4
Arquitetura do Middleware *Maestro*

O middleware declarativo apresentado neste trabalho, denominado *Maestro*, faz parte da proposta para o padrão de sincronismo de mídias do modelo de referência do Sistema Brasileiro de TV Digital. A arquitetura desse middleware foi elaborada de forma modular, tendo como módulo central, ou núcleo, uma reestruturação, direcionada à TV digital, do Formatador HyperProp (Rodrigues, 2003). As seções seguintes descrevem a arquitetura modular do *Maestro*, bem como as APIs oferecidas por seus principais módulos.

4.1.
Arquitetura Modular

O middleware declarativo *Maestro* possui a arquitetura modular ilustrada pela Figura 15. Entre os principais módulos, que fazem parte dessa arquitetura, estão: Sintonizador, Filtro de Seções, DSM-CC, Núcleo e Exibidores. As próximas seções descrevem cada um dos módulos. Uma atenção especial será dada a um sub-módulo específico do Núcleo, denominado módulo Gerenciador de Bases Privadas, por ser uma contribuição que estende o subconjunto do Formatador HyperProp (Rodrigues, 2003), definido como Núcleo do middleware *Maestro*.

É interessante ressaltar que as APIs dos módulos *Sintonizador*, *Filtro de Seções* e *DSM-CC* são baseadas nas APIs definidas no padrão DVB (DVB, 2004). Os sub-módulos Filtro de Seções, Sintonizador, DSM-CC, Gerenciador de Bases Privadas e Gerenciador DSM-CC, apresentados na Figura 15, foram modelados de acordo com as discussões realizadas, mas suas implementações são citadas entre os trabalhos futuros definidos nesta dissertação.

PUC-Rio - Certificação Digital Nº 0410848/CA

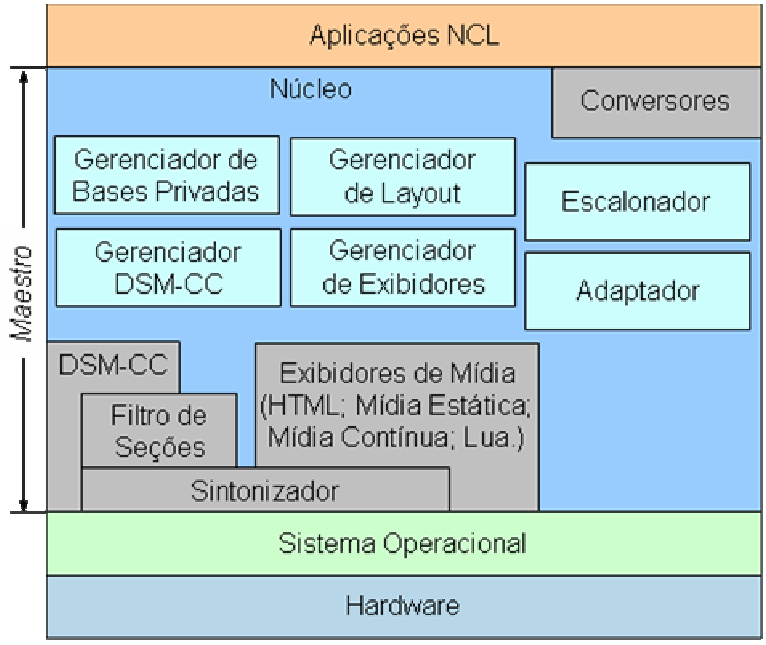

Figura 15: Arquitetura Modular do Middleware *Maestro*.

PUC-Rio - Certificação Digital Nº 0410848/CA

4.2. Módulo Sintonizador

A justificativa para a definição de um módulo sintonizador, na arquitetura do middleware *Maestro*, está na forma com que é realizada a transmissão dos fluxos de transporte. No contexto da TV digital, cada fluxo de transporte é enviado em um canal distinto, através de um determinado sistema de transmissão por difusão, que pode ser um sistema de difusão terrestre, por satélite ou por cabo (Morris & Chaigneau, 2005). Os canais podem ser multiplexados na freqüência, ou no tempo, como, por exemplo, em redes IP, onde os canais podem ser determinados por um endereço IP *multicast* (Kurose & Ross, 2003), somado ao número de uma porta lógica para a recepção do fluxo.

