5
O Fomatador NCL *Xlet*

Atualmente, o Formatador NCL encontra-se implementado em duas linguagens: JAVA e C++. Com o GEM oferecendo um ambiente JAVA para a execução global de aplicações interativas, tem-se a possibilidade de portar a implementação do Formatador para sistemas que implementem tal *framework*.

No entanto, a implementação Java do Formatador NCL é voltada para a plataforma Java SE. Para que ela possa executar com o comportamento esperado em sistemas de TV digital, são necessárias algumas adaptações e otimizações a serem discutidas neste capítulo.

Para cumprir seus propósitos este capítulo está organizado da forma a seguir. A Seção 5.1 detalha como desmembrar a arquitetura do Formatador em componentes independentes e as possíveis otimizações que podem ser realizadas nessa arquitetura. A Seção 5.2 trata da implementação do ambiente a partir das propostas das seções anteriores e descreve o processo de implantação (*deployment*) do sistema proposto nesta dissertação. Ao final da seção, são apresentados mecanismos para a integração desse mesmo sistema com o padrão MHP. E, por fim, a Seção 5.3 apresenta o ambiente onde foram realizados os testes da implementação.

5.1.
Os Componentes do Formatador NCL Xlet

Nas seções a seguir serão descritos os componentes que compõem o Formatador NCL.

5.1.1. Gerenciador de Leiaute

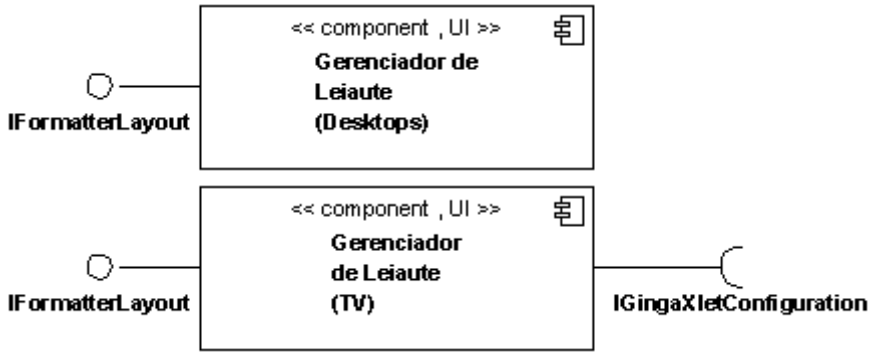

Figura 23 – Duas implementações do componente Gerenciador de Leiaute

Esse componente é responsável por controlar a interface gráfica do Formatador. Ele possui a capacidade de criar as regiões onde serão exibidos os objetos de mídia. Por esse motivo ele deve sofrer uma customização para cada ambiente gráfico (como X Window System) (Scheifler & Gettys, 1996) no qual tais objetos serão exibidos.

A interface fornecida por esse componente é a *IFormatterLayout*, descrita na Seção 4.4.4, e as interfaces necessárias para sua execução irão depender da implementação. A Figura 23 ilustra duas implementações diferentes desse componente: uma para *desktops* e outra customizada para TV. Na implementação para TV percebe-se a necessidade da interface *IGingaXletConfiguration*. Maiores detalhes sobre essa interface e essa implementação são encontrados na Seção 4.4.4.

Esse componente não é obrigatório. Apesar de na maioria das vezes haver a necessidade de seu uso, em apresentações onde for necessário apenas o sincronismo entre objetos de mídia do tipo áudio, não há a necessidade do carregamento desse componente.

5.1.2.
Gerenciador de Documentos

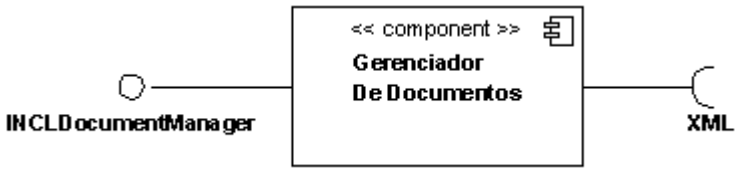

Figura 24 – Componente Gerenciador de Documentos.

Esse componente é responsável por gerenciar **uma** base de documentos. Nesse processo o componente irá adicionar, remover e alterar os documentos da base. Dentre outras coisas, o componente Gerenciador de Documentos é capaz de receber a especificação de documentos XML, verificar se estes possuem uma sintaxe/semântica NCL válida e convertê-los para objetos Java. Para realizar esse processo, o componente necessita da presença de uma biblioteca capaz de processar documentos XML. Basicamente, o componente é formado pelas seguintes entidades, mostradas na **Figura 16**: Gerenciador de Documentos, Conversor NCL e a Base de Documentos.

O recebimento de documentos XML é feito através da interface *INCLDocumentManager*, mostrada na Figura 24. O conjunto de métodos providos por essa interface permitem ao componente processar os Comandos de Edição NCL.

