4
Implementação do Gerente de Contexto

Como discutido no Capítulo 2, o Sistema HyperProp (Soares et al., 2003a) é um sistema hipermídia baseado no modelo NCM. Ao longo dos últimos anos, a implementação do sistema vem passando por diversos processos de melhoria, necessários para incorporar ao sistema os refinamentos do modelo NCM e acoplar novas funcionalidades.

Este capítulo se limita a discutir a implementação da estrutura de gerenciamento de contexto utilizada pelo sistema, baseada nos aspectos discutidos no Capítulo 3, tais como: estrutura de dados para representação da informação de contexto, vocabulário adotado, existência de um servidor de contexto centralizado e intercâmbio de informações através de diversos tipos de protocolo de comunicação.

A arquitetura genérica para gerência de contexto concebida neste trabalho é representada através de dois *frameworks*. A representação conceitual utilizando *frameworks* é devida à facilidade de identificação dos pontos de flexibilização através dos quais esta arquitetura genérica pode ser especializada, pelo projetista, para cenários específicos, de forma a tornar possível a sua validação. A adoção de *frameworks* (Fayad et al., 1999) favorece a reutilização e, ao mesmo tempo, permite a personalização do modelo apresentado em diferentes contextos, o que se deve à sua capacidade de modularizar, combinar e estender seus componentes.

Os *frameworks* concebidos foram:

- *Framework* de Modelagem da Informação Contextual;
- *Framework* para Gerência de Contexto;

Outro objetivo deste capítulo é apresentar uma instanciação para provisão da informação contextual ao Sistema HyperProp (Soares et al., 2000) servindo como avaliação dessa proposta. A fim de melhor demonstrar o funcionamento global dos *frameworks,* eles deverão ser integrados ao formatador hipermídia adaptativo construído no Laboratório Telemídia (Rodrigues, 2003), bem como conter um

exemplo de adaptação realizada no servidor, para melhor demonstrar a adaptação de documentos hipermídia com orientação ao contexto como um todo.

Para auxiliar no entendimento deste capítulo, convém lembrar que na descrição conceitual dos *frameworks,* foi adotada a linguagem UML (*Unified Modeling Language*) (UML, 1997) para sua especificação, documentação e visualização. Nos diagramas de classes foram utilizadas cores distintas para diferenciar as classes-base (em cinza), daquelas que representam possíveis instanciações dos pontos de flexibilização desses elementos (em branco). Nesses diagramas, foi ainda utilizada uma grafia diferenciada para representar as classes abstratas (itálico) e as concretas (regular).

4.1. Framework de Modelagem da Informação Contextual

Uma das principais bases para o mecanismo de gerência de contexto consiste na representação correta dos atributos envolvidos, através de um formato aberto e extensível, permitindo incorporações futuras de atributos novos ou até mesmo, na descontinuação de algum deles, sem impactos para as aplicações que consomem essa informação contextual.

O modelo da informação contextual deve conter informações que possam ser usadas para caracterizar a situação de uma entidade onde, por entidade, compreende-se uma pessoa, um lugar, ou um objeto que possa ser considerado relevante na interação com uma dada aplicação, incluindo o usuário e as aplicações propriamente ditas.

A informação contextual pode ser descrita através de vocabulários genéricos definindo um conjunto de atributos que atenda ao interesse geral de diferentes comunidades e possa ser usada por diversas aplicações, ou ser composto por um conjunto de atributos específico ao domínio de uma aplicação. A vantagem da primeira abordagem é evitar a proliferação de definições distintas para um mesmo atributo e, com isso, potencializar o reuso de definições já formalizadas por alguma comunidade. Além disso, se mesmo assim uma aplicação específica ainda tiver a necessidade de definir determinados atributos pertinentes ao seu domínio, tais aplicações podem apenas definir esse pequeno conjunto e, através de espaços de nomes, fazer o uso conjunto de múltiplos vocabulários.

Considerando esse tipo de cenário, a modelagem desse *framework* permite que o gerenciador de contexto seja capaz de lidar com múltiplos vocabulários, além de prever a possibilidade de um mesmo atributo constar da definição de mais de um tipo de entidade. Por exemplo, seja o atributo idioma, que define a língua de uma nação ou a língua peculiar de uma região; o idioma pode tanto representar as diferentes línguas que um dispositivo cliente é capaz de entender como, também, as preferências do usuário em termos de língua, quando da apresentação de um documento.

Apesar deste trabalho propor a criação de um novo vocabulário, a arquitetura concebida para o *framework* que modela a informação de contexto não está restrita a adoção/incorporação de um vocabulário único, sendo este, portanto, mais um ponto de flexibilização a ser ressaltado e que foi considerado durante a concepção deste trabalho. Deve-se, ao instanciar o *framework*, definir um vocabulário que servirá para consultas, atualizações etc., dos diversos atributos do contexto.

Como pode ser observado na Figura 4.1, os perfis e seus atributos representam outro ponto de flexibilização do *framework*. Pode-se definir novos perfis e atributos de acordo com a aplicação para a qual o gerente de contexto está sendo instanciado. Cada perfil (Perfil de Dispositivo Cliente, Perfis de Usuário, Perfil do Servidor, Perfil da Rede, etc.) irá representar um conjunto de atributos, assim como estender o vocabulário do contexto.

4.1.1. Componentes do Framework de Modelagem da Informação Contextual

A Figura 4.1 retrata o *framework de modelagem da informação contextual* através das classes e dos respectivos relacionamentos que o constituem. As classes abstratas *Context*, *Profile* e *GenericProfileAttribute* representam, respectivamente, as hierarquias de contexto, perfil e atributo que compõem um contexto. O objetivo desse *framework* é desacoplar os contextos dos perfis que os constituem, garantindo dessa forma uma maior flexibilidade na descrição da informação contextual. Através do *pattern* estrutural Bridge (Gamma, 1995) os contextos são representados como conjuntos de perfis, o que garante sua extensibilidade. Como se pode ver na Figura 4.1, as classes *Context* e *Profile* estão associadas pelo

relacionamento de agrupamento `profileList`. Os métodos `getProfile()`, `addProfile()` e `removeProfile()`, disponibilizados em *Context*, são responsáveis por, respectivamente, retornar uma instância da subclasse *Profile*, identificadas pelo atributo `profileId`, e adicionar novos atributos que definirão novos perfis, remover ou mesmo adequar-se aos já existentes. O método `addProfile()` verifica se o perfil a ser adicionado já existe, de forma a garantir que não existam perfis com mesmo `profileId`.

