [FLU95] FLUCKIGER, F. **Understanding Networked Multimedia: Application and Technology**. Prentice Hall International (UK) Limited, 1995.
- [FRA95] FRAGA, J; FARINES, J.M; FURTADO, O, J.V. et. al. **Programming Model for Real-Time Applications ind Open Distributed Systems**. 5 th Workshop on Future Trends in Distributed Computing Systems, Cheju Island, Republic of

- Korea, august 1995.
- [FUR97] FURTADO, O. J. V. **RTR – Uma Abordagem Reflexiva para Programação de Aplicações Tempo Real**. Tese de Doutorado, UFSC, Florianópolis, novembro 1997.
- [GIB91] GIBBS, S. **Composite Multimedia and Active Objects**. In proc. of THE OOPSLS'91, PHOENIX, Arizona, October 1991.
- [GIN95] GINIGE, A; LOWE D.B; ROBERTSON J. **Hypermedia Authoring**. IEEE Multimedia. vol. 2 nº 4, pp. 24-35, 1995.
- [GOL83] GOLDENBERG, A. **The Influence of Object-oriented Language on the Programming Environments**. In: ACM. COMPUTER SCIENCE CONFERENCE, 1983, Orlando, Florida, USA. Proceedings. New York: ACM, 1983. p. 35-54
- [GOM97b] GOL, M; KLEINÖDER, J. **METAXA AND THE FUTURE OF REFLECTION**, University of Erlangen – Nürnberg, 1997.
- [HAI96] HAINDL, M. **Multimedia Synchronization**. Computer Science/Department of Interactive Systems. IEEE Journal on Selected Areas in Communication. Vol. 14 nº 1, pp. 73-83, 1996.
- [HAM72] HAMBLIN, C.L. **Instants and Intervals**. In proc. Of 1st Conf. Int. Soc. For the Study of time, J.T. Fraser et1. (Eds.), Springer –Verlag, , New York, 1972, pp. 324-331.
- [HAR94] HARDMAN, L; BULTERMAN, D. C. A; ROSSUM, G.V. **The Amsterdam Hypermedia Model: Adding Time, Structure and Context to Hypertext**. Communication of the ACM, vol. 37 nº 29, pp. 50-62. February 1994.
- [HAR95] HARDMAN, L; BULTERMAN, D. C. A. **Authoring Support for Durdle Interactive Multimedia Presentations**. STAR Report in Eurographics'95.
- [HER94] HERMAN, I; Reynolds G.J. **MADE: A Multimedia Application Development Environment**. In proc. Of the Int. Conf. on Multimedia Communication and Systems, 1994, pp. 184-193.
- [HIR95] HIRZALLA, N; FALCHUK, B; A. Karmouch. **A Temporal Model for**



- Interactive Multimedia Scenarios**. IEEE Multimedia, Fall 1995, pp. 24-31.
- [HUD93] HUDSON, S.E; HSL, C. H. **A Framework for Low Level Analysis and Synthesis to Support High Level Authoring of Multimedia Documents**. Graphics Visualization and Usability Center Technical Report Gvu – 93-14. Georgia Institute of Technology, 1993.
- [HON94] HONDA, Y; TOKORO, M. **Reflection and Time-Dependent Computing: Experiences with the R2 Architecture**, Technical Report, SONY C. S. Laboratory Inc. Tokio, julho 1994.
- [IBM90] IBM Corporation, **Audio Visual Connection User's Guide and Authoring Language Reference**, Version 1.05, IBM Form, S15F-7134-02, august 1990.
- [ISO 8613] ISO 8613 Information Processing, **Text and Office Systems – Office Document Architecture (ODA) and Interchange Format (ODIF)**, 1998.
- [KIC92] KICZALES, G. **Towards a New Model of Abstraction in Software Engineering**. International Workshop On New Models For Software Architecture/Reflection And Meta-Level Architectures, TOKYO, Japan, 1992, Proceedings...: [s.l.:s.n], pp. 1-11.
- [KLA90] KLAS, W; NEUHOLD, E.J; SCHREFL, M. **Using an Object-Oriented Approach to Model Multimedia Data**, Computer Communication, 1990, vol. 13 n° 4, pp. 204-216.
- [KNI00a] KNISS, J; FURTADO, O. J. V; WILLRICH, R. **Programando Cenários Multimídia Utilizando Reflexão Computacional**. VI Simpósio Brasileiro de Sistemas Multimídia e Hipermídia, Natal, RN, 2000. pp 231-243.
- [KNI00b] KNISS, J; FURTADO, O. J. V. **Modelo Reflexivo para Programação de Aplicações Multimídia**, I Simpósio Catarinense de Computação, Itajaí, SC, 2000. pp 625-636.
- [LIS97] LISBOA, M. L. B. **Reflexão Computacional no Modelo de Objetos**, agosto 1997.
- [LIT90] LITTLE, T. D. C; GHAFOR, A. **Synchronization and Store Models for Multimedia Objects**, IEEE Journal on Selected Areas in Communication, april 1990, vol. 8 n°3.

