4
Pan: Uma linguagem de domínio específico (DSL) para VRT

Uma linguagem de domínio específico (DSL – domain-specific language) em MDSD permite tornar os aspectos relevantes de um domínio modeláveis, ou seja, expressos formalmente. Os construtos da linguagem são concebidos com foco sobre o formalismo do meta modelo e na aproximação das expressões aos conceitos facilmente interpretáveis por especialistas do domínio.

Em MDRM, devido às idiossincrasias mencionadas no Capítulo 2, outros requisitos vêm à tona para a especificação de sua DSL. O primeiro deles é a capacidade de representação de um recurso em diversos níveis de abstração, preenchido, em parte, pelo meta modelo VRT, por meio da definição de VRTs primitivas e VRTs compostas. A linguagem deve ser capaz de representar ambos os tipos de VRTs uniformemente, e prover os mecanismos necessários para as noções de composições, hierarquias e dependências, peculiares ao meta modelo.

Outro requisito diz respeito ao uso da DSL não somente por um tipo de especialista do domínio, mas por diferentes atores, que vão desde o desenvolvedor de aplicações até o administrador de um sistema. Certamente, os maiores beneficiários da DSL de MDRM são os projetistas de serviços de telecomunicações (ou atores que desempenham papel semelhante em outros cenários), uma vez que são responsáveis pela construção do modelo de gerenciamento de recursos inicialmente implantado nas plataformas participantes.

No entanto, a mesma DSL deve incluir as operações de manutenção de VRTs, o que constitui um terceiro requisito diferenciado: a inclusão de quaisquer atores (autorizados) interessados na configuração e evolução do subsistema de gerenciamento de recursos. A partir deste ponto, o termo ‘atores’ pode incluir também os componentes da infra-estrutura de provisão de QoS que, após negociações ou sintonizações de QoS, precisam estabelecer ou modificar reservas de recursos fazendo uso do meta modelo VRT.

De fato, desde o advento das redes ativas e programáveis (Bush, 2001), há uma tendência em tornar indistinguíveis os papéis dos atores no estabelecimento

de serviços de comunicação. Um subsistema de gerenciamento de recursos colaborativo se alinha a essa abordagem. A DSL do meta modelo VRT, denominada *Pan*[4], possui papel fundamental na construção de tal ambiente.

Pan é uma linguagem de aplicação XML (eXtensible Markup Language) (Bray, 2006). A adoção de tecnologias em torno de XML trouxe vários benefícios com relação aos requisitos impostos pelo meta modelo VRT. Primeiro, XML é tipicamente uma base para linguagens declarativas, tipo que melhor se adéqua à especificação de modelos de gerenciamento de recursos que contam com infra-estrutura de suporte nas plataformas. Segundo, XML é, por definição, uma linguagem hierárquica, o que vem facilitar a formação dos construtos relacionados a VRTs e florestas. Terceiro, XML possui diversas técnicas associadas para a formalização da estrutura de documentos, trazendo consigo ferramentas de validação e análise necessárias no processo de modelagem descrito por MDRM. Quarto, documentos XML podem ser transformados sucessivamente em outros documentos XML, ou ainda, em outros arquivos de diferentes linguagens, por meio de técnicas de transformação que serão fundamentais para o processo de instanciação descrito por MDRM. Quinto, documentos XML podem ser facilmente mantidos em repositórios para referências futuras, o que pode beneficiar as ações de manutenção de modelos definidas em MDRM.

A especificação da linguagem Pan utiliza as seguintes tecnologias XML: *XML Schema* (Thompson, 2004), para a validação sintática; *ISO Schematron* (ISO/IEC, 2006), para complementação da validação sintática e validação semântica; e XSLT (Extensible Stylesheet Language Transformations) (Kay, 2007a), para transformações modelo-para-modelo e modelo-para-código. Este capítulo é dedicado à descrição da linguagem Pan e aos aspectos de validação sintática e semântica de instâncias de modelos descritos em Pan.

4.1. Introdução à linguagem Pan

A especificação da linguagem Pan leva em conta não somente a representação formal das abstrações inerentes ao meta modelo VRT, mas também

[4] Nome inspirado no deus grego protetor das florestas. Coincidentemente, 'pan' é também um prefixo na língua portuguesa que significa 'todos', o que remete à preocupação com o suporte a múltiplos atores.

o oferecimento de facilidades de modelagem que incluem, sempre que possível, mecanismos de reuso e de extensão da linguagem. Esses mecanismos ficam evidentes mais adiante neste capítulo.

Um documento Pan é formado por uma seqüência de elementos XML em conformidade com a estruturação definida pelo *schema* da linguagem (listagem completa encontra-se no Anexo I), cujo *namespace* é:

```
http://mdrm.telemidia.puc-rio.br/specs/xml/Pan
```

O elemento raiz de todo documento Pan é `<panvrt>` e deve incluir as diversas declarações de *namespaces* utilizados. Dois *namespaces* devem obrigatoriamente ser incluídos: o *namespace* de Pan, mencionado acima, e o de XML *Schema Instance* (XSI), que disponibiliza atributos para a descrição de localização física de *schemas* e de especialização de elementos da linguagem, entre outros. Pan recorre ao artifício de definir alguns de seus elementos como abstratos para estabelecer pontos em que a linguagem deve ser estendida visando atender às características variáveis do domínio, como em parâmetros e categorias de serviços, regras de classificação e regras de controle de acesso. Em alguns desses pontos, um dos atributos de XSI será de fundamental importância.

De uma forma geral, documentos Pan são divididos em duas partes. A primeira parte, comum para todos os tipos de documentos, permite a atribuição de um título (elemento `<title>`) e de uma descrição (elemento `<description>`) ao documento. O conteúdo desses elementos é de livre escolha e de caráter puramente informativo. A parte inicial do documento possui, ainda, uma seqüência indeterminada de elementos para a importação de modelos já implantados (elementos `<import>`), de modo a permitir que alguns dos elementos especificados nos modelos importados sejam referenciáveis no documento corrente, visando reuso e integração.

A segunda parte de um documento Pan tem conteúdo dependente do tipo de função que deseja-se descrever, determinando, assim, o tipo do documento. São quatro os possíveis tipos de documentos, listados a seguir de forma introdutória:

1. Documento de especificação de um modelo de gerenciamento de recursos, ou, simplesmente, *modelo* (elemento `<model>`): descreve as estruturas de uma ou mais florestas (elementos `<forest>`) e suas

respectivas árvores de recursos virtuais (elementos `<rootvr>`), a serem implantadas inicialmente nas plataformas envolvidas;

2. Documento de manutenção de um modelo de gerenciamento de recursos já implantado, ou, simplesmente, *manutenção* (elemento `<maintenance>`): descreve uma seqüência de operações que visam modificar um modelo atualmente implantado, em uso. As operações são acondicionadas em transações (elementos `<transaction>`), para possibilitar a especificação de dependências entre operações;
3. Documento de resposta a uma solicitação de manutenção de modelo, ou, simplesmente, *resposta a manutenção* (elemento `<maintenance-response>`): descreve os valores de retorno de cada operação especificada e o resultado final de cada transação (elemento `<transaction-response>`). Normalmente, esse tipo de documento Pan é gerado automaticamente pela implementação de suporte a VRT em cada plataforma envolvida, como resposta às ações de manutenção previamente submetidas;
4. Documento de relatório de compatibilidade entre elementos, ou, simplesmente, relatório de compatibilidade (elemento `<compatibility-report>`): descreve a compatibilidade entre elementos distintos, porém de alguma forma interdependentes. Por exemplo, uma estratégia de admissão somente pode ser associada a um recurso virtual se for compatível com os parâmetros reunidos na categoria de serviço oferecida por ele (conforme mencionado na Seção 3.2), caso contrário uma solicitação de QoS não poderá ser corretamente avaliada quanto à sua viabilidade. Nesse exemplo, o relatório de compatibilidade gerado para a estratégia de admissão lista quais categorias de serviços são compatíveis com ela (`<sccompatibility>`).

A descrição completa de cada tipo de documento e seus elementos pertinentes encontra-se na próxima seção. Resumindo esta pequena introdução à linguagem Pan, o Quadro 2 ilustra a disposição geral dos quatro tipos de documentos Pan, ilustrando apenas seus elementos principais.

```
<?xml version="1.0" encoding="UTF-8"?>

<panvrt xmlns="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan
                     http://mdrm.telemidia.puc-rio.br/specs/xml/Pan/panvrt.xsd">

 <title>Pan: Hello World!</title>
 <description>General structure of Pan documents</description>
 <import alias="imported1"
         src="http://mdrm.telemidia.puc-rio.br/repository/instance1.xml"/>
 <import…/>

<!—
    Tipo 1          Tipo 2              Tipo 3                  Tipo 4      -->

 <model>       | <maintenance>    | <maintenace-response>    | <compatibility-report>
  <forest…>    |  <transaction…>  |  <transaction-response>  |  <sccompatibility…/>
   <rootvr…>   |   <operation…>   |   <operation-response…/> |  <sccompatibility…/>
   <rootvr…>   |   <operation…>   |   <operation-response…/> |  <sccompatibility…/>
  </forest>    |  </transaction>  |  </transaction-response> |        ...
  <forest…>    |  <transaction…>  |  <transaction-response…> | </compatibility-
 </model>      | </maintenance>   | </maintenace-response>   |   report>

</panvrt>
```

Quadro 2 – Disposição geral dos diferentes tipos de documentos Pan

4.2. Descrição da Linguagem Pan

A descrição da linguagem Pan nesta seção se baseia no modelo XML de cada elemento componente da linguagem. A descrição do modelo XML de um elemento define quais são seus atributos, tipos, elementos filhos e a obrigatoriedade de uso de cada um desses construtos. Para descrever essa gramática, a seguinte notação foi adotada:

- A vírgula (‘,’) define a ordem de utilização dos elementos filhos do elemento descrito. Por exemplo, a expressão `title,description` determina que o elemento `<title>` precede o elemento `<description>`
- A barra vertical (‘|’) representa disjunção entre os elementos filhos do elemento descrito. Ou seja, entre os elementos separados pela barra vertical, apenas um pode ser utilizado. Por exemplo, a expressão `model|maintenance` define que esses dois elementos não podem ocorrer simultaneamente.
- O sinal de interrogação (‘?’) indica que o elemento imediatamente anterior é opcional (cardinalidade 0 ou 1). Por exemplo, a expressão

`title?` determina que esse elemento pode ocorrer uma vez ou simplesmente não ocorrer.