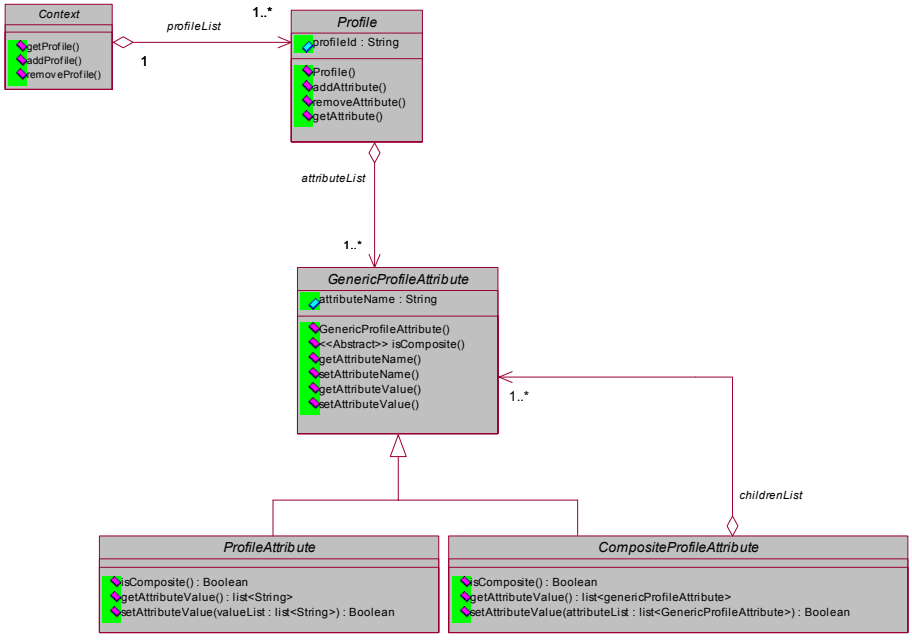

Figura 4.1– Framework de Modelagem da Informação Contextual

A classe abstrata *GenericProfileAttribute* modela os atributos de um vocabulário. Mais uma vez o *pattern* estrutural Bridge é utilizado, agora para representar os atributos como um conjunto de perfis, o que se consegue através do relacionamento de agrupamento `attributeList`. Essa representação oculta das aplicações que desejam consultar contexto, certos detalhes referentes à estruturação dos atributos, tornando tais aplicações mais independentes desses atributos e da própria forma como eles estão estruturados. Isso é obtido pelo método `getAttribute()`, definido na classe *Profile*, que retorna uma instância de uma subclasse de *GenericProfileAttribute* identificada através do atributo `attributeName`.

A classe *ProfileAttribute* implementa de forma básica as funcionalidades de um atributo simples, enquanto a classe *CompositeProfileAttribute* implementa, também de forma básica, as funcionalidades de um atributo composto, através do *pattern* estrutural Composite (Gamma, 1995).

A função `isComposite()` retorna se um determinado atributo é composto ou não. Um atributo composto *CompositeProfileAttribute* pode ser obtido pelo método `getAttributeValue()`. No caso do atributo ser simples, o método `getAttributeValue()` recebe um nome de atributo `attributeName` e retorna uma lista de valores do tipo *string,* contendo um único valor ou vários valores associados a um atributo.

Vale salientar que, de forma análoga ao método `addProfile()`, o método `addAttribute()` verifica se o atributo a ser adicionado já existe de forma a garantir que não existam atributos com mesmo `attributeName`.

4.1.2. Exemplo

A seguir será apresentado um exemplo de aplicação do *framework do modelo da informação contextual* que implementa um vocabulário definido pelo grupo WAP Forum (WAP-UAProf) e cujo objetivo é descrever as características de dispositivos móveis, sem fio. O cenário modelado representa alguns atributos de um tal dispositivo.

A classe *Profile* foi especializada na classe *WapUAProf*. Como foi apresentado na Seção 3.1.3.1.2, esse vocabulário define cinco componentes que especializam a subclasse *CompositeProfileAttribute*: *HardwarePlatform*, *SoftwarePlatform, WapCharacteristic, BrowserUA* e *NetworkCharacteristic*.

Os atributos simples que compõem esses cinco componentes especializam a subclasse *ProfileAttribute,* sendo alguns deles exemplificados na Figura 4.2.

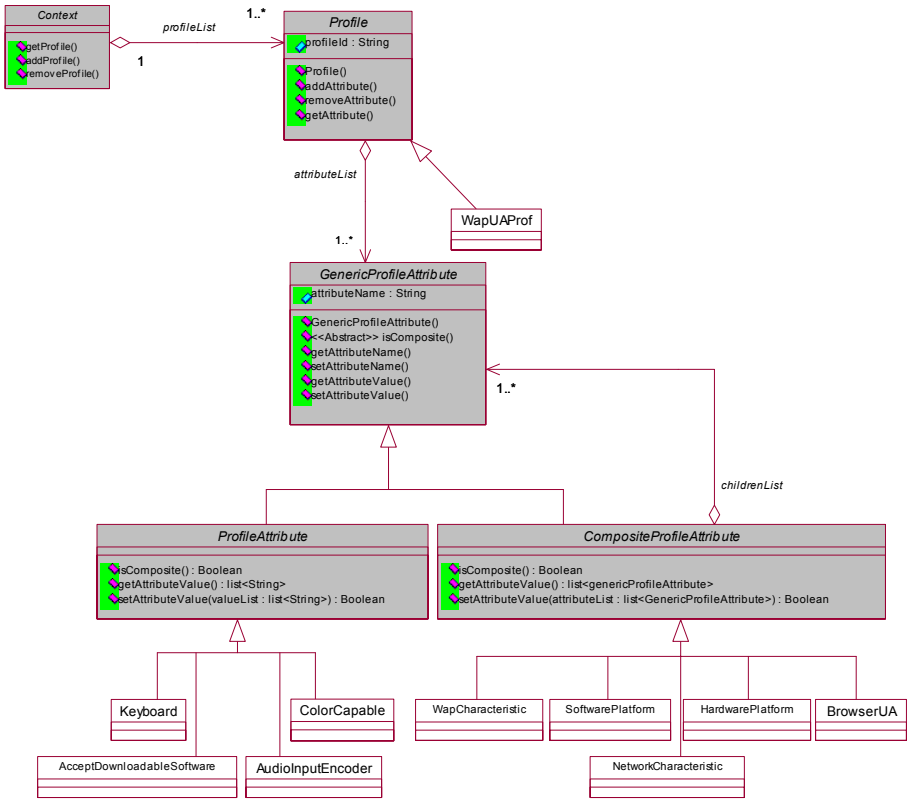

Figura 4.2 – Exemplo de aplicação do Framework de Modelagem da Informação Contextual

4.2. Framework para Gerência de Contexto

O conceito relacionado às aplicações orientadas ao contexto compreende tanto a responsabilidade por detectar, armazenar, traduzir e disseminar a informação de contexto como também, personalizar os conteúdos solicitados, procurando refletir as necessidades e preferências do usuário. Com o objetivo de retirar a sobrecarga imposta por estas tarefas de gerenciamento da informação contextual da arquitetura das aplicações orientadas ao contexto, em particular, dos sistemas hipermídia com suporte à adaptação, a idéia é poder contar com uma estrutura independente e reutilizável inteiramente responsável pelo gerenciamento da informação contextual.

Além da redução na complexidade das arquiteturas dessas aplicações, outras vantagens agregadas por esta proposta são: maior reusabilidade, isto é, prover um conjunto de funções comuns à maioria das aplicações orientadas ao contexto e aumentar o intercâmbio dessas informações através do uso de um formato de estrutura de dados padrão baseado em XML.

As funcionalidades relacionadas à gerência de contexto são modeladas no *framework para gerência de contexto*, como será apresentado na Seção 4.2.1.

4.2.1. Componentes do Framework para Gerência de Contexto

A Figura 4.3 traz uma visão conjunta dos componentes de gerência de contexto. A existência de uma única instância da classe ContextManager foi assegurada pelo uso do pattern de criação Singleton (Gamma, 1995), permitindo que a mesma saiba coordenar diferentes eventos.