- [MAD95] MADESON, O. L. **Open ISSUES in Object-Oriented Programming Software Practice and Experience**, New York, december 1995, vol. 25, nº 54, pp. 3-43.
- [MAE87] MAES, P. **Concepts and Experiments in Computacional Reflection**. Proceedings of OOPSLA'87, october 1987, pp. 147-155.
- [MAR97] MARTINS, R.F. **E-LOTOS – Aprimoramentos da Linguagem LOTOS**, Anais do II Workshop do projeto DAMD – ProTeM-CC, Florianópolis, agosto, 1997.
- [MIT96] MITCHELL, S. **TAO – A Model for the Integration of Concurrency and Synchronization in Object-Oriented Programming**, PhD. Thesis, University of York, UK, march, 1996.
- [MIT97] MITCHELL, S; BURNS, E; WELLINGS, A. J. **Developing a Real-Time Metaobject Protocol**, WORDS'97, Newport Beach, California, USA, february 1997, pp. 7-7.
- [MUR89] MURATA, T. **Petri Nets: Properties, Analysis and Application**. IEEE, april, 1989, vol. 77 nº 4, pp. 541-580.
- [NEW91] NEWCOMB, S. **The Hytime Hypermedia/Time-based Document Structuring Language**. Communications of the ACM, 1991, vol. 34 nº 11, pp. 159-166.
- [PAU96] PAULO, F. B. **Extensões HMBS para Especificação de Apresentações Hipermédia**. In: Anais do X-SBES, outubro 1996, pp. 241-256.
- [PER97] PERRY, P. **Guia de Desenvolvimento de Multimídia**. Berkeley, 1995.
- [RAD95] RADA, R. **Interactive Media**. England: Springer-Verlag, 1995.
- [SAM98] SAMPAIO, P.N.M. **Uma Metodologia para Implementação em MHEG-5 de Aplicações Multimedia Especificadas em E-LOTOS**, Dissertação de Mestrado Apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPG-CC) – UFSCar, fevereiro, 1998.
- [SCH94] SCHLOSS, G. A. WYNBLATT. M.J. **Building Temporal Structures in a**

- Layered Multimedia Data Model.** In proc. ACM Multimedia'94, 1994, pp. 271-278.
- [SÉN95] SÉNAC, P.; WILLRICH, R.; SAQUI-SANNES, P. S. **Hierarchical Time Stream Petri Nets: A Model for Hypermedia Systems.** 16th. International Conference on Application and Theory of Petri Nets. In Application and Theory of Petri Nets 1995, Lecture Notes in Computer Science no. 935, G. De Michelis and M. Diaz (Eds.), Springer, pp. 451-470.
- [SHA93] SHACKELFORD, D.E; SMITH, J.B; SMITH, F.D. **The Architecture and Implementation of a Distributed Hypermedia Storage System.** Technical Report 93-013, Department of Computer Science, The University of North Carolina at Chapel Hill, 1993.
- [SHE90] SHEPHERD, D; SALMONY, M. **Extending OSI To Support Synchronization Required by Multimedia Applications,** Computer Comm, september 1990, vol. 13, pp. 399-406.
- [STA96] STANKOVIC, J. A. **Strategic Direction in Real-Time and embedded Systems.** ACM Computing Surveys, december, 1996, vol 28 n° 4, pp. 751-763.
- [STE94] STEEL, L. **Beyond Objects.** European Conference On Object-Oriented Programming, 8., 1994, Bologna, Italy. Proceedings... Berlin: Springer-Verlag, 1994. P.1-11 (Lecture Notes in Computer Science n. 821).
- [STO90] STOTT, P.D; FURATA, R. **Temporal Hyperprogramming.** Journal of Visual Languages and Computing, 1990 vol 1, pp. 237-253.
- [STR91] STROUSTRUP, B. **The C++ Programming Language.** 2nd ed., Reading: Addison Wesley, 1991.
- [SUN97] Sun Microsystems Inc. **JavaCore Reflection – API and Specification,** Mountain View, CA, september, 1996.
- [TSI91] TSICHRITZIS, D; GIBBS, S; DAMI, L. **Active Media in Object Composition.** Ed. Genève, Switzerland: Université de Genève, Centre Universitaire d' Informatique, 1991, pp. 115-132.
- [VAZ93] VAZIRGIANNIS, M; HATZOPOULOS, M. **A Script Based Approach for**