- O asterisco (‘*’) indica que o elemento imediatamente anterior é opcional e, se ocorrer, pode ser repetido por uma ou mais vezes (cardinalidade 0 a ∞). Por exemplo, a expressão `import*` determina que esse elemento pode ocorrer zero ou mais vezes.
- O sinal positivo (‘+’) indica que o elemento imediatamente anterior é obrigatório e pode ser utilizado mais de uma vez (cardinalidade 1 a ∞). Por exemplo, a expressão `forest+` determina que o elemento deve ocorrer pelo menos uma, ou mais vezes.
- Dois pontos (‘:’) separam elementos que podem aparecer em qualquer ordem. Por exemplo, a expressão `admstrategy:schedstrategy` define que esses elementos não possuem uma ordem de ocorrência obrigatória.
- Parênteses (‘()’) são utilizados quando uma das expressões acima deve ser aplicada ao resultado de uma outra expressão, caracterizando, assim, uma expressão composta. Por exemplo, a expressão `(forest|rootvr)*` determina que a disjunção entre os elementos mencionados (‘|’) é opcional, podendo ocorrer várias vezes (‘*’).
- Por fim, os tipos dos atributos de cada elemento são baseados nos tipos disponibilizados por *XML Schema*, cujo *namespace* será identificado pelo prefixo ‘`xs:`’. Atributos em negrito são obrigatórios. A disjunção de atributos não é representável por meio do modelo XML (nem pelo *schema* XML de Pan), e por isso será descrita textualmente.

Complementando as informações de cada elemento, são discutidas outras restrições sintáticas ou semânticas que não podem ser expressas por meio do modelo XML. Algumas dessas restrições adicionais e todas as restrições descritas pelo modelo XML são possíveis de serem impostas por meio do *schema* XML de Pan, listado no Anexo I. No entanto, outras restrições são necessárias para uma completa validação de documentos Pan, impossíveis de serem descritas pelo *schema* XML da linguagem, e por isso são implementadas por um *schema* complementar em linguagem *ISO Schematron*, listado no Anexo II. Quando esse for o caso, a regra *Schematron* correspondente a uma

restrição mencionada no texto será citada entre parênteses, no formato (regra: `id-do-pattern`), para facilitar a referência.

A descrição inicia-se pelo elemento `<panvrt>`, que deve obrigatoriamente ser a raiz de todo documento Pan (regra: `docroot-must-be-panvrt`). Normalmente, o elemento raiz de um documento XML contém atributos para a definição dos *namespaces* disponíveis para uso. Conforme mencionado anteriormente, dois *namespaces* são obrigatórios (regra: `required-namespaces`) em Pan:

- `http://mdrm.telemidia.puc-rio.br/specs/xml/Pan`
- `http://www.w3.org/2001/XMLSchema-instance`

O primeiro *namespace* citado é o da linguagem Pan e não precisa ser especificado como *namespace default* do documento. O segundo *namespace* disponibiliza os atributos XSI necessários para a localização física de *schemas* XML e para a especialização de elementos abstratos que são explorados por Pan para extensibilidade e uniformização.

O projetista que constrói modelos Pan em ambientes de desenvolvimento XML pode declarar no elemento `<panvrt>` a seguinte localização de acesso público do *schema* XML da linguagem:

- `http://mdrm.telemidia.puc-rio.br/specs/xml/Pan/panvrt.xsd`

O *schema* complementar de Pan em *Schematron* também está disponível publicamente e pode ser usado nesses ambientes de desenvolvimento, se suportado tal tipo de validação, por meio de alguma diretiva de configuração. A localização pública do *schema* complementar é:

- `http://mdrm.telemidia.puc-rio.br/specs/xml/PanSchematron/panvrt.sch`

Assim, todo documento Pan pode ser iniciado da seguinte forma:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Primitiva de configuracao Schematron própria do ambiente OxygenXML -->
<?oxygen SCHSchema="
         http://mdrm.telemidia.puc-rio.br/specs/xml/PanSchematron/panvrt.sch"?>

<panvrt xmlns="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan
                       http://mdrm.telemidia.puc-rio.br/specs/xml/Pan/panvrt.xsd”>

...
```

O elemento <panvrt> pode incluir um atributo id para possível identificação entre documentos em repositórios, por exemplo. Porém, seu uso é opcional e, caso ocorra, deve ser um identificador único em todo o documento.

```
<panvrt
   id = xs:ID
>

   title?, description?, import*,
   (model | maintenance | maintenance-response |
    compatibility-report)

</panvrt>
```

O elemento <title>, filho de <panvrt>, é opcional e deve ser usado para atribuir um título que sintetiza a função e finalidade do documento, por meio de uma cadeia de caracteres de formato livre.

```
<title>
   character data
</title>
```

Uma descrição textual mais detalhada da função e finalidade do documento pode, opcionalmente, ser feita por meio do elemento <description>, também filho de <panvrt>. Seu conteúdo descritivo é uma cadeia de caracteres de formato livre.

```
<description>
   character data
</description>
```

O elemento <import>, que também tem <panvrt> como pai, especifica um modelo Pan já implantado a ser importado, ou seja, a ter seus elementos disponíveis para reuso ou referência pelo modelo ou manutenção descrito no documento corrente. O elemento <import> é opcional e pode se repetir pelo número de vezes igual ao número de documentos Pan a serem importados. Seu atributo src, obrigatório, descreve uma URI que leva ao repositório onde o documento-alvo é mantido depois de ter seu modelo implantado. O atributo alias, também obrigatório, atribui ao modelo referenciado um identificador local, único no documento corrente, que deve ser usado como prefixo na definição de uma referência a um elemento do documento importado (e.g. alias1#element1).

```
<import
   alias = xs:ID
   src   = xs:anyURI
/>
```

Na validação do documento, o modelo importado deve estar disponível para acesso (regra: `import-doc-available`), para que as referências estabelecidas no documento corrente possam ser validadas.

Os demais elementos filhos de `<panvrt>` ocorrem de forma exclusiva entre si. São eles: `<model>`, `<maintenance>`, `<maintenance-response>` e `<compatibility-report>`. Conforme já mencionado, esses elementos determinam o tipo de função descrita pelo documento e, para melhor organização, cada um deles e seus respectivos filhos estão descritos nas subseções a seguir.

4.2.1. Descrevendo modelos de gerenciamento de recursos em Pan

Um modelo de gerenciamento de recursos baseado no meta modelo VRT descreve o conjunto de florestas e árvores que determinam a estruturação inicial dos recursos de um ambiente distribuído. Como pôde ser visto na apresentação do meta modelo, essa estrutura define quais são os tipos de serviços disponibilizados, as estratégias de gerenciamento associadas a cada recurso e as reservas inicialmente estabelecidas.

Um documento de especificação de um modelo, o primeiro dos quatro tipos de documentos Pan, é caracterizado pela declaração do elemento `<model>`, logo após a parte inicial já descrita, tendo `<panvrt>` como elemento pai.

```
<model
   id = xs:ID
>

   forest+

</model>
```

O elemento `<model>` possui como atributo opcional `id`, que poderia ser usado para sua identificação entre diversos modelos. Porém, seu uso é opcional e, caso ocorra, deve ser um identificador único em todo o documento.

Declarando florestas

O único elemento filho de `<model>` é `<forest>`, que pode ocorrer uma ou mais vezes, quantas forem as florestas a serem definidas no modelo. Uma floresta

pode conter declarações das estratégias de gerenciamento a elas associadas, das suas árvores e das florestas nela contidas. Para possibilitar a inclusão de florestas já declaradas em uma floresta sendo modelada, Pan define que florestas são referenciáveis. Referências a florestas podem ser locais ou remotas. Referências locais apontam para florestas declaradas no documento corrente. Referências remotas apontam para florestas declaradas em modelos já implantados, importados por meio do elemento `<import>`.

```
<forest
   id    = xs:ID
   refer = xs:anyURI
>
   mgmtstrategies, (forest | rootvr)*
</forest>
```

O elemento `<forest>` possui como atributos `id` e `refer`, os quais são mutuamente exclusivos, mas um deles deve ocorrer (regra: `forests-must-have-id-or-refer`). O atributo `id` atribui à floresta declarada um identificador único no modelo, pelo qual poderá ser referenciada em modelos e manutenções. O atributo `refer` torna a floresta declarada localmente uma simples referência à floresta por ele apontada. O atributo `refer` é uma URI de formato específico, tendo como prefixo o identificador atribuído ao modelo importado (`alias` de `<import>`), seguido do `id` da floresta naquele modelo (e.g.: `alias1#forest1`). Referências para florestas do modelo corrente são possíveis, bastando omitir o prefixo. O atributo `refer` deve obrigatoriamente apontar para um elemento `<forest>`, já existente (regra: `refer-correctness`). A exclusão mútua entre `id` e `refer` tem como efeito colateral desejável a impossibilidade de encadeamento de referências sucessivas, que poderia se tornar de difícil gerenciamento, ou mesmo levar acidentalmente a referências circulares.

As florestas de primeiro nível em um modelo (aquelas declaradas logo como filhas de `<model>`, e não internamente a outras florestas), não podem referenciar outras florestas (regra: `forest-cannot-be-a-reference`), pois a referência a uma floresta possui semântica de inclusão da floresta referenciada na floresta especificada. O elemento `<model>` não representa uma floresta.

Por razão óbvia, os elementos filhos de `<forest>` somente podem ocorrer se a declaração não for referência a uma floresta (regra: `element-with-refer-cannot-have-child-elements`).

O primeiro elemento filho de <forest>, opcional, é <mgmtstrategies>, que abre uma seção para a configuração da lógica de gerenciamento da floresta. Essa lógica inclui a associação de uma estratégia de admissão e das regras de controle de acesso à floresta. O elemento <mgmtstrategies> não possui atributos quando no contexto de uma floresta.

```
<mgmtstrategies
   servicecategory = xs:QName[5]
>

   schedstrategy?[6] : admstrategy? : accessctrlrules?

</management>
```

Os elementos filhos de <mgmtstrategies> são opcionais e, em florestas, incluem somente a estratégia de admissão <admstrategy> e as regras de controle de acesso <accessctrlrules>. Omitir a declaração de uma estratégia de admissão significa que o controlador de admissão adotará uma estratégia padrão, aquela em que toda solicitação é aceita no nível da floresta. Da mesma forma, omitir a declaração de regras de controle de acesso significa que o controlador de acesso adotará uma regra padrão, aquela em que toda operação de manutenção da floresta é aprovada para efetivação. Quando for o caso de omitir ambas declarações, basta não especificar o elemento <mgmtstrategies>.

O elemento <admstrategy> representa o ponto de ligação entre o subdomínio de desenvolvimento de estratégias de admissão com o domínio de gerenciamento de recursos em MDRM. Estratégias de admissão são referenciáveis, nesse caso, para reuso da especificação. Trata-se de uma forma mais simples de se associar a uma floresta ou recurso virtual uma estratégia já declarada em algum modelo ou no documento corrente. Se for uma referência, pode ser especificado que ela se atualize juntamente a qualquer manutenção na estratégia referenciada.

Tal como ocorre com qualquer elemento referenciável em Pan, os atributos id e refer de <admstrategy> são mutuamente exclusivos. Para a especificação de uma nova estratégia de admissão, os atributos id, src e type são obrigatórios. Eles não podem ocorrer simultaneamente aos atributos refer e

[5] O atributo servicecategory não deve ocorrer em <mgmtstrategies> quando este elemento for filho de <forest> ou de <leafvr>.
[6] O elemento <schedstrategy> não deve ocorrer como filho de <mgmtstrategies> quando este for elemento filho de <forest>.

`inherit-changes` (regra: `strategies-attributes`), que só devem ser usados para especificar uma referência a uma estratégia existente no documento corrente ou em modelo já implantado.

