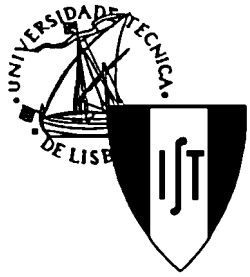

UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

UM MODELO PARA O DESENVOLVIMENTO DE APLICAÇÕES MULTIMÉDIA INTERACTIVAS DISTRIBUÍDAS

Paulo Rogério Barreiros d'Almeida Pereira

(Licenciado)

Dissertação para a obtenção do grau de
Mestre em Engenharia Electrotécnica e de Computadores

Junho de 1994

RESUMO

Apresenta-se um sistema de desenvolvimento de aplicações multimédia interactivas distribuídas.

Define-se um modelo de objectos multimédia em que as características dos objectos estão classificadas em estados, que combinam uma pequena parcela das características internas do objecto, as acções relacionadas com essa funcionalidade do objecto e os eventos que o objecto pode enviar para o exterior para anunciar mudanças no seu estado.

Descreve-se como todos os conceitos relacionados com os objectos multimédia são tipificados, sendo a informação relevante mantida numa base de dados, para que todas as aplicações e ferramentas no sistema tenham conhecimento das possibilidades disponíveis em cada momento, facilitando a expansibilidade do sistema, e permitindo verificações de consistência durante as várias fases do desenvolvimento de uma aplicação.

Define-se uma linguagem baseada na álgebra de processos CSP, utilizada para especificar aplicações multimédia, e descreve-se um editor gráfico destinado a gerar aplicações nessa linguagem para que um autor possa criar aplicações com facilidade e rapidez, utilizando os objectos disponíveis, sem necessitar dos conhecimentos técnicos relacionados com a linguagem.

Finalmente, descreve-se como um compilador converte a especificação da aplicação nessa linguagem para um gestor de sincronização, baseado numa máquina de estados para maximizar o desempenho, que comanda o desenrolar da aplicação multimédia.

ABSTRACT

A system for developing distributed interactive multimedia applications is presented.

A multimedia object model, which organizes the objects' characteristics in statuses, is defined. Each status combines a small part of the object's internal behaviour, the actions related with its functions and the events the object can send to the exterior to announce changes in its state.

All the concepts related with the multimedia objects are typed, with the relevant information being stored in a database. Therefore, all the applications and tools in the system know the possibilities available at any moment, allowing for system expansibility, and for consistency checks at each step of application development.

A language based on the CSP process algebra is defined, and used to specify multimedia applications. A graphical editor for the generation of applications in that language is described. This editor allows fast and easy application building by composing the available objects, and the author does not need the technical knowledge related with the language.

Finally, a compiler which converts the application specification in that language into a synchronization manager is described. The synchronization manager is based on a state machine in order to maximize the performance, and orchestrates the progress of the multimedia application.

PALAVRAS CHAVE

Sincronização multimédia
Composição multimédia
Sistemas multimédia distribuídos
Classificação em tipos
Objectos activos

KEYWORDS

Multimedia synchronization
Multimedia composition
Distributed multimedia systems
Type categorization
Active objects

AGRADECIMENTOS

A realização deste trabalho só se tornou possível graças ao apoio de professores, colegas, amigos e familiares.

Ao meu orientador científico, Prof. Paulo da Fonseca Pinto, agradeço a sua amizade, os seus ensinamentos, incansável apoio e permanente acompanhamento.

A amizade, a ajuda prestada e o interesse manifestado pelo meu trabalho por parte dos meus colegas do IST e do INESC[1] são também objecto da minha gratidão. Devo referir, em especial, o Prof. Vítor Vargas, o Prof. Nuno Guimarães, o Eng.º Luís Bernardo e o Eng.º Nuno Correia.

Por fim, devo agradecer à minha família o apoio e incentivo que sempre me deram ao longo de toda a minha vida de estudante.

Dedico, por isso, esta dissertação aos meus Pais.

Lisboa, Junho de 1994

Paulo Rogério Barreiros d'Almeida Pereira

[1] Instituto de Engenharia de Sistemas e Computadores.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

NOTAÇÃO

Para facilitar a interpretação do texto, utilizou-se a seguinte notação:

- Conceitos importantes estão em **negrito**.
- Código de programas, nomes de funções, tipos de dados, ou outra informação relacionada com a programação está em `caracteres de máquina de escrever`.
- Expressões em línguas estrangeiras estão em *itálico*.

CAPÍTULO 1
INTRODUÇÃO

1.1 O PROBLEMA

O Homem comunica através da escrita desde cerca de 3000 a.C., embora antes disso já transmitisse ideias através de imagens, estando as mais antigas pinturas rupestres conhecidas datadas de cerca de 20000 a.C. O cinema, isto é, a projecção de cenas animadas ou em movimento, só foi inventado em 1895 pelos irmãos Lumière.

Nestes últimos cem anos a ciência e a tecnologia têm evoluído muito rapidamente, contribuindo para a popularização da comunicação audiovisual, e estimulando a investigação e o desenvolvimento de novos produtos nesta área.

Os desenvolvimentos recentes em capacidade de processamento, memorização e arquivo dos computadores, em ambientes gráficos e interacção com utilizadores, em redes de alta velocidade e técnicas de compressão dependentes do contexto, em equipamentos e normalização, apontam no sentido de uma cada vez maior utilização de computadores para processar e combinar diversos tipos de informação em novos tipos de aplicações.

A natureza isócrona dos novos tipos de dados, tais como áudio e vídeo, levanta novos problemas relacionados com tempo real. Além disso, a integração de vários tipos de dados necessita de formas de exprimir o paralelismo das várias actividades e a ordem pela qual os eventos se devem suceder. Estes problemas tornam-se mais complexos em sistemas distribuídos e também no caso de se permitirem interacções com utilizadores.

No resto desta secção apresentam-se alguns conceitos importantes para a compreensão do trabalho desenvolvido.

A palavra **multimédia** tem origem na contracção do prefixo multi, que exprime a ideia de muitos, com a palavra latina *media*, que é o plural de *medium*, e significa meios.

Multimédia refere-se à geração, representação, processamento, arquivo, e disseminação integradas de informação, processável por computador, expressa em meios, dependentes ou não do tempo, tais como dados, gráficos, desenhos, voz, áudio e vídeo [Steinmetz 90].

O termo multimédia é utilizado nesta dissertação de uma forma geral, podendo referir-se a meios simples (monomédia), hipertexto, hipermédia, ou mesmo meios mais complexos como gráficos animados ou realidade virtual. Hipertexto é texto com referências para outros textos no mesmo ou noutro documento, permitindo navegar pela informação disponível. Hipermédia é uma extensão do hipertexto para incluir outros meios, tais como imagem, som ou animação.

As palavras meio e multimédia têm muitos significados, pelo que deve evitar-se a sua utilização isolada [ISO 93b]. Aliás, a palavra multimédia é um adjectivo, pelo deve ser utilizada associada a um substantivo que providencie o contexto, por exemplo: serviço multimédia, apresentação multimédia, objecto multimédia.

Um **objecto multimédia**, no contexto desta dissertação, é uma entidade activa com um comportamento específico, que produz e apresenta dados de certos tipos. O comportamento pode ser expresso em termos das possibilidades de interacção com o exterior.

Os meios podem classificar-se em meios **dinâmicos** ou **contínuos**, e meios **estáticos**. Meios dinâmicos são aqueles nos quais o tempo determina a apresentação dos dados. Como exemplo, temos o som como uma sucessão de amostras, e o vídeo como uma sucessão de imagens. Por outro lado, designam-se meios estáticos aqueles em que não há evolução com o tempo, como é o caso de um texto ou um gráfico.

1.2 OBJECTIVOS

O objectivo deste trabalho é o estudo e implementação de um sistema de desenvolvimento de aplicações multimédia interactivas distribuídas baseado na arquitectura da figura 1.1.

O editor gráfico permite a um programador construir, de forma fácil, uma aplicação multimédia sabendo quais os objectos multimédia existentes no sistema. O editor produz numa linguagem textual, designada *script*, a descrição das interacções

entre os vários objectos, e com eventuais utilizadores. Um compilador recebe como entrada esta descrição em linguagem textual, e produz um programa, o gestor de sincronização, que não é mais do que uma máquina de estados que controla os objectos multimédia de forma a respeitar a composição criada no editor gráfico. Um gestor de tipos reúne informação sobre todos os tipos existentes no sistema, permitindo fazer verificações durante as várias fases do desenvolvimento de uma aplicação.

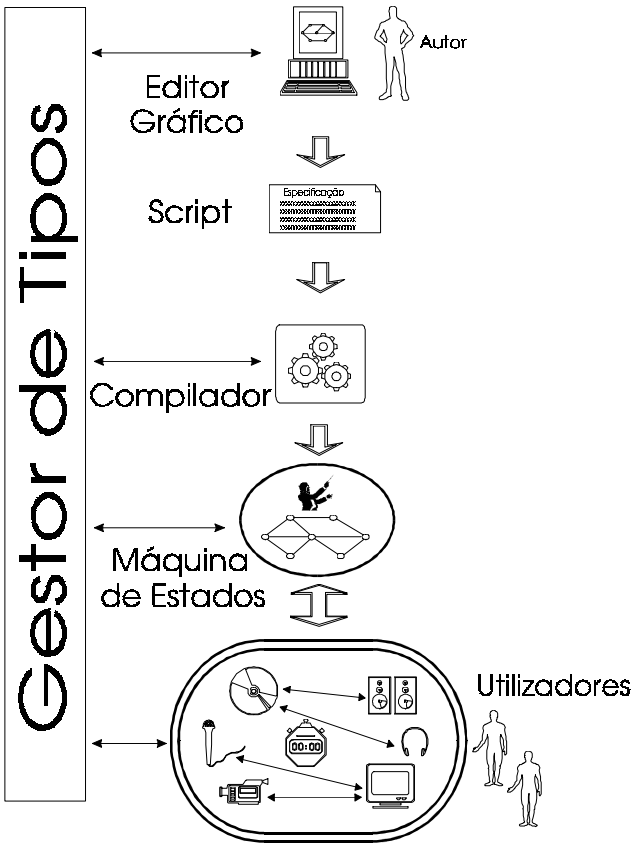

Figura 1.1: Arquitectura do sistema de desenvolvimento de aplicações multimédia interactivas distribuídas.

1.3 ESTRUTURA DA DISSERTAÇÃO

O segundo capítulo descreve modelos de composição de aplicações, modelos de sincronização e modelos de comunicação, de acordo com o trabalho de outros autores. Apresentam-se ainda alguns sistemas e normas relacionadas com o trabalho desenvolvido nesta dissertação. Para além de descrever trabalhos relacionados, tecem-se alguns comentários sobre as suas virtudes e limitações.

O terceiro capítulo descreve o modelo de objectos multimédia, a sua constituição e forma como eles interagem entre si e com o resto do sistema. Além disso, mostra-se como é possível adicionar funcionalidades ao modelo.

O quarto capítulo descreve como um gestor de nomes permite esconder a distribuição dos vários objectos na rede. Refere-se ainda como todos os conceitos relacionados com os objectos multimédia são tipificados, sendo a informação relevante mantida numa base de dados, para que todas as aplicações e ferramentas no sistema tenham conhecimento das possibilidades disponíveis em cada momento, facilitando a expansibilidade do sistema sem ter de modificar as aplicações existentes, e permitindo verificações de consistência durante as várias fases do desenvolvimento de uma aplicação.

O modelo apresentado no terceiro capítulo é uma evolução do proposto em [Pinto 93] e [Pinto$^+$ 93], tendo sido apresentado conjuntamente com o sistema de tipos e a arquitectura de desenvolvimento de aplicações multimédia em [Pinto$^+$ 94]:

> Paulo Pinto, Luís Bernardo, **Paulo Pereira**.
> A Constructive Type Schema for Distributed Multimedia Applications.
> *Proceedings of the 3rd International Conference on Broadband Islands, BRIS'94*, pág. 419-434, North-Holland, Junho 1994.

O quinto capítulo define uma linguagem baseada na álgebra de processos CSP, utilizada para descrever aplicações multimédia e apresenta um editor gráfico destinado a gerar aplicações nessa linguagem. Utilizando o editor gráfico, um autor pode criar aplicações com facilidade e rapidez, utilizando os objectos disponíveis, sem necessitar de ter os conhecimentos técnicos relacionados com uma linguagem baseada numa álgebra de processos.

O sexto capítulo descreve como foi implementado o sistema de desenvolvimento de aplicações multimédia interactivas distribuídas, apresenta um exemplo de aplicação e realça a forma como o compilador converte a especificação da aplicação para um gestor de sincronização e como este comanda o desenrolar da aplicação multimédia.

Finalmente, o último capítulo resume as conclusões deste trabalho, discutindo as possibilidades de evolução futura tanto a nível de investigação, como a nível da implementação.

CAPÍTULO 2
TRABALHO RELACIONADO

Este capítulo dá uma perspectiva do trabalho relacionado com sistemas multimédia que pode ser encontrado na literatura. Para tal, descrevem-se alguns modelos, sistemas e normas, que serviram como ponto de partida para o trabalho desenvolvido nesta dissertação.

2.1 MODELOS

Criar uma apresentação multimédia requere a especificação de quando os objectos são apresentados, e onde são apresentados, em termos de quais os dispositivos de entrada/saída a usar, por exemplo especificando a posição no écran, ou o canal áudio. No entanto, o autor não quer ter de especificar o que acontece segundo a segundo na apresentação, pelo que deve ter um sistema de autoria que lhe permita pensar em termos de alto nível, e que automaticamente gere as correspondentes temporizações e localizações da informação a baixo nível.

Um modelo é uma abstracção que representa parte da realidade, com o objectivo de melhor estudar um problema. Em multimédia, os modelos mais relevantes são: modelos de composição que descrevem como um autor constrói uma aplicação; modelos de sincronização que descrevem como as várias partes da aplicação coordenam a ordem do seu funcionamento no domínio do tempo; e finalmente, modelos de comunicação que descrevem como é trocada informação dentro do sistema. Apesar de estes três tipos de modelos estarem estreitamente relacionados entre si, serão analisados separadamente em cada uma das subsecções seguintes.

2.1.1 Modelos de Composição

Os modelos de composição descrevem como um autor constrói uma aplicação, sendo especialmente úteis os modelos que permitem a composição visual da aplicação. A

composição visual é feita construindo grafos que descrevem a aplicação. Os grafos são compostos de nós que representam os objectos a ligar, pontos de conexão que representam os tipos de conexões que um objecto suporta, e ligações que representam as conexões propriamente ditas. Para além disto, um modelo de composição atribui uma semântica aos grafos, e especifica regras para garantir que não são construídos grafos inválidos. Em [Mey+ 92] são propostos três modelos de composição para aplicações multimédia:

- Um modelo de **composição de fluxos de dados** (*dataflow*) descreve a aplicação como uma configuração de interligações de objectos (equipamento ou programas) e os fluxos de dados multimédia que fluem entre eles. Neste modelo, são definidas as classes ObjectoMultimédia, Porto e Conector, sendo possível através de um editor gráfico compor as interligações dos vários objectos.

- Um modelo de **composição de actividades** descreve o comportamento da aplicação em termos de combinações de actividades e eventos. Neste modelo, são definidas as classes Actividade, Evento e CondiçãoDisparo. Uma actividade produz e consome eventos. Os eventos podem ser de tipo **notificação** para uma actividade anunciar uma modificação do seu estado interno, ou de **aceitação** que são pontos de entrada para uma actividade modificar o seu comportamento temporal. Todas as actividades têm pelo menos os eventos de aceitação `begin` e `interrupt` para começar e interromper a actividade, e o evento de notificação `end` para anunciar o fim da actividade. Para especificar mudanças de estado na aplicação são utilizadas relações entre eventos, que associam um evento de notificação a um evento de aceitação, tais como: quando um botão é carregado numa actividade, causa o início de outra actividade.

- Um modelo de **composição temporal** descreve relações entre sequências temporais. Este modelo trata da especificação da apresentação, não em termos dos recursos envolvidos ou da forma como colaboram, mas em termos dos resultados pretendidos. A composição visual para este modelo corresponde à sincronização num eixo temporal [Gibbs 91], estando os objectos associados a um eixo que representa uma abstracção do tempo. Cada objecto converte o tempo global no seu tempo local através de uma transformação. Se houver uma discrepância maior que um determinado limite entre o tempo global e o tempo do objecto, o objecto tem de se ressincronizar com o tempo global. É possível, através de modificações na transformação do tempo global

para o tempo do objecto, variar a velocidade da apresentação, ou mesmo inverter o sentido da sua apresentação.

Estes três modelos são utilizados para a composição visual de aplicações multimédia, sendo definida a hierarquia de classes de nós, pontos de conexão e ligações, resumida na tabela 2.1, para a construção de grafos descrevendo a composição como o exemplificado na figura 2.1.

Classes genéricas de Composição Gráfica	Modelo de Composição de Fluxos de Dados	Modelo de Composição de Actividades	Modelo de Composição Temporal
Nó	ObjectoMultimédia	Actividade	PerformanceActivity ClockScheduler
PontoConexão	Porto	Evento	BeginEvent EndEvent
Ligação	Conector	CondiçãoDisparo	CondiçãoDisparo

Tabela 2.1: Classes usadas como Nós, Pontos de Conexão e Ligações na composição visual de aplicações multimédia.

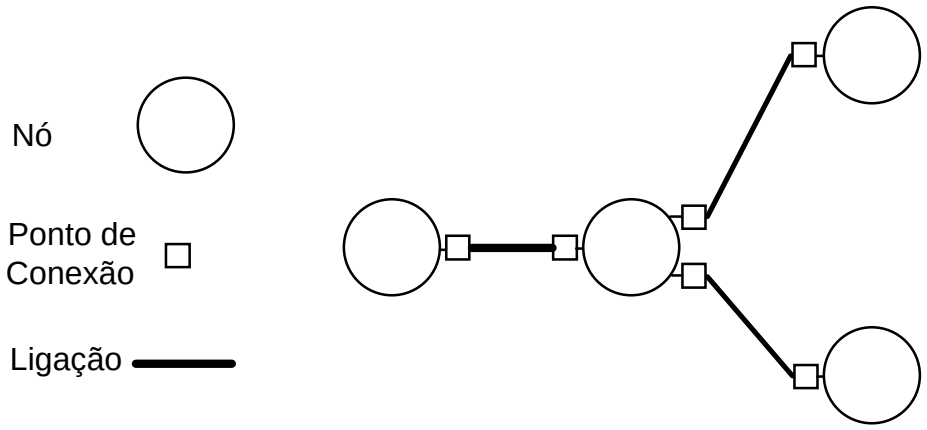

Figura 2.1: Exemplo de grafo descrevendo uma composição.

Verifica-se, no entanto, que o modelo de composição temporal é um caso particular do modelo de composição de actividades, sendo as classes indicadas na tabela 2.1 para a composição temporal, ou coincidentes, ou subclasses das classes indicadas para a composição de actividades. Embora permita descrever facilmente alguns tipos de aplicações simples, o modelo de composição temporal só permite utilizar objectos que tenham durações predefinidas e conduz a uma descrição muito rígida que não prevê interacções com utilizadores.

Em [Mey+ 93] é descrito um conjunto de componentes multimédia, e a implementação do modelo de composição de fluxos de dados, sendo exemplificado com um museu virtual.

2.1.2 Modelos de Sincronização

Os modelos de Sincronização descrevem como as várias partes da aplicação coordenam a ordem do seu funcionamento no domínio do tempo, sendo as principais abordagens encontradas na literatura caracterizadas por [Blakowski+ 92]:

- **Sincronização hierárquica**: Os objectos multimédia são vistos como árvores constituídas por nós que denotam uma apresentação sequencial ou simultânea das subárvores. A sincronização hierárquica é baseada em dois tipos de operações de sincronização: sincronização **sequencial** de acções e sincronização **paralela** de acções. As acções podem ser atómicas ou compostas. Uma acção atómica apresenta um objecto monomédia, aceita um comando do utilizador ou provoca um atraso. Uma acção composta é uma combinação de operações de sincronização e de acções atómicas. As estruturas hierárquicas são fáceis de processar e largamente divulgadas. No entanto, têm a restrição de cada acção só poder ser sincronizada no seu início e fim. Isto significa que, por exemplo, para colocar legendas num vídeo, tem de se dividi-lo em vários componentes a serem apresentados consecutivamente. Desta forma, não se permite considerar um objecto multimédia como uma unidade abstracta se for necessário sincronizá-lo entre o início e o fim da sua apresentação.

- **Sincronização baseada num eixo temporal**: Os objectos monomédia são ligados a um eixo temporal que representa uma abstracção do tempo real. Esta aproximação está directamente relacionada com o modelo de composição temporal, tendo as mesmas limitações já referidas. Além disso, este modelo não permite descrever facilmente não determinismo, como por exemplo, os decorrentes de interacções com utilizadores, e atrasos devidos à distribuição.

- **Sincronização em pontos de referência**: Os objectos monomédia são vistos como uma sequência de subunidades discretas apresentadas a intervalos de tempo constantes, por exemplo imagens de vídeo ou amostras áudio. A posição de uma subunidade num objecto é designada de ponto de referência. A sincronização entre objectos é definida associando subunidades de objectos diferentes que devem ser

apresentadas simultaneamente. Tal como a sincronização num eixo temporal, esta descrição permite sincronização em qualquer ponto de uma apresentação, e não apenas no início e fim de cada objecto. No entanto, tem a desvantagem, comparativamente com a sincronização hierárquica, de exigir mecanismos para detectar inconsistências.

Uma outra alternativa é recorrer a técnicas de especificação formal. Em [Little+ 90a] é proposta uma técnica para a especificação formal e modelação de composições multimédia respeitante a relações temporais entre objectos. O modelo é baseado na lógica de intervalos temporais e em Redes de Petri Temporizadas (*Timed Petri Nets*).

Para redes de Petri simples, o tempo que decorre desde uma transição ser disparável até disparar não está especificado, e é indeterminado, sendo o disparo de uma transição considerado um evento instantâneo. Nas redes de Petri temporizadas, este modelo é estendido, sendo atribuída uma duração a cada transição. Um outro modelo de redes de Petri temporizadas representa os processos por lugares, em vez de transições e a cada lugar na rede é atribuído um tempo de execução não negativo. Com este esquema, a noção de disparo instantâneo das transições é preservada, e o estado do sistema é claramente representado durante a execução, pois as marcas estão sempre em lugares, não em transições.

Este último modelo é preferido pelos autores por conduzir a representações mais compactas, e baseando-se nele sugerem um novo modelo: OCPN (*Object Composition Petri Nets*) que acrescenta às redes de Petri convencionais valores de duração e utilização de recursos nos lugares das redes.

Na figura 2.2 mostra-se um exemplo de uma apresentação multimédia modelada com OCPN. O tempo está indicado no eixo horizontal, e os recursos são descritos por uma dimensão, designada espaço, no eixo vertical indicando múltiplas actividades de apresentação. A representação temporal mostra a componente textual, e a imagem que persiste durante a apresentação, enquanto é apresentada uma sequência de imagens. Após um intervalo de tempo constante τ_9, começa um vídeo com som associado.

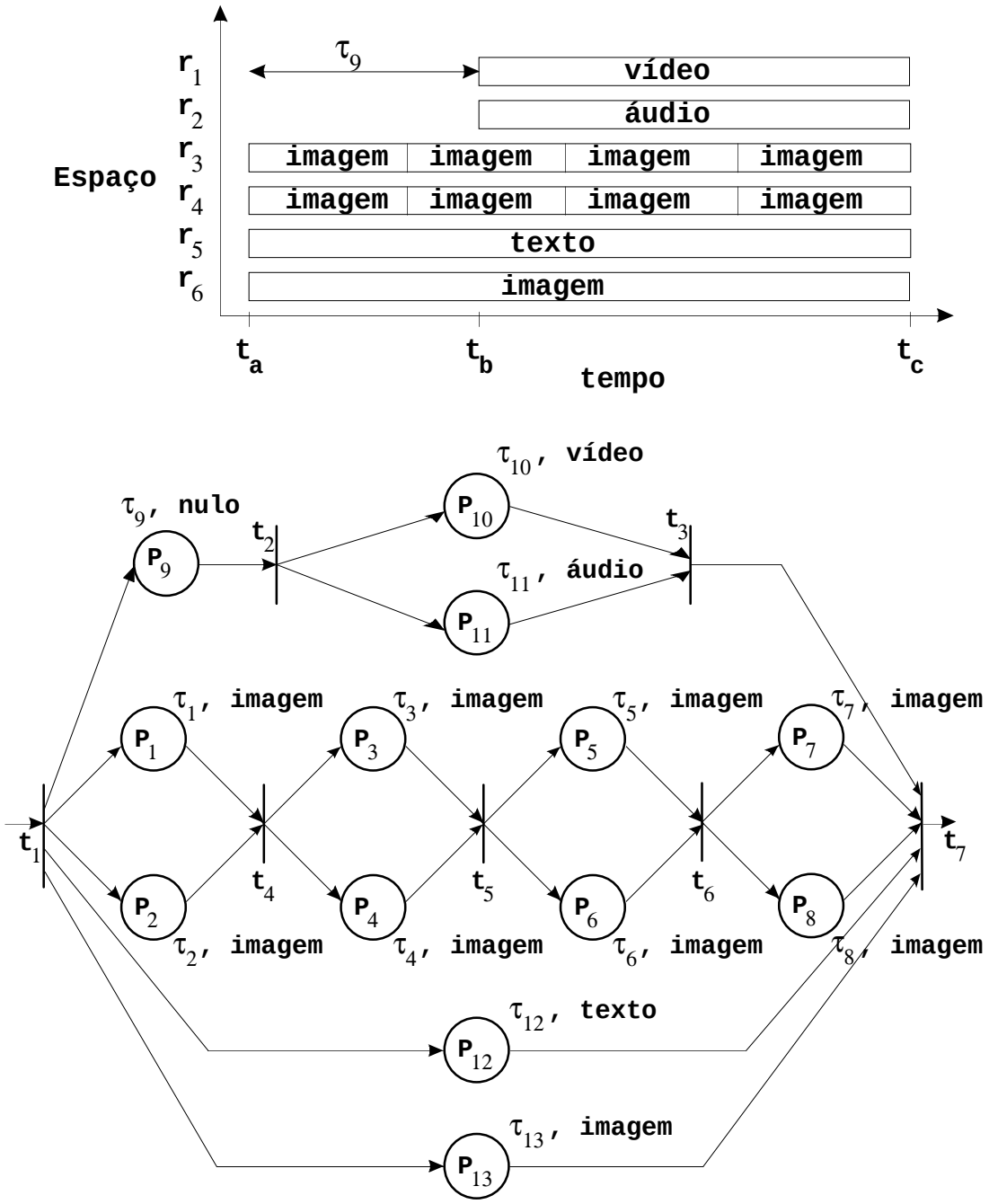

Figura 2.2: Apresentação multimédia modelada com OCPN.