A necessidade desse componente vai depender da implementação do componente do Núcleo do Formatador. Uma implementação mais simples de núcleo que, por exemplo, processe apenas um documento (incluindo os documentos por ele referenciados) por vez e não dê suporte a eventos de edição NCL não necessitaria desse componente.

5.1.3.
Núcleo do Formatador

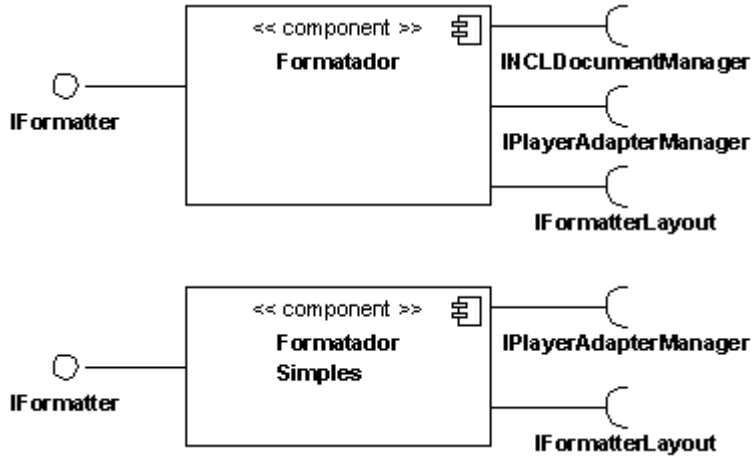

Figura 25 – Duas implementações do componente do núcleo do formatador.

Dos componentes apresentados esse é o mais importante e obrigatório. Ele será o responsável por controlar a apresentação e corresponde diretamente ao núcleo do Formatador mostrado na Seção 3.4.

Esse componente pode ter sua implementação customizada de forma a simplificar o seu funcionamento. Como mostrado na Figura 25, pode-se criar uma implementação simples de Núcleo independente do Gerenciador de Documentos. Esse componente depende ainda das interfaces *IFormatterLayout* (oferecida pelo Gerenciador de Leiaute) e *IPlayerAdapterManager* (oferecida pelo Gerenciador de Adaptadores para Exibidores, que será visto na seção a seguir).

De acordo com as otimizações propostas na Seção 4.4.2, a interface *IFormatter* possui a assinatura mostrada na Figura 26.

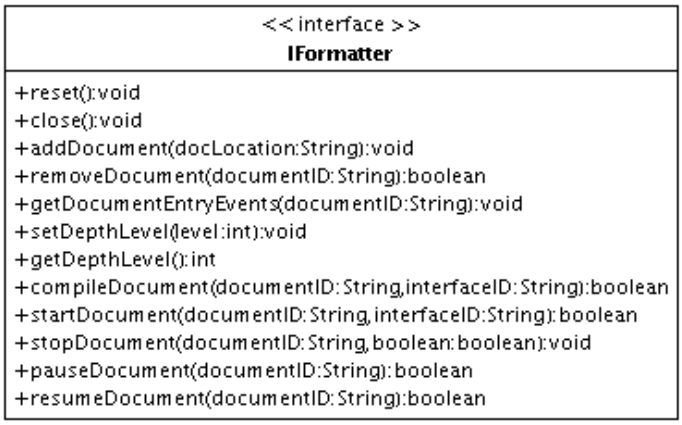

Figura 26 – Duas implementações do componente do núcleo do formatador

5.1.4.
Gerenciador de Adaptador para Exibidores

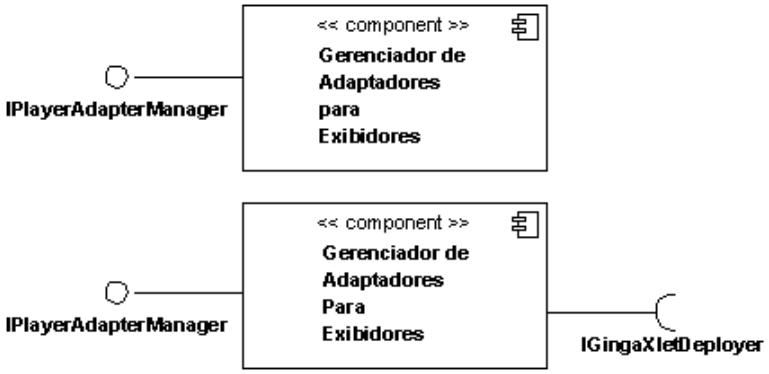

Figura 27 – Duas implementações do componente Gerenciador de Exibidores

Esse componente é obrigatório para a execução do Formatador NCL. A gestão realizada por ele compreende a identificação e instanciação do adaptador para o exibidor necessário para a apresentação de um determinado tipo de conteúdo.