Além disso, o *pattern* Facade (Gamma, 1995) foi utilizado para dividir todo o gerenciador em uma estrutura independente, representada no *framework* por seu pacote, de acordo com a arquitetura definida. Através da aplicação do design *pattern* Facade, o pacote possui sempre a classe de "fachada", que é a única classe visível ao pacote representante da camada superior, e serve justamente para sua comunicação com outras aplicações.

No diagrama de classes da Figura 4.3, pode-se observar, inicialmente, que a aplicação implementa a classe *ContextManager*. A implementação dessa classe é

obrigatória nos casos em que a aplicação necessite de informações sobre o contexto. A consulta à informação contextual é efetivamente tratada pelo método `getProfile()`, disponibilizando uma consulta a todos os atributos que definem um determinado perfil ou uma consulta a um atributo específico.

A classe abstrata *UpdatedProfileContainer* representa o armazenamento de um conjunto de perfis com informações contextuais atualizadas referente a um dado instante. O uso do *pattern* estrutural Bridge (Gamma, 1995) permite desacoplar o meio de armazenamento (cache, banco de dados etc.) do tipo de estrutura de dados (XML, CC/PP etc.) na qual os perfis atualizados estão representados.

Na arquitetura proposta, o gerenciador de contexto possui um *repositório centralizado*, previamente configurado, que armazena descrições padrões a respeito de alguns perfis. O uso do *pattern* estrutural Bridge (Gamma, 1995), novamente permite desacoplar o meio de armazenamento do tipo de estrutura de dados que representa os *perfis de referência*. O método `getRefProfile()` é responsável pela recuperação de um *perfil de referência*.

Exemplificando, se uma aplicação estiver sendo utilizada por dois ou mais dispositivos clientes do tipo Compaq Presario, o *perfil de referência* descreve todos os atributos desse tipo de dispositivo e, com isso, só é necessário que o gerente de contexto se preocupe em coletar aquele conjunto de atributos cujo valor associado foi alterado. Dessa forma, os agentes que coletam contexto de dispositivos cliente não precisam se preocupar em enviar uma descrição de contexto extensa (com todos os atributos), além da vantagem de reutilizar uma mesma definição de dispositivo para diversos clientes com mesma configuração de dispositivo.

A classe abstrata *ProfileCollector* é responsável pela obtenção e envio de informações de gerenciamento mediante comunicação com um ou mais agentes coletores de contexto. Um ponto de flexibilização importante pertinente a essa função de coleta refere-se ao diálogo gerente-agentes, podendo ocorrer em dois sentidos: *sob demanda*, isto é, o gerente envia uma solicitação explícita na qual requisita que lhe sejam informados os parâmetros e valores que sofreram modificações, comparados às descrições padrões mantidas nos perfis de referência, ou através de *notificação espontânea*, onde o(s) agente(s), periodicamente, informa(m) ao gerente o conjunto de parâmetros e valores

alterados. Se o método `verifyUpdate()` tiver na sua assinatura o parâmetro `wasCalled` com o valor "true", isto indica que o gerente de contexto está invocando o agente coletor através da classe *DiffProfileMonitor*, solicitando o envio das alterações referente a um determinado perfil. Caso o parâmetro `wasCalled` contenha valor "false", o método `verifyUpdate()` será chamado pelo agente coletor para notificar os atributos alterados de um determinado perfil. O método `mergeProfile()` mescla o conjunto de atributos de um perfil de referência com o conjunto de atributos alterados.

É importante ressaltar que o diálogo entre o Gerente de Contexto e qualquer outra aplicação pode dar-se através de qualquer protocolo de comunicação, constituindo-se, portanto, em um importante ponto de flexibilização deste *framework*. Da mesma forma, o diálogo entre o gerente e os agentes também pode utilizar diferentes protocolos de comunicação.

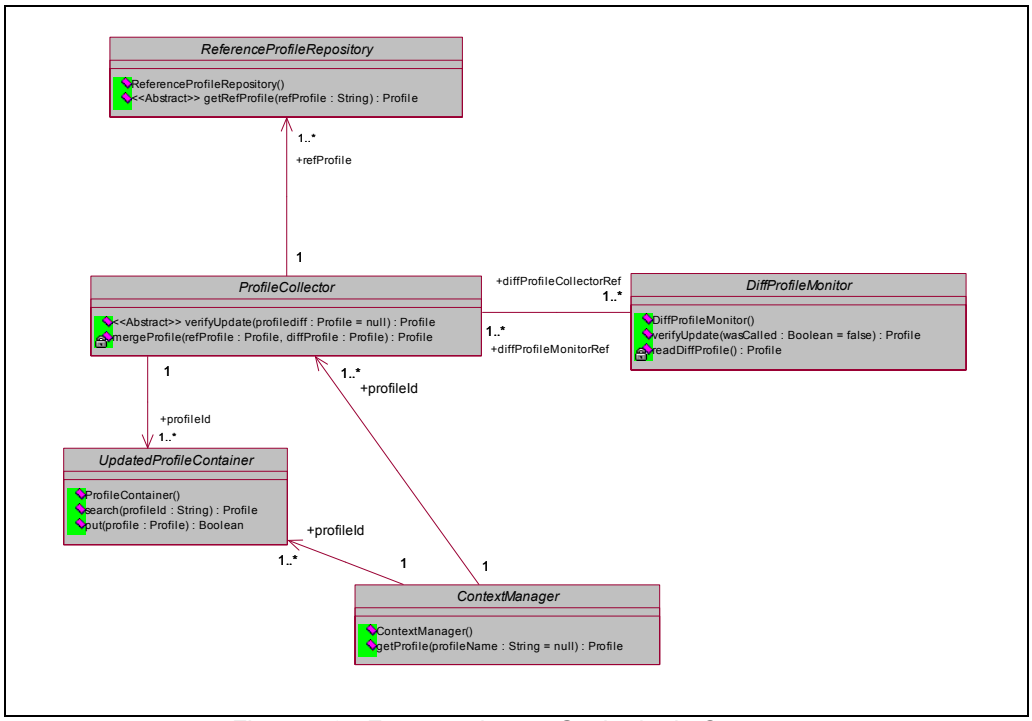

Figura 4.3 - Framework para Gerência de Contexto

4.2.2. Exemplo

O cenário demonstrado nesta seção procura exemplificar um diálogo entre o Gerente de Contexto e uma aplicação qualquer através do protocolo de comunicação SOAP. Além disso, descreve que os perfis atualizados são

agrupados em uma tabela hash, enquanto que os perfis de referência utilizam o CC/PP como forma de representação de dados. Finalmente, o diálogo entre gerente e agente coletores emprega um protocolo de comunicação proprietário da linguagem Java, denominado RMI (*Remote Method Invocation*).