Os autores demonstram haver um isomorfismo entre a lógica de intervalos temporais e as OCPN, sendo mostrada na segunda parte da figura 2.2 a OCPN correspondente à apresentação. Nesta rede, a cada lugar está associado um recurso de apresentação (dispositivo), e o tempo necessário para a apresentação de informação. As transições na rede indicam pontos de sincronização, e os lugares processamento.

Neste modelo não são tratadas considerações espaciais, tais como onde colocar múltiplas imagens no mesmo écran. Não são ainda descritas pelo modelo interacções

com utilizadores, manipulações dos objectos durante a apresentação, (tais como paragem da apresentação, inversão do sentido da apresentação ou modificação da geometria de imagens durante a apresentação), nem edição dos objectos (podendo criar inconsistências temporais), ou objectos que não mantenham relações temporais consistentes. Para além disso, não foi estudada a decomposição dos objectos multimédia e a sua distribuição.

Por outro lado, este modelo baseado em redes de Petri é adequado para modelar apresentações estáticas sem interacção com utilizadores. Tem ainda a vantagem de poder especificar requisitos de tempo real e permitir análises tais como modelação com processos de Markov.

Para resolver o problema de permitir interacções com utilizadores, é proposto em [Prabhakaran+ 93] um modelo de Redes de Petri Dinamicamente Temporizadas (*Dynamic Timed Petri Nets*). Este modelo consiste em permitir a interrupção da execução de OPCN e a modificação dos tempos de execução dos processos interrompidos, permitindo resolver os problemas de: saltar eventos, inversão do sentido da apresentação, pausa e recomeço da apresentação, e modificação do ritmo da apresentação.

Em [Qazi+ 93], para resolver o problema da comunicação e sincronização em sistemas distribuídos, é proposta uma extensão do modelo OCPN: XOCPN (*eXtended Object Composition Petri Nets*) onde, para prever os atrasos aleatórios causados pela comunicação e permitir a sincronização entre vários canais de dados, é introduzido um novo tipo de lugares na rede de Petri - os lugares de controlo - aos quais podem estar associadas acções de controlo da ligação, tais como criação de circuitos virtuais, definição da qualidade de serviço requerida, e sincronização entre canais, entre outros.

Os modelos de sincronização também podem ser classificados em níveis consoante se encontram mais próximos do equipamento ou do utilizador.

Em [Little+ 90b] são propostos três níveis de integração para fornecimento de serviços multimédia num sistema de informação multimédia distribuída (DMIS - *Distributed Multimedia Information System*):

- Nível de ***interface* com o utilizador** responsável pela apresentação dos dados em dispositivos físicos e interacção com o utilizador.

- Nível de **serviço**, responsável pelas funções de composição de objectos multimédia.
- Nível **físico**, responsável pela comunicação e arquivo dos dados multimédia.

Baseados nesta divisão em níveis do sistema, definem um modelo de sincronização de dois níveis:

- **Sincronização de objectos multimédia**, de alto nível, que pode ser modelada com OCPN, e que é utilizada no nível de interface com o utilizador e no nível de serviço.
- **Sincronização de fluxos de dados**, de baixo nível, utilizada no nível físico do sistema, e que corresponde à definição da qualidade de serviço requerida, com parâmetros tais como taxa de símbolos binários errados, taxa de pacotes errados, atraso médio e máximo, velocidade relativa da apresentação, entre outros. Será a este nível que, por exemplo, será modelada a sincronização entre áudio e vídeo que tivessem sido gravados simultaneamente de forma a acertar o som com o movimento dos lábios no vídeo.

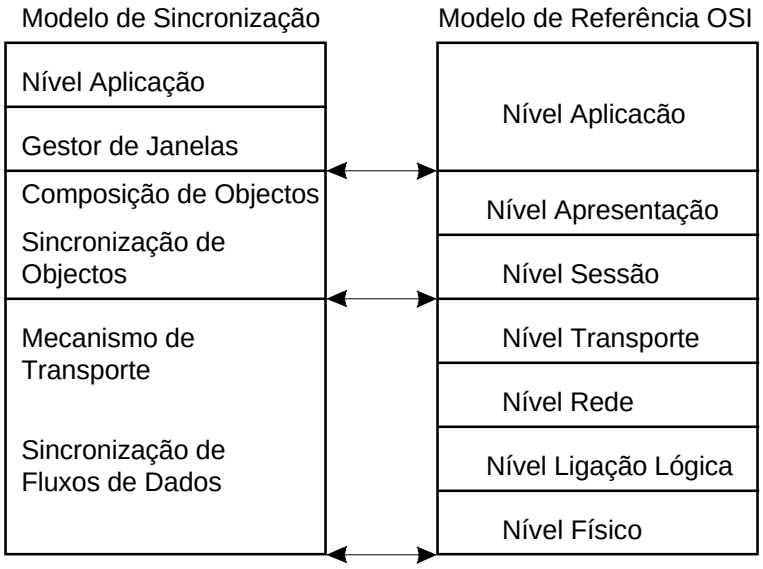

Figura 2.3: Comparação do modelo de sincronização com o modelo OSI.

Na figura 2.3 é apresentado um esquema comparativo do modelo de sincronização com o modelo de referência OSI para interligação de sistemas abertos [Rose 90]. Os níveis Aplicação e Gestor de Janelas correspondem ao anteriormente

referido nível de *interface* com o utilizador e são responsáveis pela interacção com os utilizadores. O nível de Composição de Objectos é responsável pela sincronização a nível dos objectos, e o nível de Transporte pela sincronização de fluxos de dados.

Em [Little+ 91] são descritos protocolos para os dois níveis de sincronização: o ASP (*Application Synchronization Protocol*) para o nível de sincronização de objectos e o NSP (*Network Synchronization Protocol*) para a sincronização de fluxos de dados. O ASP recebe como entrada uma estrutura multimédia complexa e produz um conjunto de fluxos de dados sincronizados prontos para apresentação. Para isso calcula as temporizações a exigir nas ligações que pede ao NSP. O NSP estabelece e mantém ligações com as características de sincronização especificadas.

2.1.3 Modelos de Comunicação

Quando se fala de computação distribuída, a palavra distribuída significa espalhada no espaço, isto é, em vários locais. No entanto, um dos problemas da computação distribuída é fornecer aos utilizadores uma visão não distribuída do sistema. Os modelos de computação geralmente considerados distribuídos são modelos de processos nos quais a actividade computacional é representada como a execução concorrente de processos sequenciais. Os modelos de processos que mais obviamente são distribuídos são modelos em que os processos comunicam entre si por troca de mensagens. Um canal é uma abstracção de uma rede de comunicação física que providencia um caminho para a comunicação entre processos.

Há vários métodos de comunicação entre processos [Andrews 91]:

- **Passagem de mensagens assíncrona**, no qual os canais têm, conceptualmente, capacidade ilimitada, pelo que o envio de uma mensagem é não bloqueante.
- **Passagem de mensagens síncrona**, em que um processo quando envia uma mensagem espera até o outro processo estar pronto para a receber, não sendo necessária qualquer capacidade de armazenamento de mensagens. Este modelo foi introduzido por Hoare na linguagem CSP (*Communicating Sequential Processes*) [Hoare 85] em que os comandos de entrada e saída para a troca de mensagens são executados sincronizadamente. Neste modelo, a comunicação e a sincronização estão interligadas, sendo a troca de uma mensagem um ponto de sincronização entre dois processos.

- **Passagem de mensagens com memorização**, no qual o canal tem capacidade limitada, havendo atraso no envio de mensagens só quando o canal está cheio.
- **Comunicação regenerativa** é semelhante à passagem de mensagens assíncrona, mas há um único canal partilhado por todos os processos.
- **Chamada de procedimentos remotos** ou ***rendez-vous***, no qual há um módulo que exporta operações que podem ser invocadas remotamente. A invocação do procedimento corresponde a uma comunicação de dois sentidos: o processo que invoca o procedimento envia os parâmetros do procedimento e fica à espera de ser servido e de receber de volta os resultados. Os nomes diferem consoante é, ou não, criado um novo processo para a invocação do procedimento.

Estas cinco aproximações são equivalentes, pois um programa escrito numa notação pode também ser escrito em qualquer das outras. No entanto, algumas são mais adequadas para certo tipo de problemas, conduzindo a diferentes desempenhos.

Há ainda modelos de comunicação por **variáveis globais partilhadas** que podem ser lidas e escritas por todos os processos, mas não são normalmente considerados modelos distribuídos porque a falha do processo que tem as variáveis bloqueia todos os outros processos.

Havendo comunicação entre processos num sistema distribuído, tem de se prever o que acontece no caso de haver falhas. Estas podem classificar-se em [Lamport+ 90]:

- **Falhas de comunicação**, assumindo-se habitualmente que resulta apenas na perda de mensagens.
- **Falhas de processos**, quando algum processo não funciona, ou funciona mal. Os modelos mais habituais são os que admitem apenas falhas de paragem - quando um processo avariado não faz nada; ou falhas de omissão - quando um processo não envia alguma mensagem.

Do ponto de vista das características da comunicação disponível, podemos ter os seguintes tipos de redes comutadas [Stallings 88]:

- **Comutação de circuitos**, caso em que sendo estabelecida ligação, fica-se com um circuito físico entre a origem e o destino, podendo-se transmitir dados a um ritmo fixo, a capacidade da linha, com um atraso devido apenas aos tempos de propagação.

- **Comutação de mensagens**, caso que se podem enviar blocos de dados de tamanho arbitrário. Não necessita de estabelecimento de ligação, mas os atrasos de transmissão podem ser grandes, pois a mensagem tem de ser toda recebida num nó antes de passar ao próximo.
- **Comutação de pacotes**, caso em que se podem enviar blocos de dimensão limitada. Neste caso, há duas variantes: **circuitos virtuais**, exigindo estabelecimento de ligação, caso em que se garante que os pacotes chegam ao destino por ordem, sem perdas ou duplicados; e **datagramas**, em que não há a noção de ligação e não se garante que os pacotes cheguem ao destino, ou cheguem por ordem. Os atrasos na transmissão de dados serão mais próximos dos obtidos com comutação de circuitos.

Existem ainda redes de **difusão**, caso em que não havendo comutação, quando se enviam dados, todos os nós da rede os podem receber - disponível por exemplo em redes locais, redes via satélite ou redes por rádio.

Factores críticos para a comunicação multimédia são os atrasos na comunicação, e as possibilidades de erros na comunicação. A menos que se use sempre comutação de circuitos, que corresponderá a um preço mais elevado, existe uma contradição entre recuperação de erros na comunicação e os atrasos que ela causa. Enquanto para transmissão de voz ou vídeo, pretende-se ter poucos atrasos, pois seriam notados no destinatário, mas pode-se tolerar a perda de uma fracção de segundo de som ou de uma imagem no vídeo, no caso de não se utilizar compressão. No caso de se transmitir informação de controlo, por exemplo comandos para iniciar a transmissão de som, ou indicação que o trecho de som terminou, já não se pode ter perdas de dados, pois poderiam conduzir ao não funcionamento das aplicações. Daqui se conclui a necessidade de separar os dados de controlo dos outros dados, de modo a permitir a utilização de circuitos virtuais com qualidade de serviço diferente para cada caso.

Retornando ao assunto das características das redes, hoje em dia caminha-se para a integração numa mesma rede dos vários tipos de serviços que no passado eram oferecidos por redes independentes.

Um exemplo disso é a futura Rede Digital com Integração de Serviços de Banda Larga, RDIS-BL (*BISDN - Broadband Integrated Services Digital Network*) [Nunes+ 92], que, tudo indica, utilizará um modo de transferência de informação

conhecido como Modo de Transferência Assíncrono (*ATM - Asynchronous Transfer Mode*) que permite a integração dos modos de comutação de circuitos e comutação de pacotes, sendo capaz de se adaptar a diferentes ritmos de comunicação utilizando de forma eficiente os recursos disponíveis na rede.

Uma Camada de Adaptação ATM (*AAL - ATM Adaptation Layer*) tem várias classes, indicadas na tabela 2.2, consoante os serviços requeridos.

Assim, verifica-se haver uma grande adequação entre os serviços fornecidos por RDIS-BL e os requeridos para aplicações multimédia.

	Classe A	Classe B	Classe C	Classe D
Relação Temporal	Existente		Não Existente	
Ritmo	Constante	Variável		
Modo de Conexão	Com Conexão			Sem Conexão
Exemplos de Aplicação	Vídeo e áudio de ritmo constante	Vídeo e áudio de ritmo variável	Transferência de ficheiros	Mensagens "sem conexão"

Tabela 2.2: Classificação dos serviços para AAL.

2.2 SISTEMAS

2.2.1 Brama

A BRAMA [Brama 94] (*Build, Run & Animate Multimedia Applications*) é uma ferramenta (figura 2.4) que permite ao utilizador construir apresentações multimédia, sem necessidade de programação, enquadrando-se no que se designa actualmente por sistemas de autoria multimédia. O utilizador pode seleccionar um conjunto de objectos (janelas e botões) e associar a esses objectos toda a informação pretendida: som, imagem estática, ou vídeo e texto. A ferramenta BRAMA permite ainda a adaptação a diferentes tipos de plataformas e a interligação com sistemas complementares, tais como bases de dados de informação administrativa ou comercial. Por exemplo, permite a invocação de

programas externos para executar operações específicas em determinado ponto da apresentação, tais como reserva de lugares para espectáculos.

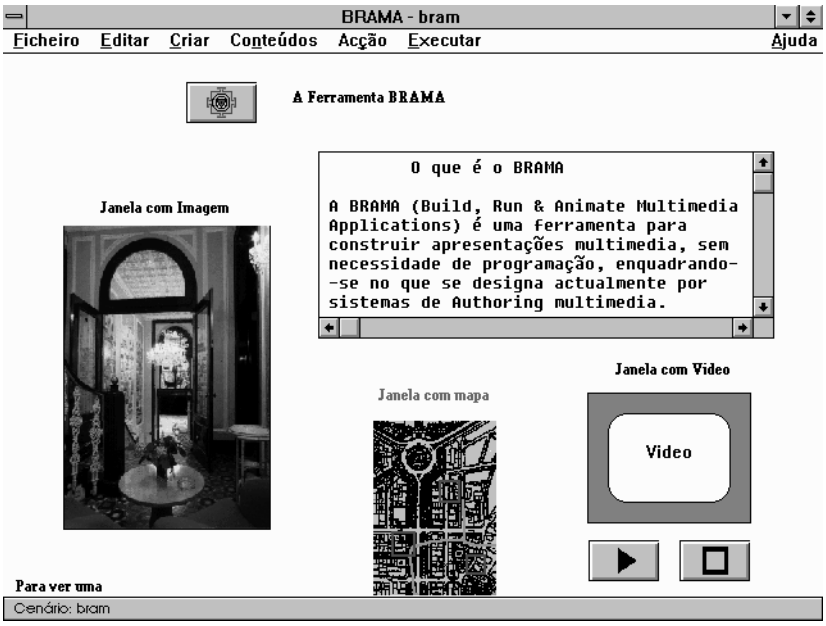

Figura 2.4: A ferramenta Brama.

A sua finalidade é a produção de documentos multimédia/hipermédia destinados a um tipo de utilização específica. Esta utilização é caracterizada pela ideia de visita guiada por se destinar essencialmente a leitores. A ferramenta BRAMA, permite por exemplo criar guias turísticos interactivos. Num mapa (imagem digitalizada ou desenhada) definem-se zonas interactivas, correspondentes a pontos de interesse particular (museus, teatros, etc.) aos quais se associam imagens, locução, música e texto, de forma a fornecer ao leitor o máximo de informação de uma forma harmoniosa. É ainda possível definir percursos (pela simples selecção de pontos de interactividade e sua ligação) e animá-los.

A ferramenta BRAMA destina-se a todos os utilizadores familiarizados com o Microsoft Windows, que pretendam construir apresentações multimédia através da simples manipulação visual de objectos e/ou linguagem de *script* em português.

Uma limitação desta ferramenta, tal como do sistema operativo em que está baseada, é não suportar distribuição. Outra limitação advém de os objectos multimédia serem tratados como entidades externas indivisíveis, pelo que apenas é possível definir acções a serem executadas quando o utilizador carrega em botões. Além disso, o gestor

de sincronização não permite associar acções a realizar quando um vídeo ou som termina, muito menos quando eles chegarem a uma dada posição intermédia. Os únicos eventos que sabe processar são a activação de botões e temporizadores cíclicos aos quais se podem associar funções.

O editor da ferramenta Brama permite facilmente posicionar os objectos no écran, dando uma ideia do aspecto final da aplicação logo durante a edição. Mas, por outro lado, é difícil ter uma ideia do desenrolar da aplicação sem a correr, pois a apresentação é toda programada por menus. Além disso, devido às facilidades de edição próximas do resultado da apresentação, é necessário dividir a apresentação em tantos ficheiros *script* quantos os écrans que se pretende obter, o que dificulta ainda mais ter uma ideia da apresentação como um todo.

Uma vez que recorre à utilização de extensões multimédia do MS Windows [Microsoft 91], esta ferramenta suporta apenas um objecto vídeo e um objecto de som de cada vez, e aceita apenas o formato de ficheiros vídeo, som e imagem standard neste ambiente, o que não é limitativo dado haver disponíveis conversores de formatos.

Outra desvantagem desta ferramenta advém de ser uma única aplicação com o editor, gestor de sincronização e objectos multimédia, pelo que qualquer extensão para acrescentar ou modificar funcionalidades poderá implicar modificar todas as componentes deste programa.

2.2.2 BERKOM

O BERKOM (*Berliner Kommunikationssystem*) é um dos mais proeminentes projectos de teste de RDIS de banda larga a nível mundial. O objectivo da segunda fase deste projecto é oferecer uma única infra-estrutura de comunicação uniforme para as futuras aplicações multimédia de banda larga. Esta fase do projecto compreende três áreas de trabalho que serão seguidamente analisadas:

- Serviço de Transporte Multimédia.
- Serviço de Correio Multimédia.
- Serviço de Colaboração Multimédia.

Serviço de Transporte Multimédia

O Serviço de Transporte Multimédia [Boecking+ 93] fornece a plataforma de comunicação audiovisual. É baseado numa versão modificada do protocolo XTP sobre o protocolo ST-II e pode utilizar uma grande variedade de redes, tais como Ethernet, anel com testemunho, FDDI, VBN, e ATM.

O XTP, *Xpress Transport Protocol* [XTP 92], é um protocolo leve, que pode ser implementado em VLSI, concebido para redes de alta velocidade e que inclui as funcionalidade dos níveis rede e transporte do modelo OSI. Este protocolo oferece serviços de circuito virtual e datagrama, suportando tempo real, controlo de ritmo, controlo de fluxo, recuperação de erros por retransmissão selectiva e comunicação multiponto fiável.

O ST-II, *Internet Stream Protocol* [CIP 90], pretende substituir o nível rede da Internet para transferência de dados isócronos, como voz e imagem. Este protocolo funciona em modo circuito virtual (*stream*) sem garantias de recuperação de erros, o que é desejável para a transferência de dados multimédia. O protocolo oferece serviços de tempo real, controlo de ritmo, multiplexagem (por exemplo: juntar voz e imagem), e comunicação multiponto.

O sistema de transporte multimédia do BERKOM usa um protocolo XTP-lite que corresponde ao XTP sem as funcionalidades de nível rede, e melhorado para incluir funções necessárias à transmissão de voz e imagem.

Verifica-se que a especificidade da transferência de dados multimédia levou ao desenvolvimento de um serviço de transporte específico. De facto, tornou-se necessária a existência de um protocolo de transporte com ligação lógica, e suportando vários parâmetros de qualidade de serviço:

- Tamanho máximo das TSDUs em octetos.
- Débito em TSDU/segundo.
- Tempo de trânsito extremo a extremo em segundos, sendo especificado um valor máximo e mínimo.
- Classe de confiança, determinando se o nível transporte deve ignorar, detectar, indicar, ou corrigir os erros de transmissão.

Serviço de Correio Multimédia

O Serviço de Correio Multimédia facilita a troca de documentos multimédia incluindo anotações áudio e vídeo. Foi concebido como uma extensão compatível com as funcionalidades de X.400 e ODA, e permite interactuar com o MIME, *Multipurpose Internet Mail Extensions*, o formato do correio multimédia da Internet.

O ODA, *Open Document Architecture* [ISO 89], define um modelo de documentos hierárquico e orientado para objectos. Os objectos representam componentes do documento, e atributos fornecem informação sobre os objectos. Um documento é visto como uma árvore, cuja estrutura é definida pela forma da árvore, e o conteúdo pelas folhas da árvore. Um documento é descrito através de uma **estrutura lógica** (*logical structure*) e uma **estrutura de exposição** (*layout structure*). A estrutura lógica divide e subdivide o documento em itens com significado para o autor ou leitor, por exemplo: capítulos, secções, títulos e parágrafos. A estrutura de exposição divide e subdivide a representação visível do documento em áreas rectangulares, por exemplo: páginas, quadros e blocos.

Actualmente o ODA define conteúdos de tipo carácter, gráfico e imagem. Foram definidas extensões ao ODA para integrar tipos de dados dependentes do tempo nesta norma, tais como áudio e vídeo [Hoepner 91]. Desta forma, a apresentação de um documento consiste num conjunto de acções temporalmente relacionadas.

Uma **acção** é a representação de algo que acontece, e um **evento** é uma unidade atómica de acção. Apenas alguns eventos têm interesse para a sincronização, sendo designados por pontos de sincronização. As acções são delimitadas por dois pontos de sincronização: o *startpoint* e o *endpoint*. Ambos podem ser definidos em termos de unidades absolutas de tempo, ou em termos de relações temporais com outras acções, por exemplo: o fim de outra acção, uma determinada duração após uma acção ou uma interacção com o utilizador. As acções podem subdividir-se em acções **atómicas**, quando não são subdivididas para efeitos de sincronização, e acções **compostas**, que são compostas de acções atómicas ou compostas, exigindo sincronização.

A sincronização das acções é definida numa notação baseada em expressões de caminho [Campbell+ 74]. As expressões de caminho descrevem a sequência de operações permitidas. Os operadores de caminho definem a sincronização das operações.

Generalizando o conceito de operação, são utilizadas acções nas expressões de caminho definindo-se os seguintes operadores:

$A \wedge B$ *Parallel-last*: As acções A e B começam num *startpoint* comum, e são executadas concorrentemente. A acção composta termina quando todas as acções participantes (A e B) terminarem.

$A \vee B$ *Parallel-first*: As acções A e B começam num *startpoint* comum, e são executadas concorrentemente. A acção composta termina quando a primeira (no tempo) acção participante (A ou B) terminar.

$A ; B$ *Sequential*: Só é permitido executar B se A for executado anteriormente. O *endpoint* de A coincide com o *startpoint* de B. A acção composta termina quando a última acção da sequência terminar.

$A \mid B$ *Selective*: É permitido executar quer A, quer B. A selecção depende de uma condição que não faz parte da expressão de caminho, sendo avaliada por outro meio. A acção composta termina quando a acção seleccionada (A ou B) terminar.

A^{i*} *Repetition*: A acção A é repetida *i* vezes. Se não for fornecido um *i*, A é repetida zero ou mais vezes, devendo o número exacto de repetições ser fornecido por algum outro meio, por exemplo o utilizador.

$N : A$ *Concurrency*: A acção A pode ser executada N vezes concorrentemente. Se N=1, correspondendo ao valor por defeito, a execução de A é mutuamente exclusiva.

Como exemplo de uma expressão de caminho temos:

```
path (A ∧ B) ; D end
path A ; C end
```

Para este exemplo, as acções A e B são iniciadas simultaneamente. Quando A terminar, deve ser começada uma acção C. Quando A e B terminarem, D é iniciado. Note-se que as acções C e D não têm nenhum ponto de sincronização em comum, e a acção A só é executada uma vez.

As extensões ao ODA propostas são aplicadas apenas à estrutura de exposição, deixando as extensões à estrutura lógica para estudo posterior. Modificando a expressão de caminho para uma notação prefixa, obtém-se uma estrutura hierárquica em árvore. Os operadores de caminho são nós, e as acções folhas. Com esta modificação, obtém-se uma correspondência com a estrutura hierárquica do ODA, levando à definição de novos atributos para os objectos de exposição:

- `CONTENT TEMPORAL TYPE`: Aplica-se a objectos básicos (não compostos), e pode tomar os valores `estático`, se o objecto conteúdo associado é invariante no tempo, ou o valor `dinâmico`, se o objecto é variante no tempo.
- `DURATION`: Aplica-se a objectos básicos dinâmicos, e define a duração da apresentação do objecto.
- `OBJECT SYNCHRONIZATION TYPE`: Aplica-se a objectos compostos, e define a ordenação temporal dos objectos subordinados. Pode tomar os valores `parallel-last`, `parallel-first`, `sequential` ou `selective`, com significados iguais aos dos operadores de caminho definidos.
- `REPETITION`: Aplica-se a objectos compostos, e define o número de vezes que o objecto corrente deve ser apresentado.
- `START FIRST`: Aplica-se a objectos compostos, e define o início da apresentação de um objecto composto através de restrições temporais, ou interacções com o utilizador.
- `START FOLLOWING`: Aplica-se a objectos básicos e compostos, e define as condições de terminação do objecto corrente e/ou início do objecto seguinte numa sequência, através de restrições temporais, ou interacções com o utilizador.

Este modelo de sincronização baseado em expressões de caminho apresenta algumas limitações. As expressões de caminho denotam propriedades dos operandos, pelo que é possível uma especificação ter várias expressões de caminho utilizando o mesmo objecto, embora ele só deva ser apresentado uma vez, o que complica a compreensão das expressões. Além disso, as interacções com utilizadores são especificadas através de atributos, separadamente das expressões de caminho, encontrando-se limitadas à activação dos objectos.

Serviço de Colaboração Multimédia

O Serviço de Colaboração Multimédia [Altenhofen+ 93] suporta o trabalho cooperativo num ambiente distribuído. Permite que utilizadores partilhem aplicações e participem em videoconferências a partir dos seus computadores. Qualquer utilizador pode lançar uma aplicação a ser partilhada, sendo a partilha conseguida distribuindo os resultados produzidos por todos os utilizadores, mas permitindo-se que apenas um utilizador, que deve possuir um testemunho, possa fornecer dados à aplicação. Uma

conferência pode ter um número variável de participantes que podem assumir papéis de: moderador, que tem privilégios de gestão da conferência; possuidor de testemunho, que pode submeter aplicações a serem partilhadas pela conferência; e observador, que participa passivamente na conferência. Para além dos testemunhos que dão o direito a controlar cada aplicação partilhada, há ainda um testemunho associado ao direito a enviar imagem e som para a conferência.