Assim como o Gerenciador de Leiaute, esse componente apresenta uma forte dependência com o ambiente no qual será executado (TV, *Desktop*). Isso porque, além de, no caso de exibidores de vídeo ou imagens, haver uma dependência quanto à interface gráfica do ambiente, a forma como são obtidos os recursos (no caso, os exibidores) depende do ambiente de execução. Num sistema de TV Digital, por exemplo, os recursos podem estar presentes no carrossel de objetos DSM-CC, sistema de arquivos local ou canal de retorno. O mecanismo utilizado para identificação de recursos num sistema GEM é tratado na Seção 4.3.

5.1.5.
Adaptadores para Exibidores

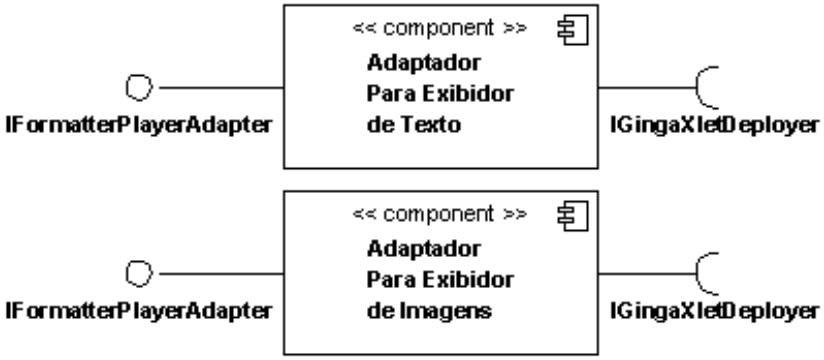

Figura 28 – Duas implementações de adaptadores para exibidores.

Esse componente constitui uma camada de adaptação para exibidores de um determinado tipo de conteúdo de forma a torná-los compatíveis com o modelo de execução do Formatador NCL. Existe uma implementação diferente desse componente para cada exibidor suportado pelo formatador.

Na apresentação de um dado documento NCL nem todos os exibidores disponíveis para o Formatador NCL são necessariamente utilizados. Existem, por exemplo, apresentações que envolvem apenas imagens estáticas, outras apenas textos. Dessa forma, a obrigatoriedade desse componente depende do tipo de conteúdo por ele processado e a presença desse conteúdo na apresentação.

Para tornar esse componente independente do protocolo utilizado para a obtenção do conteúdo a ser exibido, utiliza-se as facilidades oferecidas pela interface *IGingaXletDeployer* para o carregamento de arquivos. Isso porque, por exemplo, antes de obter um arquivo de um carrossel DSM-CC, é necessário estabelecer uma conexão. A interface *IGingaXletDeployer* abstrai todas essas peculiaridades. Mais informações sobre o funcionamento dessa classe podem ser encontradas na Seção 5.2.2 e 5.2.3.

5.2. Implementação dos Componentes para Sistemas GEM

Esta seção e suas subseções descrevem a implementação do ambiente de execução declarativo, baseado na linguagem NCL. Esta implementação é baseada na implementação Java do Formatador NCL. A partir dela são feitas adaptações para adequá-la às propostas das Seções 4.1, 4.2 e 4.3.

O primeiro aspecto a levar em consideração na implementação do Formatador NCL *Xlet* é que este deve utilizar apenas as bibliotecas fornecidas pelo JVM mínimo necessário para o GEM, ou seja, portar o código escrito para uma configuração de máquina virtual Java SE para uma configuração Java ME (CDC com PBP). A seguir são abordados aspectos particulares de cada componente.

5.2.1. O Carregamento Dinâmico

A chave para a criação de uma arquitetura modular, onde os módulos podem ser carregados dinamicamente no decorrer da execução da aplicação, é o uso das tecnologias oferecidas pelas classes `java.lang.Class` e `java.lang.ClassLoader`.

Cada classe Java disponível para uso por uma aplicação é carregada por um objeto *ClassLoader* a ela associado. Ou seja, quando o Gerenciador de Aplicações, abordado na Seção 2.4, inicia a execução do *Xlet*, todas as classes disponíveis para uso por este *Xlet* estão associadas a um *ClassLoader* uilizado para carregá-las inicialmente em memória. Ao longo da execução da aplicação, esta instância de *ClassLoader* pode ser usada para o carregamento de novas classes que antes não se encontravam disponíveis para o uso pela aplicação. A este processo dá-se o nome de *Dynamic Class Loading* (Liang & Bracha, 1998), ou Carregamento Dinâmico de Classes.

O carregamento dinâmico de classes não se restringe apenas às classes presentes no ambiente de execução da aplicação. É possível carregar classes disponíveis em ambientes remotos através da transferência do código binário destas classes via protocolos de rede. A classe `java.net.URLClassLoader` utiliza, por exemplo, uma conexão baseada no protocolo especificado via URL (desde que implementado pelo ambiente de execução Java local) para transferir e carregar o código binário das classes. O carregamento dinâmico ainda traz um benefício denominado *lazy loading*: ao atrasar o carregamento o máximo possível, ocorre uma economia de memória e uma melhora no tempo de resposta do sistema. Isso porque a classe só é carregada quando requisitado e não na inicialização do sistema.