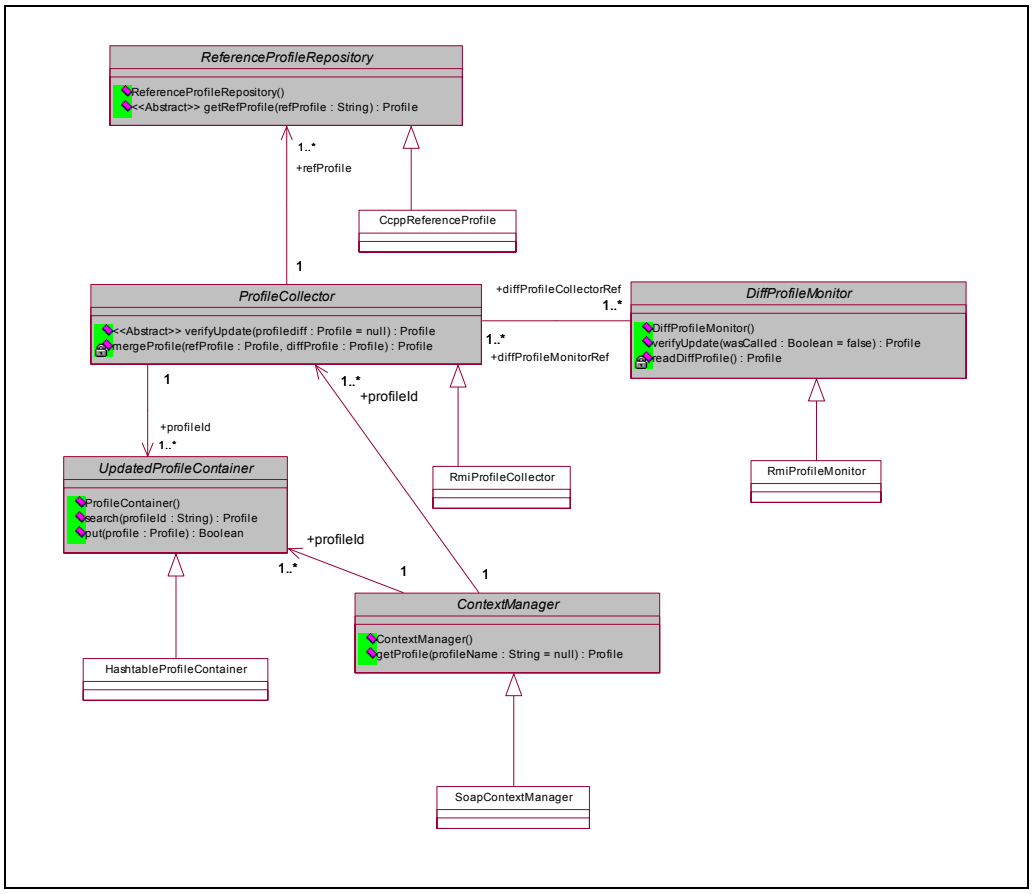

Figura 4.4 - Exemplo de aplicação do Framework para Gerência de Contexto

4.3. Ambiente de Implementação

A Seção 2.2.2 apresentou uma arquitetura de um sistema hipermídia adaptativo ideal, que não deve restringir a localização do mecanismo de adaptação a um único local. Outra contribuição desta dissertação é propor maior flexibilização à arquitetura do sistema HyperProp, distribuindo os mecanismos de adaptação entre o ambiente de armazenamento (ou em um elemento intermediário que represente a figura do ambiente de armazenamento) e o ambiente de exibição. As principais vantagens dessa proposta são: diminuir a sobrecarga tanto nas configurações dos servidores como nas dos dispositivos clientes, reduzir a complexidade dos algoritmos de adaptação localizados em um único ambiente e

otimizar a utilização dos recursos que estejam situados no servidor, no caminho até o dispositivo cliente e no próprio cliente.

Embora esse *elemento intermediário* possa estar de fato em uma entidade intermediária na rede ou acoplado ao ambiente de armazenamento do sistema Hyperprop, sua criação objetivou minimizar os impactos decorrentes de alterações no servidor desse sistema. A estrutura básica desse agente intermediário é modelada pelo diagrama da Figura 4.11 e seu funcionamento é detalhado na Seção 4.3.4.3.

4.3.1. *Context-Aware* Hyperprop - Cenário de Uso

Esta seção descreve uma instanciação dos *frameworks para Modelagem e Gerência do Contexto* através de um cenário no qual o sistema HyperProp é a aplicação que interage com o Gerente de Contexto em duas etapas distintas: durante a compilação do hiperdocumento e durante a sua apresentação. Para validar esta proposta, foi implementado um diálogo entre Gerente-Hyperprop e Gerente-Agentes baseado no protocolo de comunicação HTTP (*HyperText Transfer Protocol*), utilizado amplamente por um grande número de aplicações.

4.3.2. Simplificações Adotadas

A instanciação dos *frameworks para modelagem e gerência de contexto* tem como objetivo validar a modelagem proposta e gerenciar a informação contextual coletada, levando em consideração as requisições de consulta sobre um contexto específico. Será efetuada uma série de simplificações em relação à instanciação dos *frameworks*, para viabilizar a implementação do cenário-exemplo:

- Para definição do vocabulário deste trabalho, assume-se que uma comunidade ligada à área de sistemas hipermídia auxiliaram na definição do conjunto dos principais atributos que constam no vocabulário adotado;

- O repositório central de perfis foi previamente configurado e carregado com perfis de referência de dispositivos clientes, servidores de conteúdo e preferências de usuários;

- A forma como as preferências dos usuários foram coletadas, isto é, se de forma automática ou através de questionário não determinam impacto para estrutura do gerente de contexto e, por isso, não é detalhada no cenário apresentado;

- Diferentes aplicações têm acesso irrestrito a qualquer consulta relativa à informação de contexto de seu interesse, ou seja, as políticas de segurança e privacidade não são incorporadas no contexto deste trabalho;

- A implementação do HTTP serve de base para a instanciação do protocolo de comunicação, sendo usado tanto para consultas ao contexto, como coleta de modificações de contexto;

- A coleta de modificações no contexto de cada entidade assume um diálogo entre o gerente e um agente principal, que apenas consulta a respectiva MIB procurando identificar quais atributos que compõem o perfil em questão sofreram modificações desde a última consulta. Como a MIB foi criada e como os vários agentes atualizaram essas base de dados não são detalhados no escopo deste trabalho;

- Nesse cenário, não foram considerados o contexto da rede de acesso e do servidor de conteúdo para adaptação do conteúdo;

4.3.3. Descrição do Cenário-Exemplo

Para ilustrar o uso dos elementos *switch*, considere um documento NCL identificado por "hpropWeatherNews". O documento ilustrado pela Figura 4.5 contém um *switch* composto por dois nós de composição alternativos, selecionados pelo ambiente de armazenamento com base no contexto de exibição do mesmo.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <ncl id="hpropWeatherNews" xmlns="http://www.telemidia.puc-
rio.br/specs/xml/NCL" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.telemidia.puc-rio.br/specs/xml/NCL
http://www.telemidia.puc-rio.br/specs/xml/NCL.xsd">
+ <head>
- <body>
- <composite id="weather-news">
- <switch id="snwc-subt01">
  <bindRule rule="ruleAudioOn" component="weather2003-audio01" />
  <bindRule rule="ruleAudioOff" component="weather2003-audio02" />
```