A gestão das conferências é conseguida principalmente através da gestão de testemunhos e gestão dos grupos de participantes. O serviço de colaboração está estruturado em vários módulos. A administração das conferências é realizada por um módulo *Conference Manager* (CM), que supervisiona a gestão de testemunhos, os direitos de acesso e papéis dos participantes; a partilha de aplicações e a comunicação audiovisual. Associado ao CM, há um módulo *Audiovisual Manager* que coordena o estabelecimento de canais de comunicação entre os participantes. Um módulo *Conference Directory*, que é uma base de dados com informação estática sobre as conferências, mantém informação sobre utilizadores, grupos de utilizadores e conferências, e pode ser distribuído através dos serviços de X.500. Por cada máquina de um utilizador, há um módulo *Audiovisual Component* que estabelece ligações áudio e vídeo entre os participantes. Para a partilha de aplicações, há um módulo *Application Sharing Component* que distribui os resultados produzidos pela aplicação.

Verifica-se que a gestão da sincronização neste sistema encontra-se centralizada, apesar de um protocolo baseado em testemunhos poder ser realizado de forma descentralizada. No entanto, depois de definidos os grupos de participantes, a transferência de dados multimédia é feita directamente por ligações multiponto entre o *source* e os *sinks*.

2.3 NORMALIZAÇÃO

2.3.1 MHEG

O MHEG (*Multimedia and Hypermedia information coding Expert Group*) é um grupo de trabalho da ISO/IEC que pretende definir um standard [ISO 93b][Price 93] para a "representação codificada de objectos de informação multimédia e hipermédia na

sua forma final, que serão trocados como um todo entre, ou através de, serviços e aplicações, por qualquer meio (métodos de arquivo, redes locais, redes de telecomunicações de área ampla, ou redes de difusão)". Espera-se que este standard esteja finalizado no fim de 1994 ficando registado como ISO 13522, e fornecendo uma descrição orientada por objectos e uma notação em ASN.1 [ISO 87]. Seis meses depois, uma segunda parte providenciará notações alternativas em SGML [ISO 86] isomorfas com as da primeira parte.

Na figura 2.5 mostra-se que um objecto MHEG está apenas definido na comunicação entre duas aplicações. Quando uma aplicação A quer enviar um objecto MHEG para uma aplicação B, deve chamar um formatador MHEG para converter do formato interno de A para o formato normalizado. Quando a aplicação B recebe o objecto, descodifica-o no seu analisador para o seu formato interno. Nada impede que estes formatos internos sejam o próprio formato MHEG, embora o standard não obrigue a isso.

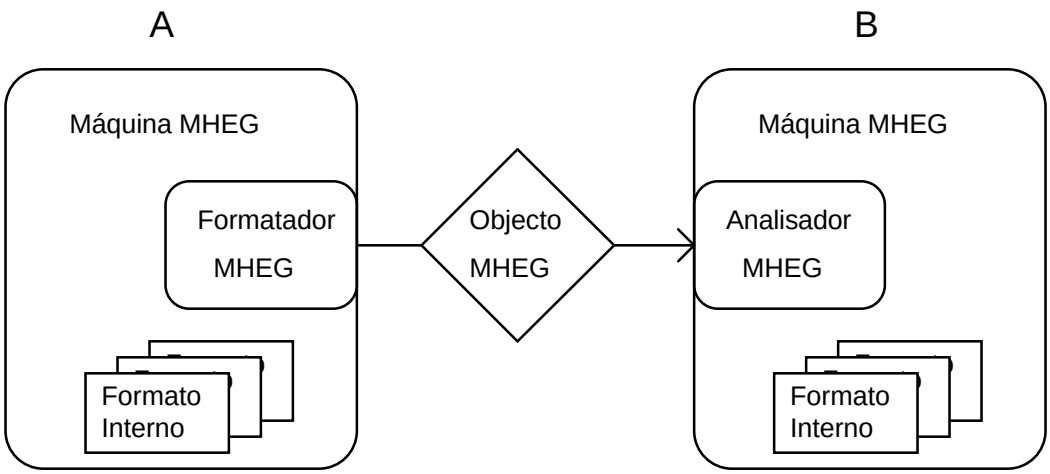

Figura 2.5: Utilização do modelo MHEG.

Na figura 2.6 mostra-se a hierarquia de classes definida na norma MHEG. Apenas podem ser trocadas instâncias das classes indicadas em **negrito**, sendo as outras classes abstractas.

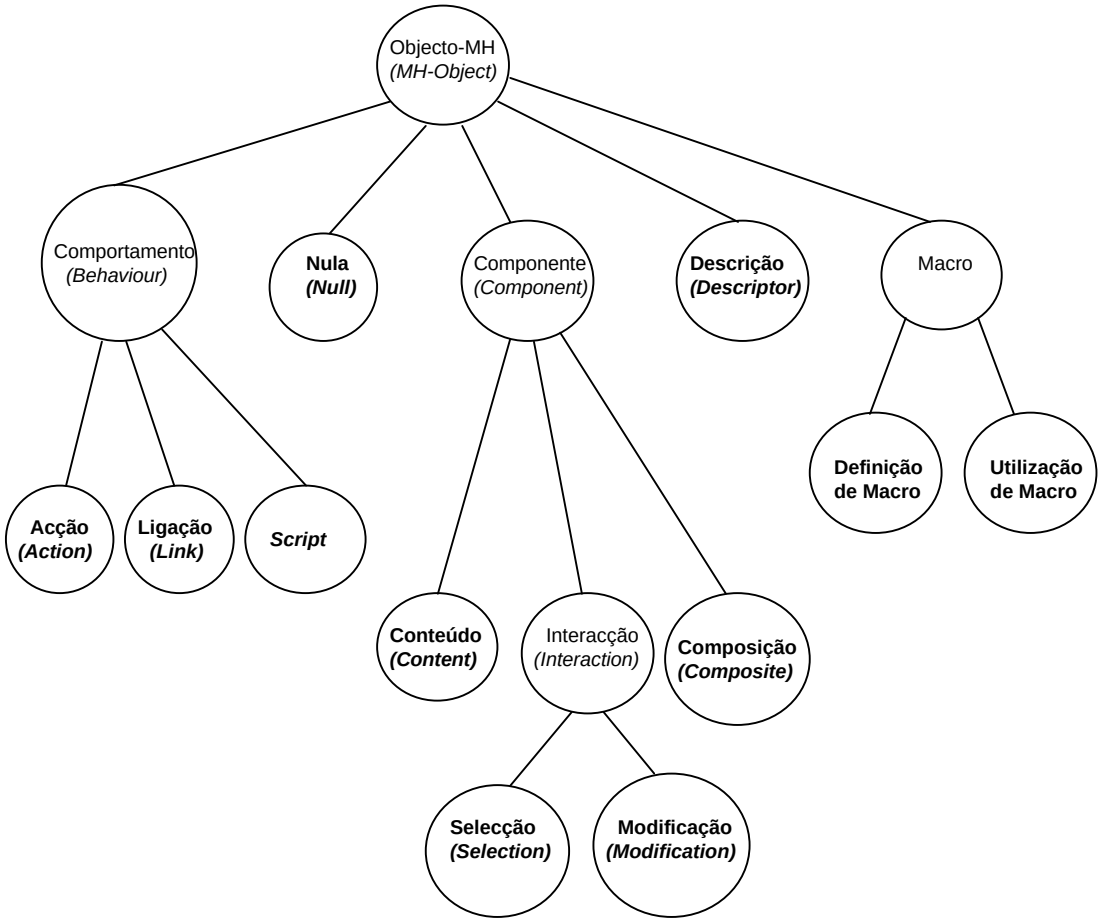

Figura 2.6: Hierarquia de classes MHEG.

A classe Macro serve para permitir a partilha e reutilização de estruturas complexas.

A classe Nula serve para testes.

A classe Descrição serve para fornecer informações gerais sobre os objectos transferidos, como por exemplo os recursos necessários para a apresentação.

A classe Conteúdo serve para transferir dados multimédia, tais como texto, imagens fixas, gráficos, som, vídeo ou outros dados. Estes dados são codificados de acordo com outras normas.

Um objecto Componente é considerado um objecto original, e pode ser referenciado num visualizador para a sua apresentação num contexto específico. A apresentação não afecta o objecto Componente original, e permite a reutilização do mesmo objecto Componente em diversos contextos da apresentação, ou em diversos visualizadores. Por exemplo, num jogo de hóquei, haverá um objecto Conteúdo com a imagem original de um jogador, e 20 visualizadores que referindo-o corresponderão aos

10 jogadores da equipa azul e aos 10 jogadores da equipa vermelha. Para além disso, para cada equipa 6 visualizadores terão posições diferentes no ringue e 4 visualizadores terão posições diferentes nos bancos, representando os jogadores de reserva.

A classe Selecção permite descrever uma árvore de selecções permitidas ao utilizador.

A classe Modificação permite a modificação de objectos Conteúdo.

A classe Composição permite agrupar objectos Componente relacionados, num único objecto, e o meio de descrever a apresentação correspondente. Contém uma parte de comportamento consistindo de listas de Ligações, Acções e *Scripts* para descrever as inter-relações (temporais, espaciais e lógicas) entre visualizadores. Contém ainda um conjunto de visualizadores que são referências a objectos Componente originais e facultativamente o correspondente conjunto de objectos Componente originais.

A classe Acção define acções que podem ser aplicadas em objectos ou visualizadores, podendo-se classificar em diversos tipos:

- Acções de **preparação**, que afectam o estado de um objecto mheg, por exemplo "preparar", "destruir".
- Acções de **informação**, que retornam informação sobre o estado de um objecto mheg.
- Acções de **apresentação**, que afectam o estado de um visualizador, por exemplo "começar", "parar".
- Acções de **projecção**, que afectam a forma como os objectos são apresentados, por exemplo "volume", "velocidade", "posição".
- Acções de **interacção**, que permitem modificar o resultado de interacções, por exemplo modificar as possibilidades de modificação e selecção de visualizadores.

A classe Ligação serve para descrever o comportamento da aplicação permitindo a execução de acções quando determinada condição for satisfeita. Uma ligação liga de forma condicional uma origem (tanto um objecto mheg como um visualizador) com um ou mais destinos (objectos mheg ou visualizadores), sendo composta de:

- **Condições** associadas à variação da avaliação de um estado de uma origem. A condição principal é designada condição de disparo, podendo-se definir condições adicionais referentes a estados do mesmo ou de outros objectos ou visualizadores.

- **Acções**, que são processadas quando a condição de disparo e condições adicionais forem satisfeitas.

Uma ligação MHEG é, assim, um mecanismo geral para descrever relações entre objectos e visualizadores, no tempo, no espaço ou relações puramente condicionais, como por exemplo: "quando o visualizador A começar a correr, destruir o objecto mheg B". De entre as possibilidades permitidas com ligações MHEG destacam-se:

- Sincronização elementar, que corresponde à sincronização de dois objectos relativamente ao mesmo instante de origem, correndo em paralelo, ou um relativamente ao outro, correndo em sequência.
- Sincronização sequencial, em que um conjunto de objectos deve ser apresentado um após o outro, em sequência.
- Sincronização cíclica, em que um ou mais objectos devem ser repetidamente apresentados.
- Sincronização condicional, em que a apresentação de um objecto está associada à satisfação de uma condição.

A classe *Script* serve para descrever ligações mais complexas que as anteriormente referidas, entre objectos ou visualizadores. Este standard não define a linguagem de script ou a sua codificação, mas providencia apenas o encapsulamento de um programa externo no formalismo mheg. Assim, este objecto compreende um atributo para especificar que linguagem se usa, e o programa propriamente dito, codificado nessa linguagem.

Há quatro tipos de sincronização referidos na norma:

- ***Script***: uma descrição geral de como a apresentação deve ser feita, podendo ter sincronizações complexas levando em conta respostas prévias dos utilizadores, valores calculados, ou os recursos disponíveis no sistema. Este nível de sincronização será providenciado por futuros standards internacionais tais como AVI *scriptware* (*Audio Visual Interactive*), podendo utilizar o standard para troca de objectos MHEG, recorrendo à classe *script*.
- **Condicional**: O estado corrente do sistema pode desencadear acções noutros objectos. Por exemplo, "quando o som terminar, fazer a pergunta".

- **Espaço-Tempo**: Definindo a posição no tempo e no espaço de um objecto relativamente a outro. Por exemplo, "Mostrar o nome do produto 2 cm acima da imagem".

- **Sistema**: Sincronização de baixo nível. Por exemplo, a sincronização entre som e imagem para que o som bata certo com o movimento dos lábios. Este nível de sincronização é providenciado por outros standards, para o exemplo referido pelo MPEG [ISO 11172].

Comparando esta divisão da sincronização em níveis com outras propostas, tal como a apresentada anteriormente que referia apenas dois níveis: um de baixo nível próximo dos objectos multimédia, e outro de alto nível próximo da aplicação, verifica-se haver coincidência de funções para a sincronização de baixo nível, embora a de alto nível se encontre aqui subdividida em três. Verifica-se por isso haver uma certa repetição de funções nestes três níveis, que pode ser explicada por o *script* ser uma alternativa de sincronização externa à norma MHEG, e pelo facto de a sincronização Condicional e a sincronização no Espaço-Tempo serem conceptualmente muito próximas uma da outra, sendo ambas realizadas com a classe Ligação.

Outro ponto importante é o facto de a portabilidade só estar garantida a nível dos objectos, não a nível das aplicações. Desta forma, tanto a apresentação de um objecto MHEG pode diferir, como pode haver problemas de portabilidade dos *scripts*.

Convém realçar que este standard ainda se encontra em estudo, e portanto sujeito a modificações.

2.3.2 WWW

Embora não seja uma norma internacional, o *World-Wide Web*[1] (WWW ou W3) [Lee+ 93], pela grande expansão que tem, está a tornar-se uma norma de facto entre os utilizadores da Internet.

O WWW é uma forma de obter informação imediatamente disponível na Internet como se fosse um meio contínuo pesquisável. Recorrendo a saltos e pesquisas em hipertexto, o utilizador navega através de um mundo de informação em parte escrito à mão, e em parte gerado por computador a partir de bases de dados e sistemas de informação existentes. Como ferramentas de interface com o utilizador, os clientes

[1] *Web* na gíria de hipertexto significa um conjunto de ligações (*links*).

WWW correm no computador deste, permitindo-lhe aceder à rede através de simples selecções com o rato, enquanto os servidores WWW, normalmente numa máquina completamente diferente, algures noutra parte do mundo, oferecem um método eficiente e simples de fornecer informação aos utilizadores.

O WWW define:

- A ideia de um mundo onde cada pedaço de informação tem uma referência pela qual pode ser acedido.
- Um sistema de endereçamento (URL - *Uniform Resource Locator*), que permite endereçar vários tipos de objectos acessíveis através de protocolos já em uso, tais como FTP, NNTP, Gopher, WAIS e HTML.
- Um protocolo de rede (HTTP - *Hypertext Transfer Protocol*) oferecido pelos servidores WWW genuínos com desempenho e possibilidades que não estariam de outro modo disponíveis.
- Uma linguagem de hipertexto com marcas de formatação[2] (HTML - *Hypertext Markup Language*) que todos os clientes WWW devem entender, e que é usada para a transmissão de coisas básicas, tais como texto, menus e informação de ajuda. Esta linguagem tem um formato compatível com o SGML, *Standard Generalized Markup Language* [ISO 86].

O SGML foi desenvolvido em resposta à necessidade de descrever documentos em termos da sua estrutura lógica, e define um modelo de documento hierárquico sob a forma de árvore. O SGML providencia uma metasintaxe para exprimir a sintaxe de cada tipo de documento. Cada uma dessas sintaxes é designada *document type definition* (DTD). A sintaxe é expressa como um conjunto de elementos lógicos do documento (*elements*) delimitados por códigos genéricos (*tags*), um conjunto facultativo de atributos e um modelo de conteúdo (*content model*) que consiste numa produção do tipo BNF[3] que especifica que tipos de dados ou elementos podem ser colocados dentro de cada elemento. Por exemplo, um elemento livro pode conter vários elementos secção, um elemento secção vários elementos parágrafo, até que o elemento terminal contém texto.

---

[2] O nome *Markup* advém da semelhança com as marcações que os editores fazem em rascunhos de documentos.

[3] BNF ou Backus-Naur *Form*, também designada gramática livre de contexto, é uma notação largamente utilizada para a especificação da sintaxe de linguagens.

Na figura 2.7 vê-se o navegador de WWW da NCSA, Mosaic em X Windows, mostrando a página de abertura de Portugal. No topo está indicado o título do documento e o endereço (URL). Tanto o texto sublinhado, como as zonas assinaladas no mapa e os ícones correspondem a ligações hipertexto, bastando seleccionar uma com o rato para se saltar para o documento correspondente.

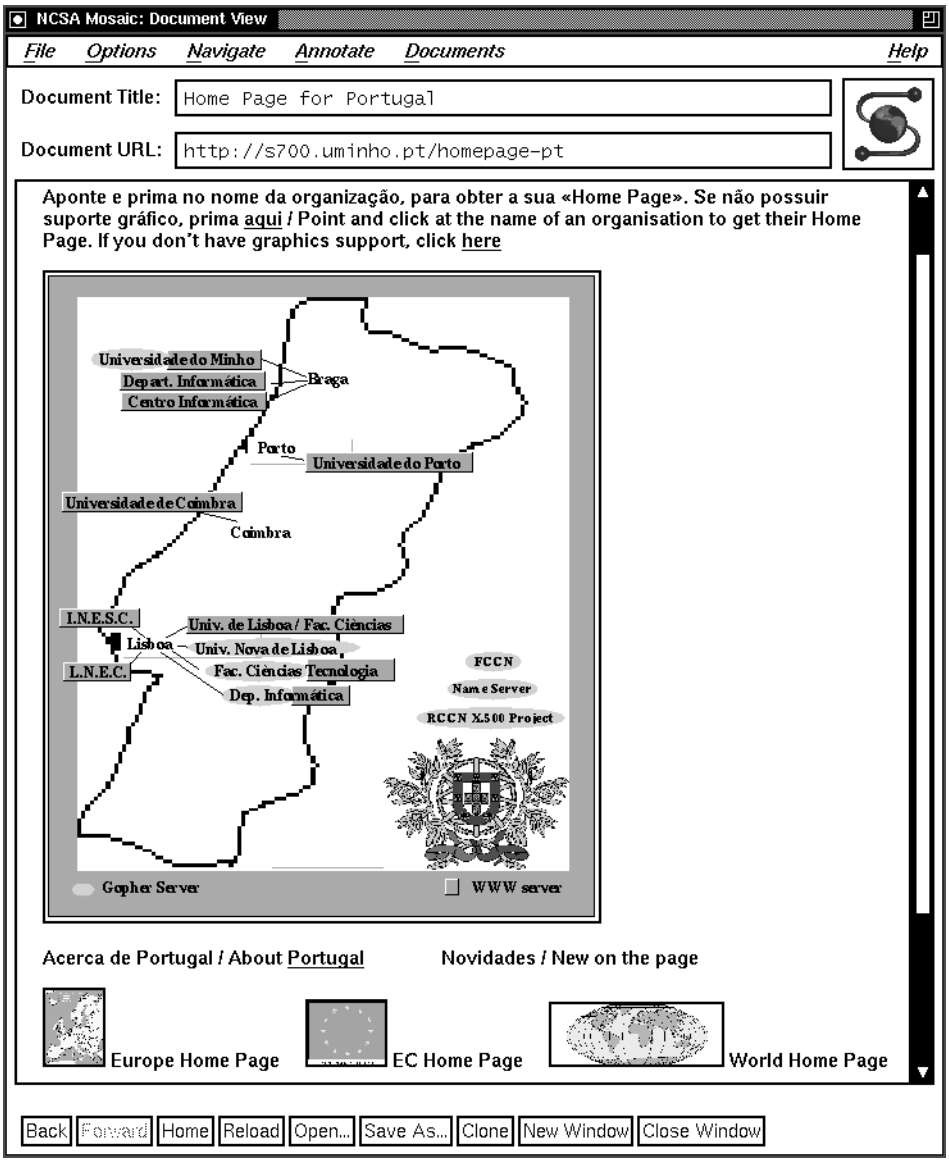

Figura 2.7: O Navegador de WWW da NCSA, Mosaic em X Windows, mostrando a página de abertura de Portugal.

Na tabela 2.3 são comparadas algumas das características de vários sistemas de pesquisa de informação em rede baseada em [Obraczka+ 93] e em [Lee+ 93]. O WAIS (*Wide Area Information Server*) [Addyman 93] resultou dos esforços combinados de

quatro companhias com interesses complementares na área de pesquisa e recuperação de informação. O Internet Gopher [Anklesaria+ 93] permite a pesquisa e navegação em informação distribuída usando um protocolo implementado sobre TCP-IP. O X.500 [CCITT 88] é o resultado do esforço de normalização do CCITT e da ISO no campo de directórios de serviços.

	WAIS	Gopher	X.500	WWW
Granularidade:	Objectos e Descritores de Objectos	Ficheiros	Descritores de Objectos	Ficheiros e Porções de Ficheiros
Formatos:				
Texto	com palavras chave	Sim	Sim	Sim
Gráficos	-	Visualizador externo	Sim	Sim
Hipertexto	-	-	-	Sim
Funções:				
Navegação		Sim		Sim
Pesquisa	Sim		Sim	
Refinamento de Pesquisa	Sim	-	-	-
Referência a outros servidores	-	Sim	Sim	Sim
Protocolos suportados pelos clientes:				
WAIS	Sim	por gateway	-	Sim
X.500	-	por gateway	Sim	por gateway
FTP	-	por gateway	-	Sim
NNTP (Net News)	-	por gateway	-	Sim
Gopher	-	Sim	-	Sim
HTTP	-	-	-	Sim

Tabela 2.3: Comparação de vários sistemas de pesquisa de informação em rede.

Os vários sistemas de informação baseiam-se em interacções cliente-servidor, no entanto, a forma principal como se procura informação é diferente nos vários sistemas. Na realidade, sistemas como o Gopher e o WWW estão mais orientados para a navegação no espaço de informação disponível, enquanto sistemas como o X.500 e o WAIS estão mais orientados para a pesquisa de informação através de perguntas a uma

base de dados. A granularidade com que os clientes manipulam a informação também é uma característica que os distingue. Em sistemas como o Gopher, os clientes acedem a ficheiros inteiros, enquanto no X.500 é possível obter informação sobre apenas um determinado atributo.

É de notar nalguns destes sistemas um esforço por manter compatibilidade com outros sistemas já existentes, que inclui a necessidade de normalização da forma de identificar e aceder aos objectos.

Conclui-se assim, que o WWW é basicamente um sistema hipertexto distribuído, não suportando meios contínuos, pelo que um cliente WWW não necessita de uma componente de sincronização, uma vez que se limita a pedir sequencialmente as várias partes de um documento ao servidor, apresentando-as ao utilizador e bloqueando-se à espera que este dê ordem para ir buscar outro documento.

CAPÍTULO 3
MODELO DE OBJECTOS

Neste capítulo apresenta-se o modelo de objectos multimédia desenvolvido, sendo realçada a forma como os objectos multimédia são construídos, a forma como interagem uns com os outros, e a forma como são usados em aplicações.

3.1 DESCRIÇÃO GERAL

Um bom modelo de objectos multimédia deve ter as características que se enumeram de seguida:

- **Evolução**. Atendendo à constante evolução da técnica, tanto a nível de equipamentos como normas de codificação de dados e métodos de os processar, o modelo, para poder ser útil por muito tempo, deve permitir uma evolução fácil possibilitando modificar ou criar novas funcionalidades com esforço mínimo.
- **Encapsular as dependências do equipamento**. Necessário para minimizar as modificações com a evolução tecnológica.
- **Distribuição**. O equipamento multimédia é caro, havendo vantagem em partilhá-lo. Além disso, o funcionamento em rede permite a criação de novos serviços, muito mais atractivos, inexistentes em sistemas centralizados.
- **Interacção com utilizadores**. O utilizador não deve ser apenas um espectador passivo, devendo ser possível que ele participe de forma activa.
- **Permitir qualquer tipo de interacção de controlo sobre os objectos**. Os objectos multimédia não devem ser tão simples que permitam apenas começar e parar, devendo ser possíveis formas de sincronização durante o funcionamento dos objectos, por exemplo quando o objecto de vídeo V chegar à imagem 50, activar o objecto de som S. Note-se que não se pretende fazer sincronização de baixo nível, como é o caso de sincronizar o som com o movimento dos lábios num vídeo, pois este tipo de

sincronização é computacionalmente intenso, e frequentemente pouco relevante do ponto de vista do autor de uma especificação multimédia.

- **Facilidade de reutilização dos objectos multimédia**. Pretende-se que seja possível interligar os objectos multimédia de formas diferentes para aplicações diferentes. Para além disso, deve ser possível utilizar os objectos para várias aplicações sem ter de os modificar.

- **Ferramentas de criação de aplicações fáceis de usar e poderosas**. Não deve ser necessário saber-se como a arquitectura foi implementada, ou ser um programador experimentado para conseguir criar aplicações multimédia. Assim, o modelo tem de ser conceptualmente simples, sendo, no entanto, suficientemente geral para permitir a grande variedade de usos possíveis em multimédia.

De seguida analisa-se como se conseguem obter estas características.

Para permitir a distribuição, torna-se necessário decompor os objectos multimédia na parte que produz o fluxo de dados multimédia e na parte que recebe esse mesmo fluxo de dados apresentando-o ao utilizador. Desta forma se permite, por exemplo, que o som seja gerado numa máquina e apresentado no altifalante de outra máquina. No entanto, o programador prefere continuar a pensar que tem apenas um objecto multimédia de som, sendo-lhe escondida a distribuição, de modo a que lhe baste mandar tocar o som, e não tenha de pensar em activar todos os componentes desse objecto que estão distribuídos pela rede.

Assim, os objectos multimédia são entidades activas compostas por objectos **componentes** que se podem subdividir em [Gibbs 91]:

- ***Sources***[1], onde são originados sinais multimédia.
- ***Sinks***, onde são consumidos sinais multimédia.
- **Filtros**, que transformam os sinais multimédia que recebem noutros sinais.

Uma vez que se trata de um sistema distribuído, os *sources* e *sinks* poderão estar localizados em diferentes locais no sistema.