A partir do momento em que a classe encontra-se disponível para uso, por não possuir uma referência estática, seu referenciamento e instanciação deverão ocorrer de forma dinâmica, ou seja, para obter uma referência para a nova classe utiliza-se o método estático "*forName*" da classe `java.lang.Class` passando como parâmetro o nome da classe a ser instanciada. Este método retornará uma instância da classe `java.lang.Class`. Com esse objeto pode-se, finalmente,

criar uma instância da classe (dinamicamente carregada) através do método "*newInstance*".

No sistema proposto foi criada uma entidade que se responsabiliza pelo carregamento dinâmico dos módulos do formatador e a extração de tais módulos do fluxo MPEG-2: o *GingaXletDeployer*.

5.2.2. O Módulo de Implantação

Esse módulo é responsável por realizar a implantação do ambiente declarativo. A implementação desse módulo vai depender da plataforma para a qual o ambiente declarativo se destina. Pode oferecer, inclusive, mecanismos sofisticados de implantação/monitoramento dos componentes. No entanto, no caso de sistemas GEM sua implementação deve ser simplificada.

Para a implementação desse componente foram criadas quatro classes: *GingaXlet*, *IGingaXletConfiguration*, *IGingaXletDeployer* e *StreamEventHandler*.

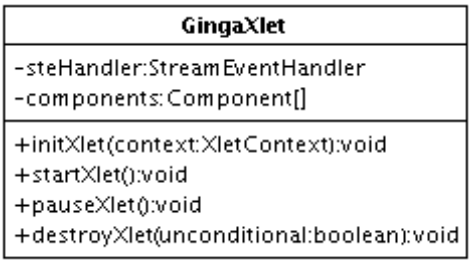

Figura 29 – Classe GingaXlet.

A classe *GingaXlet*, mostrada na Figura 29, é a implementação da interface *Xlet* necessária para o modelo de programação do Java TV. Essa classe é o ponto de entrada da aplicação e, através dela, o Gerenciador de Aplicações do receptor controlará o ciclo de vida da aplicação. A assinatura dessa classe é idêntica a de um *Xlet* mostrada na Figura 9 na Seção 2.4. A classe *GingaXlet* possui referência para todos os componentes instanciados no sistema e para a entidade *StreamEventHandler,* que será apresentada a seguir. A utilidade desses atributos será apresentada na Seção 5.2.3.

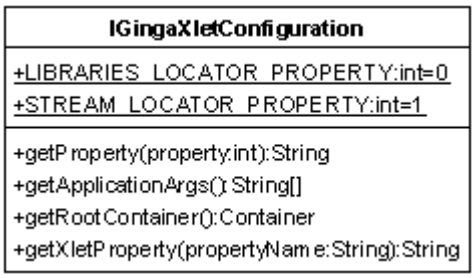

Figura 30 – Classe IGingaXletConfiguration.

A classe que implementa a interface *IGingaXletConfiguration*, mostrada na Figura 30, guarda parâmetros de configuração para a execução da aplicação. Através dela a aplicação pode acessar parâmetros provenientes do Gerenciador de Aplicações, fornecidos na inicialização da aplicação via sinalização ou presentes em arquivos de configuração. Existem dois parâmetros importantes fixados pela *IGingaXletConfiguration*, são eles: a localização das bibliotecas disponíveis, ou seja, o diretório no qual os componentes e demais bibliotecas estarão disponíveis; e a localização do objeto de eventos a partir do qual serão enviados os comando de edição NCL.

A instanciação dessa classe ocorre no momento de iniciação (chamada ao método "*initXlet*") do *Xlet*. Isso porque essa classe deve acessar o contexto do Xlet para ter acesso às suas propriedades.

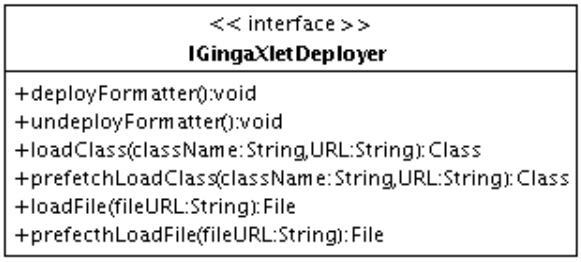

Figura 31 – Interface IGingaXletDeployer.

A classe que implementa a interface *IGingaXletDeployer*, mostrada na Figura 31, é responsável pela implantação do Formatador. O processo todo é realizado quando chamado o método *deployFormatter*. Esse método esconde toda a lógica de localização dos componentes (que podem apresentar-se no fluxo MPEG-2, no sistema de arquivos local ou serem obtidos via canal de retorno), quais componentes serão carregados e a conexão entre tais componentes. Mais detalhes sobre o funcionamento desse método são fornecidos na Seção 5.3. Em contrapartida, seu método *undeployFormatter* é responsável por realizar a

finalização dos componentes permitindo, caso o ambiente permita, o armazenamento destes para futuro reuso.