```
    - <composite id="weather2003-audio01">
  <port id="snwc-end" component="weatherPucText" port="subt06-end" />
+ <text id="weatherPucText" descriptor="faceDescriptor01"
src="file:files/midia/news/weather/20031114weather.html">
  <img id="brazilSatellitePicture" descriptor="logoDescriptor01"
src="file:files/midia/news/weather/brazil_sat_277x187.jpg" />
+ <switch id="snwc-subt01">
+ <switch id="snwc-subt02">
+ <switch id="snwc-subt03">
+ <switch id="snwc-subt04">
+ <switch id="snwc-subt05">
+ <switch id="snwc-subt06">
+ <linkBase>
    </composite>
  - <composite id="weather2003-audio02">
  <port id="snwc-end" component="weatherPucText" port="subt06-end" />
  <text id="weatherPucTextAlt" descriptor="faceDescriptor03"
src="file:files/midia/news/weather/2003114weather.html" />
  <img id="brazilSatellitePictureAlt" descriptor="logoDescriptor01"
src="file:files/midia/news/weather/brazil_sat_277x187.jpg" />
+ <linkBase>
    </composite>
        </switch>
  </composite>
  </body>
  </ncl>
```

Figura 4.5 – Representação de um documento NCL identificado por “weather-news”

O nó de composição “weather2003-audio01” representa o documento constituído por um nó de texto descrevendo a previsão meteorológica da cidade do Rio de Janeiro em quatorze de Novembro de 2003, apresentado por uma aplicação de animação facial composta por uma personagem e sua fala (Lucena, 2002) e um nó de imagem representando uma fotografia do satelite mostrando as diferentes densidades de nuvens que cobrem a referida cidade. Por outro lado, o nó de composição “weather2003-audio02” contém os mesmos nós de texto e nó de imagem da composição anterior exceto pelo fato de a apresentação não envolver a aplicação de animação facial para apresentação sintetizada do nó de texto.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <ncl id="hpropWeatherNews" xmlns="http://www.telemidia.puc-
rio.br/specs/xml/NCL" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.telemidia.puc-rio.br/specs/xml/NCL
http://www.telemidia.puc-rio.br/specs/xml/NCL.xsd">
  - <head>
      - <layout>
      - <topLayout id="hpropWindow" title="HyperProp Weather News" width="800"
          height="600">
            <region id="faceRegion01" width="250" height="400" />
            <region id="subtRegion01" top="400" width="250" height="200" />
            <region id="faceRegion02" left="550" width="250" height="400" />
            <region id="videoRegion" top="150" width="450" height="300" />
            <region id="logoRegion01" left="250" width="200" height="200" />
            <region id="logoRegion02" left="250" width="300" height="600" />
        </topLayout>
      </layout>
  + <descriptorBase>
  - <presentationRuleBase>
```

```
        <presentationRule id="ruleAudioOn" var="systemAudioDesc" op="eq"
            value="true" />
        <presentationRule id="ruleAudioOff" var="systemAudioDesc" op="eq"
            value="false" />
      - <compositePresentationRule id="ruleEnCaptionsTrue" op="and">
            <presentationRule id="ruleAudioTrue" var="systemLanguage" op="eq"
                value="en" />
            <presentationRule id="ruleCaptionsTrue" var="systemCaptions" op="diff"
                value="false" />
        </compositePresentationRule>
      - <compositePresentationRule id="rulePtCaptionsTrue" op="and">
            <presentationRule id="ruleAudioTrue" var="systemLanguage" op="eq"
                value="pt-br" />
            <presentationRule id="ruleCaptionsTrue" var="systemCaptions" op="diff"
                value="false" />
        </compositePresentationRule>
    </presentationRuleBase>
  </head>
+ <body>
</ncl>
```

Figura 4.6 – Representação da base de regras

A seleção de uma das composições alternativas depende dos parâmetros de áudio e de legenda obtidos a partir da consulta ao contexto em um instante específico. Além disso, a legenda pode ser exibida em dois idiomas, de acordo com a preferência do usuário. Essa seleção corresponde ao processamento do documento NCL apresentado pela Figura 4.5, gerando um novo documento NCL, adaptado, contendo apenas um dos nós de composição, especificando ou não a apresentação do nó de texto de forma sintetizada. Além disso, o documento NCL adaptado especifica também o idioma da legenda de apresentação.

Depois de gerar o documento NCL adaptado, o sistema HyperProp foi utilizado para importar o documento final e apresentá-lo com o uso do formatador (Rodrigues, 2003). A Figura 4.7 ilustra a visão estrutural do documento importado, usando o browser gráfico do HyperProp.

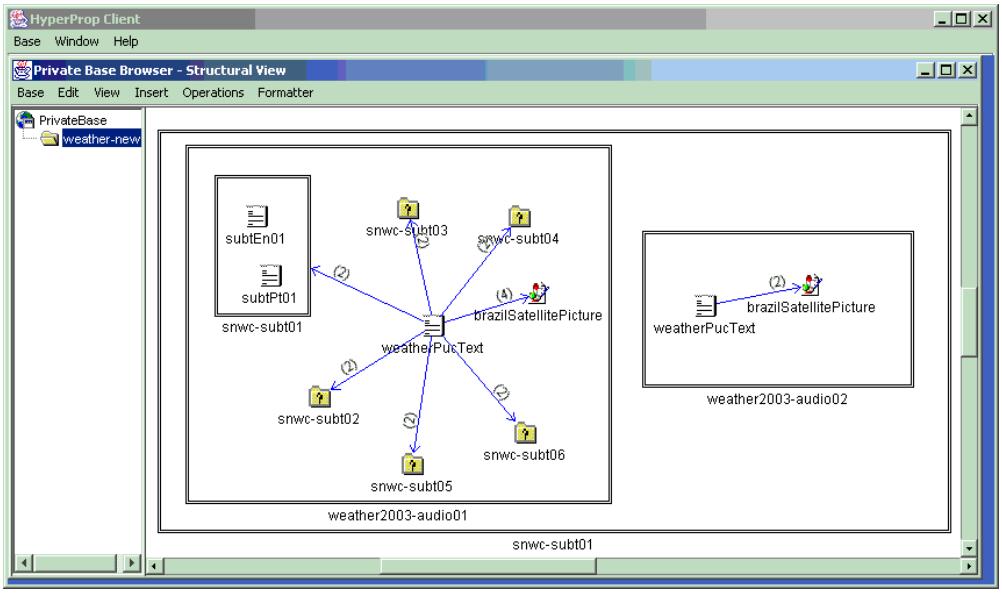

Figura 4.7 - Visão estrutural do documento final no sistema HyperProp

Ao executar o documento final "hpropWeatherNews" usando o formatador HyperProp, cada trecho do áudio é sincronizado com a parte da legenda correspondente e no idioma preferido pelo usuário como mostra a Figura 4.8. As características de apresentação de cada nó em um documento NCL são definidas através de um elemento *descritor*, que deve ser associado a cada um dos nós de mídia: o nó de texto apresentado pela aplicação de animação facial, o nó de imagem representado a figura do satélite e os nós de texto que representam a legenda no idioma desejado.