Para esconder essa distribuição do utilizador, os objectos multimédia oferecem um *interface* único para o exterior através de um ***dispatcher***. O *dispatcher* do objecto

[1] Preferiu-se utilizar os termos anglo-saxónicos *source* e *sink.* Seria, no entanto, possível traduzir *source* por fonte, ou origem, ou produtor de informação, e *sink* por destino, ou apresentador, ou consumidor de informação.

multimédia decide o que fazer com as interacções que receba, encaminhando-as para o *source* ou *sink* conveniente.

Como tinha sido referido no capítulo 1, a especificação criada no editor gráfico será compilada produzindo um programa, designado **Gestor de Sincronização**, que coordena o funcionamento dos objectos multimédia de forma a que eles respeitem a composição criada.

Para permitir qualquer tipo de interacção de controlo sobre os objectos multimédia, eles devem poder gerar **eventos** que descrevam alguma situação referente ao seu funcionamento que seja relevante para o gestor de sincronização, por exemplo, `Ev_End` quando terminarem de tocar. Os eventos podem classificar-se em eventos **determinísticos**, que ocorrem sempre pela mesma ordem durante o funcionamento de um objecto, e eventos **não determinísticos** que modelam situações que podem não ocorrer, tais como intervenções do utilizador. Esta classificação serve apenas para verificações de consistência.

Além disso, os objectos devem ainda poder receber comandos, designados **acções**, para controlar o seu funcionamento, por exemplo `Play` para começar a tocar.

Para encapsular as dependências do equipamento, os objectos devem oferecer para o exterior um *interface* padrão. No entanto, para que seja fácil a evolução do sistema, pretende-se poder acrescentar novos objectos com novas funcionalidades, ou modificar os objectos multimédia existentes, mantendo compatibilidade com as versões anteriores. Para que isso seja possível, os objectos devem suportar um *interface* padrão muito simples com operações para os casos mais comuns e que ofereça um mecanismo genérico pelo qual possa ser estendido o tipo de interacções possíveis, de forma a permitir-lhe interagir a nível temporal, espacial ou lógico com outros objectos no sistema.

Podem ainda existir outros objectos, que se chamam **painéis de controlo**, que permitem mudar certas propriedades dos objectos como a sua geometria (recorrendo ao gestor de janelas do sistema), a velocidade de apresentação, o volume do som e a filtragem de cores entre outros.

Na figura 3.1 está ilustrada a forma como os objectos multimédia interagem com o resto do sistema. Os objectos multimédia são entidades compostas por *sources*, *sinks* e filtros. Estes objectos componentes são representados por círculos, e têm **portos** com tipos associados, que são interligados, representando-se por uma seta, de acordo com

uma configuração. No caso mais simples, eles terão um *source* e um *sink*. Os portos de saída representam-se por quadrados salientes, e os portos de entrada por quadrados no interior do círculo.

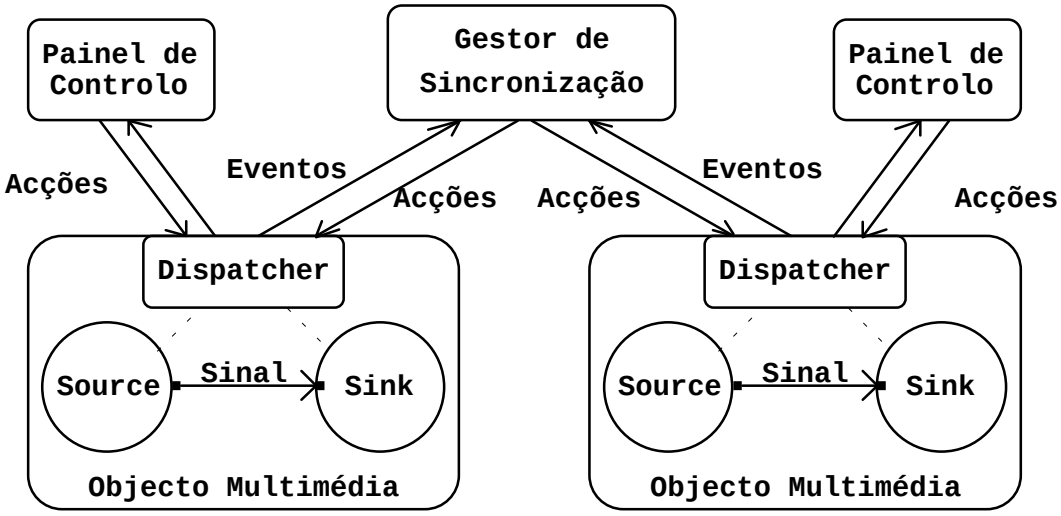

Figura 3.1: Objectos multimédia.

A nível de controlo, a comunicação é feita por troca de **mensagens**. Uma mensagem pode ser uma invocação de uma acção de um estado, que é convertida no objecto receptor na invocação de um método, ou pode ser a emissão de um evento que chega ao receptor como uma invocação de um método de recepção de eventos. De entre os modelos de comunicação entre processos introduzidos na secção 2.1.3 (página 13), o mais adequado para a comunicação entre os objectos é a passagem de mensagens assíncrona, pois o envio de mensagens não é bloqueante. Para as acções que retornem parâmetros, também seria adequado o modelo de chamadas a procedimentos remotos. No entanto, é preferível, quando necessário, retornar parâmetros através do mecanismo de eventos, pois os objectos multimédia podem ter tempos de resposta elevados.

Embora na figura 3.1 se represente apenas a troca de acções e eventos com o gestor de sincronização e com os painéis de controlo, nada impede que os próprios objectos multimédia ou componentes também troquem eventos ou invocações de acções entre eles, o que exige uma maior inteligência dos objectos e uma gestão da sincronização descentralizada. Esta possibilidade é também adequada, por exemplo, para gerir a sincronização dos fluxos de dados multimédia, de baixo nível, de forma mais eficiente.

3.2 OBJECTOS COMPONENTES

Um **objecto componente** é uma entidade activa que pode produzir **sinais** de saída de acordo com tipos bem definidos, (por exemplo: áudio PCM de 16 bits), receber sinais de entrada com tipos bem definidos, produzir **eventos** de acordo com tipos específicos para eventos, e oferecer métodos para a recepção de **acções** a executar. O componente tem variáveis de estado internas, agrupadas em **estados**, que podem ser modificadas pela invocação de métodos, designados acções, ou por iniciativa do próprio objecto. A cada tipo de estado está associado um conjunto de acções próprias, que também são tipificadas. Existe também um conjunto de eventos associado a cada estado, que permite informar outras entidades no sistema de algo relacionado com esse estado do objecto. Os eventos podem ter, ou não, **parâmetros** associados. Os componentes podem ser configurados, quando são criados, através de **propriedades** que fornecem valores iniciais para as variáveis de estado internas.

Exemplificando, pode haver estados para Velocidade, Geometria, Volume, entre outros. Associados ao estado Volume, ter-se-ão acções tais como `SetVolume` e `GetVolume`, e eventos tais como `Ev_Volume`, para informar que o volume do som foi modificado, e qual o valor com que ficou. Se um dado objecto não suportar o estado Volume, não poderá interagir com o exterior para modificar o valor que controla o volume do som que produz.

Em termos de modelo computacional, um estado não introduz nenhum conceito novo, mas apenas uma forma de relacionar *interfaces*. Um estado constitui um conjunto fechado de características associadas com alguma particularidade do sistema. No entanto, é possível que a interacção com um estado possa influenciar outros estados do mesmo objecto. Isso é um aspecto interno ao objecto, tendo o programador a liberdade de poder implementar as interdependências que quiser, desde que os mecanismos de comunicação sejam suportados. Um exemplo de influência cruzada é um estado relacionado com a largura de banda disponível no canal (estado qualidade de serviço), que pode modificar a geometria com que o objecto é apresentado para reduzir a quantidade de informação que é necessário transferir, originando a geração de um evento de mudança de geometria a ser enviado para todos os objectos no sistema que tenham

registado interesse em ser informados de mudanças na geometria. Outro aspecto interno ao objecto, é como se pode usar herança para constituir o estado. Mais uma vez, esta é uma possibilidade relacionada com a construção dos objectos que não é relevante para a sua utilização.

Na figura 3.2 está exemplificado um componente com três estados, cada um com um *interface* com acções associadas e podendo gerar eventos para o exterior. Os eventos podem ser gerados por iniciativa própria dos objectos, por exemplo à medida que o tempo passa, ou como resultado da invocação de acções do seu *interface*.

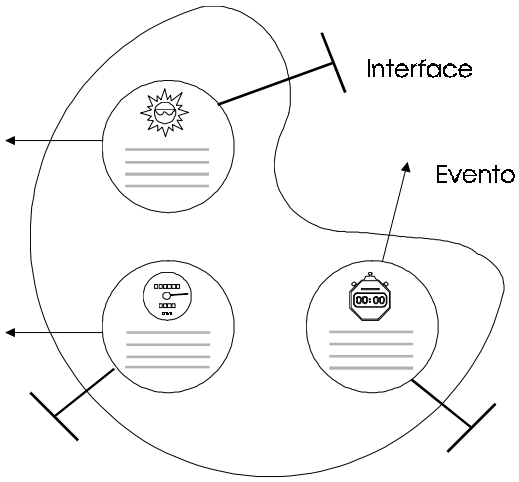

Figura 3.2: Um componente com três estados.

O *interface* completo de um objecto multimédia, ou componente, é o conjunto de todos os *interfaces* dos estados que ele suporta.

O conceito de estado é muito útil pelas seguintes razões:

- Evita a necessidade de conceber um *interface* comum a todos os objectos para os utilizar de forma eficiente. Não é necessário definir funções vazias quando elas não tiverem significado (por exemplo, `SetVolume` para diapositivos), nem redefini-las (*overload*) com funções de significados menos intuitivos.
- Evita a necessidade de prever um *interface* que seja compatível com qualquer evolução futura da tecnologia dos componentes. Com este modelo, basta acrescentar um estado novo.

- Permite que ferramentas feitas numa certa versão do sistema possam ainda trabalhar com objectos que tenham funcionalidades ainda não conhecidas nessa altura.
- Os sistemas multimédia podem ser construídos por etapas, acrescentando-se objectos e funcionalidades novas à medida das necessidades.

Tal como os estados, eventos e acções são entidades tipificadas, também os componentes têm tipos associados. A um tipo de componente corresponde, não só um determinado conjunto de portos disponíveis, como ainda um conjunto de estados suportados e a lógica que os relaciona em termos de quando são gerados os eventos. Por exemplo, um *source* vídeo que nunca gere o evento correspondente ao fim do filme, deixando essa tarefa para o *sink*, pertencerá a um tipo diferente de um *source* que envia esse evento quando envia a última imagem do filme, mesmo que os portos sejam iguais em número e tipo.

Esta informação associada ao tipo é útil para decidir quando é que dois componentes podem ser interligados, permitindo verificações mais fortes do que apenas compatibilidade de tipos de sinais multimédia.

3.3 OBJECTOS MULTIMÉDIA

O objecto multimédia é a entidade base utilizada na especificação de uma composição. Ele é constituído por um certo número de objectos componentes interligados e, tal como estes, oferece para o exterior um *interface* constituído pela reunião dos *interfaces* dos estados que ele suporta. Desta forma, os objectos multimédia têm características semelhantes aos objectos componentes, excepto que não produzem nem consomem sinais multimédia.

Os objectos multimédia também são entidades tipificadas. Um tipo de objecto multimédia representa um conjunto de estados suportados e um conjunto de tipos de componentes que podem ser interligados para fornecer esse serviço. Por exemplo, um vídeo de certo tipo pode ter como origem um ficheiro em disco, e ser apresentado no écran de um computador. O *sink* também poderia ser substituído por um aparelho de televisão se ele tivesse o mesmo conjunto de estados que interagissem da mesma maneira, isto é, se fossem ambos componentes do mesmo tipo. No entanto, se o *source*

passasse a ser uma câmara de vídeo, que não suporta mudanças de velocidade (câmara lenta), seria classificado como um objecto multimédia de tipo diferente.

Tal como foi ilustrado anteriormente na figura 3.1, um objecto multimédia tem, para além dos vários componentes que o constituem, um ***dispatcher*** que tem como responsabilidade encaminhar as acções recebidas para os componentes adequados. O que ele faz na realidade é **delegar** a execução das acções nos componentes que tenham os estados correspondentes. Por exemplo, as acções para começar e terminar a apresentação, e a acção para destruir o objecto são enviadas para todos os componentes, mas acções como mudança de velocidade estão normalmente num único componente que passa a produzir dados a um ritmo diferente, sendo acompanhado pelos outros componentes.

Esta delegação é transparente para quem usa o objecto porque quando é pedida uma referência para o *interface* de um determinado estado a um objecto multimédia, o que ele faz é devolver directamente uma referência para o estado do componente onde o estado está implementado, se isso for possível.

3.4 COMPOSIÇÃO E SINCRONIZAÇÃO

Comparando as possibilidades de composição do modelo de objectos definido com os modelos apresentados na secção 2.1.1 (página 5), verifica-se que se tem aqui dois modelos de composição:

- Um **modelo de composição de objectos**, que descreve a configuração dos objectos multimédia em termos de componentes e os fluxos de dados multimédia entre eles. Este modelo facilita mais o trabalho dos autores de composições do que o modelo de composição de fluxos de dados do capítulo 2, pois classifica os objectos no sistema em objectos multimédia e componentes. Os objectos multimédia são os objectos que o autor usa na descrição da apresentação, escondendo-se a distribuição. Os componentes encapsulam as dependências do equipamento, produzindo e consumindo dados multimédia.
- Um **modelo de composição de controlo**, que descreve o funcionamento da aplicação em termos de relações entre eventos e invocações de acções. Este modelo é

muito geral e engloba as possibilidades do modelo de composição de actividades e do modelo de composição temporal apresentados no capítulo 2.

Estes dois modelos de composição prestam-se facilmente à composição visual de aplicações, como será visto no capítulo 5.

Comparando agora as possibilidades de sincronização do modelo de objectos definido com os modelos de sincronização apresentados na secção 2.1.2 (página 8), verifica-se que se tem apenas um modelo de sincronização.

Este é um modelo de sincronização baseado em **pontos de sincronização**. Os objectos geram eventos que estão de alguma forma relacionados com o seu estado interno, e por consequência com os sinais multimédia que produzem ou consomem. A especificação da sincronização corresponde a definir as acções a realizar quando os eventos são recebidos.

O modelo de sincronização hierárquica apresentado no capítulo 2 tem a limitação de considerar os objectos multimédia como entidades indivisíveis, impossibilitando a sincronização entre o início e o fim da apresentação. Esta limitação não existe no modelo baseado em pontos de sincronização, pois os objectos podem gerar eventos em qualquer altura.

O modelo de sincronização baseada num eixo temporal apresentado no capítulo 2 permite sincronização durante o funcionamento dos objectos, mas não permite modelar facilmente não determinismo, ao passo que o modelo baseado em pontos de sincronização permite.

O modelo de sincronização baseado em pontos de referência apresentado no capítulo 2 tem a desvantagem de só prever relações temporais entre os vários objectos, não prevendo relações que não sejam temporais.

Os modelos baseados em redes de Petri também têm algumas limitações, que foram em parte ultrapassadas por diversas extensões ao modelo inicial. No entanto, mantêm-se as dificuldades de exprimir acontecimentos não determinísticos que obrigam a sobrecarregar a rede com transições para todas as possibilidades. Além disso, é difícil exprimir relações entre vários objectos que não sejam puramente temporais.

O modelo de sincronização baseado em pontos de sincronização que se apresentou assume que os objectos geram eventos de forma coerente, o que obriga a ter mecanismos para detectar e corrigir inconsistências. Estas verificações devem sempre

que possível ser realizadas durante a fase de autoria da apresentação, mas algumas só poderão ser detectadas durante a execução.

3.5 ESTADOS, ACÇÕES E EVENTOS

De entre os vários estados já definidos, apresentam-se nesta secção os mais relevantes.

O estado Contexto, porque é o único obrigatório nos objectos multimédia e componentes, e oferece mecanismos para localizar os outros estados suportados.

Os estados Vida e VidaComponentes pois são necessários para a criação de, respectivamente, objectos multimédia e componentes.

O estado Anotação, pois permite formas de sincronização temporal e lógica durante a apresentação do objecto.

3.5.1 Estado Contexto

Todos os componentes e objectos multimédia suportam o estado Contexto, que oferece as funções básicas necessárias para o funcionamento dos objectos no sistema. Consoante a sua natureza podem suportar outros estados.

Desta forma, um objecto pode ser identificado univocamente por uma referência para o *interface* deste estado.

Além disso, é o estado Contexto que permite esconder a distribuição interna do objecto.

Ao estado Contexto estão associadas as acções:

```
Play()
Stop()
Destroy()
Inform()
receiveEvent()
registerStatusInterest()
unregisterStatusInterest()
```

A acção `Play` inicia os fluxos de dados multimédia, começando a apresentação desse objecto. A acção `Stop` termina um `Play`. A acção `Destroy` termina o objecto de forma normal, libertando os recursos reservados.

A acção `Inform` permite que se obtenha uma lista de referências para os *interfaces* correspondentes aos estados fornecidos como parâmetro. Quando é criado um novo objecto, só é devolvida uma referência para o estado Contexto. Invocando esta acção permite-se saber quais referências para as outras funcionalidades oferecidas pelo objecto. Notar que o conhecimento de que essas funcionalidades de facto existem, encontra-se no gestor de tipos (descrito no próximo capítulo). Desta forma, permite-se esconder a organização interna do objecto, sendo devolvida pelo *dispatcher* do objecto multimédia, uma referência directamente para o componente onde o estado se encontra implementado. Um utilizador só usa as funcionalidades do objecto que compreende, reduzindo-se deste modo a necessidade de evoluções constantes de todo o sistema, quando um pormenor é mudado.

A acção `receiveEvent` permite que o objecto receba eventos vindos do exterior, sendo invocada sempre que alguma entidade no sistema lhe envia um evento. Nesta acção é implementado o processamento de eventos, de forma a que o objecto lhes possa reagir. Notar que os eventos são uma das formas de comunicação. Pode também haver uma chamada directa a uma acção.

A acção `registerStatusInterest` permite registar interesse em receber todos os eventos de um conjunto de estados. É fornecido como parâmetro uma referência para o objecto que deve receber os eventos correspondentes a cada estado, com o objectivo de os eventos poderem ser enviados para qualquer objecto no sistema, e não apenas para o gestor de sincronização. Isto permite prever o facto de posteriormente se poder ter mais do que um gestor de sincronização, ou se ter uma gestão descentralizada. A acção `unregisterStatusInterest` permite cancelar o efeito da acção anterior.

Para além desta acção, podem existir outras noutros estados, que permitam uma selecção mais fina de quais os eventos do estado correspondente que se pretende receber, de forma a reduzir o número de eventos que é necessário enviar durante a execução. Optou-se por dividir as funcionalidades desta forma, pois uma selecção parcial de eventos pode obrigar a verificações de consistência demasiado complexas para estarem num estado obrigatório, como é o caso do estado Contexto. Um exemplo disso, é o estado Anotação, no qual a selecção de eventos muda o aspecto qualitativo do objecto, pelo que exige um controlo mais fino da selecção de eventos. No entanto, esta selecção para o caso das anotações não pode ser feita apenas por tipo de evento, pois

todos os eventos são do tipo `Ev_Annotation`, o que obriga a fornecer também o tipo de anotação e eventualmente os parâmetros.

Ao estado Contexto estão associados os eventos:

```
Ev_Ready
Ev_Start
Ev_Stop
Ev_End
```

O evento `Ev_Ready` serve para anunciar que a criação do objecto foi completada, e é gerado apenas nos casos em que foi realizada assincronamente. O evento `Ev_Start` indica que foi feito `Play`, e o evento `Ev_Stop` indica que foi feito `Stop`. O evento `Ev_End` indica que um `Play` terminou por se ter chegado ao fim normal do objecto.

3.5.2 Criação de Componentes e Objectos Multimédia

Para criar uma nova instância de um objecto multimédia, é necessário num primeiro passo criar novas instâncias de todos os componentes, e num segundo passo interligar os seus portos.

A função de criação de novas instâncias de objectos multimédia é realizada por servidores de objectos multimédia, que suportam apenas o estado Vida, que oferece funções para criação e destruição de instâncias de objectos multimédia, e não tem nenhum tipo de eventos. Esta é uma técnica semelhante ao conceito de fábrica de [Ansa 93].

A acção para criar uma nova instância de um objecto multimédia, `prepare`, tem como parâmetros principais a identificação do utilizador, a lista de componentes *sources*, filtros e *sinks*, a topologia da rede de interligação dos componentes sob a forma de uma lista de ligações a efectuar entre os seus portos, e ainda uma lista de propriedades do objecto que têm significado apenas para o próprio objecto e permitem parametrizá-lo. Exemplos de propriedades são: nomes de ficheiros, geometria inicial do objecto, volume inicial do som.

A operação `prepare` devolve, como resultado, uma referência para o estado Contexto do objecto criado, que permite identificar univocamente o objecto no sistema. Se o utilizador permitir, e a criação do novo objecto não puder ser executada

imediatamente, pode ser realizada assincronamente, caso em que, quando tiver sido completada, o novo objecto envia um evento `Ev_Ready` para o utilizador.

A acção para destruição de objectos, `dismiss`, permite abortar as instâncias criadas, servindo apenas para recuperação em caso de erros.

De forma semelhante à criação de objectos multimédia, a função de criação de novas instâncias de componentes é realizada por servidores de componentes. Os servidores de componentes suportam apenas o estado VidaComponentes, que oferece funções para criação e destruição de instâncias de objectos componentes. O estado VidaComponentes também não tem eventos.

A acção para criar uma nova instância de um componente multimédia, `prepare`, tem como parâmetros principais a identificação do utilizador, que neste caso será o próprio objecto multimédia, a lista de portos que se pretende utilizar, e ainda uma lista de propriedades do componente que têm significado apenas para o próprio componente e permitem parametrizá-lo. Como resultado, são devolvidas referências para o estado Contexto do componente criado e ainda para os estados Porto correspondentes a cada um dos portos seleccionados do componente.

O estado Porto é suportado por cada componente servindo de base à definição de estados para portos de entrada e portos de saída específicos de cada tipo de porto. Na classe base Porto estão apenas definidos métodos `connect` e `disconnect`, que permitem interligar portos, tendo como parâmetro a lista de referências para os portos a que se pretende ligar.

Desta forma, para criar um novo objecto multimédia, o servidor de objectos multimédia localiza no sistema os servidores dos componentes necessários à construção do novo objecto, e cria instâncias de todos os componentes seleccionando os portos referidos na topologia. Seguidamente liga os portos de acordo com a mesma topologia e termina devolvendo uma referência para o estado Contexto do objecto multimédia criado. Nesta altura, o novo objecto está pronto a funcionar, podendo o utilizador obter referências para os restantes estados do objecto invocando a acção `Inform`, registar interesse na recepção de eventos invocando `registerStatusInterest` e começar a sua apresentação invocando a acção `Play`.

3.5.3 Estado Anotação

Os objectos mais simples não têm este estado, pelo que os únicos eventos relacionados com a posição da apresentação são os do estado Contexto: `Ev_Start` para identificar o início e `Ev_End` para identificar o fim, tal como se exemplifica na figura 3.3.

Figura 3.3: O objecto multimédia mais simples em termos de composição.

O estado Anotação está relacionado com a posição da apresentação de um objecto, servindo para diversificar os tipos de composição possíveis, permitindo a interacção entre objectos a nível temporal e lógico.

Criando anotações nos objectos multimédia consegue-se não só uma forma de sincronização durante o funcionamento desses objectos, mas também uma forma de subdividir o objecto em partes que podem ser apresentadas independentemente.

Quando o objecto está a tocar e chegar à posição correspondente a uma anotação é enviado o evento `Ev_Annotation` para todos os objectos no sistema que registaram interesse em recebê-lo. Pode-se registar interesse em receber todos os eventos anotação invocando a acção `registerStatusInterest` do estado Contexto, ou registar interesse em receber apenas um certo subconjunto das anotações existentes, caso em que se invoca a acção `registerEventAnnotation` do estado Anotação, com a lista ordenada das anotações que se pretende receber. No caso de se registar interesse simultaneamente em mais do que uma anotação, será verificado pelo objecto se a ordem fornecida é consistente com o seu funcionamento interno, e em caso negativo é gerado o evento `Ev_RegisterAnnotationError` como resposta.

Uma anotação pode ser classificada num de vários tipos, estando associado a cada tipo um domínio de valores de parâmetros que são diferentemente interpretados. Na tabela 3.1 apresentam-se alguns exemplos de tipos de anotações e parâmetros. Por exemplo: se numa dada sequência de imagens, a passagem do carro amarelo estiver

associada à imagem 120, então o evento anotação correspondente seria, por exemplo, do tipo `Label` e teria o valor "CarroAmarelo", sendo responsabilidade do objecto fazer a conversão do valor do parâmetro para a correspondente posição no objecto. Outro tipo de anotação possível é o tipo `Locator`, que tem significado físico, pelo que à posição anteriormente referida também poderia corresponder uma anotação de tipo `Locator` com parâmetro 120.

Tipo	Parâmetros		
Label	"CarroAmarelo"	"CarroVermelho"	"Avião"
Etiqueta	"Capítulo1"	"Capítulo2"	"Capítulo3"
Locator	"0", "1", "2", "3", ...		

Tabela 3.1: Exemplo de vários tipos de anotações com alguns parâmetros.

O estado Anotação suporta ainda uma acção `getLocator`, que devolve informação sobre a posição actual da apresentação do objecto, permitindo, por exemplo, a um editor de anotações associar novas anotações de diversos tipos a um objecto.

Na figura 3.4 apresenta-se um exemplo de um objecto multimédia, que para além de ter os eventos `Ev_Start` e `Ev_End` correspondentes ao estado Contexto, tem ainda algumas anotações associadas, que permitem saber com maior precisão qual a posição da apresentação do objecto, ou ainda tocar apenas uma parte, por exemplo o capítulo 2.

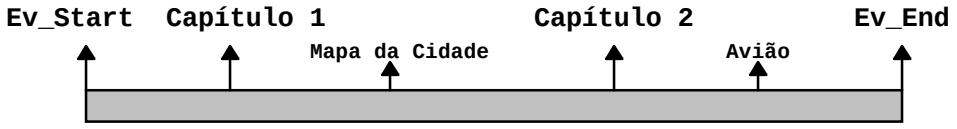

Figura 3.4: Um objecto multimédia com vários tipos de anotações.

3.5.4 Extensibilidade

Para permitir a evolução do sistema, acrescentando-lhe funcionalidades, tem de se poder adicionar novos estados à medida das necessidades. A informação sobre os estados existentes no sistema é guardada num gestor de tipos, descrito no próximo

capítulo, permitindo-se aos programas existentes (editor gráfico e compilador) saber quais as possibilidades oferecidas em cada momento ao autor de uma aplicação multimédia.