```
<admstrategy
   id   = xs:ID
   src  = xs:anyURI
   type = xs:string
   refer           = xs:anyURI
   inherit-changes = xs:boolean : false
/>
```

O atributo `id` de `<admstrategy>` deve ser um identificador único no modelo e será usado para possíveis referências no mesmo documento ou em outros modelos após a implantação. O atributo `src` é uma URI que descreve a localização do arquivo que implementa a estratégia de admissão, podendo ser local ou remota, advinda de repositórios Pan ou não. Obrigatoriamente, o arquivo deve estar disponível no momento da validação. O atributo `type` é uma cadeia de caracteres que informa qual é o tipo de implementação contida no arquivo, como o nome de alguma linguagem específica ou genérica, compilada ou interpretada, ou ainda já pronta para implantação. Exemplos de tipos de estratégias de admissão são: uma DSL específica, *Lua*, *Java Servlet*, *C*, *kernel module*, programa executável, etc. A lista de tipos possíveis corresponde aos tipos criados pelas soluções de desenvolvimento no subdomínio de estratégias de admissão ou àqueles suportados pelas plataformas do ambiente distribuído.

O atributo `refer` é uma URI no formato específico já mencionado, e deve obrigatoriamente apontar para um outro elemento `<admstrategy>`, já existente (regra: `refer-correctness`). O atributo opcional `inherit-changes` somente deve ser usado aliado a `refer` (regra: `inherit-changes-needs-a-refer`). Ele determina que se houver manutenções na estratégia de admissão referenciada, elas serão notadas também pela floresta (ou recurso virtual) que possui associada tal referência.

Todos os aspectos comentados sobre `<admstrategy>` valem para o elemento `<schedstrategy>` (que não é usado em florestas), e em parte poderiam valer para o elemento `<accessstrlrules>`, se este for considerado um subdomínio de MDRM. Porém, na especificação atual, o suporte a regras de controle de acesso está embutido no *schema* XML de Pan.

O elemento `<accessctrlrules>` lista, para cada operação de manutenção de modelos mencionada na Seção 3.6 quais são as regras que permitirão ou negarão sua efetivação. O atributo `id` é opcional, e se usado deve conter um identificador único no documento.

Em florestas, os possíveis filhos de `<accessctrlrules>` são: `<initrules>`, `<releaserules>` e `<adaptrules>`, que correspondem às regras de controle de acesso das operações possíveis em florestas: *init*, *release* e *adapt*, respectivamente. Os demais elementos filhos apresentados no modelo XML abaixo não podem ocorrer em regras de controle de acesso de florestas.

```
<accessctrlrules
   id = xs:ID
>
   initrules : splitrules : tunerules : releaserules :
   adaptrules

</accessctrlrules>
```

Cada elemento filho especifica uma lista de regras abstratas, compostas (por meio de elementos de conjunção `<all>` ou de disjunção `<any>`) ou simples. Pan lança mão de elementos e tipos abstratos para possibilitar a extensão da linguagem para regras personalizadas aos modelos. Cada regra é formada pela tripla genérica nome/operação-de-teste/valor. Mais detalhes sobre como construir regras de controle de acesso podem ser encontradas na Subseção 4.2.6. Como exemplo, o trecho de código abaixo associa regras de controle de acesso a uma floresta:

```
...
<forest id="TeleMidiaForest">
    <mgmtstrategies>
        <admstrategy id="admstrat1"
            src="http://mdrm.telemidia.puc-rio.br/strategies/periodic.pmt"
            type="Prometheus"/>
        <accessctrlrules>
            <adaptrules>
                <any>
                    <all>
                        <acr1:permituser op="eq" value="moreno"/>
                        <acr1:permittime op="eq" value="20:00-03:00"/>
                    </all>
                    <acr1:denyuser op="eq" value="*"/>
                </any>
            </adaptrules>
        </accessctrlrules>
    </mgmtstrategies>
</forest>
...
```

Com os elementos descritos até este ponto, já é possível construir um modelo de gerenciamento de recursos em Pan que contenha florestas avulsas e florestas que contenham outras florestas, porém ditas vazias por não terem árvores. O código Pan descrito no Quadro 3 ilustra um exemplo de tal tipo de

modelo, com o intuito de resumir os elementos já descritos e apresentar uma visão mais aberta da disposição de um documento de especificação de modelo.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Primitiva de configuracao Schematron própria do ambiente OxygenXML -->
<?oxygen SCHSchema="
         http://mdrm.telemidia.puc-rio.br/specs/xml/PanSchematron/panvrt.sch"?>

<panvrt xmlns="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
     http://mdrm.telemidia.puc-rio.br/specs/xml/Pan
     http://mdrm.telemidia.puc-rio.br/specs/xml/Pan/panvrt.xsd"
   id="MyFirstModelDocument">

   <title>My First Model Document</title>

   <description>
      This is my first model document, I hope that it is correct!
      I don't know how to plant trees in my forests yet :(
   </description>

   <import alias="imported1"
           src="http://mdrm.telemidia.puc-rio.br/repository/instance1.xml"/>

   <model id="MyFirstModel">

      <forest id="MyFirstForest">

         <mgmtstrategies>
            <admstrategy id="MyFirstAdmStrat"
                src="http://mdrm.telemidia.puc-rio.br/strategies/default.pmt"
                type="Prometheus"/>
            <accessctrlrules>
               <initrules/>
               <releaserules/>
               <adaptrules/>
            </accessctrlrules>
         </mgmtstrategies>

         <forest refer="#MySecondForest"/>

      </forest>

      <forest id="MySecondForest">

         <mgmtstrategies>
            <admstrategy refer="imported1#admstrat1" inherit-changes="true"/>
         </mgmtstrategies>

      </forest>

   </model>

</panvrt>
```

Quadro 3 – Exemplo de modelo descrito em Pan - Somente florestas

Declarando árvores de recursos virtuais

As árvores de recursos virtuais, primitivas ou compostas, são declaradas dentro das florestas por meio do elemento `<rootvr>`, que pode se repetir como filho de `<forest>` indefinidamente. Por definição, `<rootvr>` é um elemento abstrato e deve ser especializado para ser validado como um dos três tipos de especificação de árvores possíveis: primitiva (tipo *PrimitiveVRT*), composta (tipo *CompositeVRT*) ou referência (tipo *ReferVRT*). Para a especialização, deve ser

usado o atributo `xsi:type`, assim os validadores podem assumir corretamente o tipo especializado de `<rootvr>`.

Quando assume o tipo *PrimitiveVRT*, `<rootvr>` deve obrigatoriamente apresentar o atributo `id`, único no documento, usado para possíveis referências no próprio documento, em outros modelos ou em manutenções.

```
<rootvr xsi:type="PrimitiveVRT"
   id = xs:ID
>

   ((targetresource, mgmtstrategies) | targetvr),
   (innervr | leafvr)*

</rootvr>
```

Se a VRT primitiva a ser declarada é voltada a gerenciar um recurso real, o primeiro elemento filho de `<rootvr>` deve ser `<targetresource>`, que é usado para especificar qual é esse recurso real (através do atributo `type`) e sua localização (através do atributo `host`).

```
<targetresource
   type = xs:string
   host = xs:string
/>
```

O atributo `type`, obrigatório, é uma string que pode assumir os seguintes valores: *cpu*, *network*, *diskspace*, *diskaccess* e *memory*. Se necessário, outros recursos podem ser adicionados a essa lista, no *schema* Pan usado pelos validadores. O atributo `host`, também obrigatório, especifica por meio de endereços de rede (endereços IP, por exemplo) qual é o nó da rede a ter aquele recurso gerenciado pela estrutura de VRT primitiva sendo especificada.

Ainda para o caso de VRT gerenciando um recurso real, o próximo elemento, obrigatório, é `<mgmtstrategies>`, já descrito anteriormente[7] no contexto de uma floresta. Como filho de `<rootvr>`, `<mgmtstrategies>` define a categoria de serviço e o conjunto de estratégias de gerenciamento associadas ao recurso virtual raiz da árvore.

Categorias de serviço são mais um ponto de extensibilidade da linguagem Pan, pois podem ser construídas de acordo com as necessidades do modelo. Toda categoria de serviço deve ser especificada em um *schema* XML à parte, que

[7] Elementos já descritos não terão seus modelos XML novamente citados. O leitor deve fazer uso do modelo XML apresentado anteriormente para o mesmo elemento.

entrará no processo de validação para a verificação dos parâmetros de QoS que compõem a categoria de serviço. Detalhes sobre a personalização de parâmetros e categorias de serviço podem ser encontrados na Subseção 4.2.5. A categoria de serviço associada ao recurso virtual raiz é informada por meio do atributo `servicecategory` de `<mgmtstrategies>`. Por ora, é importante mencionar apenas que seu valor deve ser um nome qualificado (`xs:QName`) de um tipo criado em XML schema que especifica a referida categoria.

As estratégias de gerenciamento a serem associadas a um recurso virtual raiz de uma VRT primitiva são especificadas pelos elementos `<admstrategy>` e `<accessctrlrules>`, já mencionados no contexto de florestas, e pelo elemento `<schedstrategy>`, que especifica a estratégia de escalonamento a ser associada. Toda a descrição sobre o elemento `<admstrategy>` feita anteriormente vale para `<schedstrategy>`, uma vez que esses elementos são idênticos quanto aos atributos, possibilidade de serem referenciados e restrições.

```
<schedstrategy
   id   = xs:ID
   src  = xs:anyURI
   type = xs:string
   refer           = xs:anyURI
   inherit-changes = xs:boolean
/>
```

As estratégias declaradas por `<admstrategy>` e `<schedstrategy>`, devem ser compatíveis com a categoria de serviço informada no atributo `servicecategory` de `<mgmtstrategy>`. Um modelo válido é aquele que respeita essa compatibilidade, que pode ser verificada consultando-se um relatório de compatibilidade associado a cada estratégia (regra: `strategies-correctness`). Esses relatórios compreendem o quarto tipo de documentos Pan e são discutidos na Subseção 4.2.4. Essa compatibilidade é necessária para que uma solicitação de reserva ao recurso raiz, especificada por meio dos parâmetros de uma categoria de serviços, possa ser corretamente interpretada e avaliada pelo controlador de admissão e pelo escalonador do recurso, no uso de suas respectivas estratégias.

