5
Implementação do ProxyFramework

O Capítulo 4 apresentou a arquitetura proposta para sistemas de disseminação de conteúdo com adaptação baseada no contexto dos clientes. Nesta arquitetura, a função de adaptação de conteúdo está concentrada na *camada de adaptação baseada em contexto*. Mas, dado que a maioria das adaptações está relacionada com características do enlace sem fio ou do dispositivo móvel, estas adaptações devem ser realizadas "próximo ao enlace sem fio". Propõe-se, então, a utilização de um proxy da aplicação que fique responsável pelas funções de adaptação baseada em contexto, como mostrado na Figura 5.1. Ter a adaptação realizada por um proxy apresenta as seguintes vantagens: *i*) não há necessidade de alterar o provedor de conteúdo, *ii*) proxies possuem a flexibilidade de poder ser implantados nas "bordas"da rede fixa (perto dos pontos de acesso) mais próximos dos clientes.

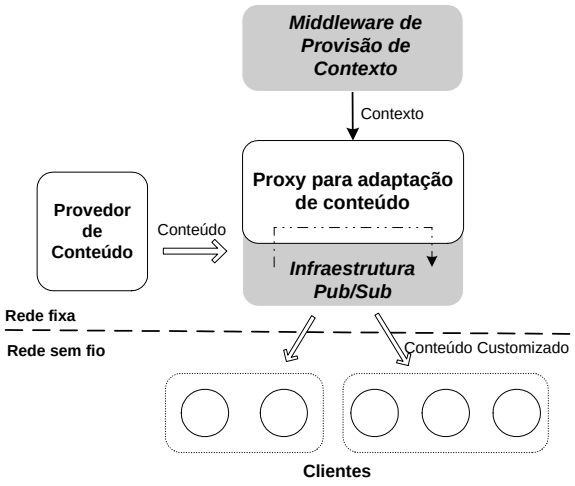

Figura 5.1: Proxy para adaptação de conteúdo

Este capítulo apresenta o ProxyFramework, desenvolvido seguindo o projeto da camada de adaptação da arquitetura proposta, que implementa os principais componentes relacionados ao suporte à adaptação de conteúdo. O ProxyFramework também contempla parte da infraestrutura Pub/Sub e de sua interface com a camada de adaptação, bem como a interface com o middleware provedor de contexto. Além disso, são fornecidos exemplos para criação de re-

gras de adaptação e seus adaptadores (camada de aplicação). A Seção 5.1 apresenta uma visão geral do framework e a Seção 5.2 discute os principais passos para instanciação de um proxy para adaptação de conteúdo utilizando o framework.

5.1 Visão Geral do ProxyFramework

A adaptação de conteúdo, quando oferecida, pode ser realizada no middleware, na aplicação ou em ambos, como em [86]. Quando a capacidade de adaptação é provida pela camada de middleware, a aplicação não precisa monitorar o ambiente e pode deixar as adaptações completamente a cargo do middleware. Nesta abordagem, o middleware provê adaptação de melhor esforço para computação móvel geral, onde a informação de contexto é completamente escondida da aplicação. Entretanto, as tomadas de decisões de como e quando adaptar um processamento ou dado são muito relacionadas às características da aplicação.

Na abordagem adotada neste trabalho, conforme a arquitetura proposta no Capítulo 4, as regras que levam às adaptações e os adaptadores são definidos/implementados na camada de aplicação. Entretanto, a monitoração de contexto e o controle sobre a execução das adaptações são efetuados por uma camada de middleware de proxy. Essa leva em consideração o contexto corrente de cada cliente e determina, em tempo de execução, qual adaptação ou adaptações devem ser empregadas para cada cliente.

Para facilitar o desenvolvimento de proxies capazes de adaptação de conteúdo sensível aos contextos de clientes móveis, foi implementado um framework (ProxyFramework [121]) como parte da arquitetura MoCA (Seção 2.4). O ProxyFramework intermedeia a comunicação entre o servidor da aplicação e seus clientes móveis e também serve como interface com serviços da MoCA. Além disso, o framework visa aumentar o reuso de código, permitindo sua extensão e personalização para criar instâncias de *proxies de aplicação* de acordo com as necessidades específicas da aplicação. A parte concreta (*frozen-spot*) do framework é responsável por identificar o contexto corrente dos dispositivos clientes e acionar as adaptações apropriadas, de acordo com regras definidas pelo desenvolvedor da aplicação. Essa parte provê também mecanismos para o armazenamento temporário (*buffering*) de mensagens em momentos de desconexão ou de fraca conectividade do cliente, bem como a seleção e invocação de adaptações selecionadas.

A Figura 5.2 apresenta a estrutura do ProxyFramework, que foi desenvolvida seguindo o projeto da Camada de Adaptação de Conteúdo da arquitetura

proposta na Seção 4.2.