Ter um estado implica implementar todas as acções a ele associadas, podendo-se, no entanto, não implementar todos os eventos associados a esse estado, caso em que são ignorados pelo objecto os pedidos de registo de interesse nos eventos não implementados.

Na tabela 3.2 apresentam-se exemplos de tipos de objectos multimédia e os estados que eles poderiam suportar.

Objecto	**Estados Suportados**		
Texto	Contexto Geometria	Anotação	Click
Áudio	Contexto Velocidade	Anotação Volume	Pausa QOS
Vídeo	Contexto Pausa QOS	Anotação Velocidade	Click Geometria
Diapositivos	Contexto Geometria	Anotação Brilho	Click Contraste
Botão	Contexto	Click	
Relógio	Contexto	Anotação	

Tabela 3.2: Alguns tipos de objectos multimédia e os estados associados.

Por motivos de optimização, alguns objectos podem ser criados recorrendo aos mecanismos definidos sem ter de recorrer a novos estados. Por exemplo, um objecto temporizador simples podia ser um objecto que só tivesse o estado vida e que na inicialização com `prepare` tinha um parâmetro que definia a duração do temporizador. Quando o tempo estabelecido tivesse passado, o objecto terminava enviando um evento `Ev_End`. Para ter temporizadores mais elaborados poder-se-ia recorrer ao estado Anotação, tendo as anotações significado de instantes de tempo.

3.6 GESTOR DE SINCRONIZAÇÃO

O gestor de sincronização é um programa, criado de forma automática por compilação de um *script*, que coordena o funcionamento dos objectos multimédia de maneira a que eles respeitem a composição expressa nesse *script*.

O gestor de sincronização tem a responsabilidade de criar instâncias dos objectos multimédia necessários à apresentação, devendo para tal contactar com o servidor de objectos e fornecer-lhe a lista de componentes que constituem cada objecto, a topologia das interligações entre eles, e os valores das propriedades para parametrizar os objectos. Seguidamente, o gestor de sincronização regista interesse junto de cada objecto em receber os eventos relevantes para a composição. Note-se que estas duas operações devem ser efectuadas apenas durante a inicialização de uma apresentação, pois demoram algum tempo, e causariam atrasos evitáveis se fossem feitas depois de iniciada a apresentação.

Depois de completados estes dois passos de inicialização, o gestor de sincronização funciona como uma máquina de estados que invoca acções sobre os objectos, e aguarda a recepção de eventos. Cada evento recebido pode desencadear novas invocações de acções e mudanças de estado.

Quando a composição chegar ao fim, o gestor de sincronização destrói os objectos multimédia para libertar os recursos reservados.

O gestor de sincronização também se pode comportar como um objecto multimédia, oferecendo um interface constituído por estados, de forma a poder ser controlado por um painel de controlo, ou mesmo utilizado noutras composições, sob as ordens de outro gestor de sincronização, formando assim uma **hierarquia** de gestores de sincronização.

Para além do estado Contexto, o gestor de sincronização pode ainda ter, por exemplo, o estado Pausa para permitir parar temporariamente toda a aplicação, o estado Velocidade para mudar a velocidade da apresentação como um todo, ou fazê-la andar para trás, e o estado Anotação para passar para o exterior algumas das anotações dos objectos internos.

Usando esta perspectiva de controlo centralizado, simplificam-se os objectos multimédia, pois eles nunca precisam de receber e interpretar eventos, tendo apenas de executar acções e gerar eventos. Além disso, o gestor de sincronização tem conhecimento sobre todos os objectos sob o seu domínio, reduzindo inconsistências causadas pela distribuição.

Uma outra alternativa para a sincronização dos objectos seria ter um controlo totalmente descentralizado, em que cada objecto multimédia teria o seu próprio gestor de sincronização, que seguiria apenas a parte do *script* que lhe dissesse respeito directamente, interagindo com os outros objectos no sistema e os seus gestores de sincronização. Esta alternativa foi apresentada em [Bernardo 94].

CAPÍTULO 4
GESTOR DE NOMES E GESTOR DE TIPOS

Neste capítulo apresentam-se os gestores de nomes e de tipos, que são fundamentais para o funcionamento num sistema distribuído e permitem que o modelo de objectos seja facilmente extensível, pois as aplicações e ferramentas existentes podem saber em cada momento quais as possibilidades oferecidas pelo sistema sem terem de ser modificados.

4.1 GESTOR DE NOMES

O gestor de nomes oferece um serviço de localização dos objectos no sistema distribuído. Os objectos são conhecidos por nomes, e é função do gestor de nomes resolver o nome para a referência que o objecto tem no sistema. Os objectos que pretendam ser conhecidos no sistema devem exportar o seu nome e uma referência para o seu *interface* que querem anunciar para o gestor de nomes, de modo a poderem ser localizados no sistema, e utilizados por qualquer aplicação que deles necessite. Além do nome e da referência, os objectos também podem anunciar atributos que permitam uma selecção mais fina. Um exemplo de atributo é o nome da máquina em que se encontram.

Já tinha sido referido que para criar novos objectos multimédia era necessário contactar um servidor de objectos para criar uma nova instância do objecto pretendido. Este servidor de objectos, por sua vez, contacta com os servidores de componentes para criar novas instâncias dos componentes necessários. Assim, os servidores de objectos exportam o seu nome, que é o nome dos objectos que sabem criar, e uma referência para o estado Vida, para o gestor de nomes. Os servidores de componentes exportam o seu nome e uma referência para o estado VidaComponentes.

Também os próprios objectos multimédia podem anunciar uma referência para o seu estado Contexto, para poderem ser utilizados por outras entidades no sistema para além da entidade que os criou. Exemplos disso são o controlo directo dos objectos por painéis de controlo, e a construção de hierarquias de gestores de sincronização.

O gestor de tipos, descrito na próxima secção, também exporta o seu nome e *interface* para o gestor de nomes para poder ser consultado por qualquer entidade no sistema.

Ao exportar uma referência para o gestor de nomes, é-lhe associado um tipo, que é o tipo do estado correspondente ao *interface*, e ainda um conjunto de propriedades com o objectivo de facilitar a selecção do *interface* correcto. São utilizadas propriedades com o nome da máquina onde se encontra o objecto, o instante de criação do objecto para permitir seleccionar a versão mais recente, e quando for o caso, o tipo do objecto multimédia ou componente.

O modelo de objectos foi implementado sobre os serviços de comunicação entre processos oferecidos pelo ANSA [Ansa 93], sendo o gestor de nomes realizado com o Trader do ANSA.

4.2 GESTOR DE TIPOS

Ao desenvolver programas, a produtividade dos programadores e a confiança no sistema é melhorada quando se utilizam linguagens que oferecem alguma forma de sistema de tipos. De facto, ao detectar antecipadamente os erros, ganha-se em produtividade e confiança.

Os tipos são sujeitos às seguintes utilizações [Campin+ 91]:

- **Definição**, podendo ser incluídos como parte intrínseca na linguagem, ou introduzidos (implícita ou explicitamente) durante a compilação.
- **Associação** a expressões ou valores.
- **Verificação**, em termos de equivalência, subtipificação e compatibilidade.

Algumas linguagem, sendo Pascal um exemplo, apenas usam os tipos durante a compilação, tendo uma verificação de tipos estática. Outras, como alguns dialectos de Lisp, deixam todas as operações de verificação para a fase de execução, tendo uma verificação de tipos dinâmica. Uma linguagem que pretenda suportar tipificação dinâmica

deve, idealmente, suportar as duas variantes de modo a apanhar os erros resultantes da utilização estática de tipos (provavelmente a maioria) o mais cedo possível.

Linguagens com tipificação dinâmica requerem que tanto o compilador como o sistema de suporte à execução tenham acesso aos tipos. Neste caso, é sensato factorizar o código correspondente às três utilizações acima indicadas num módulo separado designado por gestor de tipos.

Ao dissociar a gestão de tipos do compilador e do sistema de suporte à execução, levanta-se um novo problema: como armazenar os tipos mantendo-os facilmente acessíveis. Nos sistemas tradicionais, os tipos estão agrupados como parte do programa, e não têm existência independente. Se estiverem armazenados separadamente, devem poder ser novamente agrupados, quando necessário, com dois objectivos. Em primeiro lugar, no caso de tipificação dinâmica, devem estar acessíveis para executar os programas. Em segundo lugar, devem estar disponíveis para ser reutilizados em novos programas. Linguagens tais como Modula-2, que colocam informação sobre a definição de *interfaces* em ficheiros separados, oferecem uma solução parcial, dependente da linguagem, para este problema.

Para resolver estes problemas de forma geral, incluiu-se no sistema de desenvolvimento de aplicações multimédia um gestor de tipos. O gestor de tipos tem como função armazenar e gerir, a nível global ao sistema, a informação sobre os tipos existentes e relações entre eles, tais como saber quais os estados suportados por cada objecto, saber quais as acções e eventos correspondentes a cada estado, e quais os tipos dos parâmetros das acções e dos eventos.

Colocou-se no gestor de tipos toda a informação necessária para maximizar o grau de verificações feitas durante a edição e compilação, e minimizar o número de verificações a fazer pelos próprios objectos durante a execução.

O gestor de tipos é utilizado por todo o sistema:

O editor gráfico utiliza a informação guardada no gestor de tipos para saber quais os tipos de objectos disponíveis ao autor de uma apresentação, e as suas características, e ainda para validar possíveis novos objectos que sejam criados a partir dos componentes existentes.

O compilador recorre ao gestor de tipos para fazer verificações de consistência durante a compilação, evitando que muitos erros ocorram durante a execução. A geração

do código correspondente a novos estados, ainda não conhecidos aquando da implementação do compilador, é também auxiliada pelo gestor de tipos.

O gestor de sincronização e o *dispatcher* dos próprios objectos usam o gestor de tipos para obter os códigos dos tipos e fazer verificações durante a execução.

Para permitir compatibilidade com aplicações já desenvolvidas, basta que não se modifiquem tipos existentes, acrescentando-se tipos novos quando for necessário. Assim, como optimização, é possível incluir directamente nas várias aplicações existentes no sistema os códigos dos tipos utilizados e informação sobre as assinaturas das acções e eventos que já estão definidas, de forma a tornar a execução mais rápida, pois desta forma só será necessário consultar o gestor de tipos para obter informação sobre os tipos acrescentados posteriormente ao sistema.

Desta forma, quando o compilador constrói uma nova aplicação multimédia pode fazer todas as consultas necessárias ao gestor de tipos, colocando a informação necessária no programa criado, e evitar assim que sejam feitas consultas ao gestor de tipos durante a execução, melhorando o desempenho das aplicações criadas.

O gestor de tipos guarda informação sobre os seguintes tipos existentes no sistema:

- tipos de objectos multimédia
- tipos de componentes
- tipos de sinais de entrada/saída
- tipos de estado
- tipos de acções
- tipos de eventos
- tipos de anotações

O gestor de tipos oferece acções para pesquisa de informação sobre um dado tipo, para obtenção de listas de tipos, para modificação da informação disponível, e ainda para verificações de compatibilidade de tipos para serem usadas pelo editor gráfico e pelo compilador.

Com o gestor de tipos permite-se uma fácil extensibilidade do sistema pois as aplicações podem ser escritas independentemente dos tipos que existam no sistema, não sendo necessário modificá-las quando se acrescentam novos tipos. Na realidade, as

aplicações quando em execução (editor gráfico e compilador) consultam o gestor de tipos obtendo a informação necessária. Como exemplo, mostra-se na figura 4.1 o menu para modificação das propriedades de um objecto de som no editor gráfico. O editor gráfico pergunta ao gestor de tipos quais são as propriedades suportadas pelos objectos de som, obtendo ainda os valores de defeito para cada propriedade construindo assim o menu apresentado. Destas propriedades importa realçar a `anotfile` que é o nome do ficheiro com as anotações deste objecto, que lhe permite suportar o estado anotação sabendo quais são os tipos de anotações que pode gerar e quais as posições correspondentes em termos do número da amostra de som. Também a propriedade `node` é importante, pois permite indicar qual o nó onde preferencialmente deve ser procurado o servidor de objectos no sistema, para o caso de haver mais do que um disponível no gestor de nomes, possibilitando assim uma escolha do nó onde o objecto será criado.

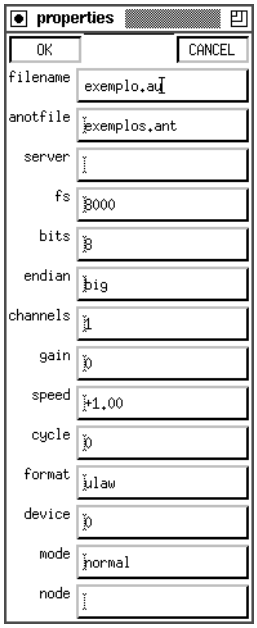

Figura 4.1: Menu para a modificação das propriedades de um objecto de som no editor gráfico.

CAPÍTULO 5
LINGUAGEM

Neste capítulo apresenta-se a linguagem *script* definida, e o editor gráfico que permite a um autor criar *scripts* sem ter de conhecer a linguagem. O editor não suporta todas as possibilidades oferecidas pela linguagem, permitindo no entanto criar facilmente uma grande variedade de aplicações. O *script* é lido pelo compilador para produzir o programa Gestor de Sincronização.

5.1 LINGUAGEM UTILIZADA

Uma boa linguagem deve ter as seguintes características:

- **Expressividade**. A linguagem deve poder descrever uma larga gama de aplicações.
- **Eficiência**. A linguagem deve poder produzir código eficiente numa grande variedade de arquitecturas.
- **Ajudar o programador**. A linguagem deve ter uma sintaxe simples, uma semântica compreensível, e ser de fácil aprendizagem. A linguagem deve ajudar o programador nas suas tarefas de especificação, projecto, implementação, verificação e validação de aplicações complexas.
- **Abstracta** e **formal**. A linguagem deve ser suficientemente abstracta para ser independente da implementação. Deve ser formal para oferecer uma descrição da aplicação precisa e sem ambiguidades. Além disso, sendo formal, o significado das expressões é mais claro sendo mais fácil construir compiladores.

Para cumprir estes objectivos escolheu-se uma linguagem baseada na álgebra de processos CSP [Hoare 85], com uma sintaxe semelhante à da linguagem Occam [Galletly 90].

Um programa está dividido em duas partes principais, a primeira correspondente ao modelo de composição de objectos que descreve a configuração dos objectos, e a segunda correspondente ao modelo de composição de controlo que descreve o funcionamento da aplicação.

```
SPECIFICATION NomeEspecificação DefiniçãoExterna;
  DefiniçãoObjectosMM

  DefiniçãoAlias
  DeclaraçãoVariáveis
  DefiniçãoProcessos

BEHAVIOUR
  ExpressõesComportamento
ENDSPEC;
```

Figura 5.1: Estrutura da linguagem.

Na figura 5.1 apresenta-se a estrutura da linguagem. A secção `Definição-Externa` serve para declarar a visibilidade exterior de estados, eventos e acções de forma a permitir a utilização da especificação noutras especificações. A secção `DefiniçãoObjectosMM` declara os objectos multimédia que são utilizados nesta especificação, identificando as suas propriedades, os objectos componentes, e as suas interligações. Note-se que todos os objectos declarados nesta secção são criados e inicializados automaticamente antes de a aplicação começar. A secção `Definição-Alias` permite definir nomes alternativos para operadores com o objectivo de simplificar a sintaxe. A secção `DeclaraçãoVariáveis` serve para declarar variáveis globais. A secção `DefiniçãoProcessos` serve para atribuir nomes a processos com o objectivo de facilitar a estruturação da programação e a reutilização de código. Esta secção tem uma estrutura semelhante à estrutura da linguagem. A secção `ExpressõesComportamento` corresponde à descrição do funcionamento da aplicação em termos de relações entre eventos e invocações de acções.

O comportamento das aplicações é expresso, à semelhança do que acontece em CSP, como um conjunto de processos sequenciais que podem ser executados concorrentemente com outros processos. Os processos podem interagir, ou comunicar, com outros processos através de operações de entrada e saída síncronas.

Os processos primitivos são, à semelhança de CSP (na linguagem adoptou-se a sintaxe de CSP):

- Processos de saída, que correspondem a invocar uma acção de um objecto multimédia:

```
objecto ! acção(parâmetros, ...)
```

- Processos de entrada, que correspondem a aguardar que um evento chegue de um objecto multimédia:

```
objecto ? evento(parâmetros, ...)
```

- Processos de atribuição de variáveis:

```
variável:= expressão
```

Há ainda um outro processo primitivo: ABORT, que aborta toda a especificação, destruindo todos os objectos multimédia.

De acordo com o modelo definido na linguagem CSP, as entradas e as saídas são síncronas. Para evitar que o gestor de sincronização ou um objecto multimédia fique bloqueado numa operação de entrada ou saída, os objectos devem estar sempre prontos a receber um pedido de invocação de uma acção. Além disso, a maior parte das acções são invocações de operações que retornam imediatamente, sendo o eventual retorno de parâmetros feito usando o mecanismo de eventos.

Estes processos primitivos podem ser agrupados em construções de mais alto nível formando uma estrutura hierárquica de processos (apresenta-se também a sintaxe correspondente de CSP):

- Composição sequencial de processos, que corresponde a um conjunto ordenado de processos que são executados pela ordem dada:

```
SEQ p1; p2; ... pN; END
```
Em CSP: `p1; p2; ... ; pN`

- Composição paralela de processos, que corresponde à execução simultânea de um conjunto de processos:

```
PAR p1; p2; ... pN; END
```
Em CSP: `p1 ‖ p2 ‖ ... ‖ pN`

- Composição alternativa de processos, que corresponde à execução de apenas um processo. A cada processo está associada uma guarda que pode ser um processo de entrada, ou uma condição booleana e um processo de entrada. No caso de mais de uma

guarda estar pronta, é seleccionado apenas um processo para ser executado, de forma não determinística:

```
ALT g1: p1; g2: p2; ... gN: pN; END
```

Em CSP: `(g1→p1) □ (g2→p2) □ ... □ (gN→pN)`

- Execução condicional, que corresponde a executar um de dois processos consoante uma dada condição tomar o valor lógico verdadeiro ou falso:

```
IF condição THEN
  p1;
ELSE
  p2;
```

Em CSP: `p1 ◁ condição ▷ p2`

- Execução repetitiva, que corresponde a executar repetidamente o mesmo processo enquanto uma dada condição for verdadeira:

```
WHILE condição DO
  p1;
```

Em CSP: `condição * p1`

Para facilitar a programação foram definidas algumas simplificações sintácticas, sendo a principal a correspondente à activação de um objecto ou zona de um objecto, que pode ser feita por simples indicação do seu nome, por exemplo:

```
objecto
```

que será convertida em:

```
SEQ
  objecto ! Play(Ev_Start, Ev_End);
  objecto ? Ev_End;
END
```

Repare-se que a acção `Play` activa o objecto, e retorna imediatamente, sendo gerado o evento `Ev_End` quando o `Play` terminar. No caso de activação de uma zona de um objecto, apenas os parâmetros do `Play` diferem.

É ainda possível executar processos previamente declarados como se fossem processos elementares usando simplesmente o seu nome.

A composição paralela pode ser parametrizada em termos da:

- **Semântica de terminação**. Um paralelo pode terminar quando todos os processos terminarem, quando o primeiro processo terminar, ou quando terminar um certo conjunto deles, dado por uma conjunção e disjunção de processos.
- **Influência na duração dos objectos**. Os objectos podem manter a sua duração original, ficar todos com a duração do maior, do mais pequeno, ou com uma duração dada.
- **Influência entre os objectos**. Os objectos enquanto activos devem trocar informação sobre um certo estado, por exemplo velocidade ou geometria, mantendo uma certa relação entre eles, por exemplo a mesma velocidade ou geometria. Esta possibilidade está relacionada com sincronização de baixo nível, pelo que os objectos devem interagir directamente, sem passar por um gestor de sincronização, por razões de eficiência.
- **Vida dos processos**. Quando um paralelo terminar, os processos activos são mortos, ou um determinado conjunto não é morto.
- **Informação espacial**. Para os objectos com representação espacial podem ser definidas relações sobre a posição espacial ou relações hierárquicas.

Uma composição paralela não parametrizada, assume um comportamento por defeito que corresponde a terminar apenas quando todos os processos terminarem, mantendo todos os objectos a duração original, não trocando eventos entre eles, e não sendo definidas relações espaciais entre eles.

Na versão actual do sistema já é suportada a parametrização da semântica de terminação, utilizando-se a sintaxe PAR AND para indicar que o paralelo deve terminar apenas quando todos os processos terminarem, e a sintaxe PAR OR para indicar que o paralelo deve terminar quando o primeiro processo terminar, sendo destruídos os restantes objectos ainda activos.

A declaração dos objectos multimédia segue a sintaxe indicada na figura 5.2, em que se utilizaram parêntesis rectos para indicar as secções facultativas, reticências para indicar as secções que podem ser repetidas e omitiu-se a sintaxe correspondente à declaração de objectos externos, que não se encontra implementada.

```
MMOBJECT nomeObjecto tipoObjecto
    [ WITH
         [ SUB$STRUCTURE (
              nomeZona anotaçãoInício anotaçãoFim
              ... ) ]
         nomePropriedade valorPropriedade
         ... ; ]
    [ START  anotaçãoInicial ; ]
    [ END    anotaçãoFinal ; ]
    [ SOURCE tipoSource[:instância][@nóRede] ... ; ]
    [ FILTER tipoFiltro[:instância][@nóRede] ... ; ]
    [ SINK   tipoSink[:instância][@nóRede]   ... ; ]
    [ LINK   componente[.portoSaída[.tipoSinal]]
             componente[.portoEntrada[.tipoSinal]]
             ... ; ]
ENDOBJECT ';'
```

Figura 5.2: Sintaxe da declaração de um objecto multimédia.

A declaração de um objecto consiste primeiramente numa definição de zonas e uma lista de propriedades. Uma zona é apenas uma simplificação sintáctica que permite utilizar as várias partes de um objecto como se fossem objectos autónomos. Na declaração de um objecto, seguem-se as anotações a utilizar para o início e fim do objecto, quando se invoca o objecto pelo seu nome, e não se pretende que seja de `Ev_Start` a `Ev_End`. Segue-se a definição da topologia do objecto, com a declaração dos componentes *sources*, filtros e *sinks* seguida das ligações entre eles.

Há, ainda, a possibilidade de declarar vectores de objectos multimédia, componentes, ou portos, acrescentando aos seus nomes a dimensão dos vectores dentro de parêntesis rectos. Para exemplificar, considere-se a topologia da figura 5.3, onde um componente mesa de mistura tem cinco portos de saída, dos quais o primeiro pretende-se que produza um sinal PCM de 16 bit e seja enviado para um componente auscultadores, e as outras quatro saídas sejam ligadas às entradas de um outro componente colunas.

```
SOURCE mesaMistura;
SINK   auscultadores;
       colunas;
LINK   mesaMistura.out[1].PCM16  auscultadores
       mesaMistura.out[2..4]     colunas.in[1..4];
```

Figura 5.3: Exemplo de topologia.

Repare-se que no caso, mais habitual, de só haver um componente de cada tipo, basta indicar o tipo do componente para o identificar e, no caso de se pretender usar o porto de defeito de cada componente, pode-se omitir o seu nome e o tipo de sinal multimédia pretendido, simplificando a sintaxe. Também é possível omitir as propriedades cujos valores por defeito se pretenda manter.

Na figura 5.4 apresenta-se um exemplo de um *script* produzido pelo editor gráfico para a composição que será apresentada na figura 5.5. Estão declarados dois objectos multimédia, cada um com uma lista de propriedades e os valores que devem tomar, seguida da lista de componentes que constituem o objecto multimédia e as ligações entre eles. Na parte final encontra-se descrito o comportamento da aplicação em termos de álgebra de processos. O PAR OR corresponde a uma composição paralela que termina quando o primeiro dos processos terminar, de forma a que quando a música terminar destrua o texto com a letra da música. Ao destruir o texto com a letra, o processo que está à espera do evento `Ev_Click` vindo do texto também é destruído, pelo que o PAR principal também termina, terminando a apresentação.

Verifica-se que uma linguagem baseada numa álgebra de processos é adequada para exprimir concorrência, que seria difícil numa linguagem sequencial, pois conduz a encadeamentos de fluxos de controlo complexos. Além disso, convém reforçar o facto de a linguagem utilizada ser uma linguagem formal, que tem um conjunto reduzido de conceitos com grande poder expressivo, e que permite um grande número de verificações e validações que são um precioso auxiliar para um programador.

É, no entanto, de difícil aprendizagem para pessoas sem conhecimentos técnicos adequados, daí a vantagem de ter um editor gráfico que gera o código de forma simples.

```
SPECIFICATION fortuna;

/* Generated by Visual Editor V1.00 */

MMOBJECT Oh_Fortuna audioType
        WITH    filename        "samples/fortuna.au"
                anotfile        ""
                server          ""
                fs              "8000"
                bits            "8"
                endian          "big"
                channels        "1"
                gain            "0"
                speed           "+1.00"
                cycle           "0"
                format          "ulaw"
                device          "0"
                mode            "normal"
                node            "";
        SOURCE  sound;
ENDOBJECT;

MMOBJECT Lyrics textType
        WITH    filename        "texts/ohfortuna.txt"
                anotfile        ""
                windowname      "exemplo"
                rows            "43"
                columns         "52"
                geometry        "+150+80"
                mode            "allfile"
                node            "";
        SOURCE  text;
        SINK    scrolledwindow;
        LINK    text  scrolledwindow;
ENDOBJECT;

BEHAVIOUR

PAR
    PAR OR
        Oh_Fortuna;
        Lyrics;
    END; /* PAR */
    SEQ
        Lyrics ? Ev_Click;
        ABORT;
    END; /* SEQ */
END;

ENDSPEC; /* fortuna */
```

Figura 5.4: Exemplo de *script* produzido pelo editor gráfico.

5.2 EDITOR GRÁFICO

O editor gráfico foi construído com o objectivo de um autor poder criar aplicações multimédia sem ter conhecimentos sobre a linguagem de programação. A figura 5.5 mostra o editor gráfico para X Windows com uma especificação muito simples, correspondente ao exemplo apresentado anteriormente.