- Interactive Multimedia Application.** In proc. Of Multimedia Modelling'93, Singapore, november 1993, pp. 129-143.
- [WAHL94] WAHL, T, ROTHERNEL, K. **Representing Time in Multimedia Systems,** In Proc. Of International Conference Multimedia Computing and systems, 1994, pp. 538-543.
- [WIL96] WILLRICH, R. **Conception Formelle de Documents Hypermedias Portables.** Tese apresentada ao Laboratoire d'Analyse et d'Architecture des Systèmes du C.N.R.S. em vista de obtenção de título de Doucteur pela Universidade Paul Sabatier. Toulouse (França), setembro 1996.
- [WILL96a] WILLRICH, R; SÉNAC, P.S-S; Diaz, M. **A Formal Framework for the Specification, Analusys and Generation of Standardized Hypermedia Documents.** Third IEEE International Conference on Multimedia Computing and systems (ICMS'96), Japão, 1996, pp. 399-406.
- [WIL97] WILLRICH, R; SENAC, P. S-S. **Concepção Formal de Aplicações Multimedia Java** INE/CTC/UFSC. Florianópolis SC, Brasil.
- [WU97] WU, Z; SCHWIDERSKI, S. **Reflective Java: Making Java Even More Flexible.** FTP: Architecture Projects Management Limited. ([apm@ansa.co.uk](mailto:apm@ansa.co.uk)), Cambridge, UK, 1997.
- [WYN95] WYNBLAT, M. Y. **Position Statement Multimedia Synchronization.** In proc. Of the IEEE Workshop on Multimedia Synchronization (Sync'95). In. URL: <http://spiderman.bu.edu/sync95/sunc95.html> Tysons Corner, Virginia, 95.
- [YAN96] YANG, C.C; HUANG, J.H. **A Multimedia Synchronization Model and Its Implementation in Transport Protocols.** IEEE Journal on Selected Areas in Communications, vol. 14 n° 1, pp. 212-255.
- [YOK92] YOKOTE, Y. **The Apertos Reflective Operating Systems: The Concetps and its Implementation,** OOPSLA'92 Proceedings, 1992, pp. 414-434.
- [YO88] YOKOTE, Y. **The Apertos Reflective Operating Sytems: The Concept and its Implementation,** OOPSLA'92 Proceedings, 1992, pp. 414-434.
- [YON86] YONEZAWA, A. **Object-Oriented Concurrent Programing in ABCL/1.** SIGPLAN Notices, novembro 1986, vol. 21 n° 11.

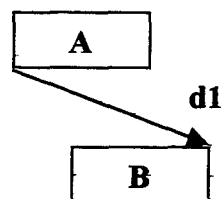
## ***APÊNDICE A***

## Apêndice A – Pseudo-Código dos Métodos que Verificam a Consistência e Sincronização para os Operadores de Intervalos

Como exposto no capítulo 5, métodos que implementam o comportamento dos operadores de intervalos, afim de verificar se os valores atribuídos aos atributos (**tinicio** e **tfim**) de cada mídia, estão de acordo com a sincronização multimídia desejada, podem ser implementados de acordo com a filosofia do modelo RTR, abstraindo do programador da aplicação os aspectos relativos a especificação da sincronização entre as mídias componentes de um cenário. A implementação desses operadores deverá estar disponível em bibliotecas para o usuário, sendo que a flexibilidade inerente ao modelo permite que o programador possa desenvolver novas bibliotecas de acordo com as necessidades específicas da aplicação. Neste apêndice, por questão de uniformidade, representaremos todos os métodos que representam a apresentação de mídias por “A() ou B()”.

### A.1 –Operador Beforeendof

- Representação Gráfica

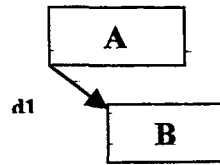


```

void beforeendof(tiA,tfA,tiB, tfB){
  // Verifica consistência da sincronização
  If (tfA>tiA) && (tiB<tfB) && (tiB>tiA) then{
  // Ativa método Sincroniza do MOG
  Sincroniza();
  }
  Else
  {
  // Os tempos (tinicio, tfim) das mídias não são consistentes com
  // a sincronização desejada.
  }
}
  
```

## A.2 –Operador Cobegin

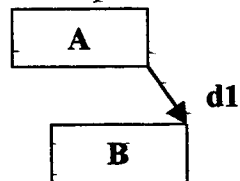
### Representação Gráfica



```
void cobegin (ti_A, tf_A, ti_B, tf_B){  
  // Verifica consistência da sincronização  
  If (tf_A > ti_A) && (ti_B >= ti_A) && (tf_B > ti_B) then{  
    // Ativa método Sincroniza do MOG  
    Sincroniza();  
  }  
  Else  
  {  
    // Os tempos (tinício, tfim) das mídias não são consistentes com  
    // a sincronização desejada.  
  }  
}
```