É importante mencionar que há uma variante de VRT primitiva, cuja raiz está associada não a um recurso real, mas a um recurso virtual de outra VRT primitiva. Para declarar tal VRT, no lugar do elemento `<targetresource>` é usado o elemento `<targetvr>`. Ele possui um único atributo, `refer`, para

especificar por uma URI o recurso virtual a ser gerenciado pela nova VRT. Usa-se o mesmo formato de URI já mencionado para outros atributos `refer`. Obrigatoriamente, o recurso virtual a ser gerenciado tem que ser intermediário e pertencer a outra VRT primitiva (regra: `primitive-targetvr-correctness`). Nessa variante, o elemento `<mgmtstrategies>` não ocorre como filho de `<rootvr>`, uma vez que as estratégias para a raiz já estão especificadas no recurso virtual referenciado.

```
<targetvr
   refer = xs:anyURI
/>
```

A especificação de recursos virtuais filhos de `<rootvr>` em VRTs Primitivas se dá pelo encadeamento de elementos `<innervr>` até elementos `<leafvr>`. As regras sintáticas e semânticas para esses elementos são uniformes entre os tipos de VRT, e, por isso, serão apresentados após esgotadas as discussões sobre a criação do recurso virtual raiz dos diferentes tipos de VRTs.

Quando `<rootvr>` assume o tipo *CompositeVRT*, ou seja, quando for dedicado a declarar um VRT composta, deve obrigatoriamente apresentar o atributo `id`, único no documento, usado para possíveis referências no próprio documento, em outros modelos ou em manutenções.

```
<rootvr xsi:type="CompositeVRT"
   id = xs:ID
>

   targetvr+, mgmtstrategies, (innervr | leafvr)*

</rootvr>
```

O primeiro elemento de `<rootvr>`, nesse caso, é `<targetvr>`, já mencionado para VRTs primitivas que gerenciam um recurso virtual de outras VRTs primitivas. Dessa vez, para representar a composição de um ou mais recursos virtuais, o elemento `<targetvr>` deve ocorrer uma ou mais vezes. Seu atributo `refer` deve obrigatoriamente apontar para um recurso virtual folha, conforme restrição comentada no meta modelo (regra: `refer-correctness`). Outra restrição do meta modelo, que deve ser satisfeita na validação do modelo, é a obrigatoriedade de uso exclusivo do recurso virtual folha pela VRT composta, ou seja, o recurso virtual folha já referenciado por alguma raiz de VRT composta não pode mais ser referenciado por outras (regra: `composite-targetvr-`

`single-reference`). A verificação deve ocorrer de duas formas: a primeira, checando se o recurso virtual folha (elemento `<leafvr>`) já não possui seu atributo `busy` com o valor *true*, o que significa que ele já está em uso por alguma composição; a segunda, checando se no documento corrente não há duas ou mais referências ao mesmo recurso virtual por elementos `<targetvr>` em VRTs compostas.

O segundo elemento filho de `<rootvr>`, também obrigatório, é `<mgmtstrategies>`, e segue a mesma sintaxe e semântica que a descrita para o esse elemento em VRTs primitivas associadas a recursos reais. Os demais elementos filhos de `<rootvr>` como raiz de VRT composta são as declarações de recursos virtuais filhos.

O último tipo de VRT é usado apenas para referenciar uma outra VRT, primitiva ou composta, para fins de inclusão da VRT referenciada em florestas sendo modeladas. Nesse caso, o elemento `<rootvr>` deve assumir o tipo *ReferVRT* e conter apenas um atributo, `refer`.

```
<rootvr xsi:type="ReferVRT"
   refer = xs:anyURI
/>
```

Tendo sido apresentadas as especificações do recurso virtual raiz dos quatro diferentes tipos de declarações de VRTs (primitiva associada a recurso real; primitiva associada a recurso virtual de outra primitiva; composta; e referência), pode-se agora retomar a definição de seus recursos virtuais.

Encerra-se, neste ponto, as discussões sobre a especificação do recurso virtual raiz dos quatro diferentes tipos de declarações de VRTs: primitiva associada a recurso real; primitiva associada a recurso virtual de outra primitiva; composta; e referência.

VRTs primitivas e compostas apresentam sua hierarquia de recursos virtuais por meio de elementos opcionais `<innervr>` (recursos virtuais intermediários) encadeados, que podem culminar em elementos opcionais `<leafvr>` (recursos virtuais folhas). O elemento `<innervr>` pode ocorrer várias vezes como filho de `<rootvr>` e possui um atributo `id`, obrigatório, usado para identificá-lo univocamente no modelo, e, assim, poder ser referenciado por elementos `<targetvr>` no próprio documento ou em outros modelos, e por operações de manutenção.

```
<innervr
  id = xs:ID
>

  reservation, mgmtstrategies, (innervr | leafvr)*

</innervr>
```

O primeiro elemento filho de <innervr>, obrigatório, é <reservation>, usado para especificar qual a parcela alocada a partir do recurso virtual pai (que pode ser <rootvr> ou outro <innervr>). A especificação da alocação é feita por meio dos parâmetros de QoS reunidos na categoria de serviço oferecida pelo recurso virtual pai, onde foi informada por meio do atributo servicecategory do elemento <mgmtstrategies>. Por isso, o elemento <reservation> é abstrato e precisa ser especializado no modelo, para se tornar uma instância do conjunto de parâmetros da categoria de serviço do recurso virtual pai. A especialização é feita por meio do atributo xsi:type, tal como feito em <rootvr>, desta vez para os tipos criados pelo projetista para especificar categorias de serviço. Se uma categoria de serviço reúne, por exemplo, os parâmetros taxa média e taxa de pico, eles serão tratados como elementos filhos de <reservation>. No âmbito de um modelo Pan voltado à arquitetura *IntServ*, esse exemplo resultaria na seguinte especificação:

```
...
<reservation xsi:type="cl:ControlledLoad">
    <cl:tspec-r value="1000000" unit="Bps"/>  <!-- taxa média -->
    <cl:tspec-p value="1200000" unit="Bbps"/> <!-- taxa de pico -->
    ...
</reservation>
...
```

A validação dos nomes e valores desses parâmetros é feita por meio do *schema* construído pelo projetista na especificação da categoria de serviço informada em xsi:type. As diretivas para guiar a construção desses *schemas* podem ser encontradas na Subseção 4.2.5. O nome qualificado da categoria de serviço informado em xsi:type deve ser equivalente ao nome qualificado da categoria de serviço oferecida pelo pai, informada no atributo servicecategory de <mgmtstrategies> do recurso virtual pai (regra: reservation-matches-servicecategory).

O próximo elemento filho de <innervr>, obrigatório, é <mgmtstrategies>, que define a lógica de gerenciamento do recurso virtual intermediário tal como descrito no uso do mesmo elemento em um recurso virtual

raiz, sintática e semanticamente. Os demais elementos filhos de <innervr> são opcionais e compreendem outros elementos <innervr> (perfazendo a hierarquia de recursos virtuais) e elementos <leafvr>.

O elemento <leafvr> é usado para a especificação de recursos virtuais folhas, que não podem compartilhar sua capacidade com outros recursos virtuais. O elemento possui um atributo id obrigatório, usado para identificá-lo de forma única no modelo, e, assim, poder ser referenciado por elementos <targetvr> no próprio documento ou em outros modelos, e por operações de manutenção.

```
<leafvr
   id = ID
>

   reservation, classifrules?, mgmtstrategies?

</leafvr>
```

O primeiro elemento filho de <leafvr>, obrigatório, é <reservation>, que descreve a parcela de alocação relativa ao recurso virtual pai, exatamente da mesma forma já descrita para o mesmo elemento como filho de <innervr>, sintática e semanticamente.

O segundo elemento filho de <leafvr> é <classifrules>, usado opcionalmente para descrever as regras de classificação que caracterizam as unidades de informação dos fluxos que devem ser escalonados pelo recurso virtual folha. As regras são muito semelhantes às regras de controle de acesso definidas para cada operação de manutenção. O elemento <classifrules> especifica uma lista de regras abstratas, compostas (por meio de elementos de conjunção <all> ou de disjunção <any>) ou simples. Novamente, Pan lança mão de elementos e tipos abstratos para possibilitar a extensão da linguagem para regras personalizadas aos modelos. Cada regra é formada pela tripla genérica nome/operação-de-teste/valor. Mais detalhes sobre como construir regras de classificação podem ser encontrados na Subseção 4.2.7. A título de exemplo, o trecho de código Pan abaixo associa algumas regras de classificação a um recurso virtual folha:

```
...
<classifrules>
   <all>
      <any>
         <ip:destaddr op="eq" value="139.82.95.11"/>
         <ip:destaddr value="139.82.95.12"/> <!-- Operação eq é default -->
         <ip:tos op="eq" value="3"/>
      </any>
```

```
      <ip:sourceaddr op="eq" value="139.82.35.135"/>
   </all>
</classifrules>
...
```

O último elemento filho de <leafvr> é <mgmtstrategies>, opcional, usado para definir a lógica de gerenciamento do recurso virtual folha. A sintaxe dele é semelhante à usada pelo mesmo elemento em recursos virtuais intermediários, a não ser pelo fato de não ocorrerem o atributo servicecategory e o elemento filho <admstrat>, já que folhas não oferecem serviços sobre sua parcela alocada.