Para sintonizar um canal, com objetivo de receber um fluxo de transporte, primeiro é necessário definir entre as interfaces de redes, presentes em um terminal de acesso, quais estão aptas a receber fluxos de transporte. Por exemplo, um terminal de acesso que possui uma interface de rede receptora de sinais de TV terrestre e outra receptora de TV por satélite. Dessa forma, é especificado, no módulo sintonizador, um gerenciador de interfaces de rede, denominado *NetworkInterfaceManager*, responsável por gerenciar todas as instâncias de interface de rede. Cada instância dessas interfaces é denominada *NetworkInterface*. O gerenciador de interfaces de rede oferece a API apresentada

na Tabela 12. Note, na Tabela 12, que o gerenciador de interfaces é definido utilizando o padrão de projeto *Singleton* (Gamma et al, 1995), possuindo uma instância única na arquitetura, que pode ser obtida através do método *getInstance()*.

Método	Descrição
getInstance()	Retorna a instância única do gerenciador, uma vez que o mesmo é definido como *Singleton* (Gamma et al, 1995).
getNetworkInterfaces()	Retorna todas as interfaces de rede presentes no terminal de acesso.
setCurrentNetworkInterface(NetworkInterface)	Define qual interface de rede, passada como parâmetro, pode ser utilizada para sintonizar um fluxo de transporte.
getCurrentNetworkInterface()	Retorna a interface de rede definida pelo método setCurrentNetworkInterface().

Tabela 12: API oferecida pelo *NetworkInterfaceManager*.

Cada instância *NetworkInterface* encontrada pelo gerenciador de interfaces de rede representa uma forma possível de se obter fluxos de transporte. Geralmente, para terminais que apresentam mais de uma interface de rede, o usuário seleciona, através do controle remoto, qual das interfaces de rede deve ser utilizada para sintonizar os canais. Para isso foi especificado o método *setCurrentNetworkInterface()*. No entanto, para que o gerenciador de interfaces ofereça informações ao usuário de sua API sobre quais interfaces de rede estão aptas a sintonizar fluxos de transporte, deve-se percorrer todas as instâncias de interfaces de rede, obtidas através do método *getNetworkInterfaces()*. Essa verificação pode ser realizada, por exemplo, rastreando os intervalos de freqüências (para redes de TV por difusão terrestre, cabo e satélite); ou endereços IPs somados às portas lógicas (em um sistema de TV sobre IP) que cada interface alcança.

A API de uma *NetworkInterface*, apresentada na Tabela 13, oferece o método *listAccessibleTransportStreams()*. Esse método realiza suas tarefas, apresentadas na Tabela 13, apenas depois de verificar que o valor retornado pelo método *isReserved()* é falso. Isso significa que nenhum usuário está, concomitantemente, utilizando a interface de rede para sintonizar um canal específico.

PUC-Rio - Certificação Digital Nº 0410848/CA

Para obter o fluxo de transporte relativo ao canal que a interface de rede está atualmente sintonizada, deve-se utilizar o método *getCurrentTransportStream()*. Os usuários da API de uma *NetworkInterface* específica, que manipulam constantemente o fluxo de transporte retornado por esse método (por exemplo, módulo Filtro de Seções), devem registrar-se como *listener* dessa interface de rede. Dessa forma, esses usuários recebem notificações quanto à ocorrência de uma nova sintonização (i.e. deve ser considerado um novo fluxo de transporte a ser manipulado).

Para obter informações sobre o tipo de sistema de transmissão que a interface de rede suporta, deve-se utilizar o método *getDeliverySystemType()*.

Método	Descrição
listAccessibleTransportStreams()	Retorna uma lista vazia se o valor do método isReserved() retornar verdadeiro. Caso contrário, realiza um rastreamento nos canais (intervalo de freqüências para redes de TV por difusão terrestre, cabo e satélite; ou endereços IPs somados às portas lógicas pré-determinadas em um sistema de TV sobre IP) que a interface alcança e retorna uma tabela, relacionando, para cada fluxo de transporte acessível, o canal correspondente.
getCurrentTransportStream()	Retorna o fluxo de transporte correspondente ao canal atualmente sintonizado. Retorna um valor nulo se nenhum canal válido estiver sintonizado.
getDeliverySystemType()	Retorna um valor inteiro que identifica o tipo de sistema de transmissão que a interface de rede suporta. Os sistemas de transmissão possíveis são: transmissão de TV a cabo, TV por satélite ou TV terrestre, bem como um sistema de transmissão de TV sobre IP *multicast*.
isReserved()	Informa se a interface está ou não reservada. Interfaces de rede são reservadas via um controlador de interface de rede, conforme explicado na Tabela 14.
addNIListener(NIListener)	Adiciona um *listener*, passado como parâmetro, à interface de rede. Os *listeners* são notificados que uma nova sintonização ocorreu e recebem uma referência para o recurso da interface de rede por onde o fluxo de transporte está sendo recebido.
removeNIListener(NIListener)	Remove um *listener*, passado como parâmetro, da interface de rede.