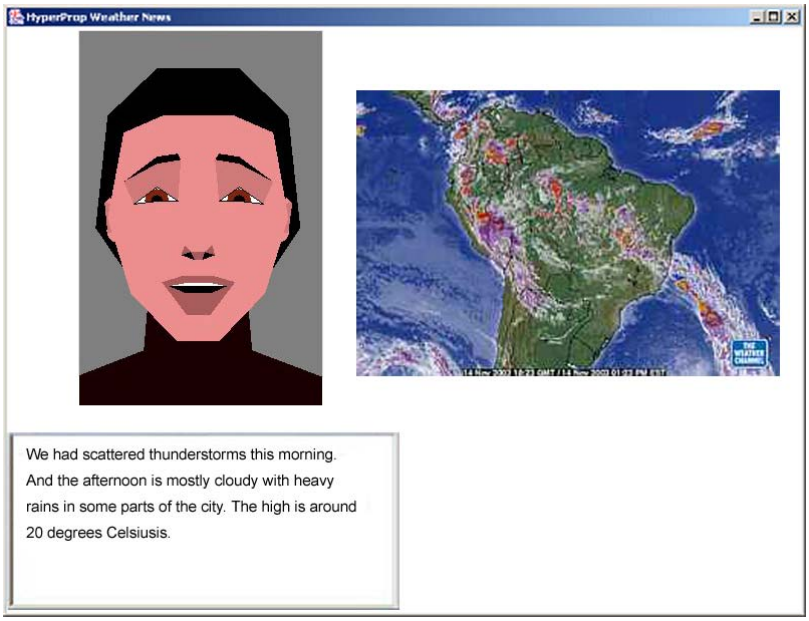

Figura 4.8 - Tela da apresentação do documento "weather-news"

4.3.4.
Instanciando os Frameworks para Gerência de Contexto

4.3.4.1.
Definindo o modelo de informação contextual

Como detalhado no Capítulo 3, este trabalho criou um vocabulário mais abrangente em termos de descrição de características de dispositivo, chegando a estender essa descrição para servidores de conteúdo, rede de acesso e preferências dos usuários. Entretanto, é importante lembrar que a característica de adaptação presente no sistema HyperProp também demonstrou a necessidade de parâmetros mais específicos e particulares aos sistemas hipermídia adaptativos. A Figura 4.9 ilustra o vocabulário contemplado na implementação descrita neste capítulo.

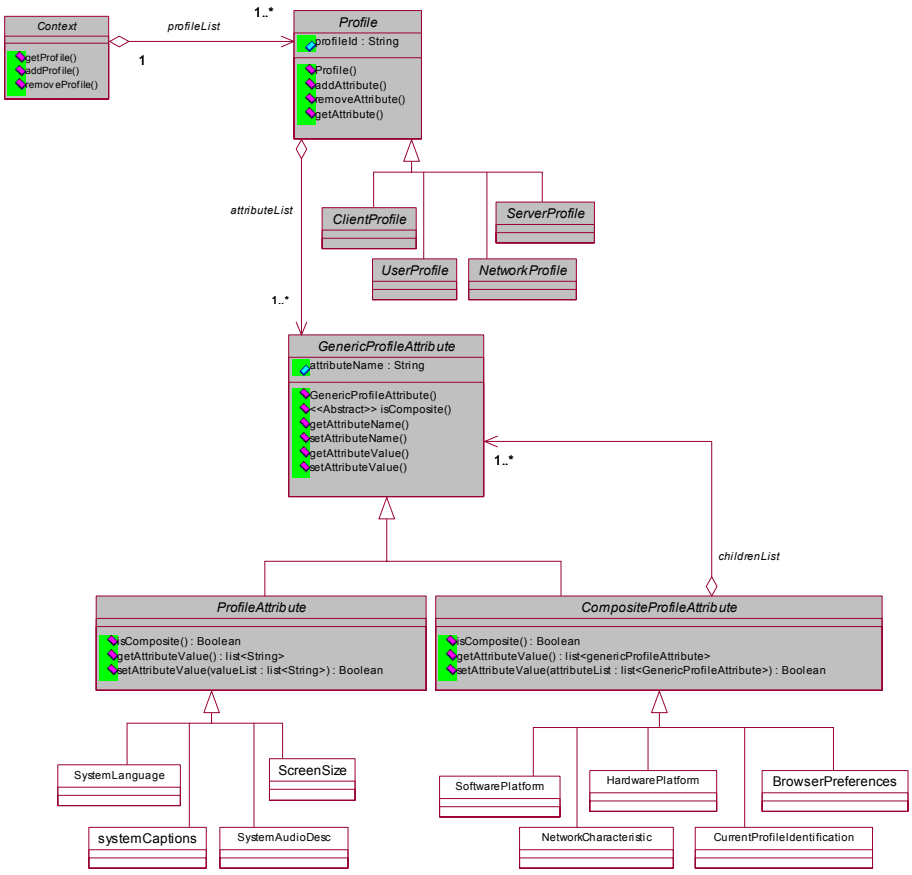

Figura 4.9 – Vocabulário “Hypermedia Context Vocabulary”

4.3.4.2. Definindo a estrutura de gerência de contexto

A Figura 4.10 ilustra o diagrama de classes da estrutura de gerência de contexto que foi definida para o funcionamento deste framework integrado ao Sistema HyperProp.

Como é possível observar, nesta implementação, a troca de mensagens entre o sistema HyperProp-Gerente de Contexto, o Gerente de Contexto-AgenteCliente e o Gerente de Contexto-AgenteUsuário através do protocolo HTTP, como já explicado no Capítulo 3.

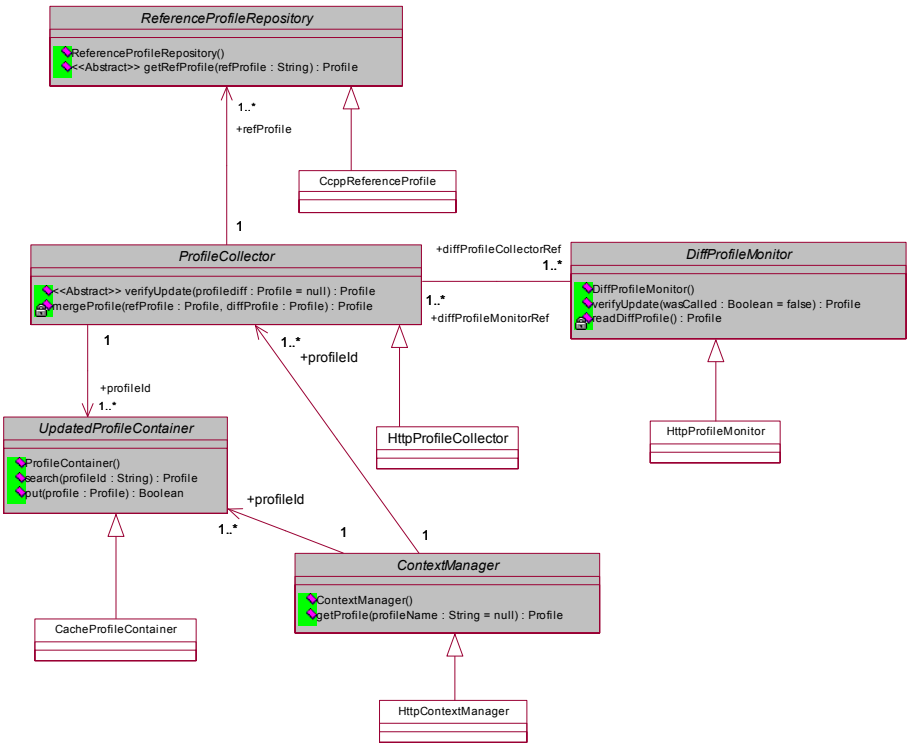

Figura 4.10 – Framework de Gerência de Contexto instanciado para integração ao Sistema HyperProp

Finalmente, para intercâmbio da informação contextual, a representação dos dados utiliza o *framework* CC/PP.