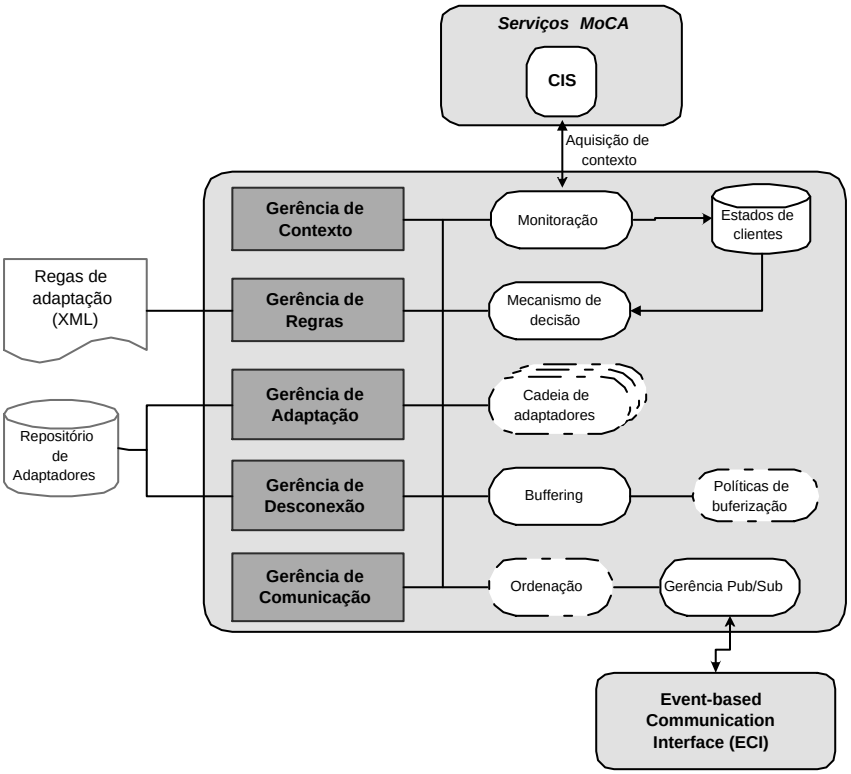

Figura 5.2: Arquitetura Proxy Framework

O serviço de **provisão de contexto** utilizado é o CIS da MoCA (Seção 2.4). Este serviço permite a definição de contextos complexos através de expressões lógicas que combinam diversas informações de contexto, tanto estáticas (i.e. características do dispositivo móvel, como tamanho de tela, número de cores, etc), quanto dinâmicas (tais como: qualidade do enlace, ponto de acesso, quantidade de energia da bateria).

A **gerência de contexto** é responsável por coletar e gerenciar o estado de contexto corrente dos clientes. Tais estados são definidos pelos contextos de interesse da aplicação (configurados via XML). A obtenção das informações de contexto necessárias para detecção do estado de contexto para cada cliente é realizada através do registro do proxy junto ao CIS da MoCA para recebimento de notificações de alteração de contexto de cada cliente. Os estados de contextos dos clientes são armazenados em um repositório local no proxy para agilizar a sua consulta pelo *gerente de regras*. Essa consulta é feita no momento da tomada de decisão de quais adaptações serão necessárias para cada cliente.

O módulo de **gerência de regras de adaptação** consulta as regras para adaptação (contextos que disparam as adaptações) definidas através de um arquivo de configuração em XML. Esse módulo é responsável por

verificar o estado de contexto corrente do cliente e indicar qual a seqüência de adaptadores a ser usada (cadeia de adaptadores) para a mensagem. A avaliação dos contextos dos clientes destinatários ocorre sempre que há o envio de uma mensagem para eles, e é neste momento que se determina a necessidade de adaptação e quais os adaptadores deverão ser utilizados. A descrição detalhada da configuração de regras é apresentada na Seção 5.2.2. Na implementação atual as regras são carregadas apenas na iniciação do proxy e não se permite uma atualização dinâmica dessas regras de adaptação.