Tabela 13: API oferecida pelo *NetworkInterface*.

PUC-Rio - Certificação Digital Nº 0410848/CA

Para controlar uma interface de rede, de forma a utilizá-la para sintonizar um canal, com o objetivo de receber um fluxo de transporte, faz-se necessário um controlador de interface de rede, denominado *NetworkInterfaceController*. A API oferecida pelo controlador de interface é apresentada na Tabela 14.

Método	Descrição
reserve()	Realiza uma associação entre a interface de rede definida no gerenciador de interfaces de rede, através do método *setCurrentNetworkInterface()*, e o controlador de interface de rede, reservando a interface de rede para realizar uma sintonização. Essa operação é realizada apenas se o método isReserved() da interface definida retornar falso.
tune(int)	Utiliza a interface de rede associada ao controlador, através do método reserve(), para sintonizar o fluxo de transporte correspondente ao canal passado como parâmetro. Retorna uma referência para o fluxo de transporte sintonizado.
release()	Desfaz a associação com a interface de rede, liberando seus recursos para possibilitar uma futura sintonização.
getNetworkInterface()	Retorna a interface de rede associada ao controlador.

Tabela 14: API oferecida pelo *NetworkInterfaceController*.

Após criar um controlador de interface de rede, é necessário associá-lo à interface desejada. Essa associação é realizada através do método *reserve*() descrito na Tabela 14, para depois sintonizar o fluxo de transporte desejado, com o método *tune*(). Uma vez sintonizado, o fluxo de transporte é recebido através de um recurso da interface de rede.

4.3. Módulo Filtro de Seções

Como discutido no Capítulo 2, as seções privadas MPEG-2 são utilizadas para transportar dados que variam de tabelas SIs a sistemas de arquivos. Um middleware de TV digital deve permitir que partes específicas de um fluxo de transporte, relativas a uma seção, sejam obtidas. Para isso, o middleware *Maestro* define um filtro, denominado *SectionFilter*, que oferece uma API capaz de retornar seções privadas contidas no fluxo de transporte MPEG-2 sintonizado, de acordo com alguns critérios especificados como, por exemplo, “filtre as seções cujo tipo de dados são tabelas SIs”. Uma vez que um *SectionFilter* realiza processamento sobre o fluxo de transporte sintonizado, é necessário que o mesmo

PUC-Rio - Certificação Digital Nº 0410848/CA

se cadastre como *listener* da *NetworkInterface*, discutida na seção anterior, que possui o fluxo sintonizado.

Segundo a arquitetura do middleware *Maestro*, ilustrada na Figura 15, a API do módulo Filtro de Seções é utilizada pelos módulos DSM-CC e Núcleo. O módulo DSM-CC normalmente solicita as seções contendo carrossel de objetos ou eventos DSM-CC. O Núcleo, por sua vez, solicita as seções contendo um determinado tipo de tabela SI, como será discutido na Seção 4.5.

A API de um *SectionFilter*, apresentada na Tabela 15, permite que restrições afirmativas e restrições de negação sejam especificadas no filtro para designar as seções desejadas. Por exemplo, um usuário da API *SectionFilter* que deseja receber apenas carrossel de objetos que não possua aplicações NCL especificaria o seguinte filtro: "filtre todas as seções com dados de um carrossel de objetos (`table_id` = 10, no exemplo), mas que não contenha aplicações NCL (`table_id_extension` ≠ 32, no exemplo)".

Método	Descrição
startFiltering(Pid, table_id, posFilerMask, negFilerMask);	Inicia o processamento do fluxo de transporte sintonizado, retornando uma referência para as seções encontradas de acordo com o filtro especificado. Os parâmetros são opcionais. Pid especifica o identificador de fluxo, table_id especifica o identificador da seção, posFileMask são restrições afirmativas e negFilerMask definem restrições de negação.
setTimeOut(Time)	Especifica um tempo limite para que um temporizador, passado como parâmetro, pare o processamento do filtro no fluxo de transporte. Sempre que uma seção for encontrada, o temporizador é reiniciado.
stopFiltering()	Pára o processamento do filtro sobre o fluxo de transporte.