O código Pan descrito no Quadro 4, a seguir, ilustra a utilização de muitos dos elementos descritos nesta seção, para criar um modelo de gerenciamento de recursos em um roteador baseado na arquitetura *IntServ*. As categorias de serviço e parâmetros utilizados são, por enquanto, considerados já existentes. A construção desses tipos especializados é abordada na Subseção 4.2.5.

```
<?xml version="1.0" encoding="UTF-8"?>

<panvrt
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan
                        http://mdrm.telemidia.puc-rio.br/specs/xml/Pan/panvrt.xsd
                        ..."
    xmlns="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan"
    xmlns:ir="http://mdrm.telemidia.puc-rio.br/sc/IntServ/IntServRoot"
    xmlns:cl="http://mdrm.telemidia.puc-rio.br/sc/IntServ/ControlledLoad"
    xmlns:gs="http://mdrm.telemidia.puc-rio.br/sc/IntServ/GuaranteedService"
    id="MySecondModelDocument">

    <title>My Second Model Document</title>

    <description>
        My Second Model Document is much better. Now I can plant my trees!
        For the time being, I'm unable to create service categories, clssification
        rules, etc :(
        But I'm using some categories readily available in TeleMidia,
        to create an Intserv-based resource management model
    </description>

    <model id="MySecondModel">

        <forest id="MyIntServForest">

            <rootvr id="IntservNode" xsi:type="PrimitiveVRT">

                <targetresource type="network" host="10.0.0.2"/>
                <mgmtstrategies servicecategory="ir:IntServRoot">
                    <schedstrategy id="rootSchedStrat"
                        src="http://mdrm.telemidia.puc-rio.br/strategies/wfq.chr"
                        type="Chronos"/>
                    <admstrategy id="rootAdmStrat"
                        src="http://mdrm.telemidia.puc-rio.br/
                             strategies/refuseAll.pmt"
                        type="Prometheus"/>
                </mgmtstrategies>

                <innervr id="GuaranteedServiceVR">
                    <reservation xsi:type="ir:IntServRoot">
                        <ir:avguse value="50" unit="%"/>
                    </reservation>
```

```
                    <mgmtstrategies servicecategory="gs:GuaranteedService">
                        <schedstrategy refer="#rootSchedStrat"/>
                        <admstrategy id="gsAdmStrat"
                            src="http://mdrm.telemidia.puc-rio.br/
                                strategies/avgRate.pmt"
                            type="Prometheus"/>
                    </mgmtstrategies>
                </innervr>

                <innervr id="ControlledLoadVR">
                    <reservation xsi:type="ir:IntServRoot">
                        <ir:avguse value="50" unit="%"/>
                    </reservation>
                    <mgmtstrategies servicecategory="cl:ControlledLoad">
                        <admstrategy id="clAdmStrat"
                            src="http://mdrm.telemidia.puc-rio.br/
                                strategies/avgRate10.pmt"
                            type="Prometheus"/>
                    </mgmtstrategies>
                </innervr>

            </rootvr>

        </forest>

    </model>

</panvrt>
```

Quadro 4 – Exemplo de modelo descrito em Pan - IntServ

4.2.2. Descrevendo manutenções de modelos em Pan

O segundo tipo de documento Pan trata da manutenção dos modelos já implantados. Ele descreve uma seqüência de operações de manutenção de árvores e florestas, acondicionadas em transações. A organização em transações permite que operações sejam agrupadas de forma a definir uma interdependência tal que, se uma operação falhar, todas as outras serão canceladas e/ou desfeitas (*rollback*). As operações de manutenção correspondem a cada uma daquelas descritas na Seção 3.6.

Operações de manutenção podem falhar devido, primeiramente, à falta de permissão para efetivá-la, reportada pelo controlador de acesso da árvore ou floresta envolvida. Há outros casos de falha, como a dependência de algum outro tipo de controle, como o de admissão para a operação *split*. Pode também haver falha se desde o momento da validação até a execução da operação o recurso virtual ou floresta envolvido tiver sido modificado em algum aspecto relevante ou mesmo removido.

Toda operação de manutenção em Pan é focada na modificação de alguma parte de um modelo já implantado, presente em algum repositório MDRM. Essa modificação culmina na alteração textual no conteúdo do documento do modelo.

Dessa forma, os novos elementos, as modificações ou remoções serão fisicamente realizados nos modelos implantados. Os modelos que serão modificados por um documento de manutenção devem ser importados por meio de elementos `<import>`, conforme já descrito.

Um documento de manutenção de um modelo é caracterizado pela declaração do elemento `<maintenance>`, logo após a parte inicial em comum já descrita, tendo `<panvrt>` como elemento pai. O elemento `<maintenance>` possui como elementos filhos uma série indefinida de elementos `<transaction>`, quantos forem necessários para especificar as transações desejadas na manutenção.

```
<maintenance>

   transaction+

</maintenance>
```

O elemento `<transaction>` possui um atributo `id` obrigatório, usado como identificador único no documento de manutenção, de forma que ele possa ser referenciado em documentos de resposta a manutenções.

```
<transaction
   id = xs:ID
>

   (init | split | release | tune | adapt)+

</transaction>
```

As operações de manutenção ocorrem como elementos filhos de `<transaction>` em qualquer ordem, bastando que haja pelo menos uma operação em cada transação. Por simplificação, a operação *Merge* está ausente em Pan, pois é suplantada pela operação *Init*, conforme mencionado na Seção 3.6.

O elemento `<init>`, filho de `<transaction>` é usado para especificar a operação de manutenção *Init*, que permite a criação de novas árvores ou florestas dentro de uma dada floresta. O elemento `<init>` possui um atributo obrigatório `id`, usado como identificador único no documento de manutenção, para fins de referência por documentos de resposta a manutenções.

O elemento `<init>` possui também um atributo `targetforest`, que especifica uma URI, de mesmo formato dos atributos `refer` usados em modelos, que aponta para a floresta já implantada a receber as novas florestas ou árvores. O

atributo deve obrigatoriamente apontar para uma floresta (regra: `targetforest-attribute-must-be-a-forest`).

```
<init
   id = xs:ID
   targetforest = xs:anyURI
>

   (forest | rootvr)*

</init>
```

Os elementos filhos de `<init>` podem ser repetições desordenadas de elementos `<forest>` e elementos `<rootvr>`, exatamente com as respectivas sintaxes e semânticas possíveis de seus homônimos usados em modelos. Isso significa que, em uma única operação de gerenciamento, uma nova árvore pode ser definida por inteiro, assim como uma floresta que contenha outras florestas e árvores.

Observa-se que, na especificação das novas florestas, árvores, recursos virtuais, estratégias, etc., cada elemento receberá valores para o atributo `id` que podem ser garantidos como únicos no documento de manutenção, quando de sua validação contra o *schema* XML de Pan. Porém, esses elementos vão fazer parte de um modelo já implantado, ou seja, o modelo será modificado para ter incluídos os novos elementos. Nesse ponto, pode haver conflito de identificadores entre os elementos já presentes no modelo e os elementos novos especificados na manutenção. Esse problema não pode ser detectado pela validação contra o *schema* XML de Pan. Mas bastaria checar se os atributos `id` especificados na manutenção já não estão em uso no modelo, e isso é feito na validação contra o *schema Schematron* de Pan (regra: `id-uniqueness-after-operation`). O problema não é exclusivo da operação *Init* e, portanto, a regra considera os atributos `id` dos elementos em todas as operações especificadas.

O próximo elemento filho de `<transaction>`, `<split>`, especifica uma operação *Split*, que permite a criação de um recurso virtual filho a partir de algum recurso virtual já existente em um modelo implantado. Essa operação está sujeita à avaliação do controlador de admissão associado à árvore que possui o recurso virtual candidato a pai. O elemento `<split>`, como todo elemento que especifica uma operação de manutenção, possui um atributo `id` obrigatório para identificá-lo univocamente.

```
<split
   id = xs:ID
   targetvr = xs:anyURI
>

   innervr | leafvr

</split>
```

O atributo `targetvr` especifica uma URI (do formato de `refer`) para identificar qual recurso virtual será "dividido" para a criação do novo recurso virtual especificado. O recurso virtual candidato a pai não pode ser folha, uma vez que não suportaria a concessão da garantia solicitada (regra: `split-targetvr-must-be-nonleaf`).

Um dos possíveis elementos filhos de `<split>` especifica o novo recurso virtual a ser criado. Se for um novo recurso virtual intermediário, o elemento `<innervr>` será usado. Se for um novo recurso virtual folha, o elemento `<leafvr>` será usado. Esses elementos possuem, respectivamente, a mesma sintaxe e semântica que seus homônimos usados em modelos.

Obrigatoriamente, o elemento `<reservation>`, filho de `<innervr>` ou `<leafvr>` (conforme o caso), deve ser especializado da mesma forma que o mesmo elemento o é em modelos, sendo que o nome qualificado da categoria de serviço escolhida deve ser equivalente ao nome qualificado da categoria de serviço oferecida pelo recurso virtual candidato a pai (regra: `split-matches-servicecategory`).

O próximo elemento filho de `<transaction>` é `<release>`, usado para especificar uma operação de manutenção *Release*, responsável por liberar a parcela alocada de um dado recurso virtual, ou seja, destruí-lo. A operação também pode ser usada para eliminar uma floresta e suas árvores. Seu atributo `id` é obrigatório, usado para identificá-lo univocamente.

```
<release
   id = ID
   targetvr = xs:anyURI
   targetforest = xs:anyURI
   recursive = xs:boolean : false
/>
```

O elemento `<release>` possui, também, os atributos `targetvr` e `targetforest`, os quais são mutuamente exclusivos, sendo que o uso de um deles é obrigatório (regra: `adapt-and-release-must-have-`

targetforest-or-targetvr), pois trata-se da escolha entre remover um recurso virtual ou uma floresta. Ambos os atributos são URIs no mesmo formato que usado em refer. O atributo targetvr aponta, obrigatoriamente, para um recurso virtual (regra: targetvr-attribute-must-be-a-vr), e targetforest aponta, obrigatoriamente, para uma floresta (targetforest-attribute-must-be-a-forest).

A operação falha se o recurso virtual apontado por targetvr tiver recursos virtuais filhos ou se a floresta apontada por targetforest incluir outras florestas ou árvores. Se a operação for voltada para a remoção proposital de toda uma subárvore ou toda uma floresta, o atributo opcional recursive deve ser usado com o valor *true*.

O próximo elemento filho de <transaction> é <tune>, usado para especificar uma operação de manutenção *Tune*, responsável por realizar a modificação dos parâmetros de QoS que especificam a parcela alocada a um recurso virtual. O elemento <tune> possui um atributo id obrigatório, usado para identificá-lo univocamente.

```
<tune
   id = ID
   targetvr = anyURI
>

   reservation

</tune>
```

Seu atributo targetvr também é obrigatório e deve informar por meio de uma URI qual é o recurso virtual a ter seus parâmetros de reserva modificados. Obrigatoriamente, targetvr deve apontar para um recurso virtual que não seja raiz, uma vez que raízes representam a alocação total do recurso real, virtual ou composto gerenciado (regra: tune-targetvr-must-be-nonroot).

O único elemento filho de <tune> é <reservation>, que possui a mesma sintaxe e semântica que seu homônimo usado em modelos. Da mesma forma que nos modelos, ele deve se especializar a um tipo criado de categoria de serviço e descrever uma lista de parâmetros de QoS. O tipo da categoria de serviço usado deve ser o mesmo que o oferecido pelo recurso virtual pai do recurso virtual sendo modificado (regra: tune-matches-servicecategory).

Por fim, o último elemento filho possível de `<transaction>` é `<adapt>`, usado para especificar uma operação de manutenção *Adapt*, responsável por modificar a lógica de gerenciamento de um recurso virtual ou de uma floresta. O elemento `<adapt>` possui um atributo `id` obrigatório, usado para identificá-lo univocamente.

```
<adapt
   id = ID
   targetvr = anyURI
   targetforest = anyURI
>

   mgmtstrategies

</adapt>
```

O elemento `<adapt>` possui, também, os atributos `targetvr` e `targetforest`, os quais são mutuamente exclusivos, sendo que o uso de um deles é obrigatório (regra: `adapt-and-release-must-have-targetforest-or-targetvr`), pois trata-se da escolha entre modificar as estratégias de gerenciamento de um recurso virtual ou de uma floresta. Ambos atributos são URIs no mesmo formato que usado em `refer`. O atributo `targetvr` aponta, obrigatoriamente, para um recurso virtual (regra: `targetvr-attribute-must-be-a-vr`), e `targetforest` aponta, obrigatoriamente, para uma floresta (`targetforest-attribute-must-be-a-forest`).