A **gerência de adaptadores** é responsável pela execução dos adaptadores selecionados na ordem definida pelas regras. Caso o adaptador não esteja disponível, p.ex. se uma regra passa a ser válida pela primeira vez, ele faz a carga dinâmica, buscando o adaptador em um repositório, e o instancia no proxy[1]. Para execução dos adaptadores é utilizado o algoritmo de otimização discutido na Seção 4.3.4, que tenta agrupar clientes de acordo com situações de contextos iguais. As adaptações podem ser implementadas por um único módulo, que abrangeria todas as alternativas, ou por vários pequenos módulos que podem ser compostos ou encadeados. Como cada contexto/cliente possui demandas diferentes de adaptação de mensagens, é mais producente desenvolver e usar componentes que implementem uma única adaptação simples e que possam ser combinados para criar configurações, i.e. cadeias, de adaptações, arbitrariamente complexas e específicas para cada cliente ou contexto. Por exemplo, em uma aplicação com disseminação de imagens, devido às condições da rede sem fio, poderia eventualmente ser preciso reduzir o tamanho (em bytes) da imagem, reduzindo sua qualidade, enquanto que características do dispositivo poderiam impedir a apresentação da imagem em seu formato original, sendo preciso, por exemplo, converter a imagem para um formato aceito pelo dispositivo (ex: JPEG para BMP) ou redimensionar a imagem devido a uma tela de dimensões pequenas. Esse componente atende a necessidade de composição de módulos de adaptação, possibilitando assim que múltiplas adaptações possam ser aplicadas consecutivamente de acordo com as diferentes variáveis que compõe o contexto corrente.

A **gerência de desconexão** requisita informações sobre eventos de desconexão e reconexão do dispositivo da gerência do contexto. Na ocorrência de desconexões, pode ativar o buffering de mensagens do cliente desconectado. A política de bufferização adotada deve ser implementada pelo desenvolvedor do proxy (ver Seção 5.2.3).

O componente **gerência de comunicação** é responsável pela interação

[1]Os adaptadores são classes Java, que são carregadas para o ProxyFramework através de uma extensão da classe ClassLoader (do pacote java.lang)

do proxy com a infra-estrutura de comunicação utilizada. A infra-estrutura Pub/Sub usada no ProxyFramework é o ECI da MoCA, por este fornecer as funcionalidades de interceptação de notificações necessárias ao funcionamento do proxy. Entretanto, um outro middleware publish/subscribe (como Siena[54]) poderia também ser utilizado, sendo necessário apenas adaptar a interface de interação, para permitir ao proxy interceptar as notificações desse middlware. Além da comunicação assíncrona (publish/subscribe), o framework também permite a interação com clientes através de diferentes protocolos de comunicação síncrona (*request/reply*), possibilitando assim a sua utilização em uma gama maior de aplicações. A gerência de comunicação também possibilita a adoção de diferentes estratégias de ordenação na entrega de mensagens, tais como ordenação por cliente ou segundo alguma prioridade. Estas podem ser escolhidas e/ou implementadas pelo desenvolvedor do proxy, seguindo a interface descrita na Seção 5.2.4.

O conjunto de componentes concretos (*frozen-spot*) do ProxyFramework implementam as funcionalidades básicas necessárias a um proxy de adaptação de conteúdo e constituem os componentes de gerência: de contexto, de regras, de execução de adaptações, de desconexão e de comunicação. As interfaces de componentes abstratos (*hot-spot*), representam os pontos de extensão e configuração que podem ser implementadas de acordo com a necessidade de cada aplicação. Os componentes *hot-spots* são: os adaptadores de conteúdo, a política de caching, a definição das regras de adaptação e a estratégia de ordenação de entrega das mensagens. A maneira como esses componentes devem ser desenvolvidos e acoplados ao ProxyFramework é explicada na Seção 5.2.

O framework, na versão atual, ainda não provê suporte à definição de prioridades de escalonamento das mensagens para a adaptação (discutido na Seção 4.2.4).

Implementação do ProxyFramework

De modo a garantir um maior desacoplamento entre os diversos componentes do framework, bem como permitir concorrência entre suas partes, sua implementação foi subdivida em elementos chamados `receivers`, que representam cada um dos componentes de gerência discutidos anteriormente. Esses elementos são integrados a partir de um elemento central, chamado `Dispatcher`, seguindo o padrão de projeto comportamental *Mediator* [122]. Toda comunicação entre os *receivers* é intermediada pelo `Dispatcher`, evitando que cada `receiver` precise ter uma referência para os outros elementos com os quais queira se comunicar. O modelo UML do projeto do ProxyFramework, com dia-

gramas de classes e seqüência, está detalhado no apêndice B.

Cada componente do framework possui uma fila própria de mensagens a serem processadas. Existe um método para adição de mensagens, que são processadas em ordem FCFS. Um *pool* de *threads* é utilizado para a retirada e o processamento das mensagens em cada fila. No ProxyFramework é possível definir a quantidade de *threads* utilizada por cada componente. Em geral, reserva-se um maior número de *threads* para execução das adaptações.

A Figura 5.3 [2] apresenta o fluxo lógico de uma mensagem dentro do ProxyFramework, após o recebimento da mensagem do produtor até, o envio para os clientes.