Tabela 15: API oferecida pelo *SectionFilter*.

4.4. Módulo DSM-CC

Para oferecer informações e conteúdo interativo em conjunto com a programação audiovisual, um provedor de conteúdo deve enviar aplicações (documentos NCL, por exemplo) aos terminais de acesso. Um dos mecanismos utilizados para que essas aplicações cheguem aos terminais de acesso é o mecanismo de transmissão cíclica, especificamente o carrossel de objetos DSM-CC. Nesse caso, o módulo Filtro de Seções é utilizado para entregar o fluxo de

PUC-Rio - Certificação Digital Nº 0410848/CA

dados ao módulo DSM-CC, responsável por tratar as funcionalidades desse protocolo. Uma das principais funções do módulo DSM-CC consiste em decodificar o carrossel de objetos, criando um sistema de arquivos para esse carrossel, de acordo com o sistema operacional da plataforma. Uma outra importante função do módulo DSM-CC é monitorar a chegada de eventos de fluxo (*stream events*) e notificar aqueles usuários que tenham se registrado como *listeners* desses eventos.

O módulo DSM-CC foi definido na arquitetura do middleware *Maestro* com o objetivo de oferecer APIs para manipular o carrossel de objetos, bem como eventos de fluxo, obtidos através do módulo Filtro de Seções.

O padrão DSM-CC (ISO, 1998b) define o conceito de "*service domain*" como um conjunto de objetos, pertencentes a um carrossel de objetos, que formam um sistema de arquivos. Para representar um conjunto de objetos, o módulo DSM-CC define um *ServiceDomain*, que oferece a API apresentada pela Tabela 16.

Método	Descrição
attach(Carousel)	Realiza uma associação do *ServiceDomain* a um carrossel de objetos, passado como parâmetro, montando esse carrossel em um diretório do sistema de arquivos.
attach(Locator, Id)	Recebe como parâmetro uma referência para um serviço do fluxo de transporte sintonizado, bem como o ID de um carrossel de objetos específico, presente nesse serviço. Para encontrar esse carrossel de objetos, a API do módulo Filtro de Seções é utilizada. Depois, realiza uma associação do *ServiceDomain* ao carrossel de objetos, retornado pelo módulo Filtro de Seções, montando esse carrossel em um diretório do sistema de arquivos.
getMountPoint()	Retorna o diretório onde o carrossel de objetos foi montado.
detach()	Realiza um *unmount* no sistema de arquivos, removendo os arquivos criados, e desfazendo a associação realizada por qualquer um dos métodos *attach*.
addEventListener(ObjectEventListener)	Adiciona o *listener* passado como parâmetro, para ser notificado sobre eventos gerados pelo carrossel de objetos.
removeEventListener(ObjectEventListener)	Remove o *listener* passado como parâmetro.

Tabela 16: API oferecida pelo *ServiceDomain*.

Um *ServiceDomain* deve ser associado a um carrossel de objetos, através do método *attach()* (qualquer uma das duas opções), apresentado na Tabela 16. Esse método é responsável por montar (semelhante ao comando `mount` do sistema

PUC-Rio - Certificação Digital Nº 0410848/CA

operacional UNIX (Tanenbaum, 1992)), em um diretório do sistema operacional do terminal de acesso, o sistema de arquivos representado pelos objetos do carrossel de objetos associado. Note que nenhum dos métodos apresentados na Tabela 16 permite especificar em qual diretório um carrossel deve ser montado. Isso é feito para evitar conflitos de diretórios definidos por usuários da API. Assim o método *attach()* define internamente onde montar o carrossel de objetos. O local definido pelo método *attach()* pode ser obtido através do método *getMountPoint()*.

Para ser notificado que um objeto contido no carrossel de objetos foi completamente recebido, um usuário da API *ServiceDomain* (módulo Núcleo, conforme ilustra a Figura 15) pode se cadastrar como um *listener* de eventos. Na notificação é passada como parâmetro a localização, no sistema de arquivos, desse objeto.

Um dos recursos mais utilizados do carrossel de objetos é a atualização de arquivos, ou seja, o provedor de conteúdo pode enviar novas versões de módulos no carrossel (Seção 3.2.1). Cada atualização gera também uma notificação aos *listeners* de eventos.

Após utilizar os objetos de um carrossel de objetos, o usuário da API pode removê-los através do método *detach()*. O método *detach()* pode ser chamado também quando um novo fluxo de transporte for sintonizado.