O único filho do elemento `<adapt>` é `<mgmtstrategies>`, que possui a mesma sintaxe e semântica do elemento homônimo usado em modelos. Os elementos filhos de `<mgmtstrategies>` a serem incluídos devem ser somente os que deverão ser modificados. Se a adaptação tem como alvo um recurso virtual folha, apenas os elementos `<schedstrategy>` e `<acessctrlrules>` podem ser especificados como filhos de `<mgmtstrategies>`. Se o alvo for uma floresta, apenas os elementos `<admstrategy>` e `<accessctrlrules>` poderão ser usados (regra: `adapt-correctness`).

O Quadro 5, a seguir, ilustra um exemplo de documento de manutenção de modelos. Todas as operações desse exemplo têm como alvo o modelo descrito anteriormente no Quadro 4, agora implantado.

```
<?xml version="1.0" encoding="UTF-8"?>

<panvrt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan
```

```
                        http://mdrm.telemidia.puc-rio.br/specs/xml/Pan/panvrt.xsd"
                        ...
    xmlns="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan"
    xmlns:ir="http://mdrm.telemidia.puc-rio.br/sc/IntServ/IntServRoot"
    xmlns:cl="http://mdrm.telemidia.puc-rio.br/sc/IntServ/ControlledLoad"
    xmlns:gs="http://mdrm.telemidia.puc-rio.br/sc/IntServ/GuaranteedService"
    xmlns:acr1="http://mdrm.telemidia.puc-rio.br/acr/ACRules1"
    id="MyFirstMaintenanceDocument">

    <title>My First Maintenance Document</title>

    <description>
        My First Maintenance Document is nice. Now I can take care of my trees!
        But I'm still unable to create my own service categories, classification
        rules, etc :(
    </description>

    <import alias="My2ndModel"
            src="http://mdrm.telemidia.puc-rio.br/repository/intserv-forest.xml"/>

    <maintenance>

        <transaction id="MyFirstTransaction">

            <split targetvr="My2ndModel#GuaranteedServiceVR" id="split1">
                <leafvr id="IPTVFlow27">
                    <reservation xsi:type="gs:GuaranteedService">
                        <gs:rspec-R value="1200000" unit="Bps"/>
                        <gs:tspec-p value="1500000" unit="Bps"/>
                        <gs:tspec-r value="1200000" unit="Bps"/>
                    </reservation>
                </leafvr>
            </split>

            <init targetforest="My2ndModel#MyIntServForest" id="init1">
                <rootvr xsi:type="CompositeVRT" id="network-10-0-0-11">
                    <targetvr refer="#IPTVFlow27"/>
                    <mgmtstrategies
                        servicecategory="ir:IntservRootServiceCategory"/>
                </rootvr>
            </init>

        </transaction>

        <transaction id="MySecondTransaction">

            <release recursive="true" targetvr="My2ndModel#IntservNode"
                     id="release1"/>

            <adapt targetforest="My2ndModel#MyIntServForest" id="adapt1">
                <mgmtstrategies>
                    <admstrategy id="rootAdmStrat"
                        src="http://mdrm.telemidia.puc-rio.br/
                             strategies/acceptAll.pmt"
                        type="Prometheus"/>
                </mgmtstrategies>
            </adapt>

        </transaction>

    </maintenance>

</panvrt>
```

Quadro 5 – Exemplo de documento de manutenção de modelos

4.2.3. Descrevendo documentos de resposta a manutenções

Toda manutenção submetida às plataformas está sujeita a falhas, conforme apresentado na subseção anterior. Por isso, o resultado de cada operação e

transação deve ser informado ao solicitante de uma manutenção. Documentos de resposta a manutenções são listas de resultados gerados pela submissão de um documento de manutenção. Normalmente, documentos de resposta a manutenções são gerados automaticamente pelos componentes responsáveis pela implantação de modelos e operações. Os documentos possuem uma estrutura muito simples, com elementos que mapeiam operação a operação e transação a transação, quais foram seus resultados efetivos.

Um documento de resposta a manutenções é caracterizado pela declaração do elemento `<maintenance-response>`, logo após a parte inicial em comum já descrita, tendo `<panvrt>` como elemento pai. O elemento `<maintenance-response>` possui como elementos filhos uma série indefinida de elementos `<transaction-response>`, quantas forem as transações submetidas no documento de manutenção.

```
<maintenance-response>

   transaction-response+

</maintenance-response>
```

O elemento `<transaction-response>` denota a resposta a uma determinada transação. Seu atributo `id`, obrigatório, o identifica de forma unívoca no documento, e deve obrigatoriamente ser preenchido com o `id` do elemento `<transaction>` que ele expressa o resultado.

```
<transaction-response
   id = xs:ID
>

   (init-response | split-response | release-response |
    tune-response | adapt-response)+, succeeded

</transaction-response>
```

Os elementos filhos de `<transaction-response>` denotam o resultado individual de cada operação, ocorrendo um número indefinido de vezes, quantas forem as operações contidas na transação que o elemento `<transaction-response>` responde. Os elementos de resposta a operações podem ser: `<init-response>`, `<split-response>`, `<release-response>`, `<tune-response>` e `<adapt-response>`. Eles expressam os resultados das operações descritas pelos elementos `<init>`, `<split>`, `<release>`, `<tune>` e `<adapt>`, respectivamente. Todos os elementos possuem a mesma sintaxe. Cada um possui

um atributo `id`, obrigatório, usado para identificá-lo univocamente no documento, e deve ser preenchido com o mesmo valor do atributo `id` fornecido no elemento da operação submetida que ele expressa o resultado.

```
<(init|split|release|tune|adapt)-response
   id = xs:ID
>

   accepted

</(init|split|release|tune|adapt)-response>
```

Todos os elementos de resposta a operações possuem um único filho, `<accepted>`, cujo valor de conteúdo pode ser *true* ou *false*. Por meio desses valores, o elemento obrigatório `<accepted>` denota se a operação, à qual seu elemento pai responde, foi aceita. Isso não quer dizer que a operação foi efetivada, uma vez que isso depende dos resultados de outras operações na mesma transação. Se uma certa operação foi aceita, mas uma outra operação na mesma transação não o foi, a transação não foi efetivada, e, portanto, as operações que seriam aceitas foram canceladas.

```
<accepted>

   "true" | "false"

</accepted>
```

Após todas as respostas a operações terem sido especificadas como elementos filhos de `<transaction-response>`, um último elemento filho, `<succeeded>`, denota o resultado final da transação à qual seu elemento pai responde.

```
<succeeded>

   "true" | "false"

</succeeded>
```

O elemento obrigatório `<succeeded>` deve ser preenchido com os valores *true* ou *false*, de acordo com a efetivação ou não de todas as operações contidas na transação que `<transaction-response>` responde. É o resultado expresso em `<succeeded>` que denota se uma operação marcada como aceita em uma transação foi realmente efetivada.

O código Pan listado no Quadro 6 é um exemplo de documento de resposta a manutenções, uma possível instância gerada como resultado na submissão do

documento de manutenção anteriormente listado no Quadro 5. Nota-se que para cada operação de manutenção, um elemento de resposta correspondente com o mesmo valor de atributo `id` é especificado.

```
<?xml version="1.0" encoding="UTF-8"?>

<panvrt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan
                        http://mdrm.telemidia.puc-rio.br/specs/xml/Pan/panvrt.xsd"
    xmlns="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan"
    id="MyFirstMaintenanceResponseDocument">

    <title>My First Maintenance Response Document</title>

    <description>
        My First Maintenance Response Document is a possible result after
        deploying MyFirstMaintenanceDocument. Let's see which operations have
        worked well.
    </description>

    <maintenance-response>

        <transaction-response id="MyFirstTransaction">

            <split-response id="split1">
                <accepted>true</accepted>
            </split-response>

            <init-response id="init1">
                <accepted>true</accepted>
            </init-response>

            <succeeded>true</succeeded>

        </transaction-response>

        <transaction-response id="MySecondTransaction">

            <release-response id="release1">
                <accepted>true</accepted>
            </release-response>

            <adapt-response id="adapt1">
                <accepted>false</accepted>
            </adapt-response>

            <succeeded>false</succeeded>

        </transaction-response>

    </maintenance-response>

</panvrt>
```

Quadro 6 – Exemplo de documento de resposta a manutenções

4.2.4. Descrevendo documentos de relatório de compatibilidade

O último tipo de documentos Pan permite que sejam descritas informações de compatibilidade de algum aspecto de um modelo Pan em relação a outros aspectos relacionados ou dependentes. O principal caso de tal dependência ocorre

com as estratégias de admissão e de escalonamento em relação às categorias de serviços.

Uma estratégia de admissão deve receber parâmetros de QoS informados na solicitação de um serviço, saber manipulá-las para a avaliação de viabilidade da admissão e, baseada nessa avaliação, emitir seu parecer. Uma estratégia de escalonamento deve receber parâmetros de QoS informado na solicitação de um serviço e saber manipulá-las para configurar sua lógica interna de escolha periódica ou espacial dos fluxos que farão uso do recurso. Em ambos os casos, "saber manipulá-las" representa uma dependência entre tais estratégias e os tipos de parâmetros informados na solicitação de serviços. Como os parâmetros de QoS podem ser reunidos em uma categoria de serviços, fica então estabelecida a dependência entre estratégias de gerenciamento e categorias de serviços.

Assim, o projetista que constrói uma estratégia de gerenciamento (lembrando que elas são tratadas como subdomínios em MDRM), deve gerar um relatório de compatibilidade para descrever quais são as categorias de serviço possíveis de serem conjugadas com seu produto. À medida que novas categorias de serviço vão sendo criadas, elas podem ser adicionadas a esse relatório, se analisadas como compatíveis à estratégia de gerenciamento reportada.

Com um documento de relatório de compatibilidade acompanhando cada estratégia de gerenciamento, validações podem ser realizadas em tempo de modelagem, para verificar a coerência da associação das estratégias a um recurso virtual e à sua categoria de serviço oferecida. Um documento de relatório de compatibilidade Pan possui estrutura muito simples. Por enquanto, sua validação considera apenas a compatibilidade do item reportado com categorias de serviço. Outros aspectos podem ser adicionados com a evolução da linguagem Pan. A associação entre a estratégia de gerenciamento e seu relatório de compatibilidade não é feita no documento, mas externamente. Uma possível associação poderia ser feita mantendo o nome do arquivo de relatório igual ao do arquivo que implementa a estratégia, acrescido de uma extensão; e mantê-los sempre juntos, em um mesmo repositório ou diretório.