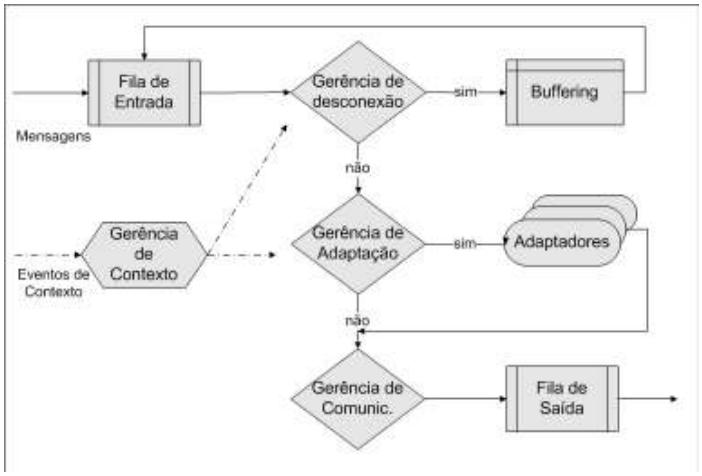

Figura 5.3: Fluxo lógico das mensagens no ProxyFramework

Toda mensagem, ao chegar ao proxy, é primeiramente inserida em uma fila de entrada, da qual é retirada pelo Gerente de Desconexão, que verifica a condição dos clientes e determina, para cada cliente, se a mensagem deve ser armazenada, ou se pode ser encaminhada aos clientes. No estágio seguinte, a mensagem é encaminhada à Gerência de Regras que define quais adaptações devem ser aplicadas à mensagem, para cada cliente. A mensagem é repassada ao Gerente de Adaptações, que executa os adaptadores definidos pela regra, utilizando o algoritmo *Context-Aware Client Grouping Algorithm* com adaptações parametrizadas, descrito na Seção 4.3.4. Depois que todas as adaptações tiverem sido aplicadas, as mensagens são encaminhadas ao Gerente de Comunicação, que determina a ordem de entrega da mensagem e a coloca na fila de saída (FIFO), de onde será repassada ao middleware de comunicação.

Quando o Gerente de Desconexão verifica a necessidade de armazenamento temporário (*buffering*), as mensagens sem alterações são guardadas de acordo com a política de "buferização"definida/implementada pelo desenvol-

[2] A Gerência de Regras foi omitida para simplificar a figura.

vedor da aplicação. Alguns exemplos de políticas podem ser o armazenamento FIFO com número limitado de mensagens, com descarte por tempo de expiração, ou tamanho da mensagem. Quando o contexto do cliente muda (p.ex, foi detectado que ele se reconectou), as mensagens armazenadas retornam para a fila de entrada, como se tivessem acabado de chegar ao proxy. As mensagens não são encaminhadas diretamente à Gerencia de Regras, porque existe a possibilidade de, enquanto essas mensagens estiverem sendo processadas, o contexto do cliente se alterar novamente para "desconectado", fazendo com que seja novamente necessário o armazenamento das mensagens.

A decisão de verificar primeiro a necessidade de armazenamento antes da necessidade de adaptação se baseia no fato de que as adaptações, que em geral são de intensivo processamento, devem ser executadas sempre de acordo com o contexto atual do cliente, ou seja, apenas imediatamente antes à sua transmissão ao cliente. De outra maneira, tais adaptações poderiam não ser eficazes ou até inúteis, principalmente se o contexto analisado envolver informações dinâmicas, por exemplo, qualidade do enlace sem fio. Caso se detecte a desconexão de um cliente após o início da adaptação, ele deve ser removido do grupo de clientes (para o qual a mensagem adaptada será enviada) e a mensagem com o conteúdo original de ser armazenada. Pelos motivos explicados acima, quando o cliente se reconecta a mensagem volta para a fila de entrada.

5.2 Passos para utilização do ProxyFramework

Para criação de um proxy de aplicação através da instanciação do ProxyFramework, deve-se seguir alguns passos principais[3]: primeiro, implementar as ações de adaptação de acordo com as necessidades específicas da aplicação; segundo, deve-se definir as regras de adaptação que determinam quando (i.e em que condição de contexto) as adaptações devem ser aplicadas; por último, determinar a estratégia de ordenação da entrega das mensagens aos clientes.

5.2.1 Criação das adaptações

O ProxyFramework permite que determinadas ações sejam executadas de acordo com o estado corrente do cliente. Por serem específicas para cada aplicação, as ações devem ser implementadas pelo desenvolvedor do proxy, que deve criar seus próprios mecanismos de adaptação.