Conforme discutido no Capítulo 2, além de arquivos e diretórios, um carrossel de objetos pode possuir objetos de evento, responsáveis por relacionar nomes textuais, conhecidos pelas aplicações, aos identificadores numéricos dos eventos que serão criados pelos descritores. Quando um objeto de evento é recebido, *ServiceDomain* notifica seus *listeners*, passando esse objeto como referência. Para representar um objeto de evento foi definido um *StreamEventObject*, que oferece a API apresentada na Tabela 17.

PUC-Rio - Certificação Digital Nº 0410848/CA

Método	Descrição
subscribe(eventName, StreamEventListener)	Cadastra o *listener* passado como parâmetro, com o objetivo de ser notificado sobre ocorrências de eventos do tipo eventName, também passado como parâmetro. Retorna o identificador do tipo de evento solicitado.
unsubscribe(eventId, StreamEventListener)	Desfaz o cadastro do *listener* passado como parâmetro no tipo de evento correspondente a um identificador, também passado como parâmetro.
getEventList()	Retorna uma tabela relacionando identificadores aos tipos de eventos que podem ocorrer.

Tabela 17: API oferecida pelo *StreamEventObject*.

Um usuário da API oferecida pelo módulo DSM-CC pode cadastrar-se para receber um determinado tipo de evento DSM-CC através do método *subscribe()*. Para saber quais os tipos de eventos estão disponíveis para cadastro, é utilizado o método *getEventList()*. Esse método retorna uma tabela relacionando tipos de eventos aos seus identificadores, conforme discutido no Capítulo 2.

Além do carrossel de objetos, eventos DSM-CC podem ser obtidos através do módulo Filtro de Seções. Os eventos DSM-CC, especificados por um descritor de eventos, são representados no módulo DSM-CC do middleware *Maestro* por um *StreamEvent*. A API oferecida por um *StreamEvent*, apresentada na Tabela 18, permite acesso aos principais campos de um evento DSM-CC.

Método	Descrição
getEventName()	Retorna o nome do evento, utilizado para definir seu tipo.
getEventId()	Retorna o identificador do evento.
getEventTimeReference()	Retorna a referência temporal para a ocorrência do evento. Retorna um valor negativo caso o evento seja um evento "*do it now*".
getEventData	Retorna os dados específicos da aplicação.

Tabela 18: API oferecida pelo *StreamEvent*.

Quando um evento é obtido pelo módulo DSM-CC, sua referência temporal é verificada com o objetivo de agendar sua ocorrência. Para um evento "*do it now*", imediatamente é utilizado o método *getEventName()* para que todos os *listeners* cadastrados para receber esse tipo de evento sejam notificados. Para um evento com uma referência temporal ainda válida (se for uma referência temporal passada, o evento é descartado), um monitor é instanciado para disparar sua

PUC-Rio - Certificação Digital Nº 0410848/CA

ocorrência no instante apropriado. A notificação da ocorrência no instante apropriado é análoga à notificação dos eventos "*do it now*".

4.5. Núcleo e Exibidores

O núcleo do middleware *Maestro* consiste em um formatador responsável por controlar a apresentação de documentos NCL. Os principais sub-módulos que compõem o Núcleo foram apresentados na Figura 15, página 66.

Através de um conversor, um documento NCL recebido pelo middleware *Maestro* é traduzido para uma estrutura de execução apropriada para o controle da apresentação dos documentos. O modelo de execução proposto para o middleware *Maestro* é baseado no modelo de contextos aninhados (NCM) (Soares et al, 2003).

Entre as principais entidades do modelo NCM estão: nós de composição, nós de mídia (ou objetos de mídia), descritores, âncoras, eventos e elos. Simplificadamente, as *composições* permitem estruturar os documentos e os *nós de mídia* representam os objetos cujos conteúdos são unidades de informações em uma determinada mídia a serem exibidos. Uma composição NCM é formada por uma coleção de nós e elos (explicados mais adiante), podendo esses nós serem objetos de mídia ou outras composições. *Descritores* podem ser associados a objetos de mídia, indicando como esses objetos devem ser apresentados. *Âncoras* representam um conjunto de unidades de informação marcadas de um nó, ou um atributo do nó e permitem definir eventos. Os *elos* modelam as relações entre os eventos. Finalmente, *eventos* são ocorrências no tempo que podem ser instantâneas ou de duração finita. Em cada objeto de mídia, vários eventos podem ser estabelecidos, como a apresentação de uma âncora (exibição de um trecho de um vídeo, por exemplo), a seleção de uma âncora pelo usuário telespectador etc. (Soares et al, 2003).