4.3.4.3. Definindo o compilador NCL 2.0 para NCL 2.0 adaptada

A linguagem NCL apresenta um certo nível de complexidade, refletido nas onze *Áreas Funcionais* que a compõem. Para facilitar o desenvolvimento de

Compiladores NCL, foram identificadas algumas funcionalidades que poderiam ser implementadas uma única vez e reaproveitadas por cada um deles. Compiladores são programas estendidos, que fazem uma análise do documento NCL e geram como resultado uma estrutura de dados refletindo, total ou parcialmente, o documento NCL inicial. Como resultado dessa análise, (Oliveira e Silva et al., 2003) desenvolveu um parser em linguagem Java, oferecendo uma estrutura genérica (*framework*) para a implementação de *compiladores NCL* específicos.

Este trabalho implementa um compilador NCL – AdaptedNCL, que responde pela função: avaliar regras de seleção de um nó *switch* e regras de exibição, gerando uma estrutura representando um documento NCL, contendo apenas uma única alternativa de nó *switch.*

Como demonstrado pelo diagrama de classes da Figura 4.11, essa função pode ser desempenhada pela implementação do *proxy,* que representa o ambiente de armazenamento (classe *NclAdaptedNclPresentationControlCompiler*), utilizando as informações contextuais coletadas junto à estrutura de gerência de contexto (classe *ContextManager*) e que, neste trabalho, também é implementado através do mesmo *proxy.*

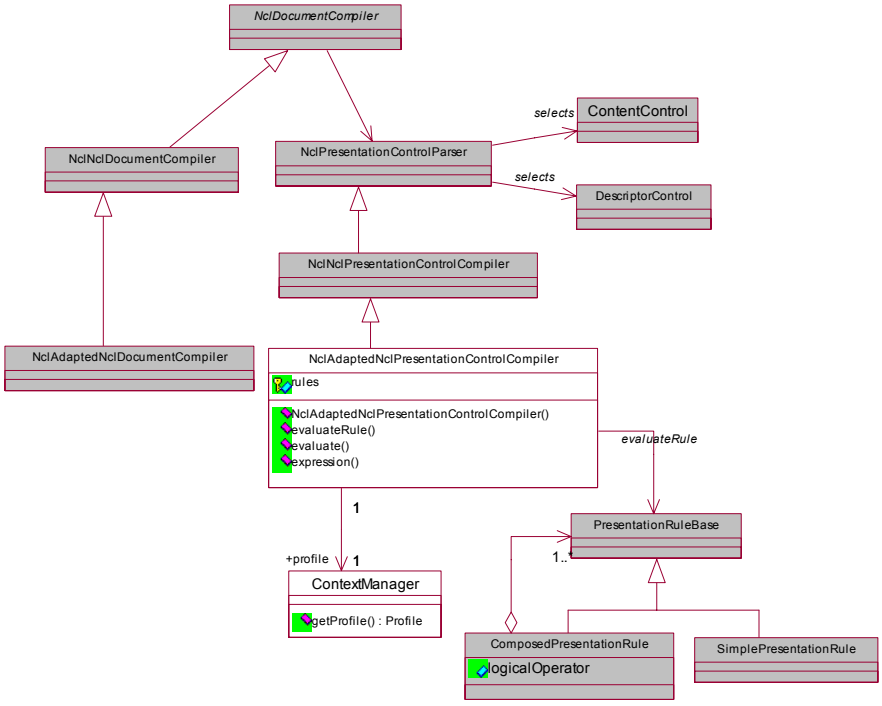

Figura 4.11– Diagrama de classes para regras de apresentação

O parser genérico contém um pacote (pacote *ncl.parser*) de classes abstratas (e algumas concretas), onde a estruturação das classes segue a modularização de NCL, existindo uma classe para cada área funcional da linguagem. As principais classes que constituem o parser são, portanto: *NclComponentsParser*, *NclConnectorParser*, *NclInterfaceParser*, *NclLayoutParser*, *NclLinkingParser*, *NclPresentationControlParser*, *NclPresentationSpecificationParser*, *NclStructureParser*, *TemplateProcessor*. Além dessas nove classes, o parser genérico também contém uma outra classe, denominada *NclDocumentCompiler*, que provê uma estrutura básica de gerência e alguns métodos para facilitar a criação de compiladores. Vale salientar que, exceto a classe *NclPresentationControlParser*, as demais classes não serão discutidas neste trabalho, de forma que, maiores detalhes sobre o parser e suas demais classes podem ser encontrados no relatório técnico (Oliveira e Silva et al., 2003).

A subclasse *NclAdaptedNclPresentationControlCompiler* estende a classe *NclPresentationControlParser*, representante da área funcional *Presentation Control* da NCL. Seu objetivo é especificar alternativas de conteúdo e, ainda, de exibição para um documento de forma similar à área funcional *Content Control* da linguagem SMIL 2.0 (SMIL, v. 2.0).

Como ilustra a Figura 4.12, o módulo *PresentationRuleBase* permite a definição de uma base de regras, que pode conter regras simples, e/ou regras compostas. A regra simples é formada apenas por um id, atributo de contexto, um valor associado e um operador que pode ser do tipo `gt`, `ls`, `gteq`, `lseq`, `equal`, `diff`. Já a regra composta é formada por um id, um operador do tipo `and` ou `or` e um conjunto de regras aninhadas.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <ncl id="hpropNews" xmlns="http://www.telemidia.puc-rio.br/specs/xml/NCL"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.telemidia.puc-rio.br/specs/xml/NCL
http://www.telemidia.puc-rio.br/specs/xml/NCL.xsd">
- <head>
- <presentationRuleBase>
    <presentationRule id="rule30" var="screenDepth" op="eq" value="4" />
  - <compositePresentationRule id="xrule333" op="or">
    - <compositePresentationRule id="xrule32" op="and">
      - <compositePresentationRule id="xrule32" op="and">
          <presentationRule id="rule222" var="screenDepth" op="diff" value="4" />
          <presentationRule id="rule222" var="screenDepth" op="gteq" value="32" />
          <presentationRule id="rule223" var="screenDepth" op="eq" value="32" />
        </compositePresentationRule>
      </compositePresentationRule>
```

```
        </compositePresentationRule>
    </presentationRuleBase>
</head>
```

Figura 4.12– Módulo PresentationRuleBase de Regras em um documento NCL

O módulo *NclPresentationControl* especifica o elemento *switch*, tal como em SMIL, permitindo a definição de nós componentes alternativos do documento escolhidos em tempo de compilação do documento, avaliando as regras, definidas pelo módulo *PresentationRuleBase*. Entretanto, como mostra a Figura 4.13, o elemento *switch* de NCL não tem conteúdo idêntico ao elemento homônimo de SMIL, pois pode conter, além de objetos de mídia, elementos *composite*, definidos pelo módulo *BasicComposite* de NCL 2.0.