A classe possibilita, ainda:

- O carregamento de classes a partir da URL de uma biblioteca Java;
- A pré-busca (*prefetch*) de classes a partir da URL de uma biblioteca JAVA;
- O carregamento de arquivos a partir de uma URL de localização do arquivo;
- E o pré-carregamento de arquivos a partir de uma URL de localização do arquivo.

A classe *StreamEventHandler* é responsável por tratar os eventos DSM-CC enviados no fluxo MPEG-2. Sua assinatura, mostrada na Figura 32, é herdada da interface `org.dvb.dsmcc.StreamEventListener` descrita na especificação do MHP.

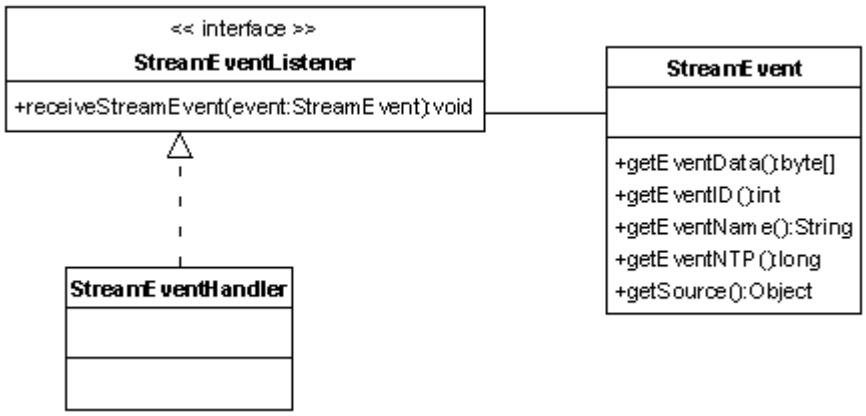

Figura 32 – A classe StreamEventHandler e suas associações.

A implementação do método *receiveStreamEvent* dessa classe determinará, por exemplo, o tratamento apropriado para os comandos de edição. Esse tratamento envolve o envio dos Comandos de Edição NCL para o componente Gerenciador de Documentos ou a alteração do estado da apresentação do Formatador. Dessa forma, essa classe possuirá uma referência para o Componente Gerenciador de Documentos e o Componente do Núcleo do Formatador.

Para exercer suas funções, essa classe deverá se cadastrar, para notificação, em um objeto de eventos (*Stream Event Object*) específico do Carrossel de Objetos. Esse objeto de eventos é obtido através da interface *IGingaXletConfiguration*. Ao se cadastrar nesse objeto, a classe será notificada à medida que os eventos DSM-CC aparecerem no fluxo MPEG-2.

5.2.3.
O Processo de Implantação do Formatador

A seguir será descrito o processo de implantação do formatador partindo do produtor de conteúdo.

5.2.3.1.
O Produtor

O produtor de conteúdo será responsável por enviar os componentes e os documentos para a apresentação. Para isso, o produtor deverá sinalizar, via AIT, a localização no Carrossel DSM-CC dos componentes de software e dos documentos e mídias envolvidas na apresentação.

Caso seja adotada a estratégia de pré-conversão do documento NCL produtor deverá instanciar o Componente Gerenciador de Documentos, adicionar os documentos necessários para a apresentação e adicionar este novo objeto Java serializado no mesmo diretório onde se localizarão os componentes.

5.2.3.2.
O Receptor

O processo de implantação do Formatador *Xlet* inicia-se quando sua presença é sinalizada para o receptor (por exemplo, através da AIT). Nessa fase é passada como parâmetro para a aplicação, obrigatoriamente, a localização dos exibidores e do *Stream Event Object* através do qual serão recebidos os eventos de edição.

Depois de recebida a sinalização, o Gerenciador de Aplicações assume o controle do Formatador com a instanciação da classe *GingaXlet*. O gerenciador, então, inicia o ciclo de vida do *GingaXlet* com a chamada ao seu método *initXlet*, herdado da interface *Xlet* do Java TV.

No processo de preparação da classe *GingaXlet*, ilustrado na Figura 33, será realizada a configuração do ambiente através da classe *GingaXletConfiguration* (classe que implementa a interface *IGingaXletXonfiguration*). A essa classe é passado o Contexto de execução do *Xlet* de onde é obtida parte dos parâmetros de configuração utilizados pela aplicação, dentre eles os argumentos passados para a

aplicação (como o Localizador das bibliotecas e o Localizador do objeto de eventos que conterá os eventos para a identificação de recursos exposta na Seção 4.3 e demais eventos de edição NCL).

Após a criação da entidade de configuração, é instanciada a entidade de implantação do Formatador denominada *GingaDeployer* (classe que implementa a interface *IGingaXletDeployer*). Essa entidade, como já mencionado, está diretamente ligada à versão do Formatador a ser implantada. Ela fará o carregamento de cada componente realizando sua instanciação e resolvendo suas dependências.