No middleware *Maestro* cada documento NCL é convertido para uma composição NCM, chamada de *contexto*. As entidades de um documento NCL são diretamente convertidas para entidades do modelo NCM, uma vez que, como comentado no Capítulo 1, a linguagem teve por base o Modelo de Contextos Aninhados.

PUC-Rio - Certificação Digital Nº 0410848/CA

Com o objetivo de agrupar um conjunto de documentos NCL, é utilizado o conceito (também definido no NCM) de um tipo especial de composição denominada *base privada*. Cada vez que um fluxo de transporte é sintonizado, o middleware abre uma nova base privada para agrupar os documentos transmitidos. A manutenção desses agrupamentos é a função atribuída ao sub-módulo Gerenciador de Bases Privadas. Esse sub-módulo utiliza a API do módulo Filtro de Seções para identificar cada base privada com o identificador do fluxo de transporte[7] sintonizado. Além disso, esse sub-módulo registra-se como *listener* do módulo sintonizador. Assim, a cada notificação recebida (sobre uma nova sintonização), o sub-módulo Gerenciador de Bases Privadas exclui a base privada que estava sendo utilizada e cria uma nova, através do identificador do novo fluxo de transporte sintonizado. Isso significa que é utilizada apenas uma base privada por vez. Vale ressaltar que, antes de sintonizar um novo canal, é possível salvar a base privada, que está sendo utilizada, em um repositório. Esse recurso, evidentemente, irá depender da capacidade do terminal de acesso.

As manipulações sobre bases privadas são oferecidas pelo sub-módulo Gerenciador de Bases Privadas através da API de um *PrivateBaseManager*, apresentada na Tabela 19. Através das bases privadas, os documentos NCL, que estão sendo apresentados (ou já foram apresentados), podem ser organizados.

Método	Descrição
openBase (baseId)	Abre uma base privada existente ou cria uma nova base, caso ainda não exista.
saveBase (baseId)	Salva conteúdo da base privada.
closeBase (baseId)	Fecha a base privada especificada.
deleteBase(baseId)	Exclui a base privada especificada.

Tabela 19: API oferecida pelo *PrivateBaseManager*.

O sub-módulo Gerenciador DSM-CC, ilustrado na Figura 15, permite, entre outras funcionalidades, que os provedores de conteúdo acessem as bases privadas, iniciando, ou mesmo organizando, suas aplicações NCL, nos terminais de acesso. Esse sub-módulo recebe, através do módulo DSM-CC, todos os eventos DSM-CC gerados pelo provedor de conteúdo. Esses eventos são então interpretados e

[7] Normalmente, os fluxos de transporte possuem uma identificação fixa e única para cada provedor de conteúdo.

PUC-Rio - Certificação Digital Nº 0410848/CA

mapeados para chamadas aos métodos de outros sub-módulos do middleware *Maestro*.

A apresentação de um documento NCL transmitido tem seu início quando o sub-módulo Gerenciador DSM-CC recebe um evento DSM-CC que transporta, no campo "nome do evento", o valor "*prepareNCL*", e no campo "dados específico das aplicações", as seguintes informações: uma referência para um carrossel de objetos (serviço do fluxo de transporte sintonizado e o ID do carrossel) que está sendo transmitido; uma referência para o documento NCL presente nesse carrossel; um identificador opcional da base privada onde as entidades do documento serão dispostas.

A partir do recebimento desse evento, o sub-módulo Gerenciador DSM-CC utiliza a API do *ServiceDomain* para montar o carrossel no sistema de arquivos, registrando-se como *listener* desse *ServiceDomain* (para receber notificações sobre atualizações nos módulos, por exemplo). Normalmente, quando esse evento DSM-CC chega ao terminal de acesso, o carrossel de objetos já está disponível, em um dispositivo (*device* - /dev do sistema operacional) do terminal de acesso, para ser montado no sistema de arquivos. Caso não esteja disponível, o sub-módulo Gerenciador DSM-CC deve aguardar uma notificação da disponibilidade por parte do *ServiceDomain*.

Após ser montado o sistema de arquivos, correspondente ao carrossel de objetos especificado, o sub-módulo Gerenciador DSM-CC solicita o serviço de um compilador NCL, presente no módulo Conversores. Essa chamada entrega como parâmetro a referência do documento NCL (localização do documento no sistema de arquivos) e retorna para o Gerenciador DSM-CC a composição NCM representando o documento.