[3]Para maiores detalhes ver apêndice C

As ações são definidas pela classe base `Action`. Esta classe fornece métodos comuns, como por exemplo, o de recuperação de parâmetros. Essencialmente, existem dois tipos de ações: *adapters*, que modificam mensagens, e *listeners*, que modificam algum estado interno do proxy relativo a um cliente. Essas ações são implementadas por *adaptadores de conteúdo*, por ex. *transcoding*; e *adaptadores de funcionalidade*, respectivamente. A principal atividade do proxy é, em geral, adaptação de conteúdo, ou seja, as adaptações são efetuadas sobre o conteúdo das mensagens e a avaliação do contexto e verificação da necessidade, ou não, de adaptação ocorre no momento do envio da mensagem ao cliente. O outro tipo de adaptação diz respeito à alteração no funcionamento da aplicação ou do proxy, no qual funções podem ser acionadas/desativadas devido à ocorrência de um determinado contexto. Nesse caso, a adaptação é ativada no momento da ocorrência da mudança de contexto do cliente, como por exemplo, em uma desconexão, poder-se-ia ativar a bufferização das mensagens no proxy, a fim de postergar o envio das mensagens até a reconexão do cliente.

As ações do tipo *adapter* são acionadas no momento do envio de uma mensagem a um cliente, dependendo de seu contexto corrente. Para criar uma ação deste tipo, deve-se estender a classe abstrata `Adapter`, implementando o método `execute`. Esse método recebe informações sobre os destinatários da mensagem e a própria mensagem a ser adaptada, e retorna a mensagem adaptada ou uma referência nula. Note que se o retorno de um adaptador for nulo o fluxo de adaptações é interrompido, já que isto significa que a mensagem foi descartada.

```
public abstract class Adapter extends Action {

   public abstract Message execute(
                       ArrayList<ClientInfo> clientInfo,
                       Message msg)
      throws AdaptationException;

   public ArrayList<String> getUsedContextAttributes();

}
```

Além disso, existe a possibilidade (e necessidade) de criar adaptadores parametrizados por contexto, para que seja possível adequar a adaptação às características do dispositivo e seu ambiente de execução. Adaptadores parametrizados por contexto utilizam o valor corrente dos atributos de contexto

dos clientes para ajustar a adaptação. O contexto corrente de cada cliente pode ser obtido através do método `getCurrentContext` da classe `ClientInfo`. Além disso, o método `getUsedContextAttributes` deve indicar quais os atributos de contexto são utilizados pelo adaptador parametrizado.Para adaptadores não parametrizados, o método deve retornar `null`;

Apesar de bibliotecas de adaptadores não fazerem parte do ProxyFramework, ele fornece um conjunto de adaptadores de imagem, descritos no Apêndice C.2. Estes adaptadores servem apenas como exemplos de adaptadores de conteúdo, tanto parametrizados como não parametrizados, para serem reutilizados pelos usuários do framework (desenvolvedores).

As ações do tipo *listeners* reagem às mudanças no estado de contexto de clientes, podendo alterar alguma funcionalidade do proxy, ou interagir com o cliente. Para implementar um *listener*, basta estender a classe abstrata `StateListener`, que possui dois métodos abstratos: `matches` e `unmatches`. O primeiro é executado sempre que o estado de contexto (descrito por uma expressão) muda de inativo (OFF) para ativo (ON), e o segundo é executado quando o estado de contexto muda de ativo(ON) para inativo (OFF). Em ambos os casos, são fornecidas as informações do cliente que sofreu a mudança de estado, bem como informações que permitem ao adaptador encaminhar mensagens para outros componentes do proxy.

```
public abstract class StateListener extends Action {
   public abstract void matches(ClientInfo info,
                                Dispatcher disp)
       throws StateListenerException;

   public abstract void unmatches(ClientInfo info,
                                  Dispatcher disp)
     throws StateListenerException;
}
```

5.2.2 Definição de regras de adaptação

As regras de adaptação, que associam as situações de contexto com as adaptações necessárias, podem ser definidas de duas formas principais: através de interfaces programáveis ou através de um arquivo de configuração (como discutido na Seção 3.1.1).

Segundo [27], as regras configuradas apresentam um nível mais alto de abstração para as adaptações do que a implementação em uma linguagem de

programação, sendo mais fácil de serem definidas, além de não demandarem maiores conhecimentos sobre detalhes intrínsecos da lógica interna do sistema. Outra vantagem é que a configuração do proxy pode ser alterada pelo administrador do sistema, e não somente pelo desenvolvedor do proxy. Além disso, não há necessidade de recompilação ou redistribuição de código no sistema, o que é necessário nos casos em que as regras de adaptação são programadas (*hard coded*). Essas características facilitam a manutenção de uma aplicação, que ao longo do tempo precisa se adequar à evolução dos dispositivos, das tecnologias de redes sem fio, e a novos requisitos dos usuários.

Entretanto, através da programação de regras via código pode-se obter uma maior expressividade e flexibilidade, permitindo-se implementar praticamente qualquer tipo de reação (adaptação) às mudanças de contexto.