O emprego de relatórios de compatibilidade é de alguma forma similar ao uso de interfaces descritas em IDL (*Interface Description Language*) na programação baseada em componentes (Szyperski, 2002), porém em um nível de

abstração mais alto e adequado à modelagem colaborativa do gerenciamento de recursos.

Um documento de relatório de compatibilidade é caracterizado pela declaração do elemento `<compatibility-report>`, logo após a parte inicial em comum já descrita, tendo `<panvrt>` como elemento pai. O elemento `<compatibility-report>` possui como elementos filhos uma série indefinida de elementos `<sccompatibility>`, quantas forem as categorias de serviço a serem reportadas como compatíveis com a estratégia que o relatório acompanha.

```
<compatibility-report>

  sccompatibility+

</compatibility-report>
```

O elemento `<sccompatibility>` possui um único atributo, obrigatório, deonominado `servicecategory`, que deve ser preenchido com um nome qualificado (xs:QName) de um tipo criado em *schema* XML para especificar a categoria de serviço compatível. Mais detalhes sobre os nomes qualificados de categorias de serviço podem ser encontrados na próxima subseção.

```
<sccompatibility
  servicecategory=xs:QName
/>
```

O código Pan listado no Quadro 7 é um relatório de compatibilidade de uma estratégia de admissão que aceita os parâmetros período e tempo de uso, ou seja, ideal para serviços com garantias de periodicidade, como serviços de tempo real severo.

```
<?xml version="1.0" encoding="UTF-8"?>

<panvrt  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan
    http://mdrm.telemidia.puc-rio.br/specs/xml/Pan/panvrt.xsd
    xmlns:hsc="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan/
                    HardRTServiceCategory"
    xmlns:rmc="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan/RMServiceCategory"
    xmlns="http://mdrm.telemidia.puc-rio.br/specs/xml/Pan">

    <description>
        Pan compatibility report for the Periodic Admission Strategy
        (http://mdrm.telemidia.puc-rio.br/strategies/periodic.apl)
    </description>

    <compatibility-report>
        <sccompatibility servicecategory="hsc:HardRTServiceCategory"/>
        <sccompatibility servicecategory="rmc:RateMonotonicServiceCategory"/>
    </compatibility-report>

</panvrt>
```

Quadro 7 – Exemplo de documento de relatório de compatibilidade

4.2.5. Construindo parâmetros e categorias de serviço

Conforme mencionado na Seção 3.2, uma categoria de serviço descreve qual tipo de serviço um recurso virtual oferece, reunindo um conjunto de parâmetros de QoS e de caracterização de tráfego a serem usados nas solicitações de serviço. Por exemplo, uma categoria de serviços de tempo real severo descreve parâmetros tais como a periodicidade e o tempo de uso de um recurso virtual. Uma outra categoria, de tempo real suave, poderia descrever o parâmetro percentual de uso do recurso virtual. Esses parâmetros são usados pelo solicitante, na especificação de suas necessidades de QoS; pelas estratégias de admissão na avaliação da viabilidade de aceitação de tal caracterização do serviço; e pelas estratégias de escalonamento para configurarem sua lógica de compartilhamento.

Em Pan, a definição de categorias de serviço e de seus parâmetros é tarefa para os projetistas. Pan não especifica o conjunto de parâmetros disponíveis para uso, uma vez que isso inviabilizaria a evolução dos modelos com o surgimento de novos requisitos. Para prover a adaptabilidade necessária, Pan define parâmetros de categorias de serviço com um elemento abstrato, a ser especializado caso a caso. Nota-se, porém, que há a necessidade de validação desses parâmetros e categorias de serviço em tempo de projeto.

O termo "estender a linguagem Pan para atender às necessidades individuais dos serviços" deve ser sempre entendido como disponibilizar novos elementos à linguagem, desde que eles possam ser validados. A forma de validação de todas as extensões de Pan deve se basear em *XML Schema*. É através de uma *schema* XML (derivado do *schema* XML de Pan) que categorias de serviço são especificadas. Esta subseção se dedica a descrever como isso pode ser feito.

Todo *schema* XML é baseado em tipos, elementos e atributos. Um elemento é uma instância de algum tipo. O nome qualificado (QName) de um tipo em *XML Schema* é a conjunção do *namespace* onde o tipo está definido e o nome localmente dado ao tipo, identificando-o, assim, de forma não-ambígua.

Quando define-se em um *schema* que um certo elemento e seu tipo são abstratos, eles não podem ocorrer no documento na forma como foram especificados. Eles precisam ser especializados em um elemento e um tipo

concreto que herdam do elemento e tipo abstratos definidos. Se o elemento for corretamente especializado, a validação ocorre sem contratempos.

O método descrito no parágrafo anterior permite que conteúdo variável seja validado por *schemas*, restringindo a forma como os elementos e tipos são criados (XML-Dev, 2001). Para ferramentas de edição de código Pan, esse método habilita a funcionalidade de dicas automáticas para completar código, baseando-se apenas em *schemas*.

O elemento `<reservation>` é definido no *schema* Pan como abstrato e, conforme mencionado na Subseção 4.2.2, deve ser especializado por meio do atributo `xsi:type`. O tipo para o qual ele deve ser especializado deve ser derivado do tipo abstrato *pan:ServiceCategoryType*. Pelo trecho do *schema* XML de Pan logo abaixo, nota-se que esse tipo consiste em uma lista de elementos `pan:servicecategoryparam`. Esse elemento, da mesma forma, é abstrato, assim como seu tipo *pan:ParamType*, também abstrato.

```
<!-- ParamType - Abstract Type for a generic parameter -->
   <xs:complexType abstract="true" name="ParamType" block="extension">
      <xs:attribute name="value" use="required" type="xs:anySimpleType"/>
      <xs:attribute name="unit" type="xs:string"/>
   </xs:complexType>

<!-- servicecategryparam - Abstract element to be replaced by parameters -->
   <xs:element abstract="true" name="servicecategoryparam" type="ParamType"
               block="extension"/>

<!-- ServiceCategoryType - Abstract Type for service categories -->
   <xs:complexType abstract="true" name="ServiceCategoryType"
                   block="extension">
      <xs:choice maxOccurs="unbounded">
         <xs:element ref="servicecategoryparam" maxOccurs="unbounded"/>
      </xs:choice>
   </xs:complexType>
```

Em resumo, o que deve ser feito por um *schema* que descreve uma categoria de serviços é i) especializar o tipo *pan:ParamType* para tipos concretos X que definem parâmetros de QoS no formato nome/valor/unidade; ii) definir elementos N com tipos X que substituam `pan:servicecategoryparam`; e iii) especializar o tipo *pan:ServiceCategory* para um tipo concreto Y que faça uso dos elementos N criados em sua lista de parâmetros.

Como exemplo, seja *SoftRTServiceCategory* a nova categoria a ser criada. Primeiro, deve ser criado o tipo de cada parâmetro de QoS a fazer parte da categoria de serviço. Nesse caso, o parâmetro de QoS é “percentual médio de uso” (tipo). O trecho de schema abaixo define o tipo concreto *AvergeUseParamType*, derivado do abstrato *pan:ParamType*. Nota-se que a especialização deve ser por restrição.

```
<xs:complexType name="AverageUseParamType">
   <xs:complexContent>
      <xs:restriction base="pan:ParamType">
         <xs:attribute name="value" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:positiveInteger">
                  <xs:maxInclusive value="100"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="unit" use="optional">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:pattern value="%"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:restriction>
   </xs:complexContent>
</xs:complexType>
```

Com o tipo do parâmetro definido, deve-se criar o elemento que substitui `pan:servicecategoryparam`, em dois passos. Primeiro, um elemento ainda abstrato (`softrtparam`) para aparecer como único na lista de parâmetros da categoria a ser criada. Depois, um elemento concreto do tipo *AverageUseParamType*, denominado `avguse`, habilitado para substituir `softrtparam`. Essa forma de especificação dos elementos de parâmetros em dois passos permite que a validação seja mais restrita, aceitando apenas parâmetros que substituam `softrtparam` e não o elemento mais genérico `servicecategoryparam`.

```
<xs:element name="softrtparam" abstract="true"
            substitutionGroup="pan:servicecategoryparam"
            type="pan:ParamType"/>

<xs:element name="avguse" substitutionGroup="softrtparam"
            type="AverageUseParamType"/>

<!-- Outros parâmetros de outros tipos aqui, análogos a avguse -->
```

Finalmente, pode-se criar o tipo *SoftRTServiceCategoryType*, herdando de *pan:ServiceCategoryType*, modificando apenas o nome do elemento que deve ser substituído. Nota-se, novamente, que a especialização deve ser por restrição:

```
<xs:complexType name="SoftRTServiceCategory" block="extension">
   <xs:complexContent>
      <xs:restriction base="pan:ServiceCategoryType">
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="softrtparam" maxOccurs="unbounded"/>
         </xs:choice>
      </xs:restriction>
   </xs:complexContent>
</xs:complexType>
```

Se o *namespace* da nova categoria for adicionado aos *namespaces* disponíveis em um documento Pan, o novo elemento e o novo tipo criados poderão ser usados com garantias de validação. É interessante mencionar que os tipos criados para parâmetros não precisam estar, necessariamente, no mesmo *namespace* da categoria de serviço. Observa-se, também, que todos os parâmetros de QoS criados serão uniformes, constituídos da tripla nome/valor/unidade, por serem obrigatoriamente derivados de *pan:ParamType*. Isso é muito importante no momento da implantação do modelo, para um tratamento também uniforme desses parâmetros pelas plataformas. Para concluir o exemplo, o trecho de código Pan abaixo faz uso da categoria de serviço criada e de seu parâmetro.

```
<leafvr id="app2process1">
   <reservation xsi:type="ssc:SoftRTServiceCategory">
      <ssc:avguse value="30" unit="%"/>
      <!-- Outros parâmetros da categoria poderiam vir aqui -->
   </reservation>
</leafvr>
```

Contudo, torna-se evidente que a tarefa de especificação de uma categoria de serviço não é trivial. Somente programadores especialistas em *XML Schema* conseguiriam formular os novos tipos de categorias rapidamente. Porém, é muito simples a construção de ferramentas, com uma interface simples de formulário, para implementar categorias de serviço. Além disso, os atores mais leigos em MDRM agirão muito mais na manutenção de modelos, e se precisarem usar categorias de serviços, normalmente vão buscá-las em repositórios onde já se encontram prontas.

4.2.6. Construindo regras de controle de acesso

Conforme mencionado na Subseção 4.2.1, regras de controle de acesso também são definidas por Pan como elementos extensíveis da linguagem. Isso porque não há como prever todos os tipos de regras possíveis de serem usadas por controladores de acesso.