O Gerenciador DSM-CC passa então ao Gerenciador de Bases Privadas uma solicitação para inserir a composição NCM na base privada que estiver aberta no momento. Ao receber um evento DSM-CC que transporta, no campo "nome do evento", o valor "*startNCL*", o Gerenciador DSM-CC solicita que a composição seja apresentada a partir da sua porta de entrada. Conforme definido no modelo NCM, a porta de entrada de uma composição é um mapeamento para um nó diretamente contido na composição e uma interface desse nó, que pode ser uma âncora do nó ou uma outra porta, se o nó interno for uma outra composição.

PUC-Rio - Certificação Digital Nº 0410848/CA

O sub-módulo Gerenciador de Bases Privadas permite também que, através do sub-módulo Gerenciador DSM-CC, um autor especifique (no provedor de conteúdo) modificações na estrutura das aplicações NCL (localizadas nos terminais de acesso). Para isso, o sub-módulo Gerenciador DSM-CC possui a capacidade de mapear, de forma trivial, os eventos definidos na Tabela 20 em chamadas aos métodos também oferecidos pela API de um *PrivateBaseManager*. Note que esses métodos não foram apresentados na Tabela 19, evitando repetir suas respectivas descrições.

Nome do Evento	Dados Específicos da Aplicação	Descrição
addNode	compositeId, node	Insere nó em uma dada composição da base privada em uso.
removeNode	compositeId, nodeId	Retira nó de uma dada composição da base privada em uso.
addDescriptor	descriptor	Insere descritor na base privada em uso.
removeDescriptor	descriptorId	Remove descritor da base privada em uso.
addAnchor	nodeId, anchor	Insere âncora em um nó da base privada em uso. Caso a âncora exista, atualiza seus atributos.
removeAnchor	nodeId, anchorId	Remove e destrói âncora de um nó da base privada em uso.
addLink	compositeId, link	Insere elo em uma dada composição da base privada em uso.
removeLink	compositeId, linkId	Remove elo de uma dada composição da base privada em uso.

Tabela 20: Eventos DSM-CC interpretados pelo sub-módulo Gerenciador DSM-CC.

PUC-Rio - Certificação Digital Nº 0410848/CA

É importante ressaltar que as modificações em uma aplicação declarativa, especificadas no provedor de conteúdo por um ambiente de autoria, são coerentemente atualizadas nos terminais de acesso, preservando todos os relacionamentos, incluindo aqueles que definem a estruturação lógica do documento NCL. Essas modificações podem ser realizadas em tempo de exibição.

Três observações devem ser feitas em relação aos métodos mapeados através da Tabela 20. A primeira observação é sobre o método *addLink*, que pode especificar uma âncora inexistente de um nó como ponto terminal do elo. Caso isto ocorra, uma âncora deve ser criada no nó, com seu conteúdo ainda indefinido. Esse conteúdo deverá ser preenchido, posteriormente, pelo comando *addAnchor*. Por exemplo, no caso em que a âncora represente um conjunto de unidades de

informação de um objeto de mídia, a âncora é criada com o conjunto vazio, a ser preenchido como indica a segunda observação.

A segunda observação é sobre o método *addAnchor*, que pode criar uma âncora totalmente nova ou substituir o conteúdo de uma âncora já criada. O conteúdo de uma âncora pode especificar o conteúdo de um atributo do nó, no caso da âncora ser um atributo. No caso da âncora especificar um conjunto de unidades de informação do nó, a âncora deve definir esse conjunto ou especificar um instante de tempo mais um deslocamento *d*. Essa última opção é usada para criar uma âncora em um nó que está sendo gerado e exibido ao vivo, cujo evento de apresentação ocorrerá no instante de tempo especificado no comando e durará *d* unidades de tempo.

Finalmente, a terceira observação é, novamente, sobre o método *addLink*. Permitir inserir elos em uma composição, significa, também, possibilitar que o provedor de conteúdo dispare, através da inserção de um elo com condição já satisfeita, a apresentação de um nó específico (relacionado por esse elo). Isso caracteriza uma alternativa para o processo de apresentação de um documento NCL. Note, que esse tipo de elo permite, na verdade, que o provedor de conteúdo execute qualquer ação de apresentação (i.e. *play*, *pause*, *stop* e *abort*) sobre os nós de uma base privada.