Uma terceira opção é utilização de uma linguagem de *script* para definição do arquivo de configuração das regras de adaptação. Essa opção representa a junção das outras duas alternativas, pois esse tipo de linguagem permite definições declarativas (como as regras) e procedurais (como na programação). A escolha da maneira de definição das regras de adaptação requer uma solução de compromisso entre simplicidade e flexibilidade.

No ProxyFramework, adotou-se a abordagem de configuração de regras, devido seu uso mais simples e por ela ser suficiente para descrição das regras para o tipo de adaptação suportado (adaptação de conteúdo).

É importante ressaltar que a escolha dos adaptadores de conteúdo em geral não depende apenas do contexto corrente do cliente, mas também de algumas condições relativas à mensagem, como exemplo, o tipo de conteúdo. Por exemplo, para um contexto de baixa largura de banda, um proxy pode ser configurado para comprimir o dado em transmissão. Quando o conteúdo consiste de um texto, pode ser selecionado um adaptador que implemente alguma forma de compressão no texto, enquanto que no caso de um vídeo, pode ser selecionado um adaptador que diminua a qualidade dos quadros enviados. Neste exemplo fica evidente que, embora a tarefa de compressão de dados tenha o mesmo propósito, o método de compressão utilizado para texto não será o mesmo para vídeo e, portanto, um adaptador diferente deve ser selecionado. É comum proxies utilizarem formatos MIME para definição do tipo de conteúdo da mensagem, como em [18].

Pelo motivo supracitado, o framework permite a formulação de regras, para seleção de adaptadores, que levam em consideração não apenas o contexto corrente, mas também um conjunto de atributos, como o tipo do conteúdo, da comunicação, destinatário, etc.

Configuração das Regras

O ProxyFramework utiliza o paradigma de políticas de adaptação baseada em regras para determinar quais ações (adaptações) são necessárias de acordo com o contexto do cliente, que inclui o estado do dispositivo e a qualidade do enlace sem fio. A configuração das regras de adaptação deve ser feita manualmente pelo administrador do sistema, através de um arquivo XML. Por questões de simplicidade, optou-se por utilizar um esquema XML proprietário, ao invés de utilizar um esquema de definição de regras, como RuleML, mais completo e complexo, mas acima das necessidades do nosso sistema.

No arquivo de configuração, cada estado é definido por uma *expressão de contexto*. A cada estado pode estar associada uma ação do tipo *listener*, e um número não determinado de regras para adaptação de mensagens. Cada regra é composta por um filtro e adaptadores. O filtro serve pra validar as mensagens que devem ser adaptadas por determinado adaptador. Além disso, cada regra possui uma prioridade que determina qual a ordem em que ela deve ser aplicada. Se em uma regra não for definida uma prioridade, ela será considerada como de menor prioridade. Se diferentes regras tiverem a mesma prioridade, elas serão aplicadas na mesma ordem em que foram definidas no arquivo XML.

Na Figura 5.4 é apresentado um exemplo de um arquivo de configuração. Nesse exemplo, são definidas três expressões de contexto: uma para desconexão e duas para adaptação de conteúdo. As expressões de contexto são expressões lógicas que combinam atributos de contexto, tanto estáticos (como tamanho de tela do dispositivo), quanto dinâmicos (como energia disponível). Nesse exemplo, define-se que para uma desconexão superior a dois segundos, as mensagens serão armazenadas em uma fila temporária, seguindo a estratégia FIFO. O arquivo também indica que mensagens, cujo conteúdo seja uma imagem, deverão sofrer adaptações nas seguintes situações: *i*) caso o dispositivo móvel esteja com pouca memória e CPU disponíveis (contexto dinâmico), a imagem deve ser reduzida em 50%; *ii*) caso seja um PDA (contexto estático), utiliza-se um adaptador parametrizado que recorta a imagem para o tamanhdo das dimensões de tela do PDA.

```
<ProxyConf>
  <State>
    <Expression>
      <![CDATA[ OnLine = false AND DeltaT > 2000 ]]>
    </Expression>
    <Action class="proxy.listeners.DefaultCacheListener">
      <Parameter name="cacheClassName"> FIFOCacher </Parameter>
    </Action>
  </State>
  <State>
    <Expression>
      <![CDATA[ CPU > 80 AND FreeMemory < 10000 ]]>
    </Expression>
    <Rule priority="1">
      <Filter>
        <!– message data type –>
        <StartWith>
          <FieldValue> <Literal>datatype</Literal> </FieldValue>
          <Literal>image/</Literal>
        </StartWith>
      </Filter>
      <Action class="proxy.adapters.ScaleImageAdapter">
        <Parameter name="factor">0.5</Parameter>
      </Action>
    </Rule>
  </State>
  <State>
    <Expression>
      <![CDATA[ DeviceProfile.HardwarePlatform.DeviceType LIKE 'PDA' ]]>
    </Expression>
    <Rule priority="1">
      <Filter>
        <!– message data type –>
        <StartWith>
          <FieldValue> <Literal>datatype</Literal> </FieldValue>
          <Literal>image/</Literal>
        </StartWith>
      </Filter>
      <!– Recortar pelo tamanho de tela do PDA –>
      <Action class="proxy.adapters.CropCenterParamAdapter">
      </Action>
    </Rule>
  </State>
</ProxyConf>
```

Figura 5.4: Exemplo de Arquivo de Configuração do Proxy.