Reiterando, o termo “estender a linguagem Pan”, deve ser entendido como disponibilizar novos elementos à linguagem, desde que eles possam ser validados, mantendo controle sobre o formato dos novos elementos.

A forma de criar regras de controle de acesso é muito similar àquela usada para criar os parâmetros de QoS, descrita na subseção anterior. Uma regra de controle de acesso é formada sempre por uma tripla nome/operação-de-teste/valor. As regras podem ser compostas no momento de uso nos modelos, sendo colocadas como filhas de elementos de conjunção `<all>` e de disjunção `<any>`.

Tirando proveito de toda a introdução a conteúdo variável em *XML Schema* feita na subseção anterior, parte-se para a análise dos elementos e tipos abstratos definidos no *schema* de Pan, cujo trecho relevante encontra-se listado abaixo:

```
<!-- RuleOperatorType - Concrete type for valid rule matching operations -->
   <xs:simpleType name="RuleOperatorType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="eq"/> <!-- equal to                  -->
         ...
         <xs:enumeration value="rx"/> <!-- regular expression match -->
      </xs:restriction>
   </xs:simpleType>

<!-- RuleType - Abstract type for a single generic rule -->
   <xs:complexType abstract="true" name="RuleType" block="extension">
      <xs:attribute name="op" type="RuleOperatorType" default="eq"/>
      <xs:attribute name="value" type="xs:anySimpleType"/>
   </xs:complexType>

<!-- accessctrlrule - Abstract element to be replaced by specialized
     accessctrlrule elements -->
   <xs:element abstract="true" name="accessctrlrule" type="RuleType"
               block="extension"/>
```

Cada regra de controle de acesso a aparecer em um documento Pan deve ser uma especialização do elemento abstrato `pan:accessctrlrule`. Esse, por sua vez, é instância do tipo abstrato *pan:RuleType*, formado por um atributo que define o operador de comparação (*pan:RuleOperatorType*) e um outro que define um valor, de qualquer tipo simples.

Em resumo, o que deve ser feito por um *schema* que descreve uma ou mais regras de controle de acesso é i) especializar o tipo *pan:RuleType* para tipos concretos X que definem regras de controle de acesso no formato nome/operador-de-teste/valor; e ii) definir elementos instanciados dos tipos X que substituam `pan:accessctrlrule`.

Como exemplo, sejam *PermitUserRule* e *PermitTimeRule* duas regras de controle de acesso a serem criadas. Primeiro, devem ser criados os tipos de cada regra de controle de acesso. No primeiro caso, o nome da regra é `permituser`, voltada para liberar um certo usuário a realizar alguma tarefa de manutenção. A segunda regra, `permittime`, diz respeito aos horários permitidos para a execução de alguma tarefa de manutenção. O trecho de *schema* abaixo define os tipos

concretos *PermitUserRule* e *PermitTimeRule*, derivados do tipo abstrato *pan:RuleType*. Nota-se a diferenciação do conteúdo do atributo `value` pelas expressões regulares usadas para validar um nome bem formado de usuário e de hora. Observa-se, também, que a especialização deve ser por restrição.

```
<xs:complexType name="PermitUserRule">
        <xs:complexContent>
            <xs:restriction base="pan:RuleType">
                <xs:attribute name="value">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="([a-z]|[A-Z])([a-z]|[A-Z]|[0-9]|
                                                              [_])*"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="PermitTimeRule">
        <xs:complexContent>
            <xs:restriction base="pan:RuleType">
                <xs:attribute name="value">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="([0-1][0-9]|2[0-3])\:[0-5][0-9]-
                                               ([0-1][0-9]|2[0-3])\:[0-5][0-9]"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
```

Com os tipos das regras definidos, deve-se então criar os elementos que substituem `pan:accessctrlrule`. No exemplo, basta criar um elemento concreto do tipo *PermitUserRule*, denominado `permituser`, e outro elemento concreto, do tipo *PermitTimeRule*, denominado `permittime`.

```
<xs:element name="permituser" substitutionGroup="pan:accessctrlrule"
            type="PermitUserRule"/>
<xs:element name="permittime" substitutionGroup="pan:accessctrlrule"
            type="PermitTimeRule"/>
```

Se o *namespace* das novas regras de controle de acesso for adicionado aos *namespaces* disponíveis em um documento Pan, os novos elementos poderão ser usados com garantias de validação. Observa-se, novamente, que todas as regras de controle de acesso criadas serão uniformes, constituídas da tripla nome/operador-de-teste/valor, por serem obrigatoriamente derivados de *pan:RuleType*. Para concluir o exemplo, o trecho de código Pan abaixo faz uso das regras de controle de acesso criadas.

```
<accessctrlrules>
    <adaptrules id="adapt1">
        <any>
            <all>
                <acr1:permituser op="eq" value="mfm03"/>
                <acr1:permittime op="eq" value="19:59-01:00"/>
            </all>
        </any>
    </adaptrules>
</accessctrlrules>
```

Assim como ocorre para categorias de serviço, mesmo sendo menos complexa, a tarefa de especificação de uma regra de controle de acesso não é trivial. Pode-se, no entanto, contar com ferramentas auxiliares e repositórios de regras para facilitarem o processo.

4.2.7. Construindo regras de classificação

Conforme mencionado na Subseção 4.2.1, regras de classificação definem as informações que determinam qual fluxo deve ser escalonado em um recurso virtual folha. Da mesma forma que ocorre para regras de controle de acesso, Pan define regras de classificação como elementos extensíveis da linguagem. Claramente, isso se deve ao fato de que não há como prever todos os tipos de regras possíveis de serem usadas por classificadores.

Reiterando, o termo "estender a linguagem Pan", deve ser entendido como disponibilizar novos elementos à linguagem, desde que eles possam ser validados, mantendo controle sobre o formato dos novos elementos.

A forma de criar regras de classificação é quase idêntica à usada para criar regras de controle de acesso. Uma regra de classificação também é formada pela tripla nome/operação-de-teste/valor e pode ser composta no momento de seu uso nos modelos, como filhas de elementos de conjunção `<all>` e de disjunção `<any>`.

A única parte diferenciável na construção de regras de classificação em relação a regras de controle de acesso está no fato de que, dessa vez, o elemento abstrato a ser substituído é pan:classifrule. Pode-se notar, entretanto, pelo trecho do *schema* XML de Pan listado abaixo, que o tipo abstrato é o mesmo: *pan:RuleType*.

```
<!-- RuleOperatorType - Concrete type for valid rule matching operations -->
   <xs:simpleType name="RuleOperatorType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="eq"/> <!-- equal to                -->
         ...
         <xs:enumeration value="rx"/> <!-- regular expression match -->
      </xs:restriction>
   </xs:simpleType>

<!-- RuleType - Abstract type for a single generic rule -->
   <xs:complexType abstract="true" name="RuleType" block="extension">
      <xs:attribute name="op" type="RuleOperatorType" default="eq"/>
      <xs:attribute name="value" type="xs:anySimpleType"/>
   </xs:complexType>

<!-- classifrule - Abstract element to be replaced by specialized
                   classifrule elements -->
   <xs:element abstract="true" name="classifrule" type="RuleType"
               block="extension"/>
```

Assim, cada regra de classificação a aparecer em um documento Pan deve ser uma especialização do elemento abstrato `pan:classifrule`. Esse, por sua vez, é instância do tipo abstrato *pan:RuleType*. Em um *schema* que descreve uma ou mais regras de classificação, deve-se i) especializar o tipo *pan:RuleType* para tipos concretos X que definem regras de classificação no formato nome/operador-de-teste/valor; e ii) definir elementos instanciados dos tipos X que substituam `pan:classifrule`.

Como exemplo, sejam *TOSRule, SourceddrRule e DestAddrRule* regras de classificação de pacotes de rede a serem criadas. Primeiro, devem ser criados os tipos de cada regra de classificação. No primeiro caso, o nome da regra é `tos`, a ser verificada contra o campo TOS dos cabeçalhos de pacotes IP. A segunda regra, `sourceaddr`, é o endereço IP de origem também presente no cabeçalho. A terceira regra, `destaddr`, diz respeito ao endereço IP de destino. O trecho de *schema* abaixo define os tipos concretos *TOSRule*, *SourceAddrRule* e *DestAddrRule*, derivados do tipo abstrato *pan:RuleType*.

```
<xs:complexType name="TOSRule">
        <xs:complexContent>
            <xs:restriction base="pan:RuleType">
                <xs:attribute name="value">
                    <xs:simpleType>
                        <xs:restriction base="xs:unsignedByte"/>
                    </xs:simpleType>
                </xs:attribute>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>

<xs:complexType name="SourceAddrRule">
        <xs:complexContent>
            <xs:restriction base="pan:RuleType">
                <xs:attribute name="value">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="(([1-9]?[0-9]|1[0-9][0-9]|
                                                2[0-4][0-9]|25[0-5])\.){3}
```

```
                                        ([1-9]?[0-9]|1[0-9][0-9]|
                                         2[0-4][0-9]|25[0-5])"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="DestAddrRule">
    <xs:complexContent>
        <xs:restriction base="pan:RuleType">
            <xs:attribute name="value">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:pattern value="(([1-9]?[0-9]|1[0-9][0-9]|
                                             2[0-4][0-9]|25[0-5])\.){3}
                                        ([1-9]?[0-9]|1[0-9][0-9]|
                                         2[0-4][0-9]|25[0-5])"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

Finalmente, deve-se criar os elementos que substituem `pan:classifrule`. No exemplo, basta então criar um elemento concreto do tipo *TOSRule*, denominado `tos`, um outro, do tipo *SourceAddrRule*, denominado `sourceaddr`, e um último, do tipo *DestAddrRule* denominado `destaddr`.

```
<xs:element name="tos" substitutionGroup="pan:classifrule" type="TOSRule"/>

<xs:element name="sourceaddr" substitutionGroup="pan:classifrule"
            type="SourceAddrRule"/>

<xs:element name="destaddr" substitutionGroup="pan:classifrule"
            type="DestAddrRule"/>
```

Se o *namespace* das novas regras de classificação for adicionado aos *namespaces* disponíveis em um documento Pan, os novos elementos poderão ser usados com garantias de validação. Para concluir o exemplo, o trecho de código Pan abaixo faz uso das regras de classificação criadas.

```
<classifrules>
   <all>
      <any>
         <ip:destaddr op="eq" value="139.82.95.11"/>
         <ip:tos value="3"/>
      </any>
      <ip:sourceaddr op="eq" value="139.82.35.135"/>
   </all>
</classifrules>
```

A tarefa de especificação de uma regra de classificação é tão árdua quanto a criação de regras de controle de acesso. Novamente, ferramentas auxiliares e repositórios de regras podem, contudo, facilitar muito o processo.