## A.3 –Operador Coend

### ▪ Representação Gráfica



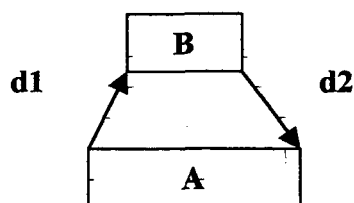
```

void coend (ti_A,tf_A,ti_B, tf_B){
// Verifica consistência da sincronização
If (tf_A>ti_A) && (ti_B <tf_B) && (tf_B>=tf_A) then{
// Ativa método Sincroniza do MOG
Sincroniza();
}
Else
{
// Os tempos (tínicio, tfim) das mídias não são consistentes com
// a sincronização desejada.
}
}

```

#### A.4 –Operador While

- Representação Gráfica



```

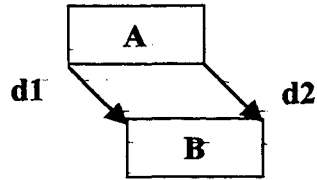
void while(ti_A,tf_A,ti_B, tf_B){
// Verifica consistência da sincronização
If (tf_A>ti_A) && (ti_B >=ti_A) && (tf_B<=tf_A) then{
// Ativa método Sincroniza do MOG
Sincroniza();
}
Else
{
// Os tempos (tínicio, tfim) das mídias não são consistentes com
// a sincronização desejada.
}
}

```

#### A.5 –Operador Delayed

- Representação Gráfica



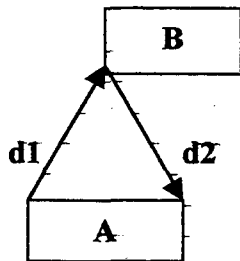


```

void delayed(tiA,tfA,tiB, tfB){
// Verifica consistência da sincronização
If (tfA>tiA) && (tiB >=tiA) && (tfB>=tfA) then{
// Ativa método Sincroniza do MOG
Sincroniza();
}
Else
{
// Os tempos (tinício, tfim) das mídias não são consistentes com
// a sincronização desejada.
}
}
}
    
```

**A.6 –Operador Startin**

- Representação Gráfica

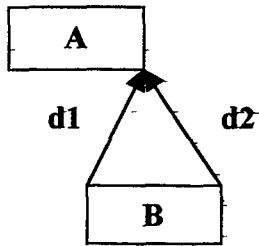


```

void startin(tiA,tfA,tiB, tfB){
// Verifica consistência da sincronização
If (tfA>tiA) && (tiB >=tiA) && (tfB>tiB) then{
// Ativa método Sincroniza do MOG
Sincroniza();
}
Else
{
// Os tempos (tinício, tfim) das mídias não são consistentes com
// a sincronização desejada.
}
}
}
    
```

## A.7 –Operador Endin

- Representação Gráfica



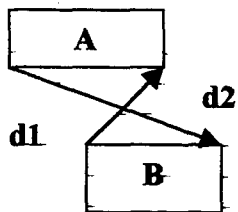
```

void endin(tiA,tfA,tiB, tfB){
// Verifica consistência da sincronização
If (tfA>tiA) && (tiB <=tfA) && (tfB>=tfA) then{
// Ativa método Sincroniza do MOG
Sincroniza();
}
Else
{
// Os tempos (inicio, fim) das mídias não são consistentes com
// a sincronização desejada.
}
}

```

## A.8 –Operador Cross

- Representação Gráfica



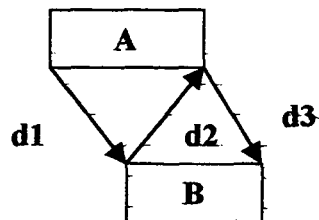
```

void cross(ti_A,tf_A,ti_B, tf_B){
// Verifica consistência da sincronização
If (tf_A>ti_A) && (ti_B <=tf_A) && (tf_B>ti_B) then{
// Ativa método Sincroniza do MOG
Sincroniza();
}
Else
{
// Os tempos (inicio, tfim) das mídias não são consistentes com
// a sincronização desejada.
}
}

```

## A.9 –Operador Overlaps

- Representação Gráfica



```

void overlaps(ti_A,tf_A,ti_B, tf_B){
// Verifica consistência da sincronização
If (tf_A>ti_A) && (ti_B >=ti_A) && (tf_B>ti_B) && (tf_A>=ti_B) && (tf_B>=tf_A)
then{
// Ativa método Sincroniza do MOG
Sincroniza();
}
Else
{
// Os tempos (inicio, tfim) das mídias não são consistentes com
// a sincronização desejada.
}
}

```