No arquivo de configuração, o elemento `State` representa um estado de contexto a ser monitorado. Todo elemento `State` possui um único elemento `Expression`, que corresponde à expressão de contexto que será registrada no serviço de contexto CIS da MoCA (Seção 2.4) para monitoramento periódico e entrega de notificações, no momento em que a expressão se torna verdadeira (estado ativo), e no momento seguinte quando passar a ser falsa. Quando ocorre uma mudança em qualquer direção, uma ação do tipo *listener* será executada. Sua configuração é feita através do elemento `Action`, no qual é possível indicar a classe que implementa a ação desejada, como por exemplo, buffering com política FIFO (como mostrado no exemplo de configuração).

Cada estado pode ter inúmeros elementos do tipo `Rule`, que agregam adaptadores de conteúdo que serão executados no momento do envio da mensagem. Estas regras possuem um atributo que lhes confere um grau de prioridade e estabelece uma ordem de execução (entre regras) - da mais alta para a mais baixa prioridade. A prioridade é um número inteiro não negativo para a qual 0 significa a menor prioridade. Caso duas regras possuam a mesma prioridade, o critério de desempate será a ordem de adição da regra no conjunto de regras do estado.

Os adaptadores de uma regra são executados caso o estado para o qual foram cadastrados esteja ativo (ON) e a condição relacionada à mensagem (tipo) e/ou do seu destinatário seja satisfeita. A condição é determinada através do elemento `Filter`, que pode ser configurado utilizando-se de uma série de operadores lógicos (AND, OR, NOT) e de seletores disponíveis (datatype, protocol, client, subject).

Outros operadores, que podem ser utilizados na composição de filtros, são apresentados abaixo:

- **Equals** - Operador binário que recebe duas strings como operandos. Assume valor verdadeiro se os operandos são iguais e falso caso contrário. Obs: Esta operação não leva em conta letras em caixa alta ou baixa (*case insensitive*).
- **StartWith** - Operador binário que recebe duas strings como operandos. Verifica se o segundo operando é prefixo do primeiro operando. Obs: Esta operação é *case insensitive*.
- **Literal**- usado para definir uma string literal, ou seja, um valor fixo.
- **FieldValue** - permite referenciar uma string cujo valor depende da mensagem sendo avaliada pelo filtro. FieldValue recebe um Literal como argumento para indicar o nome do campo a ser recuperado.

Uma vez que o filtro tenha aceito uma mensagem, a seqüência de adaptadores cadastrados para a regra de adaptação será executada. Adaptadores de conteúdo, i.e, ações do tipo `adapter`, são também configurados na regra através do elemento `Action`. É possível fornecer parâmetros tanto para os *listeners* quanto para os *adapters*, utilizando-se o elemento `Parameter`, que é definido por um nome e um valor, como mostrado no exemplo (Figura 5.4). Internamente a um *listener* ou a um *adapter*, pode-se obter o valor de um parâmetro através do método:

```
protected final Object getParameter(String paramName)
```

Este método é herdado da classe `Action` da qual tanto `StateListener` quando `Adapter` descendem. É importante ressaltar que na denominação dos parâmetros, as diferenças entre letras maiúsculas e minúsculas são levadas em consideração (*case sensitive*).

5.2.3 Definição da estratégia de armazenamento

O armazenamento de mensagens de clientes exige um certo cuidado. Guardar uma excessiva quantidade de mensagens pode comprometer o funcionamento do ProxyFramework e até mesmo derrubar o sistema em casos de completa exaustão de recursos. Deste modo, uma política de armazenamento adequada precisa ser definida para poder garantir que o serviço seja provido sem grande risco para o sistema. Por outro lado, a escolha de qual será a política adequada pode depender das características das aplicações cujas mensagens serão intermediadas pelo proxy.

Deste modo, o ProxyFramework provê uma flexibilidade para seus usuários, permitindo que eles escolham qual será a estratégia de armazenamento de mensagens. Um exemplo de estratégia pode ser FIFO com tempo de expiração. Esta flexibilidade é provida pela interface `Cacher`, que estabelece os métodos que devem ser desenvolvidos para todas as classes que implementem políticas de armazenamento a serem utilizadas pelo ProxyFramework, e representa a forma como as mensagens serão armazenadas.