Para que o Núcleo seja capaz de apresentar as entidades NCM, presentes na base privada em uso, é necessário transformá-las em um conjunto, denominado contêiner, de entidades esperadas por ele. Novamente, o módulo Conversores é solicitado pelo sub-módulo Gerenciador de Bases Privadas. É interessante ressaltar que o módulo Conversores pode ser solicitado também para permitir que o ambiente de execução do Núcleo controle a apresentação de documentos especificados em outras linguagens como, por exemplo, documentos SMIL (Bulterman, 2004). Entretanto, para manter uma arquitetura leve, atendendo aos requisitos discutidos no Capítulo 1, foram definidos no *Maestro* apenas os conversores necessários à apresentação de documentos NCL (tradução da linguagem NCL para o modelo conceitual NCM e a tradução do modelo NCM para o ambiente de execução do Núcleo (Rodrigues, 2003)). Outros conversores (Rodrigues, 2003) são especificados como opcionais.

Um contêiner pode conter entidades como objetos de execução, elos e *layout*. O objeto de execução representa a instância de um nó a ser exibido,

PUC-Rio - Certificação Digital Nº 0410848/CA

definido no documento NCL, contendo todas as suas informações, inclusive as suas características de apresentação provenientes do descritor associado. Os *layouts* presentes em um contêiner consistem em janelas e regiões, definidas na autoria, para a apresentação dos nós. O módulo Gerenciador de *Layout*, apresentado na Figura 15, é responsável por mapear as janelas e regiões, definidas no documento NCL, nos componentes gráficos disponíveis no terminal de acesso. Através das janelas e regiões criadas, bem como de possíveis relacionamentos espaciais[8] definidos na autoria, o Gerenciador de *Layout* define as características espaciais dos exibidores e, conseqüentemente, dos objetos de execução.

As instanciações dos exibidores de mídia (exibidores dos objetos de mídia) são controladas pelo módulo Gerenciador de Exibidores, apresentado na Figura 15. Estratégias de adaptação ao contexto (por exemplo, opções de resolução segundo o perfil do usuário telespectador, entre outras (Rodrigues, 2003)) podem ser acionadas, através do módulo Adaptador, para determinar uma configuração que atenda restrições do autor ou do contexto em questão. Finalmente, o Núcleo inicializa o módulo Escalonador, responsável por orquestrar a apresentação dos objetos de execução, de acordo com as relações definidas na autoria.

É importante que o Núcleo e os Exibidores estejam fortemente integrados, pois são os exibidores que decodificam os conteúdos, recebidos via fluxo de transporte (através de DSM-CC ou fluxos elementares) ou via canal de retorno (por exemplo, por TCP/IP), e são capazes de perceber o ponto exato em que a exibição de um determinado objeto se encontra para informá-lo ao Núcleo. Além disso, as interações do usuário telespectador são realizadas sobre os exibidores e não com o Núcleo, cabendo a eles repassar as informações a respeito dessas interações. Assim, é definida em (Rodrigues, 2003) uma interface para troca de mensagens entre os módulos Núcleo e Exibidores, adotada na implementação da comunicação entre o núcleo do *Maestro* e todos os exibidores implementados.

Note, na Figura 15, que um navegador HTML está entre os exibidores definidos para o middleware *Maestro*. Isso significa que aplicações desenvolvidas na linguagem HTML podem ser apresentadas pelo *Maestro*. Além disso, a fim de permitir que código executável fosse integrado a programas exibidos no

[8] Relacionamentos espaciais podem mudar o posicionamento de objetos de execução.

PUC-Rio - Certificação Digital Nº 0410848/CA

middleware declarativo, um interpretador Lua (Ierusalimschy et al, 2003) foi incorporado para atribuir mais recursos à autoria de documentos NCL. Essa integração faz com que o middleware puramente declarativo seja estendido para funcionar como um middleware híbrido (declarativo+procedural). O uso de Lua oferece vantagens: além de ser um projeto de código aberto e desenvolvido em C, Lua combina programação procedural com poderosas construções para descrição de dados, baseadas em tabelas associativas e semântica extensível; Lua é tipada dinamicamente, interpretada a partir de *bytecodes*, e tem gerenciamento automático de memória com coleta de lixo. Segundo , Lua se destaca pelo alto desempenho e baixo consumo de recursos apresentados, quando comparada com outras linguagens interpretadas. Essas características fazem de Lua uma linguagem ideal para configuração, automação (*scripting*) e prototipagem rápida (Ierusalimschy et al, 2003). O próximo capítulo discorre sobre a implementação da versão corrente do middleware *Maestro*.

PUC-Rio - Certificação Digital Nº 0410848/CA