Feita a implantação do Formatador, o *GingaDeployer* passará o controle desta entidade ao *GingaXlet*. Assim, qualquer alteração no estado da aplicação (*Xlet*) poderá ser refletido no estado da apresentação do Formatador via *GingaXlet*, ou seja, no momento em que os métodos *pauseXlet* ou *destroyXlet* dessa classe forem chamados, a execução do Formatador será interrompida.

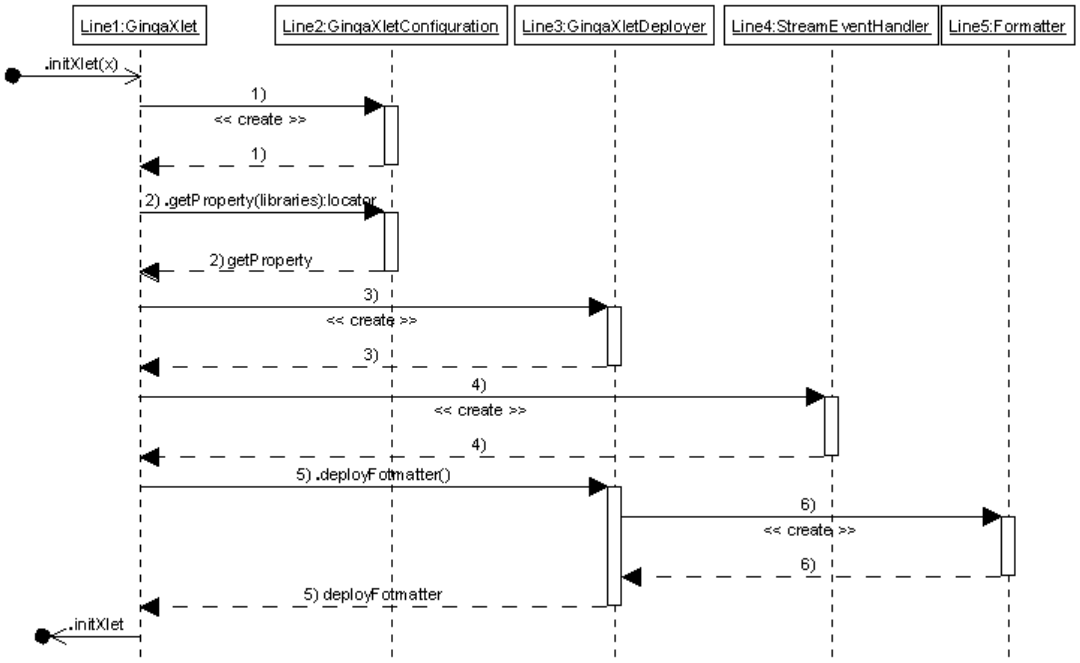

Figura 33 – Diagrama de seqüência do processo de preparação do Formatador *Xlet*.

Ainda no método de iniciação do *Xlet* será criado o *StreamEventHandler*. A essa classe serão passadas as referências para o Componente Gerenciador de Documentos e o Componente do Núcleo do Formatador.

No método *startXlet* da classe *GingaXlet*, seu atributo que contém a referência para o *StreamEventHandler* se cadastrará em um objeto de eventos DSM-CC (fornecido pelo *IGingaXletConfiguration*). Esse objeto é representado pela classe *DSMCCStreamEvent* do MHP. Após o cadastro, o

StreamEventhandler receberá os comandos de edição e repassará para os Componentes do Núcleo do Formatador e Gerenciador de Documentos, permitindo o início da apresentação. Esse processo é mostrado na Figura 34.

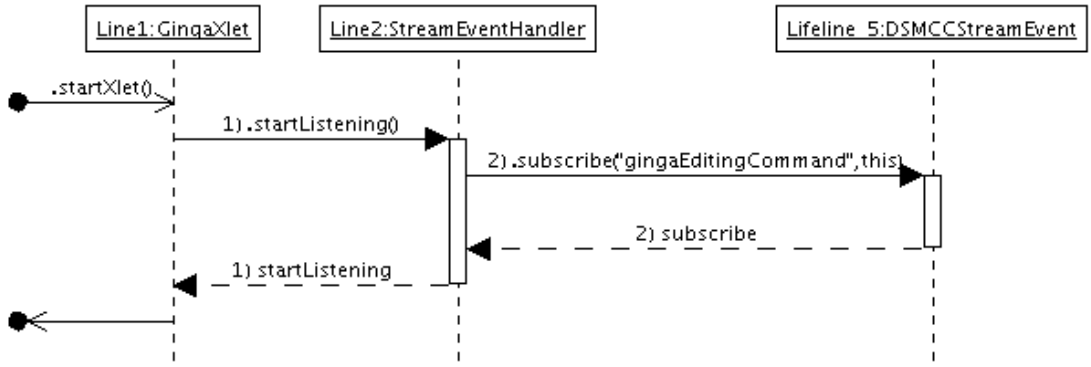

Figura 34 – Diagrama de seqüência do processo de inicialização do Formatador *Xlet*.