```
- <switch id="s1">
        <bindRule rule="rule1" component="ts1" />
        <bindRule rule="xrule5" component="ts3" />
        <bindRule rule="xrule6" component="ts5" />
       <text descriptor="text_d1" id="ts1"
      src="file:files/midia/exemplo01/musicas/coisadepele/letra/versos01.html" />
        <audio descriptor="text_d1" id="ts2"
      src="file:files/midia/exemplo01/musicas/coisadepele/letra/versos01.html" />
        <text descriptor="text_d1" id="ts3"
      src="file:files/midia/exemplo01/musicas/coisadepele/letra/versos01.html" />
        <text descriptor="text_d1" id="ts4"
      src="file:files/midia/exemplo01/musicas/coisadepele/letra/versos01.html" />
        <text descriptor="text_d1" id="ts5"
      src="file:files/midia/exemplo01/musicas/coisadepele/letra/versos01.html" />
</switch>
```

Figura 4.13– Módulo ContentControl em um documento NCL

A classe *NclPresentationControlParser* possui o método *parseSwitch*, que é responsável por compilar e retornar um *switch*. Inicialmente é criado um objeto representando o switch através do método *createSwitch*. Em seguida, têm-se algumas possibilidades, dependendo de dois atributos booleanos da classe: *alwaysParseComponent* e *parseAfterFirstMatch*. O método *setAlwaysParseComponent* pode ser utilizado para alterar o valor de *alwaysParseComponent*. Já o método *setParseAfterFirstMatch* pode ser utilizado para alterar o valor de *parseAfterFirstMatch*.

Se o atributo *alwaysParseComponent* tiver o valor verdadeiro, sempre todos os elementos contidos no elemento *switch* são compilados, independente dos atributos que compõem a regra ao qual esteja associado. Já se o atributo *alwaysParseComponent* tiver o valor falso, as regras de apresentação são avaliadas primeiramente, antes de se compilar os elementos contidos no switch. Se a regra tiver um valor falso, o elemento associado a esta regra não será compilado.

Especificamente em relação à avaliação de uma regra associada ao elemento *switch*, quando o atributo *parseAfterFirstMatch* tiver o valor verdadeiro, após a primeira regra verdadeira, o parser continua analisando os demais elementos do *switch*. Já no caso do atributo *parseAfterFirstMatch* ter o valor falso, após a primeira regra ser avaliada como verdadeira, o parser finaliza o tratamento do elemento *switch*.

O método *parseSwitchComponent* é responsável por compilar um componente de *switch*, analisando o elemento recebido como parâmetro (*composite*, *switch* ou *BasicMedia*). Caso seja um switch, chama o método *parseSwitch*, se for *composite*, chama o método *parseComposite* de *NclComponentParser* e se for um objeto de mídia, chama o método *parseBasicMedia* de *NclComponentParser*.

Duas questões merecem ser destacadas. A primeira delas é que como alguns elementos do *switch* podem não possuir regras associadas, eles são considerados como tendo regras sempre verdadeiras. Esses elementos são adicionados ao *switch*, preservando sua ordenação no arquivo, ao final da avaliação das regras. Esses elementos são adicionados ao *switch* através de uma variação, também abstrata, do método *addComponentToSwitch* da classe *NclComponentParser*.

A segunda questão refere-se ao fato de que aqueles compiladores que precisam resolver regras durante a compilação, como é o caso do *NclAdaptedNclCompiler*, devem re-implementar o método *evaluateRule* para avaliar as regras. Nesses casos, se o compilador desejar substituir todos os *switchs* por um nó que tenha uma regra verdadeira, ele deve atribuir de maneira consistente os atributos booleanos dessa classe, ou seja: *alwaysParseComponent* falso e *parseAfterFirstMatch* também falso. Se o compilador desejar apenas excluir os elementos com regras falsas, deve atribuir *alwaysParseComponent* falso e *parseAfterFirstMatch* verdadeiro. Se o compilador não for avaliar regras, deve atribuir *alwaysParseComponent* verdadeiro e *parseAfterFirstMatch* verdadeiro.

4.3.5.
Cenário de Uso

Para ilustrar o funcionamento da estrutura de gerência de contexto apresentada nas seções anteriores, foram criados dois cenários. O primeiro cenário implementa um *proxy* para seleção de alternativas de nó *switch,* acoplado ao ambiente de exibição. Já no segundo cenário de implementação, o *proxy* é separado do ambiente de apresentação, onde foi usada uma segunda estação, dentro da rede do laboratório Telemídia, para hospedar este *proxy* Figura 4.14.

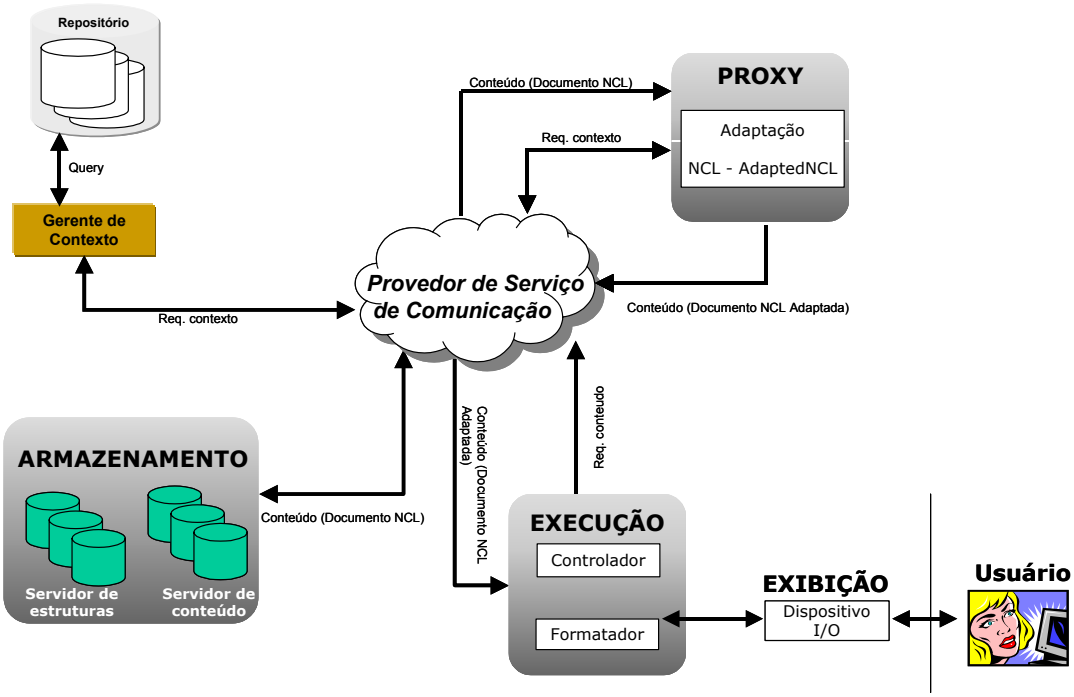

Figura 4.14 – Cenário de uso

4.3.6.
Resultados obtidos

Para realizar as implementações citadas, foi criado uma pequena aplicação browser através da qual foi possível simular a alteração dos valores de alguns parâmetros do contexto, em particular, os citados no cenário-exemplo (Seção 4.3.3). Durante os testes, esses atributos sofreram alterações de valores, gerando mensagens de notificação, onde os agentes cliente e usuário enumeram apenas os parâmetros alterados para o Gerente de Contexto.