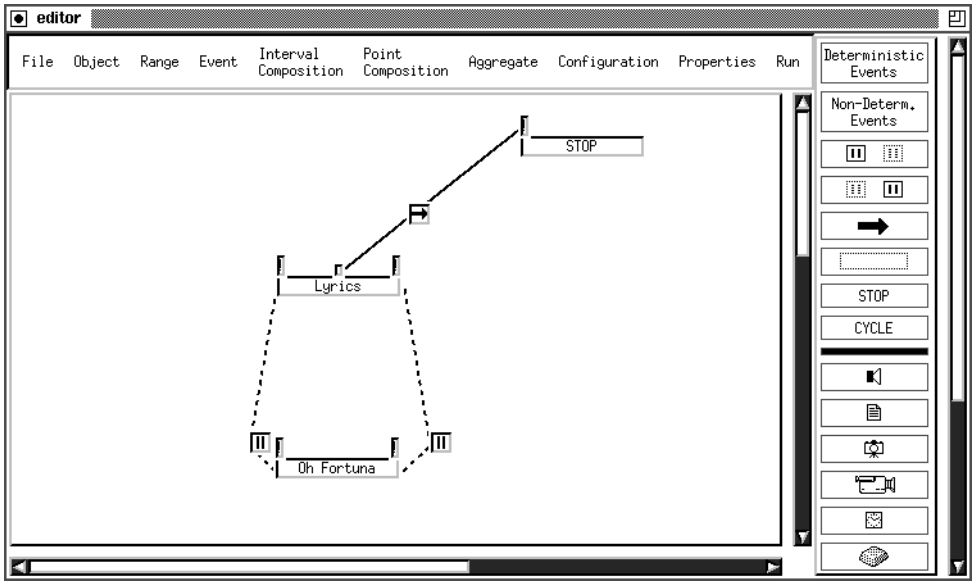

Figura 5.5: Editor gráfico de composições multimédia.

O editor consiste numa linha com menus, uma área de trabalho, e uma paleta. A paleta tem os eventos, os operadores mais comuns, e alguns objectos multimédia debaixo do separador preto. O autor constrói uma aplicação arrastando os objectos da paleta e largando-os na área de trabalho. Os objectos ficam com a configuração por defeito que está no gestor de tipos.

É possível modificar as características de um objecto, não só em termos dos seus objectos componentes, como da topologia das suas interligações, através do menu `Configuration`, podendo mesmo criar-se, desta forma, novos tipos de objectos multimédia[1]. É, também, possível modificar as propriedades dos objectos, tais como nomes de ficheiros, através do menu `Properties`.

[1] Esta última possibilidade não se encontra implementada.

Quando um objecto é largado na área de trabalho, fica automaticamente com os eventos `Ev_Start` e `Ev_End`, que existem em todos os objectos por fazerem parte do estado Contexto que é obrigatório. É possível colocar mais eventos nos objectos, arrastando-os da paleta e largando-os na posição pretendida do objecto. Nessa altura, é apresentado um menu para que o autor defina qual o evento pretendido, de entre os suportados por aquele objecto, o que o editor sabe perguntando ao gestor de tipos. Os eventos determinísticos são representados graficamente por um ícone mais alto do que os eventos não determinísticos. É ainda possível dar nomes diferentes a zonas dos objectos delimitadas por eventos determinísticos. Para isso, utiliza-se o menu `Range` para subdividir o objecto em zonas. Uma zona é apenas uma simplificação sintáctica criada na linguagem para a tornar mais legível, tendo-se optado por mantê-la no editor.

Podem-se criar composições sequenciais e paralelas arrastando os símbolos correspondentes da paleta, e configurando-as, respectivamente com os menus `Point Composition` e `Interval Composition`. As composições sequenciais são representadas graficamente por uma seta, e as composições paralelas por duas barras verticais. É possível associar acções a executar quando ocorrerem eventos, e modificar as acções a executar quando composições sequenciais forem processadas. Este aspecto não é suportado, por exemplo, em [Mey+ 92] onde existe uma acção `play` associada às composições sequenciais.

As composições sequenciais podem ser configuradas para ser executadas sempre que os eventos que lhe dão origem ocorrerem, ou apenas um certo número de vezes, podendo ainda definir-se outras condições a verificar para a sua execução[2].

As composições paralelas podem ser configuradas quanto à forma de as terminar, podendo terminar quando um, alguns ou todos os seus elementos terminarem, quanto ao efeito no tamanho dos objectos e ainda quanto às relações entre eles, como foi explicado na secção anterior.

Os menus `Object`, `Range` e `Event` permitem fazer operações simples sobre os objectos, zonas de objectos e eventos, tais como: mover ícones, modificar os seus tamanhos ou nomes, e apagá-los. O menu `Aggregate` e o símbolo da paleta com o rectângulo a ponteado permitem agregar vários objectos num objecto composto e definir

[2] Esta última possibilidade não se encontra implementada, pois o editor não suporta a utilização de variáveis.

quais os eventos visíveis exteriormente, e a correspondência com eventos internos. O menu `Run` permite gerar o *script*, compilá-lo e testá-lo. O menu `File` permite gravar e ler composições em ficheiro, bem como terminar o editor.

No exemplo apresentado na figura 5.5 mostra-se uma música que deve ser tocada ao mesmo tempo que o texto com a letra é apresentado, correspondendo a uma composição paralela. Se um evento não determinístico for gerado pelo objecto de texto, a especificação é terminada, correspondendo este facto a uma composição sequencial. Embora não esteja patente graficamente, o evento é emitido quando o utilizador premir o rato sobre o objecto texto quando a especificação estiver a correr. As informações sobre a configuração dos objectos também não são representadas graficamente, podendo ser consultadas através dos menus.

Comparando as possibilidades do editor gráfico descrito com as da ferramenta Brama, cujo editor se apresentou na figura 2.4 (página 17), verifica-se que o editor do Brama permite durante a edição ter uma ideia do aspecto final da aplicação, sendo muito fácil posicionar os objectos no écran. No entanto, é mais difícil ter uma ideia do desenrolar da aplicação sem a correr, pois a apresentação é toda programada por menus. Também seria possível ter incluído no editor gráfico descrito neste capítulo a possibilidade de configurar de forma visual as propriedades dos objectos multimédia respeitantes ao posicionamento dos objectos no écran. Para isso, bastaria criar uma nova área de trabalho para apresentar ao autor o aspecto final da aplicação, possibilidade que não se encontra implementada.

CAPÍTULO 6
IMPLEMENTAÇÃO

Neste capítulo descreve-se o ambiente de desenvolvimento sobre o qual o sistema foi implementado, a forma como se construíram os objectos multimédia e os componentes que os constituem. Descreve-se, ainda, o compilador e a forma como ele constrói as aplicações, bem como o funcionamento das aplicações. Seguidamente, avalia-se o desempenho do sistema e apresenta-se um exemplo de uma aplicação e algumas variações sobre o exemplo apresentado. O gestor de nomes, gestor de tipos e editor gráfico já foram analisados nos capítulos precedentes, sendo apenas referidos alguns pormenores da sua integração no sistema. Por fim, avalia-se a compatibilidade do sistema com as normas já apresentadas e abordam-se as limitações do sistema.

6.1 AMBIENTE DE DESENVOLVIMENTO

O sistema de desenvolvimento de aplicações multimédia distribuídas foi implementado e testado em estações de trabalho correndo o sistema operativo Unix, recorrendo aos serviços de comunicação entre processos oferecidos pelo ANSAware [Ansa 93].

ANSA - *Advanced Networked Systems Architecture*, é uma arquitectura para processamento distribuído aberto (ODP - *Open Distributed Processing*), seguindo o modelo de referência MR-ODP [ISO 93a]. O ANSAware é um pacote de software baseado nesta arquitectura.

O ANSAware oferece um modelo computacional cliente-servidor, baseado em objectos. Os objectos encapsulam processamento, armazenamento e transferência de dados. Os objectos servidores oferecem serviços, através de *interfaces*, que são utilizados pelos objectos clientes. Um objecto pode ser, simultaneamente, cliente e servidor de vários serviços.

O modelo de engenharia suporta o modelo computacional sobre vários modelos tecnológicos possíveis. Os objectos do modelo de engenharia designam-se cápsulas (*capsule*) e constituem a unidade de operação autónoma no ANSAware, sendo realizados através de processos do sistema operativo. Uma cápsula inclui um núcleo que suporta a execução concorrente de tarefas (*threads*) sem preempção, e um sistema de comunicação baseado em chamadas a procedimentos remotos (RPC - *Remote Procedure Call*), usando o protocolo REX (*Remote EXecution*), baseado no trabalho de [Birrel+ 84], sobre um serviço de passagem de mensagens.

O modelo tecnológico pode suportar-se em diversos sistemas operativos: Unix, MS-DOS, VMS, etc., e recorre a um sistema de mensagens baseado nos protocolos Arpanet: UDP, TCP. No caso do sistema Unix também é suportado IPC por *named pipes*.

Se bem que, como se referiu anteriormente (secção 3.1, página 36), o modelo de comunicação mais adequado fosse a comunicação baseada em passagem de mensagens assíncrona fiável, a linguagem pressupõe um modelo de comunicação síncrona (secção 5.1, página 58). Um modelo de comunicação síncrona pode ser facilmente realizado através de um modelo baseado em chamadas a procedimentos remotos. Um modelo baseado em chamadas a procedimentos remotos é adequado para interacções cliente-servidor, sendo aceitável para a troca de mensagens de controlo, como é o caso de invocações de acções de objectos e envio de eventos, que se traduzem na invocação de um método de recepção de eventos. No entanto, as chamadas a procedimentos remotos já não são tão adequadas para transferência de grandes quantidades de informação, nem transmissão em tempo real, como a necessária para transferência de dados multimédia entre os vários componentes dos objectos multimédia. Além disso, tal como referido na secção 2.1.3 (página 15) a transferência de dados multimédia requer características de comunicação diferentes da comunicação a nível de controlo. Voltar-se-á a este assunto nas secções sobre a conversão para a máquina de estados e análise de desempenho.

O ANSAware providencia uma linguagem de especificação de *interfaces* (IDL - *Interface Definition Language*) para definir quais as interacções permitidas entre clientes e servidores, e uma ferramenta (*stub compiler*) que compila essa linguagem produzindo o código necessário em linguagem C [Kernighan+ 78], designado *stub*, para facilitar as interacções na presença de distribuição. Devido às dissemelhanças entre invocações locais e remotas, o ANSAware providencia uma linguagem, PREPC, para invocações de

operações e acesso a outras facilidades oferecidas pelo ANSAware. Um pré-processador permite converter programas em linguagem C com instruções PREPC embebidas em programas C compiláveis.

O ANSAware permite transparência de localização, podendo-se interagir com um objecto sem conhecer a sua localização física. Um objecto Trader permite que os clientes tenham acesso aos serviços disponíveis, fornecendo referências para os servidores. Desta forma, as várias partes de uma aplicação distribuída podem encontrar-se a pedido. O gestor de nomes descrito na secção 4.1 (página 51) corresponde directamente ao Trader do ANSA. As referências para *interfaces* de servidores são entidades complexas, pois para além de conterem a localização física, podem ter estruturas de suporte a segurança, mobilidade dos objectos, escolha dos protocolos de comunicação a usar e referências a grupos, sendo consideradas entidades opacas pelos programadores. Ao exportar uma referência para o Trader, é-lhe associado um tipo, que é o tipo do estado correspondente ao *interface*, e ainda um conjunto de propriedades com o objectivo de facilitar a selecção pelos clientes. Para o modelo desta dissertação, são utilizadas propriedades com o tipo do objecto multimédia ou componente, o nome da máquina onde se encontra, e o instante de criação do servidor para permitir seleccionar a versão mais recente.

O ANSAware permite ainda transparência no acesso aos objectos, sendo o acesso realizado da mesma forma quer se trate de objectos locais ou remotos, escondendo as diferenças na representação dos dados e nos mecanismos de chamada a procedimentos. Devido à possibilidade de uma invocação de uma função poder ser convertida numa invocação remota, o ANSAware permite o tratamento de excepções resultantes do funcionamento distribuído. De entre as excepções possíveis, em número de 26, destacam-se:

- `bindFailure`. Significa que não foi possível ligar ao serviço pretendido. Resulta normalmente de erros na importação de um serviço do Trader.
- `transmitTimeout`. Significa que foi excedido o tempo máximo permitido para contactar o servidor. Resulta de situações de sobrecarga ou de terminação do servidor.
- `illegalInterface`. Significa que o servidor recebeu um pedido de invocação de uma operação de um tipo de *interface* não suportado. Resulta de erros de programação.

- `illegalOperation`. Significa que o servidor recebeu um pedido de invocação de uma operação não suportada pelo *interface*. Resulta de erros de programação.
- `abnormalReturn`. Significa que o servidor não pôde completar a operação. Resulta habitualmente do envio de um número incorrecto de argumentos da operação.

Estas últimas quatro excepções são tratadas contactando o *relocator*, para determinar se o servidor migrou para outra localização, caso em que se tenta invocar novamente a operação na nova localização.

Normalmente, as excepções provocam a terminação da cápsula. No entanto, o utilizador pode escolher ser notificado da excepção e agir em conformidade. No caso desta dissertação, optou-se por terminar a aplicação nos casos de não se conseguir contactar um dos servidores de objectos ou componentes para criação de novos objectos multimédia, e no caso de não existirem recursos suficientes. Há ainda a possibilidade de invocar uma acção, ou receber um evento, de um objecto que já terminou, que é possível devido a atrasos na comunicação, e é aceite silenciosamente.

O ANSAware suporta ainda transparência de grupo, mascarando o uso de vários objectos com um único *interface*, e prevê a inclusão em versões futuras de outras transparências definidas no modelo de referência para ODP.

6.2 OBJECTOS MULTIMÉDIA E COMPONENTES

Na tabela 6.1 apresentam-se os tipos de objectos multimédia implementados, os estados associados, e os componentes que lhes podem dar origem.

O objecto de texto permite apresentar parte de um ficheiro de texto com caracteres[1] ISO 8859-1 numa janela de dimensões e posição configurável. Se o texto apresentado não couber na janela, o utilizador pode controlar através de barras de escorregamento qual a parte visível. O *sink* de texto foi implementado sobre Motif [OSF 92].

O objecto relógio permite gerar anotações, de acordo com uma lista de intervalos, que pode ser periódica.

[1] ISO 8859-1, ou alfabeto Latino nº 1, é uma extensão de 8 bits ao código ASCII, de grande divulgação.

Objecto Multimédia	**Estados Suportados**		***Sources* Possíveis**	***Sinks* Possíveis**
Texto	Contexto Click	Anotação	text	scrolledwindow
Relógio	Contexto Velocidade	Anotação Pausa	timer	
Vídeo	Contexto Velocidade Click	Anotação Pausa	mpegSource	mpegSink
Diapositivos	Contexto Velocidade Click	Anotação Pausa	imageSource	imageSink
Áudio	Contexto Velocidade	Anotação Pausa	sound	

Tabela 6.1: Os tipos de objectos multimédia implementados, os estados associados, e os componentes que lhes podem dar origem.

O objecto de vídeo lê ficheiros em formato mpeg e apresenta-os numa janela, a um ritmo que pode ser parametrizado. Utilizou-se descodificação em software recorrendo ao descodificador de domínio público desenvolvido na Universidade da Califórnia em Berkeley [Patel+ 93]. Uma vez que a descodificação é feita em software, ritmos muito elevados podem não ser cumpridos, dependendo da complexidade das imagens. Neste caso saltam-se imagens, o que obriga a separar a comunicação a nível de controlo da transferência de dados. O *sink* de vídeo foi implementado sobre X Windows [Gettys+ 87] [McCormack+ 91].

O objecto de diapositivos permite apresentar uma sequência de imagens numa janela a intervalos regulares. O *sink* de diapositivos foi implementado sobre X Windows. São aceites imagens nos formatos: Graphics Interchange Format (GIF), PC Paintbrush (PCX), Fax Grupo 3, GEM, MacPaint, Sun Rasterfile, Utah Raster Toolkit (RLE), Portable Bit Map (PBM, PGM, PPM), X Window Dump, X Pixmap, X Bitmap, CMU WM Raster, FBM Image, McIDAS areafile e Faces Project.

O objecto de áudio foi implementado sobre os serviços do AudioFile. O AudioFile [Levergood+ 93] é um sistema de áudio para computador desenvolvido pela Digital, portável, independente do equipamento e permitindo um acesso transparente por rede. O AudioFile permite que múltiplos clientes comuniquem com servidores de som partilhando o acesso aos dispositivos de som de uma forma semelhante à utilizada no X

Windows para partilhar um écran. Quando mais do que um cliente envia dados para um servidor AudioFile, os vários sons recebidos são adicionados, resultando na sua sobreposição no altifalante. Uma biblioteca e um *interface* de programação de aplicações permitem a construção de clientes. A versão utilizada (AF2R2) permite apenas reproduzir sinais PCM com frequências de amostragem suportadas pelo codec disponível.

Assim, bastou construir um *source* de som, que é um cliente do AufioFile, sendo o servidor AufioFile o *sink*. Embora pareça mais simples, esta opção trouxe dificuldades na realização dos estados Anotação, Velocidade e Pausa pois tiveram de ser implementados no *source*. Como o servidor do AudioFile memoriza até quatro segundos de som, para garantir a continuidade do som, mesmo na presença de pequenos atrasos na rede (*jitter*), o *source* procura sempre manter esta memória cheia. Este procedimento levanta problemas pois o relógio que controla o ritmo a que os dados são consumidos está no *sink*. Além disso, sempre que se pretende mudar de velocidade ou fazer uma pausa no som, tem de se forçar o *sink* a apagar o som que já tinha sido enviado para ser tocado, e reenviá-lo, desde o ponto em que o servidor tinha ficado, a uma nova velocidade, ou quando a pausa terminar. Se esses estados residissem no *sink*, bastava que se parasse de pedir dados ao *source* no caso de uma pausa, ou que os dados passassem a ser processados de forma diferente no caso de uma mudança de velocidade, assegurando-se neste caso a continuidade do som.

Assim, uma vez que são os *sinks* que memorizam, processam e apresentam os dados, simplifica-se a implementação e obtêm-se tempos de reacção dos objectos menores concentrando a inteligência, isto é os estados, dos objectos nos *sinks*. Torna-se, no entanto, necessário dotar os *sources* de inteligência suficiente para regular a qualidade de serviço dos dados enviados de forma a tolerar situações de sobrecarga. Para que seja possível recuperar de situações de sobrecarga, tem de se separar a comunicação a nível de controlo da transferência de dados, pois apenas esta última se pode degradar.

Em situações de sobrecarga, foram adoptados os seguintes mecanismos: O objecto de som deita fora as amostras de som que ficaram para trás, pois a versão do AudioFile utilizada não permite mudanças dinâmicas na frequência de amostragem; O objecto de vídeo salta imagens.

6.2.1 Constituição dos Objectos Multimédia

Para cada tipo de componente multimédia há pelo menos um servidor no sistema, que exporta o *interface* do seu estado VidaComponentes para o gestor de nomes com o nome do próprio componente, indicado na tabela 6.1. Todas as instâncias dos componentes criados correm no mesmo processo Unix do respectivo servidor onde foram criados.

De modo semelhante, há no sistema pelo menos um servidor de objectos multimédia, que exporta o *interface* do seu estado Vida para o gestor de nomes com o nome "Vida". O servidor de objectos cria novas instâncias de todos os componentes necessários e liga os seus portos de acordo com a topologia do objecto. O estado Contexto das instâncias dos objectos criados reside no próprio processo do servidor, estando os outros estados implementados directamente nos diversos componentes.

Conseguiu-se que o servidor de objectos multimédia fosse o mesmo para todos os objectos multimédia, pois ele pode invocar a acção `Inform` do estado Contexto dos vários componentes para obter referências para os *interfaces* de todos os estados suportados por cada componente. Assim, o servidor devolve ao utilizador dos objectos referências para os componentes onde os diversos estados se encontram implementados, delegando nos respectivos componentes os serviços oferecidos pelos diversos estados. Desta forma, o servidor de objectos multimédia só necessita de oferecer, ele próprio, o estado Contexto, sendo as restantes operações deste estado (`Play`, `Stop`, `Destroy`, `registerStatusInterest` e `unregisterStatusInterest`) realizadas invocando as acções correspondentes de todos os componentes do objecto multimédia. Exceptua-se a acção `receiveEvent` que não está implementada, pois neste sistema centralizado apenas os gestores de sincronização podem receber eventos.

Na figura 6.1 apresenta-se a constituição interna de um objecto multimédia de diapositivos. O objecto multimédia de diapositivos apresenta externamente *interfaces* para os estados Contexto, Anotação e Click, tal como tinha sido indicado na tabela 6.1. Internamente, é constituído por um *source*, implementado no servidor `imageSource`, e por um *sink*, implementado no servidor `imageSink`, e ainda um *dispatcher*, implementado no servidor de objectos multimédia. Cada um dos componentes deste objecto multimédia tem um estado Contexto que só é conhecido pelo servidor de

objectos multimédia, e um estado porto que é ligado ao porto do outro componente para a transferência das imagens. O servidor de objectos multimédia tem o estado Contexto do objecto multimédia e delega no *sink* as funcionalidades dos estados Anotação e Click.

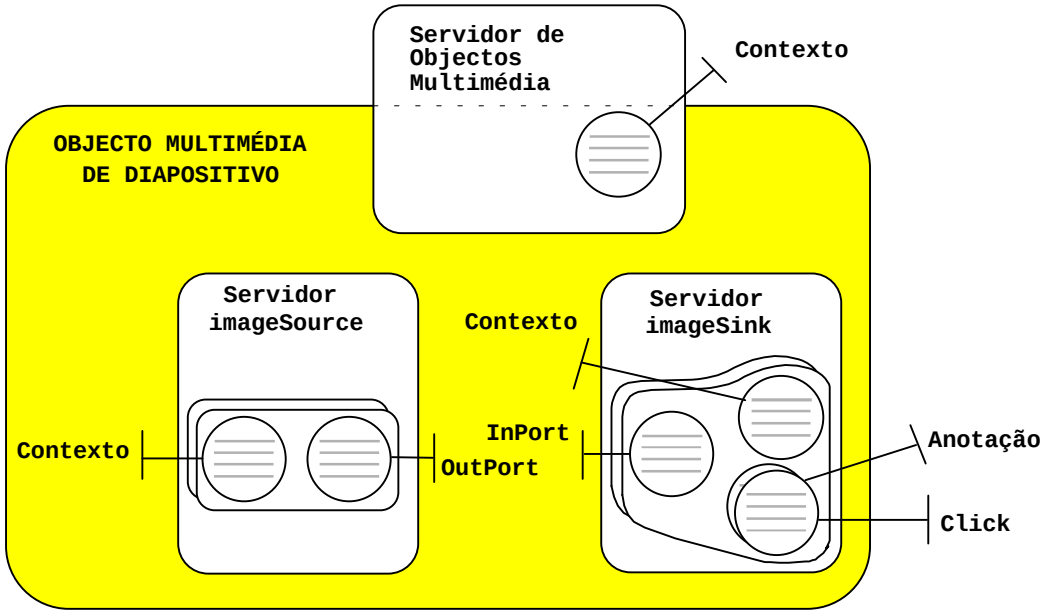

Figura 6.1: Constituição interna de um objecto multimédia de diapositivos.

Na figura 6.2 apresentam-se os passos executados pelo gestor de sincronização para construir um novo objecto multimédia. No primeiro passo, o gestor de sincronização consulta o gestor de nomes para obter uma referência para o estado Vida do servidor de objectos multimédia. No segundo passo, o gestor de nomes devolve como resposta a referência que o gestor de sincronização usa para, no terceiro passo, invocar a acção `prepare` do estado Vida do servidor de objectos multimédia. Na acção `prepare` são fornecidos, entre outros parâmetros, uma lista de componentes e a topologia das ligações dos seus portos. O servidor de objectos multimédia, nos quarto e quinto passos, que são repetidos as vezes necessárias, obtém no gestor de nomes referências para os estados VidaComponentes de todos os servidores de Componentes necessários à construção do novo objecto. No sexto passo, o servidor de objectos invoca a acção `prepare` do estado VidaComponentes, obtendo, no sétimo passo, referências para todos os componentes criados e para os portos de cada um. No oitavo passo, o servidor liga os portos dos vários componentes, invocando a acção `connect` do estado

Porto. Finalmente, no nono passo, o servidor de objectos devolve ao gestor de sincronização uma referência para o estado Contexto do novo objecto multimédia.

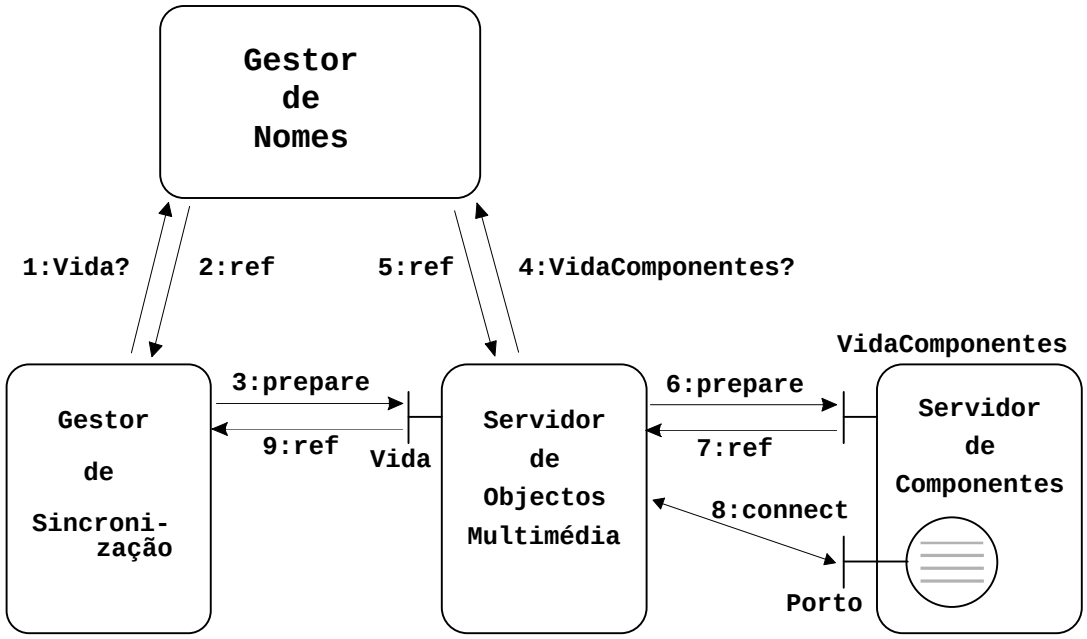

Figura 6.2: Passos executados para construir um objecto multimédia.

Como optimização futura, o servidor de objectos multimédia poderá delegar também as funções do estado Contexto num dos componentes, preferentemente num dos *sinks*. Se um dos componentes suportar esta hipótese, o servidor de objectos multimédia fornece-lhe referências para todos os componentes criados e devolve ao utilizador directamente a referência do estado Contexto do componente onde pretende delegar a responsabilidade de gestão do novo objecto multimédia. Esta possibilidade obriga a que o componente que ficar responsável pelo objecto multimédia tenha conhecimento das várias topologias possíveis, o que pode levar à necessidade de criação de novos tipos de componentes à medida que forem criadas topologias mais complexas.