No método *pauseXlet* a apresentação do Formatador é **parada** (e não apenas pausada) e o *StreamEventHandler* é descadastrado do objeto de Eventos DSM-CC deixando de receber os comandos de edição.

No método *destroyXlet* ocorre o mesmo procedimento do método *pauseXlet*, no entanto, é o método *undeployFormatter* da classe *GingaXletDeployer* é chamado para finalizar os componentes.

5.2.4. Persistência do Formatador NCL Xlet em Receptores MHP

A implementação do Formatador NCL para sistemas GEM proposta nesse trabalho se adapta facilmente tanto ao mecanismo de *plug-ins* como ao armazenamento de aplicações, mostrados no Capítulo 3, sem grandes modificações.

No caso do mecanismo de *plug-ins*, a implementação da interface `org.dvb.application.plugins.Plugin` torna-se trivial, sendo necessário apenas instanciar a implementação da classe *GingaXlet* e retorná-la no método *initApplication*. Dessa forma, o sistema agirá, na presença de um documento NCL, como se possuísse nativamente a capacidade de interpretá-lo.

No caso da utilização do mecanismo de *Application Storage*, são grandes as vantagens apresentadas. Nenhum esforço precisa ser feito em termos de implementação. A diferença das aplicações que utilizam esse mecanismo está apenas no processo de sinalização.

Um fator restritivo na utilização desses dois mecanismos é que eles devem ser utilizados isoladamente em uma rede de comunicação, ou seja, na apresentação de um programa deverá ser utilizado apenas um dos tipos de sinalização possíveis para a execução do Formatador NCL: o convencional que trata o Formatador como um aplicação Java; a sinalização de um documento NCL como uma aplicação a ser executada pelo *plug-in*; ou a sinalização para execução de uma aplicação armazenada.

5.3. Testes

Para a validação do sistema criado nesta dissertação foi criado um ambiente de testes utilizando a configuração de máquina virtual CDC (SUN, 2005b) oferecida pela SUN juntamente com o perfil PBP. Foi utilizada, ainda, uma implementação do *middleware* MHP aberta conhecida como OpenMHP (Tucs & Axel, 2005). Nesse *middleware* foram realizadas adaptações de forma a melhor refletir um ambiente de TV Digital. A plataforma escolhida foi um sistema *Linux* instalado sobre uma arquitetura *x86*.

O sistema implantado no teste é mostrado na Figura 35. Ele é composto por:

- Um Núcleo do Formatador com suporte à edição ao vivo;
- Um Gerenciador de Leiaute customizado para TV;
- Um Gerenciador de Documentos customizado para TV;
- Um Gerenciador de Exibidores customizado para TV;
- Um componente exibidor de Texto customizado para TV; e
- Um componente exibidor de Imagem customizado para TV.

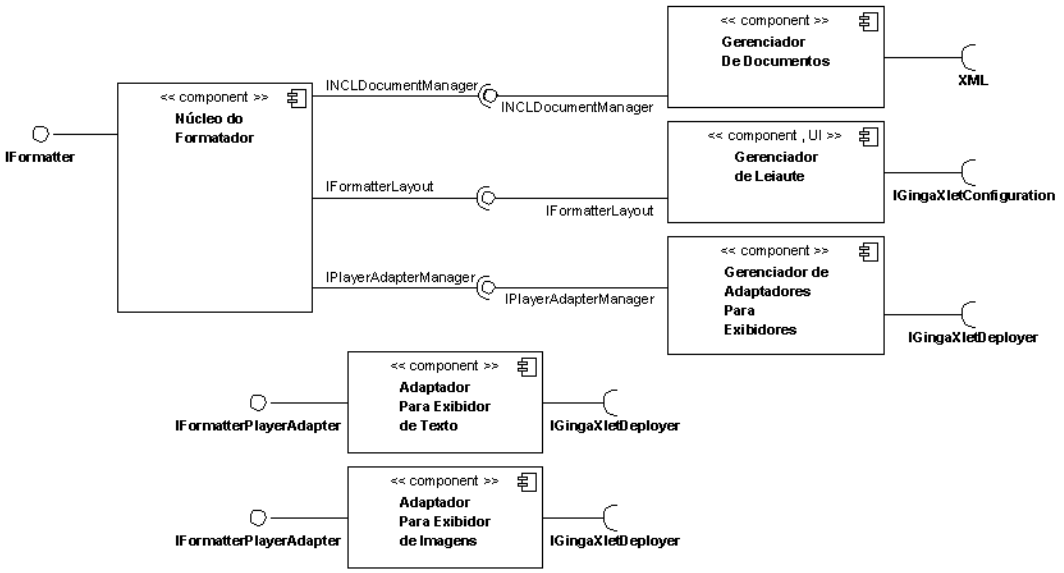

Figura 35 – Digrama de componentes do Sistema testado.