O ProxyFramework provê duas políticas padrão. Na primeira, as mensagens são simplesmente descartadas, e na outra implementa-se uma fila FIFO, com descarte de mensagens mais antigas após um certo número de mensagens. Mas, seguindo a interface `Cacher` mostrada abaixo, pode-se implementar outras políticas mais adequadas à aplicação, como por exemplo, usar o tamanho da mensagem ou sua prioridade como critério de descarte.

```
public interface Cacher {

   // Adiciona um pacote (mensagem) durante a desconexão
   void addPackage(Package pack);

   // Retorna todos os pacotes armazenados (na reconexão)
   Package[] getPackages();
}
```

Qual implementação de `Cacher` será utilizada dentro do framework deve ser informada através do arquivo de configuração do proxy (descrito na Seção 5.2.2), através do parâmetro `cacheClassName`.

5.2.4 Definição da estratégia de ordenação

O último ponto de extensão (*hotspot*) do ProxyFramework é a estratégia de ordenação de entrega de mensagens. Sem ordenação não é garantido que as mensagens sejam repassadas aos clientes na mesma ordem em que foram recebidas pelo proxy, devido principalmente a possíveis diferenças no tempo de adaptação das mesmas. Entretanto, a ordenação segundo algum critério pode ser necessária ou não, dependendo das características da aplicação, e por isso deve ser definida pelo desenvolvedor.

O desenvolvedor pode escolher entre as duas estratégias padrão oferecidas pelo ProxyFramework, ou implementar outra estratégia. As estratégias oferecidas são: sem ordenação (`NoOrderStrategy`), e ordenação por cliente (`ClientBasedDeliveryOrderStrategy`). Na segunda estratégia, a ordem de entrega de mensagens é garantida para cada cliente, mas não é uniforme entre clientes. Além disso, para comunicação Pub/Sub a ordenação além de ser por cliente é também por tópico. Assim mensagens de um tópico, não interferem na ordem de entrega de mensagens de outro tópico. Isto se faz necessário para que uma mensagem de um certo tópico que ainda esteja em processamento, por exemplo, devido a uma adaptação muito demorada, não bloqueie uma mensagem de outro tópico que esteja pronta para o envio.

Mas, como mencionado, o desenvolvedor pode definir qualquer outra estratégia, sendo necessário apenas estender a classe `DeliveryOrderMngr`, e implementar os métodos abstratos estabelecidos em sua interface (vide apêndice C.3). A utilização de uma estratégia de ordenação na entrega de mensagens impacta em diversos módulos do proxy, como na gerência de adaptação e na gerência de desconexão, devido a quebra de grupos de clientes

e replicação de mensagens. Por isso, sua implementação é mais complexa que a dos demais pontos de extensão do proxy. No apêndice C.3 pode se obter uma descrição mais detalhada de como os métodos da interface são utilizados no framework e como devem ser implementados.

Esse é o único *hotspot* não configurável pelo arquivo XML, tendo que ser definido durante a implementação do proxy (instanciação do framework), através do método `setDeliveryOrderStrategy` da classe `ProxyFramework`.

5.3 Resumo e Conclusões

Neste capítulo foi apresentado um framework para desenvolvimento de proxies para adaptação de conteúdo para aplicações móveis. Esse framework segue a proposta da arquitetura discutida na Seção 4.2, sendo uma ferramenta para facilitar a implementação da camada de serviços de adaptação.

O ProxyFramework utiliza serviços da MoCA para a obtenção de contextos dos clientes, bem como para comunicação publish/subscribe, através do ECI. Além disso, fornece mecanismos simples para incorporação de adaptadores de conteúdo e de políticas de armazenamento temporário (*buffering*), bem como a definição das regras de adaptação através de uma arquivo de configuração XML. Este arquivo permite que as regras sejam alteradas conforme a necessidade e evolução da aplicação ou de novos módulos adaptadores. Um diferencial é que o proxy pode ser utilizado para aplicações que utilizem tanto uma comunicação tipo *pull* (requisição/resposta) quanto *push* (do tipo publish/subscribe).

As primeiras experiências de utilização do ProxyFramework foram realizadas por alunos da disciplina Introdução à Computação Móvel, e obtiveram bons resultados[4]. Eles usaram o framework para desenvolver aplicações móveis de difusão de notícias com adaptação de imagens e de textos, ou seja mensagens discretas. O ProxyFramework, a princípio, não foi projetado para aplicações com fluxos de dados (*streams*), cuja adaptação pode envolver considerações sobre características particulares deste tipo de dado, que não foram objeto de estudo desta tese.

[4]Maiores detalhes na Seção 6.2