6.3 COMPILADOR

O compilador converte a especificação em linguagem *script* num programa em C, e invoca o compilador de C para o compilar e ligar a uma biblioteca de apoio à execução, produzindo assim um programa gestor de sincronização, que é uma máquina de estados que comanda o desenrolar da aplicação multimédia.

O compilador consulta o Gestor de Tipos para verificar a consistência das especificações. As acções dos objectos já definidas no gestor de tipos encontram-se pré-processadas pelos utilitários do ANSAware na biblioteca de apoio à execução, podendo ser invocadas directamente pelo código C produzido. As acções acrescentadas posteriormente, embora tendo a informação relevante no gestor de tipos, para poderem ser utilizadas no código produzido necessitam de passar pelo pré-processador do ANSA. Todas estas fases são realizadas automaticamente pelo compilador, bastando-lhe produzir um programa em linguagem C com instruções PREPC embebidas, passá-lo pelo pré-processador do ANSAware e invocar o compilador de C para compilar o código assim obtido e produzir o gestor de sincronização. A versão actual do compilador ainda não tem implementada a fase de invocação do pré-processador do ANSAware, pelo que apenas aceita a utilização das acções já definidas que se encontram em biblioteca.

Caso se pretendesse reduzir a dimensão do ambiente de desenvolvimento podia-se evitar a necessidade de ter um compilador de C disponível e o ANSAware instalado, integrando um interpretador de *scripts* no gestor de sincronização, o que seria menos eficiente do que a compilação dos *scripts*.

6.3.1 Conversão para a Máquina de Estados

A linguagem *script* é convertida numa máquina de estados que simula a execução concorrente dos processos CSP definidos na linguagem[2]. Desta forma, parte do paralelismo existente na álgebra de processos desaparece, o que é inevitável, dado ter-se centralizado a gestão da sincronização num processo Unix. Apenas se mantém o paralelismo resultante da distribuição dos vários objectos pelo sistema.

O problema da conversão de uma álgebra de processos numa máquina de estados é resolvido em [Karjoth 88] para um subconjunto da linguagem LOTOS. Em [Cardeli+ 85] é apresentada uma implementação que produz máquinas de estados em código C para uma linguagem de *interface* gráfico com o utilizador semelhante a CSP e com noção de tempo. Ambos os autores impõem restrições para garantir que se obtém uma máquina de estados finita. No entanto, apesar de proporem a reutilização de estados, e mesmo formas de modificar a especificação de modo a reduzir o número de

[2] Nesta sub-secção, sempre que se referir apenas "processos" refere-se aos processos CSP definidos na linguagem.

estados em [Karjoth 88], o número de estados da máquina de estados cresce muito rapidamente com o número de eventos de entrada possíveis na máquina de estados. Como este é um problema grave, leva a pensar numa forma alternativa de realizar a conversão para a máquina de estados.

Considere-se, como exemplo, uma especificação que tem apenas um paralelo de n=10 processos, cada um podendo apenas enviar um evento a indicar que terminou. Quando os 10 processos terminarem, o paralelo termina, e por consequência a especificação também termina. Ao manter toda a informação sobre a aplicação apenas no estado[3], ter-se-ia um número de estados dado pelo somatório dos arranjos sem repetição dos n processos em sequências de 0 a n:

$$\#estados = \sum_{i=0}^{n} {}^{n}A_i = \sum_{i=0}^{n} \frac{n!}{(n-i)!} \cong n!.e^1 \qquad (6.1)$$

Desta forma, teriam-se 9864101 estados possíveis, todos eles aguardando apenas alguns dos 10 eventos, e os 10! estados do último nível realizariam todos a mesma função: terminar o programa. Seria possível simplificar a máquina de estados obtida, agrupando estados comuns, pois por exemplo, no último nível bastaria um estado, no penúltimo 10 correspondentes às 10 possibilidades para o último processo a terminar, e assim sucessivamente. Esta situação está ilustrada na figura 6.3, para o caso de n=3 processos, estando marcado para cada estado quais os processos ainda activos.

Assim obtém-se, para o número de estados, o somatório das combinações dos n processos em subconjuntos de 0 a n:

$$\#estados = \sum_{i=0}^{n} {}^{n}C_i = 2^n \qquad (6.2)$$

[3] Modelo simples de máquinas de estado.

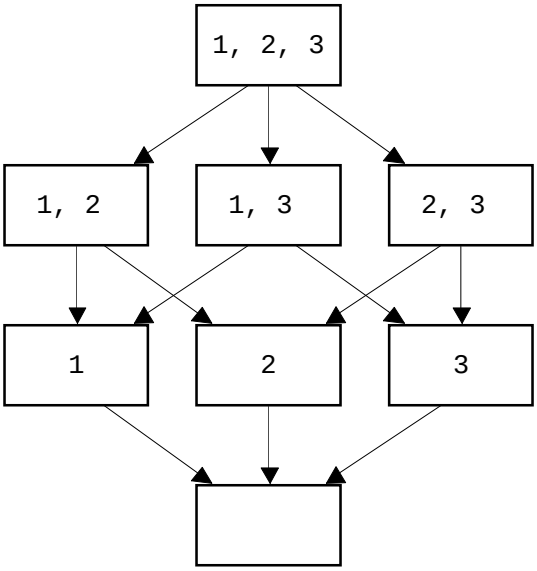

Figura 6.3: Diagrama de estados para um paralelo de três processos.

Desta forma, para n=10 processos, fica-se com 1024 estados. Como facilmente se conclui, não é prático gerar tão grande número de estados, mesmo sendo possíveis grandes simplificações. Uma forma alternativa de ver o problema, é pensar que cada um dos n processos pode apenas estar activo, ou inactivo. Desta forma, bastariam n variáveis de estado booleanas, indicando se cada um dos processos está ou não activo. Além disso, como a linguagem define claramente quais as condições de terminação das várias composições de processos, há vantagem em manter informação sobre essas composições por forma a simplificar ainda mais a implementação. De facto, no exemplo citado, a composição paralela só terminava quando todos os processos que a compunham terminassem, pelo que não seria necessário realizar qualquer acção enquanto o último processo não terminasse.

Assim, optou-se por manter o estado da máquina de estados como uma estrutura em árvore que representa os processos existentes em cada momento, e o seu encadeamento em termos de composições sequenciais e paralelas. Desta forma, é fácil saber quais as acções a tomar sempre que um processo termina, através do tipo do nó correspondente e da hierarquia de processos activos no momento.

Para cada processo que se bloqueie à espera da recepção de um evento, é mantida uma estrutura que descreve o evento pretendido, com um ponteiro para a rotina a ser invocada quando o evento for recebido. Desta forma, o compilador de especificações multimédia limita-se a produzir um conjunto de funções a serem

executadas, consoante o estado da aplicação, quando cada um dos possíveis eventos for recebido. Se esta rotina conduzir à terminação de um processo, consulta-se a hierarquia de processos existente no momento para determinar que processos devem ser terminados e iniciados de forma a preservar semanticamente a linguagem.

Esta abordagem é semelhante à referida na secção 2.2.2 (página 20) para a conversão de expressões de caminho em estruturas hierárquicas descrevendo a aplicação de acordo com as extensões ao ODA para multimédia. No entanto, os objectivos são diferentes, pelo que num documento ODA a hierarquia não é modificada durante a apresentação, pois corresponde ao documento propriamente dito. Além disso, temporizações e interacções com utilizadores são mantidas como atributos no ODA, enquanto na linguagem apresentada correspondem a processos.

Como o gestor de sincronização corre em apenas um processo Unix, não é possível ter paralelismo como havia na linguagem *script*. Desta forma, só se executa uma transição de cada vez na máquina de estados, e de forma atómica. Assim, os eventos são processados pela ordem porque chegam ao gestor de sincronização, existindo uma fila de espera para eventos a ser processados, assumindo-se que há capacidade para processar eventos a um ritmo superior ao ritmo de chegada, e que a fila tem capacidade para guardar todos os eventos que for necessário.

Outro problema resultante de não haver paralelismo é que, quando uma composição paralela é iniciada, têm de se iniciar, um a um, todos os seus processos. Podia-se sortear a ordem pela qual a inicialização seria feita para simular não determinismo, mas optou-se, por razões de eficiência e simplicidade, por iniciar os processos sequencialmente pela ordem por que aparecem no *script*.

Uma vez que as variáveis declaradas no *script* são convertidas em variáveis de tipos equivalentes em C, há a possibilidade de os processos transferirem informação entre eles, isto é comunicarem, através de variáveis. No entanto, não é permitido que um processo fique em ciclo a aguardar que uma condição dependente de variáveis se verifique, pois tal impediria os outros processos de correr, conduzindo a um ciclo infinito, visto não haver preempção de tarefas.

Como os objectos multimédia podem ter tempos de resposta elevados em virtude da tecnologia em que se baseiam, a maior parte das acções são simples invocações de operações sem retorno de parâmetros. Qualquer retorno de parâmetros é feito posteriormente usando o mecanismo de eventos. O objectivo disto é o gestor de

sincronização poder utilizar uma semântica de chamadas a procedimentos remotos e não ficar bloqueado caso a invocação seja convertida numa invocação remota. Estas foram, também, as razões porque se colocou uma fila de espera para a recepção de eventos. Com a fila de espera, permite-se que a chamada ao procedimento remoto para transferência do evento retorne rapidamente, de forma a não bloquear o objecto multimédia que emitiu o evento. Inconsistências provocadas por este tipo de actuação (por exemplo: envio de um evento para um objecto que já deixou de estar activo) são tratadas no modelo de erros do sistema sendo ignoradas. As excepções a este tipo de funcionamento assíncrono são as operações de activação de *sources* e *sinks* que necessitam de troca de identificação, mas que não causam problemas pois são apenas executadas durante a inicialização.

O compilador já suporta processos de entrada, de saída, de atribuição de variáveis, o processo ABORT, execução condicional e repetitiva de processos, composições sequenciais e paralelas. Em termos da parametrização das composições paralelas é suportada a semântica de terminação com PAR AND para terminar apenas quando todos os processos terminarem, e PAR OR para terminar o paralelo quando o primeiro processo terminar, destruindo os restantes processos ainda activos. Estes são os casos de parametrização de composições paralelas mais úteis e os únicos utilizados pela versão actual do editor gráfico. A versão actual do compilador ainda não suporta a declaração de processos, o que não é limitativo pois esta funcionalidade só serve para estruturar o código, e não é usada pelo editor gráfico. Além disso, também não é ainda suportada a composição alternativa de processos, ALT. Mas, uma vez que o gestor de sincronização está centralizado num processo Unix, que não faz preempção das tarefas executadas em resultado da chegada de eventos, pode-se obter as funcionalidades do ALT com o PAR OR.

6.3.2 Análise de Desempenho

Na tabela 6.2 apresentam-se tempos de invocações de procedimentos usando os serviços do ANSA nos casos de a invocação ser local, ou remota por rede Ethernet ou rede ATM. As máquina usadas foram duas SUN SPARCstation 10, uma com um processador, e a outra com dois processadores. A rede local ATM utilizada funciona a 100 Mbit/s.

	Mesma máquina REX/IPC	Por Ethernet REX/UDP	Por ATM REX/UDP
RPC Nula ANSA	2.04 ms	2.90 ms	2.89 ms
RPC com 3864 octetos	3.57 ms	10.00 ms	9.99 ms
RPC com 8000 octetos	19.98 ms	20.50 ms	20.08 ms

Tabela 6.2: Tempos de transmissão.

Verifica-se que os tempos obtidos para chamadas a procedimentos remotos com 8000 octetos de parâmetros são quase os mesmos, independentemente de a comunicação ser local ou remota. Este facto pode ser explicado atendendo a que a comunicação entre processos locais à mesma máquina é feita pelo ANSA através de *pipes* do Unix, que têm uma memória associada de 4 Koctetos, pelo que o desempenho das comunicações locais é muito afectado pelo número de comutações de processos. Verifica-se, ainda, que o ritmo disponível na rede praticamente não influi nos tempos obtidos, o que significa que os tempos de processamento nos vários níveis do protocolo de comunicação são muito superiores aos tempos de transmissão na rede. Desta forma, conclui-se que há um preço elevado a pagar em termos de desempenho, pela facilidade de utilizar um serviço de comunicação entre processos de alto nível como o ANSA.

Seria útil para melhorar o desempenho da transmissão de dados multimédia, que envolve transferências de grandes quantidades de informação, recorrer a mecanismos de comunicação mais eficientes e contornar os níveis de comunicação desnecessários. Uma possibilidade seria a utilização de protocolos como o XTP ou o ST-II, já descritos na secção 2.2.2 (página 18), sendo este último protocolo adequado apenas para a transferência de sinais multimédia pois não recupera de erros.

Comparando os tempos despendidos em chamadas a procedimentos remotos sobre o ANSA com os tempos gastos pelo código produzido pelo compilador, verificou-se que a diferença para o caso de invocações de acções era de cerca de 0.01 a 0.02 ms e para o envio de eventos de cerca de 0.18 ms. Estes tempos correspondem, no caso das invocações de acções, ao tempo gasto a invocar o procedimento correspondente da biblioteca do compilador. Exclui-se, portanto, o tempo despendido no processamento da acção pelo objecto, que é variável consoante a acção. No caso do envio de eventos, esta diferença de tempos corresponde apenas ao tempo de recepção do evento, colocando-o na fila de espera, ao fim do qual a rotina de recepção de eventos retorna. Exclui-se,

portanto, o tempo de invocação da rotina correspondente às acções a executar em resultado da recepção do evento, que é um tempo variável.

6.3.3 Exemplo

Na figura 6.4 apresenta-se o *script* de um exemplo de uma aplicação de informação turística, cujo aspecto durante a execução é mostrado na figura 6.5. Na figura 6.6 mostra-se como o exemplo aparece no editor gráfico.

Esta aplicação apresenta um texto e um conjunto de imagens com legendas, enquanto toca uma música de fundo. O utilizador pode fazer avançar a imagem apresentada carregando no texto com o rato. A aplicação termina quando a música terminar, ou quando o utilizador fizer passar a última imagem.

```
SPECIFICATION batalha;

MMOBJECT Musica audioType
        WITH    filename        "samples/seixas.au";
        SOURCE  sound;
ENDOBJECT;

MMOBJECT Descricao textType
        WITH    filename        "texts/batalha.txt"
                rows            "52"
                columns         "120"
                geometry        "+150+80";
        SOURCE  text;
        SINK    scrolledwindow;
        LINK    text            scrolledwindow;
ENDOBJECT;

MMOBJECT Legenda textType
    WITH SUB$STRUCTURE (
      Legenda_1 Ev_Start           <Label,"FimLegenda1">
      Legenda_2 <Label,"Legenda2"> <Label,"FimLegenda2">
      Legenda_3 <Label,"Legenda3"> <Label,"FimLegenda3">
      Legenda_4 <Label,"Legenda4"> <Label,"FimLegenda4">
      Legenda_5 <Label,"Legenda5"> <Label,"FimLegenda5">
      Legenda_6 <Label,"Legenda6"> Ev_End )
                filename        "texts/blegenda.txt"
                anotfile        "texts/blegenda.ant"
                rows            "10"
                columns         "36"
                geometry        "+160+355";
        SOURCE  text;
        SINK    scrolledwindow;
        LINK    text            scrolledwindow;
ENDOBJECT;
```

```
MMOBJECT Batalha_1 imageType
        WITH    FILENAME        "images/batalha1.gif"
                WINDOW$X        "510"
                WINDOW$Y        "280";
        SOURCE  imageSource;
        SINK    imageSink;
        LINK    imageSource     imageSink;
ENDOBJECT;

(Omitida a declaração dos objectos Batalha_2 a Batalha_6)

BEHAVIOUR

PAR OR
    Descricao;
    Musica;
    SEQ
        PAR OR
            Batalha_1;
            Legenda_1;
            Descricao ? Ev_Click;
        END;
        PAR OR
            Batalha_2;
            Legenda_2;
            Descricao ? Ev_Click;
        END;
        PAR OR
            Batalha_3;
            Legenda_3;
            Descricao ? Ev_Click;
        END;
        PAR OR
            Batalha_4;
            Legenda_4;
            Descricao ? Ev_Click;
        END;
        PAR OR
            Batalha_5;
            Legenda_5;
            Descricao ? Ev_Click;
        END;
        PAR OR
            Batalha_6;
            Legenda_6;
            Descricao ? Ev_Click;
        END;
    END; /* SEQ */
END;

ENDSPEC; /* batalha */
```

Figura 6.4: *Script* de uma aplicação de informação turística.

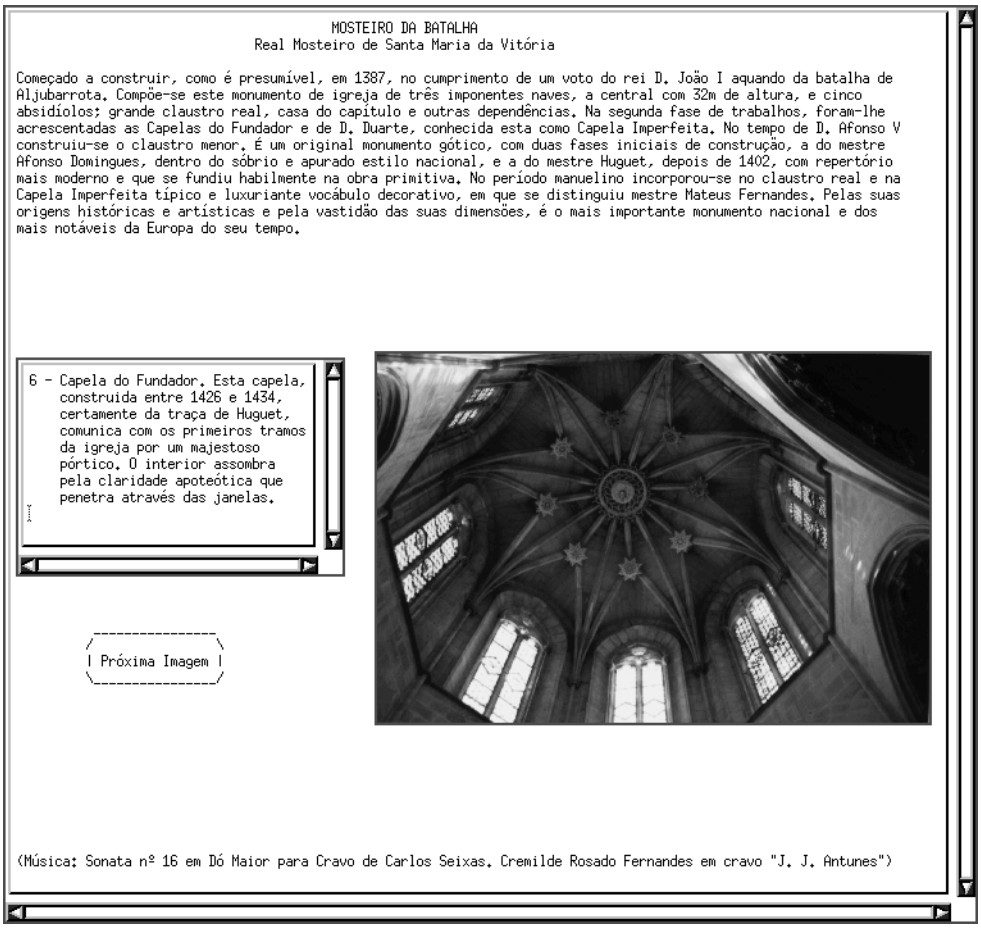

Figura 6.5: Aspecto da aplicação de informação turística.

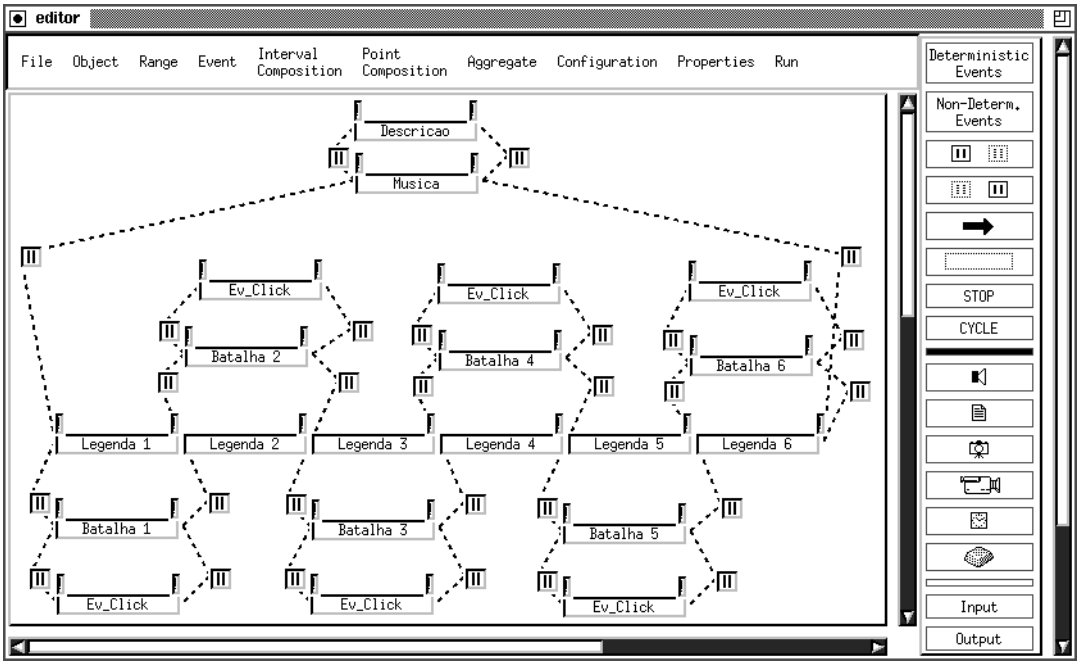

Figura 6.6: A aplicação de informação turística no editor gráfico.

Note-se que o objecto `Legenda` está divido em 6 zonas, `Legenda_1` a `Legenda_6`. A divisão em zonas é uma simplificação sintáctica que permite facilmente utilizar as várias partes de um objecto, delimitadas por eventos, como se fossem objectos autónomos. A propriedade `anotfile` contém o nome de um ficheiro onde se encontram as anotações existentes e as posições correspondentes. No caso das figuras, optou-se por ter um objecto multimédia por cada figura de forma a permitir um controlo independente da posição de cada uma, através das propriedades, pois eram de dimensões diferentes.

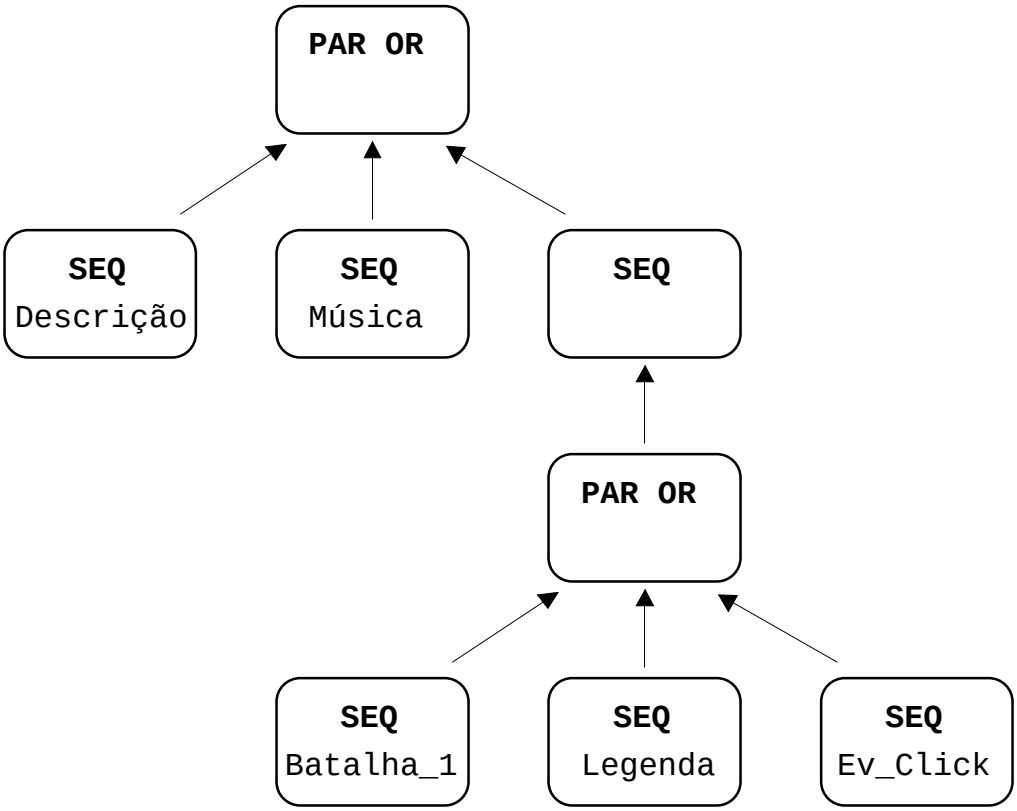

Figura 6.7: Hierarquia de processos durante a execução.

Na figura 6.7 apresenta-se a hierarquia de processos existente durante a maior parte da duração da execução da aplicação. Note-se que a invocação de um objecto multimédia corresponde a uma composição sequencial em que se invoca a acção `Play` e depois se aguarda o evento `Ev_End`. Uma vez que uma composição sequencial só executa um processo de cada vez, usa-se o mesmo nó na hierarquia para todos os processos da sequência, excepto se se tratar de uma composição paralela, caso em que é criado um nó filho. O processo que aguarda o evento `Ev_Click` é mantido por razões

de simplicidade num nó sequencial, embora seja apenas um processo, pois assim não é necessário criar um novo tipo de nó para aguardar eventos.

Uma aplicação pode terminar quando todos os processos terminarem, o que corresponde a ficar-se com uma hierarquia de processos nula, ou por inclusão explícita no *script* de um processo especial, `ABORT`, que quando executado termina a aplicação.