O tamanho final dos componentes implementados é mostrado na Tabela 1, sendo que:

- O arquivo "Deployer.jar" corresponde ao componente de implantação;
- O arquivo "Compile.jar" corresponde ao Gerenciador de Documentos;
- O arquivo "Ginga.jar" corresponde ao Núcleo do Formatador, Gerenciador de exibidores e Gerenciador de Leiaute somados;
- O arquivo "ImagePlayerAdapter.jar" corresponde ao componente do exibidor de imagens estáticas; e
- O arquivo "TextPlayerAdapter.jar" corresponde ao componetne exibidor de texto.

Componente	Tamanho
Deployer.jar	6,5 KB
Compiler.jar	111 KB
Ginga.Jar	148 KB
Diserializer.Jar	1,5 KB
ImagePlayerAdapter	4,4 KB
TextPlayerAdapter	4,3 KB
Total	275,7 KB

Tabela 1 – Componentes do formatador GingaXlet

A fim de verificar a viabilidade da serialização do componente Gerenciador de Documentos, foram realizados testes de desempenho comparando dois

cenários: serialização contra a conversão do documento XML. Na serialização o tempo foi medido desde a leitura do objeto serializado (que foi pré-convertido) até a presença de sua instância. Na conversão o tempo foi medido entre a leitura do documento XML até a presença da instância do Gerenciador de Documentos juntamente com o documento convertido. Os arquivos testados fazem parte de um conjunto de documentos elaborados pelo Laboratório de Telemídia da Puc-Rio para o teste de conformidade do Ginga-NCL. Para cada arquivo testado foram realizados 1000 ensaios. Nos testes, os documentos a serem executados foram adicionados ao Gerenciador de Documentos e este foi serializado. A implementação utilizada da interface *IGingaXletDeployer* realiza a implantação do Formatador através da desserialização do componente Gerenciador de Documentos e instanciação dos demais componentes do sistema.

	Serialização(ms)			Conversão(ms)		
Documentos	Mínimo	Máximo	Médio	Mínimo	Máximo	Médio
connectorBase.ncl	10	15	11	25	29	26
descriptor01.ncl	26	60	31	47	54	49
descriptor02.ncl	27	134	33	49	54	50
descriptor03.ncl	33	60	38	55	67	57
descriptor04.ncl	33	96	39	56	152	59
descriptor05.ncl	27	49	30	49	54	50
descriptor06.ncl	26	42	28	47	53	49
descriptor07.ncl	30	38	32	53	61	55
descriptor08.ncl	33	61	35	56	156	59
descriptor09.ncl	27	33	28	49	55	50
descriptor10.ncl	27	34	28	49	56	51
descriptor11.ncl	27	33	28	49	56	50
descriptor12.ncl	26	43	28	49	55	50
Structure01.ncl	9	21	10	11	14	12
Structure02.ncl	9	19	10	11	14	12
Structure03.ncl	9	19	10	11	18	12
Structure04.ncl	9	18	10	12	63	13
Structure05.ncl	9	14	10	11	19	12

Tabela 2 – Resultados dos testes de Serialização versus conversão

Os resultados obtidos pelos testes são mostrados na

	Serialização(ms)			Conversão(ms)		
Documentos	Mínimo	Máximo	Médio	Mínimo	Máximo	Médio
connectorBase.ncl	10	15	11	25	29	26
descriptor01.ncl	26	60	31	47	54	49
descriptor02.ncl	27	134	33	49	54	50
descriptor03.ncl	33	60	38	55	67	57
descriptor04.ncl	33	96	39	56	152	59
descriptor05.ncl	27	49	30	49	54	50
descriptor06.ncl	26	42	28	47	53	49
descriptor07.ncl	30	38	32	53	61	55
descriptor08.ncl	33	61	35	56	156	59

descriptor09.ncl	27	33	28	49	55	50
descriptor10.ncl	27	34	28	49	56	51
descriptor11.ncl	27	33	28	49	56	50
descriptor12.ncl	26	43	28	49	55	50
Structure01.ncl	9	21	10	11	14	12
Structure02.ncl	9	19	10	11	14	12
Structure03.ncl	9	19	10	11	18	12
Structure04.ncl	9	18	10	12	63	13
Structure05.ncl	9	14	10	11	19	12

Tabela 2. Pode-se perceber que: em todos os testes o cenário utilizando a serialização é, em média, ligeiramente inferior, em se tratando de tempo de espera; e que os testes não mostraram nenhum indício de que tal cenário não possa ser adotado.

Outro teste a ser realizado consiste em verificar o tamanho do arquivo NCL serializado em comparação com o arquivo em sua forma convencional e se o retardo provocado por essa diferença é definitivamente relevante. No entanto esse teste será alvo de trabalhos futuros.