Note-se que o evento `Ev_Click` tem parâmetros associados, no entanto eles são ignorados, pois não se forneceu uma lista de variáveis a ser preenchida com os parâmetros recebidos. Desta forma, o utilizador pode carregar com o rato fora da zona marcada no texto com o mesmo resultado para o funcionamento da aplicação. Podia-se corrigir este aspecto colocando um objecto diapositivo com uma imagem de um botão ou verificando a posição como exemplificado na figura 6.8, em que as anotações nos eventos `Ev_Click` geradas pelo objecto de texto têm como parâmetro a posição em termos do número de caracteres desde o início do ficheiro de texto. Uma vez que o compilador converte as variáveis do *script* em variáveis de tipos equivalentes em C, o código produzido seria muito eficiente neste caso. No entanto, a versão actual do editor gráfico ainda não permite criar *scripts* como este, pelo que se torna vantajoso ter o editor separado do compilador, permitindo a autores mais experientes contornar as simplificações admitidas no editor.

```
VAR x, y, z, t : INT;
    a : ANNOTATION;
...
PAR OR
    Batalha_1;
    Legenda_1;
    SEQ
        a.anotVal:= 0;
        WHILE (a.anotVal < 500) OR (a.anotVal > 510) DO
            Descricao ? Ev_Click(x, y, z, t, a);
    END;
END;
...
```

Figura 6.8: Modificação para testar os parâmetros de um evento.

Na aplicação descrita, a imagem apresentada muda apenas quando o utilizador carrega com o rato no texto. Caso se pretendesse que a imagem também pudesse mudar automaticamente com o decorrer de um segundo trecho de som com uma explicação

sobre o monumento, podiam-se colocar anotações no objecto de som com a explicação e modificar o *script* como se mostra na figura 6.9. Quando o objecto com a locução gerar a anotação `FimParte1`, faz terminar o paralelo em que se espera pelo evento `Ev_Click` ou pela anotação, e como consequência termina também o paralelo onde estão inseridos estes processos, conduzindo à terminação da imagem 1 e da sua legenda, e prosseguindo a apresentação com a segunda imagem. Uma outra alternativa seria dividir a explicação em zonas, o que teria a vantagem de avançar a explicação e a imagem simultaneamente.

```
PAR OR
    Descricao;
    Musica;
    Explicacao;
    SEQ
        PAR OR
            Batalha_1;
            Legenda_1;
            PAR OR
                Descricao ? Ev_Click;
                Explicacao ? <Label,"FimParte1">;
            END;
        END;
        ...
```

Figura 6.9: Modificação para também mudar a imagem com o decorrer de uma explicação.

6.3.4 Estados Suportados pelo Gestor de Sincronização

Na sua versão actual, o gestor de sincronização suporta o estado Contexto que é obrigatório para todos os objectos multimédia e permite arrancar e terminar a aplicação, bem como receber eventos. Além disso, suporta o estado Pausa, que permite parar temporariamente a aplicação e recomeçar na mesma posição, e o estado Velocidade, que permite controlar a velocidade da apresentação.

O gestor de sincronização, por defeito, arranca correndo a aplicação e depois termina. Este comportamento por defeito pode, no entanto, ser modificado por simples alteração da linha de comando. Fornecendo um nome como parâmetro na linha de

comando, o gestor de sincronização regista-se no gestor de nomes com esse nome e permite certo tipo de facilidades como:

- Controlar a aplicação com painéis de controlo.
- Utilizar a aplicação noutros gestores de sincronização, como se fosse um objecto multimédia.

A aplicação pode ainda ser parametrizada durante a execução a partir dos argumentos da linha de comando que podem ser usados no *script* em substituição de qualquer expressão do tipo cadeia de caracteres. Esta possibilidade pode ser utilizada para, por exemplo, fornecer valores das propriedades dos objectos multimédia sem ter de recompilar o *script*.

Na figura 6.10 apresenta-se um painel de controlo que permite controlar o funcionamento da apresentação multimédia do exemplo anterior. As funções dos botões são, da esquerda para a direita: inverter o sentido da apresentação, começar a apresentação no sentido normal, reduzir a velocidade da apresentação, aumentar a velocidade da apresentação, fazer ou terminar uma pausa da apresentação, parar a apresentação e finalmente terminar o programa e o painel de controlo.

Figura 6.10: Painel de controlo.

As várias operações permitidas no painel de controlo são executadas em todos os objectos multimédia sob controlo do gestor de sincronização que suportem os estados correspondentes. Para isso, o compilador quando cria o gestor de sincronização pergunta ao gestor de tipos quais os estados suportados por cada objecto multimédia, e constrói as acções dos estados Pausa e Velocidade do gestor de sincronização de acordo com as respostas obtidas. Assim, uma pausa ou mudança de velocidade é reflectida de igual modo em todos os objectos multimédia contínuos, resultando no efeito pretendido. Exceptua-se o caso do `Play`, que arranca o gestor de sincronização da mesma forma que arrancaria se não fosse controlado pelo painel de controlo.

O painel de controlo é muito geral, permitindo controlar qualquer objecto que suporte os estados Pausa e Velocidade, para além do Contexto que é obrigatório. Se o

objecto que se pretende comandar não suportar o estado Pausa ou o estado Velocidade, o painel de controlo não apresenta os botões correspondentes.

Actualmente, o gestor de sincronização ainda não suporta a inversão do sentido das apresentações, que corresponderia a atribuir uma velocidade negativa ao estado Velocidade. Se não fossem permitidas interacções com utilizadores, ou de uma forma geral eventos não determinísticos, a inversão do sentido da apresentação seria feita modificando a velocidade de cada objecto multimédia contínuo para um valor negativo, e invertendo a ordem por são executados os processos nas apresentações sequenciais. Uma vez que se permitem eventos não determinísticos, há a possibilidade de a apresentação tomar direcções diferentes consoante esses eventos forem ou não recebidos. Além disso, também a execução condicional de processos através de `IF` e `WHILE` pode conduzir a resultados diferentes se as condições de controlo forem modificadas. Desta forma, só é possível inverter o sentido da apresentação memorizando quais os eventos recebidos e os valores que as variáveis tomam, enquanto a apresentação corre no sentido normal. Esta operação de inversão do sentido da apresentação aumentaria a complexidade do gestor de sincronização e diminuiria o seu desempenho, pelo que se deixou para estudo posterior. Um problema relacionado seria arrancar a apresentação num ponto arbitrário. Tal operação obrigaria a memorizar o estado completo do gestor de sincronização, eventualmente com a história passada para permitir inverter o sentido da apresentação, de forma a poder começar a apresentação em qualquer ponto. Esta possibilidade também foi deixada para estudo posterior.

6.4 COMPATIBILIDADE E LIMITAÇÕES

Comparando as características do sistema apresentado com as características da norma MHEG descrita na secção 2.3.1 (página 23), verifica-se que existe um conjunto de conceitos comuns, e que os dois trabalhos se completam. Na realidade, a norma MHEG está mais relacionada com a troca de objectos de informação multimédia e hipermédia, não especificando como são criados esses objectos nem como são apresentados, embora refira a apresentação de uma forma geral. Além disso, a norma MHEG prevê uma alternativa de sincronização externa baseada num *script*, pelo que seria possível construir um formatador e um analisador MHEG que permitissem

converter as aplicações desenvolvidas no sistema apresentado em objectos de informação multimédia MHEG e, reciprocamente, converter objectos MHEG em aplicações.

A integração do MHEG no sistema apresentado está facilitada pelo facto de a classe Ligação usada por esta norma para descrever a sincronização ser conceptualmente semelhante ao conceito de evento apresentado, o que permite modelar a sincronização existente nos objectos MHEG através do modelo apresentado. Além disso, muitas das acções definidas na classe acção no MHEG são equivalentes às acções definidas nos estados já criados, o que facilita a compatibilidade com esta norma.

Comparando agora as características do sistema apresentado com as características do WWW descrito na secção 2.3.2 (página 28), verificamos, como ficou patente no exemplo anterior, que é possível suportar documentos mais complexos do que os permitidos no WWW. Na realidade, o WWW só suporta meios estáticos, não havendo por isso uma noção de duração nos documentos.

O sistema de nomes do WWW, baseado em endereços universais é mais versátil, e não se encontra limitado a um gestor único. No entanto, a resolução de nomes é mais lenta e complexa.

Para além de algumas limitações do sistema apresentado nesta dissertação, já descritas anteriormente, e que serão resumidas na secção de trabalho futuro do próximo capítulo, referem-se no resto desta secção mais algumas que ainda não foram indicadas.

O sistema apresentado encontra-se limitado à gestão da sincronização a um nível elevado, próximo do utilizador. Note-se que o modelo de sincronização também pode ser utilizado para sincronização de baixo nível, pois os diversos objectos podem sincronizar-se, mesmo a baixo nível, por troca de eventos e invocação de acções. No entanto, é muito ineficiente ter a gestão da sincronização centralizada neste caso, pois o número de interacções será grande, e exige tempos de resposta o menor possíveis. De facto, para realizar eficientemente uma sincronização de baixo nível é necessário que os próprios objectos comuniquem directamente.

Falta, portanto, uma forma de o autor especificar as características de qualidade de serviço requeridas e meios para exigir aos objectos multimédia que as cumpram. Uma forma possível, já sugerida, é parametrizar as composições paralelas em termos da influência entre os vários objectos multimédia. Para suportar a sincronização de baixo nível, os objectos teriam de suportar um estado qualidade de serviço, que teria a

inteligência necessária para interagir com a rede, e gerir a sincronização de baixo nível de acordo com os parâmetros fornecidos pelo autor da especificação.

Uma outra limitação advém de se ter assumido que a topologia de ligação dos componentes era fixa, e que era invisível para os utilizadores e para os autores de especificações. Além disso, na implementação só se utilizaram topologias de um *source* e um *sink*. Se bem que não deva apresentar problemas a utilização de topologias mais complexas do que um para um, há aplicações que beneficiariam da utilização de vectores de objectos multimédia e componentes, bem como da possibilidade de criação dinâmica de componentes e modificação da topologia dos objectos multimédia durante a apresentação, e ainda da possibilidade de comunicação multiponto. Um exemplo de tal aplicação é uma conferência com um número variável de participantes.

Seria possível criar um novo estado topologia que permitisse modificar dinamicamente as ligações entre os componentes, mas a nível de controlo seria mais complexo exprimir a composição pretendida e gerir a sincronização de um número variável de objectos. Uma possibilidade seria juntar os objectos em grupos de objectos com objectivos semelhantes, exprimir a composição através da definição de grupos e realizar a sincronização utilizando primitivas de comunicação em grupo, por exemplo usando o protocolo GEX (*Group EXecution*) que também é suportado pelo ANSAware.

Além disso, a criação de objectos multimédia é uma operação complexa e demorada, pelo que seriam necessários cuidados especiais para não perturbar a gestão da aplicação com a introdução de novos objectos. Para resolver este problema seria útil estudar uma forma mais simples e rápida de criar novos objectos. Uma possibilidade seria suportar vectores de objectos de dimensão variável, simplificando-se as operações de acrescentar um novo objecto ou retirar um objecto do vector por todos os objectos terem características semelhantes.

CAPÍTULO 7
CONCLUSÕES

Este capítulo resume o trabalho desenvolvido e contém sugestões de melhoramentos e extensões, bem como indicações para investigação futura.

7.1 CONCLUSÕES

Neste trabalho foi apresentado um sistema de desenvolvimento de aplicações multimédia interactivas distribuídas.

Definiu-se um modelo de objectos multimédia em que as características dos objectos estão classificadas em estados, que combinam uma pequena parcela do estado interno do objecto, as acções relacionadas com essa funcionalidade do objecto e os eventos que o objecto pode enviar para o exterior para anunciar mudanças no seu estado. Descreveu-se como todos os conceitos relacionados com os objectos multimédia eram tipificados, sendo a informação relevante mantida numa base de dados, denominada gestor de tipos, para que todas as aplicações e ferramentas no sistema tivessem conhecimento das possibilidades disponíveis em cada momento, facilitando a expansibilidade do sistema sem ter de modificar as aplicações existentes, e permitindo verificações de consistência durante as várias fases do desenvolvimento de uma aplicação. Definiu-se uma linguagem baseada na álgebra de processos CSP, utilizada para especificar aplicações multimédia e descreveu-se um editor gráfico destinado a gerar aplicações nessa linguagem. Utilizando o editor gráfico, um autor pode criar aplicações com facilidade e rapidez, utilizando os objectos disponíveis, sem necessitar de ter os conhecimentos técnicos relacionados com uma linguagem baseada numa álgebra de processos. Finalmente, descreveu-se como um compilador converte a especificação da aplicação nessa linguagem para um gestor de sincronização que comande o desenrolar da aplicação multimédia. Baseou-se o gestor de sincronização numa máquina de estados para maximizar a eficiência da aplicação.

Julga-se que o modelo de objectos apresentado pode descrever fielmente os objectos necessários à construção da maioria da aplicações multimédia, permitindo qualquer tipo de interacção de controlo sobre os objectos e qualquer forma de interacção com utilizadores. Além disso, o modelo de objectos permite encapsular as dependências do equipamento, e utilizar equipamentos distribuídos por uma rede, escondendo essa distribuição. Aliando o modelo de objectos a uma linguagem com grande poder expressivo, como é uma álgebra de processos, pode-se descrever uma larga gama de aplicações multimédia distribuídas. Um sistema de tipos construtivo permite uma fácil evolução das funcionalidades disponíveis, bem como a reutilização das funcionalidades já definidas e a possibilidade de ter aplicações fáceis de usar, poderosas e eficientes.

7.2 TRABALHO FUTURO

Neste trabalho procurou-se resolver um problema, tendo-se construído um modelo com grandes possibilidades e desenvolvido um sistema baseado nesse modelo. No entanto, muitos outros problemas se levantaram, podendo ser classificados em evoluções do modelo, que são propostas para investigação futura, e melhoramentos na implementação do sistema resultantes da experiência obtida.

7.2.1 Evolução do Modelo e Áreas de Investigação Futura

Sincronização de baixo nível. A implementação apresentada só é adequada para a gestão da sincronização a um nível elevado, próximo do utilizador. Mas também é importante a sincronização de baixo nível. Seria interessante estudar um novo estado qualidade de serviço (QOS), que permitisse exigir aos objectos um conjunto de parâmetros relacionados com sincronização de baixo nível. Também teria de se prever uma forma de o autor especificar essas características, e de as converter em qualidade de serviço. Uma forma possível é parametrizar as composições paralelas.

Outras topologias. Na implementação só se utilizaram topologias de um *source* e um *sink*, não se permitindo modificar a topologia durante a apresentação. Seria interessante estudar um novo estado topologia que permitisse modificar dinamicamente as ligações entre os componentes, e uma forma de utilizar vectores de objectos multimédia e componentes nas especificações. Além disso, poderia haver utilidade em

estudar uma forma mais simples e rápida de criar novos objectos para situações como estas.

Inversão do sentido da apresentação. O gestor de sincronização implementado permite apenas correr as aplicações no sentido normal, não sendo possível aos utilizadores fazer recuar a apresentação, ou mesmo saltar de um ponto para outro, a menos que essa possibilidade tenha sido explicitamente prevista no *script*. A solução destes problemas é complexa, pois exige a memorização do estado da aplicação no decorrer da apresentação, ficando por isso para estudo posterior.

7.2.2 Melhoramentos na Implementação

Optimização da comunicação entre processos. Como se verificou aquando da análise de desempenho, (secção 6.3.2 - página 80), a utilização de comunicação entre processos baseada em chamadas a procedimentos remotos era muito ineficiente, embora seja muito simples de usar para o programador. Desta forma, seria útil utilizar directamente mecanismos de comunicação entre processos de baixo nível para a transferência de dados multimédia pois, atendendo ao volume de dados envolvido, só assim se pode tirar partido das capacidades oferecidas pela rede. Além disso, desta forma, permitia-se a implementação do estado qualidade de serviço.

MHEG. Tal como referido na secção 6.4 (página 89), seria possível construir um formatador e um analisador MHEG que permitissem converter as aplicações desenvolvidas em objectos de informação multimédia MHEG, e reciprocamente converter objectos MHEG em aplicações.

Melhoramentos no editor gráfico. Como ficou patente na secção 5.2 (página 64), falta no editor gráfico a possibilidade de configurar graficamente os valores das propriedades correspondentes à geometria e dimensões dos objectos, dando ao autor da especificação uma ideia do aspecto final da aplicação. Falta, também, a possibilidade de modificar a topologia dos objectos multimédia e a possibilidade de interagir com o gestor de tipos para criar novos tipos, caso seja necessário.

Editores de média. Seria útil ter editores que permitissem associar facilmente anotações aos objectos multimédia.

Estado Anotação no gestor de sincronização. Como descrito na secção 6.3.4 (página 87), a versão actual do gestor de sincronização só suporta os estados Contexto,

Pausa e Velocidade. Desta forma, só é possível utilizar as aplicações criadas como objectos multimédia indivisíveis noutras aplicações, o que corresponde aos casos mais comuns de utilização. Seria interessante que o gestor de sincronização também suportasse o estado Anotação, permitindo passar algumas das anotações dos objectos multimédia internos de uma aplicação para o exterior, permitindo assim outras composições hierárquicas mais complexas.

Gestor de sincronização com interpretador de *scripts*. Para ambientes em que não seja prático ter um compilador de C disponível, embora sendo uma solução menos eficiente, seria necessário ter um gestor de sincronização que interpretasse os *scripts*.

Delegação do estado Contexto dos objectos multimédia num dos componentes. Como já foi referido na secção 6.2.1 (página 73), esta possibilidade melhoraria o desempenho dos objectos multimédia.

Seria ainda possível pensar em outros melhoramentos, como por exemplo, criar estados para permitir modificar as outras propriedades dos objectos multimédia no decorrer das aplicações, e construir painéis de controlo que permitissem modificar esses estados.

BIBLIOGRAFIA

[Addyman 93] T. Addyman. WAIS: Strengths, Weaknesses and Opportunities. *Proceedings of Information Networking 93*, Meckler, London.

[Altenhofen+ 93] Michael Altenhofen, Jürgen Dittrich, Rainer Hammerschmidt, Thomas Käppner, Carsten Kruschel, Ansgar Kückes, Thomas Steinig. The BERKOM Multimedia Collaboration Service. *ACM Multimedia 93*, pág. 457-463, Junho 1993.

[Andrews 91] Gregory R. Andrews. Paradigms for Process Interaction in Distributed Programs. *ACM Computing Surveys* 23(1):49-90, Março de 1991.

[Anklesaria+ 93] Farhad Anklesaria, Mark McCahill, Paul Lindner, David Johnson, Daniel Torrey, Bob Alberti. The Internet Gopher Protocol (a distributed document search and retrieval protocol). Internet RFC 1436, SRI Network Information Center. Março 1993.

[Ansa 93] ANSAware 4.1. *System Programming in ANSAware*. Doc. RM.101.02, Fevereiro 1993.

[Bernardo 94] Luís Filipe Lourenço Bernardo. *Especificação e Sincronização de Aplicações Multimédia com Controlo Distribuído*. Dissertação de Mestrado, Instituto Superior Técnico, Lisboa, Portugal, Junho de 1994.

[Birrel+ 84] A. Birrel, B. Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1):39-59, Fevereiro 1984

[Blakowski+ 92] Gerold Blakowski, Jens Hübel, Ulrike Langrehr e Max Mühlhäuser. Tool Support for the Synchronization and Presentation of Distributed Multimedia. *Computer Communications*, 15(10):611-618, Dezembro 1992.

[Boecking+ 93] Stefan Boecking, Spiridon Damaskos, Luca Delgrossi, Alfons Fartmann, Rainer Hammerschmidt, Gerd Hoelzing, Ilka Milouscheva, Jochen Sandvoss. The BERKOM MultiMedia Transport System. *International Conference on Open Distributed Processing 1993*, pág. 385-391, Setembro 1993.

[Brama 94] Fernando Vasconcelos. *Manual de Utilizador do Brama*, INESC, disponível a partir de Julho 1994.

[Campbell+ 74] R. H. Campbell, A. N. Habermann. The Specification of Process Synchronisation by Path Expressions. Em *Lecture Notes in Computer Science número 16, Operating Systems*, pelos editores G. Goos e J. Hartmanis, pág. 89-102. Springer-Verlag, 1974.

[Campin+ 91] Jack Campin, Richard Cooper, Francis Wai. Type Management in a Heterogeneous System. Technical Report, Department of Computer Science, University of Glasgow, 1991.

[Cardeli+ 85] Luca Cardeli, Rob Pike. Squeak: a Language for Communicating with Mice. *ACM Computer Graphics*, 19(3):199-204, Julho 1985.

[CCITT 88] CCITT Rec. X.501 (ISO/IEC JTC1/ SC21 ISO 9594-2), The Directory Models, Março 1988.

[CIP 90] CIP Working Group, Editor: C. Topolcic. Experimental Internet Stream Protocol: Version 2 (ST-II). Internet RFC 1190, SRI Network Information Center. Outubro 1990.

[Galletly 90] John Galletly. *Occam 2*. Pitman Publishing, 1990.

[Gettys+ 87] Jim Gettys, Ron Newman, Robert W. Scheiffer. *Xlib - C Language X Interface*. X Window System, X Version 11, Release 2, Setembro 1987.

[Gibbs 91] Simon Gibbs. Composite Multimedia and Active Objects. *Proceedings of the OOPSLA'91 Conference*, pág. 97-112. Association for Computing Machinery, Outubro 1991.

[Hoare 85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Hoepner 91] Petra Hoepner. Synchronizing the Presentation of Multimedia Objects - ODA Extensions. *SIGOIS Bulletin*, pág. 19-32, Julho 1991.

[ISO 86] ISO 8879: Information Processing - Text and Office Systems - Standard Generalized Markup Language, 1986.

[ISO 87] ISO 8825 ou CCITT X.209: Information Processing - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), 1987.

[ISO 89] ISO 8613: Information Processing - Text and Office Systems - Office Document Architecture (ODA) and Interchange Format, 1989.

[ISO 11172] ISO 11172: Information technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbit/s (MPEG).

[ISO 93a] ISO/IEC JTC 1/SC 21/WG7 N755. Information Technology - Basic Reference Model of Open Distributed Processing - Part 1: Overview and guide to use, Janeiro 1993.

[ISO 93b] ISO/IEC JTC 1/SC 29/WG 12. Information Technology - Coded Representation of Multimedia and Hypermedia Information Objects, Working Draft, Fevereiro 1993.

[Karjoth 88] Günter Karjoth. Implementing Process Algebra Specifications by State Machines. *Eighth International Symposium on Protocol Specification, Testing and Verification,* Atlantic City, New Jersey, Junho 1988.

[Kernighan+ 78] Brian W. Kernighan, Dennis M. Ritchie. *The C Programming Language*. Prentice-Hall, 1978.

[Lamport+ 90] Leslie Lamport, Nancy Lynch. Distributed Computing: Models and Methods. *Handbook of Theoretical Computer Science*, pág. 1159-1199. Elsevier Science Publishers,1990.

[Lee+ 93] Tim Berners-Lee, Robert Cailliau, Nicola Pellow, Arthur Secret. The World-Wide Web Initiative. *Proceedings INET'93*.

[Levergood+ 93] Thomas M. Levergood, Andrew C. Payne, James Gettys, G. Winfield Treese, e Lawrence C. Stewart. AudioFile: A Network-Transparent System for Distributed Audio Applications. *Proceedings of the USENIX Summer Conference*, Junho 1993.

[Little+ 90a] Thomas Little e Ari Ghafoor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Areas in Communications*, 8(3):413-427, Abril 1990.

[Little+ 90b] Thomas Little e Ari Ghafoor. Network Considerations for Distributed Multimedia Object Composition and Communication. *IEEE Network Magazine*, pág. 32-49, Novembro 1990.

[Little+ 91] Thomas Little e Ari Ghafoor. Multimedia Synchronization Protocols for Broadband Integrated Services. *IEEE Journal on Selected Areas in Communications*, 9(9):1368-1382. Dezembro 1991.

[McCormack+ 91] Joel McCormack, Paul Asente, Ralph R. Swick. *X Toolkit Intrisics - C Language Interface*. X Window System, X Version 11, Release 5, Agosto 1991.

[Mey+ 92] Vicki de Mey, Christian Breiteneder, Laurent Dami, Simon Gibbs e Dennis Tsichritzis. Visual Composition and Multimedia. *Proceedings Eurographics '92*.

[Mey+ 93] Vicki de Mey e Simon Gibbs. A Multimedia Component Kit. *ACM Multimedia 93*, pág. 291-300, Junho 1993.

[Microsoft 91] Microsoft Corporation. *Microsoft Windows: Multimedia Programmer's Reference*. Microsoft Press, 1991.

[Nunes+ 92] Mário Serafim Nunes, Augusto Júlio Casaca. *Redes Digitais com Integração de Serviços*. Editorial Presença, 1992.

[Obraczka+ 93] Katia Obraczka, Peter B. Danzig, Shih-Hao Li. Internet Resource Discovery Services. *IEEE Computer Magazine*, pág. 8-22, Setembro 1993.

[OSF 92] *OSF/Motif Programmer's Guide, Revision 1.2*. Open Software Foundation, Maio 1992.

[Patel+ 93] Ketan Patel, Brian C. Smith, Lawrence A. Rowe. Performance of a Software MPEG Video Decoder. *ACM Multimedia 93*, pág. 75-82, Junho 1993.

[Pinto 93] Paulo F. Pinto. An Interaction Model for Multimedia Composition. PhD thesis, University of Kent at Canterbury, Janeiro 1993.

[Pinto+ 93] Paulo F. Pinto, Peter F. Linington. A Language for the Specification of Interactive and Distributed Multimedia Applications. *International Conference on Open Distributed Processing 1993*, pág. 217-234, Setembro 1993.

[Pinto+ 94] Paulo Pinto, Luís Bernardo, **Paulo Pereira**. A Constructive Type Schema for Distributed Multimedia Applications. *Proceedings of the 3rd International Conference on Broadband Islands, BRIS'94*, pág. 419-434, North-Holland, Junho 1994.

[Prabhakaran+ 93] B. Prabhakaran, S. V. Raghavan. Synchronization Models for Multimedia Presentation with User Participation. *ACM Multimedia 93*, pág. 157-166, Junho 1993.

[Price 93] Roger Price. MHEG: An Introduction to the future International Standard for Hypermedia Object Interchange. *ACM Multimedia 93*, pág. 121-128, Junho 1993.

[Qazi+ 93] Naveed U. Qazi, Miae Woo, Arif Ghafoor. A Synchronization and Communication Model for Distributed Multimedia Objects. *ACM Multimedia 93*, pág. 147-155, Junho 1993.

[Rose 90] Marshall T. Rose. *The Open Book, a Practical Perspective on OSI*. Prentice-Hall, 1990.

[Stallings 88] W. Stallings. *Data and Computer Communications*. MacMilan, 1988.

[Steinmetz 90] Ralf Steinmetz. Synchronization Properties in Multimedia Systems. *IEEE Journal on Selected Areas in Communications*, 8(3):401-412, Abril 1990.

[XTP 92] XTP Protocol Definition, Revision 3.6. *Protocol Engines Incorporated*, Janeiro 1992.