6
O Framework VRT-FS

O meta modelo VRT pode ser instanciado em diversas plataformas e em diversos níveis de abstração. Modelos Pan que envolvem apenas recursos virtuais compostos e que não lidam com os aspectos de escalonamento de recursos reais podem ser implantados em serviços de diretório, serviços Web, componentes de *middleware*, etc. A plataforma tomada como prova de conceito para o meta modelo VRT, contudo, foi escolhida por tratar tanto recursos reais como recursos compostos: o sistema operacional.

A instanciação do meta modelo VRT em sistemas operacionais demanda a criação de uma estrutura de dados complexa e flexível o suficiente para acomodar adaptações sobre estratégias de gerenciamento. Porém, implementações leves devem ser factíveis, já que orquestrações de recursos podem envolver sistemas computacionais com poder de processamento reduzido. Ao mesmo tempo, entretanto, não é desejável limitar as características e vantagens do meta modelo VRT.

Dentre as soluções investigadas, o uso de sistemas de arquivos especiais como estrutura de dados e interface para o gerenciamento de recursos mostrou-se uma solução de projeto (Bruno, 1999; Nagar, 2004) muito interessante. As principais vantagens dessa abordagem são:

1. Sistema de arquivos é uma abstração inerentemente hierárquica, tal qual o conceito de VRTs, e já inclui convenções de nomes e de formação de caminhos prontas para uso. Além disso, os elementos que compõem sistemas de arquivos denotam entidades comuns a diversos modelos de dados, tais como elementos individuais (arquivos), conjuntos (diretórios), composições (diretórios e subdiretórios) e reuso (elos – *links*).
2. Sistemas de arquivos especiais podem representar algo além de um mecanismo para armazenamento de arquivos. Podem se tornar ainda mais atrativos ao oferecerem uma interface que permita ao usuário a modificação do comportamento do sistema através de arquivos que se

alimentam de parâmetros de configuração, ou mesmo trechos de código objeto dinamicamente carregáveis pelo sistema.

3. Operações sobre sistemas de arquivos formam uma interface padronizada, compreendendo funções como *open*, *close*, *read*, *write*, *chdir* e *mkdir*. Elas podem ser facilmente mapeadas nas operações de gerenciamento de recursos virtuais descritas na seção anterior. Isso evitará a criação de novas APIs e chamadas de sistema.
4. A abstração de sistema de arquivos pode ser encontrada em qualquer sistema operacional, até mesmo em implementações leves e embutidas, com a mesma semântica. Poucos são os sistemas operacionais que não possuem uma API para desenvolvimento de sistemas de arquivos suficientemente genérica para suportar a criação de sistemas de arquivos especiais.
5. As transformações de código Pan para chamadas do interpretador de comandos (*shell*) para manipulação de arquivos podem ser extremamente simplificadas, se todas as operações de manutenção de modelos forem mapeadas para o sistema de arquivos. Dessa forma, simples transformações XSLT podem ser o bastante para geração de código de gerenciamento de recursos.

A instanciação do modelo VRT baseada em um sistema de arquivos especial é denominada VRT-FS e sua especificação deve ser vista como um *framework*. De fato, VRT-FS descreve a estrutura de arquivos, diretórios e elos, e o comportamento esperado do sistema frente às interações com tais elementos. No entanto, detalhes sobre como a estrutura de arquivos deve ser integrada internamente aos subsistemas de gerenciamento de cada recurso são deixados em aberto, dado que são idiossincrasias de cada sistema.

A Figura 16 apresenta a estrutura de diretórios de primeiro nível de VRT-FS. A raiz de VRT-FS é o diretório `/vrt`, sob o qual encontram-se outros três diretórios: `/vrt/primitive`, onde são representadas VRTs primitivas; `/vrt/forest`, que abriga florestas; e `/vrt/composite`, onde VRTs compostas são mantidas.

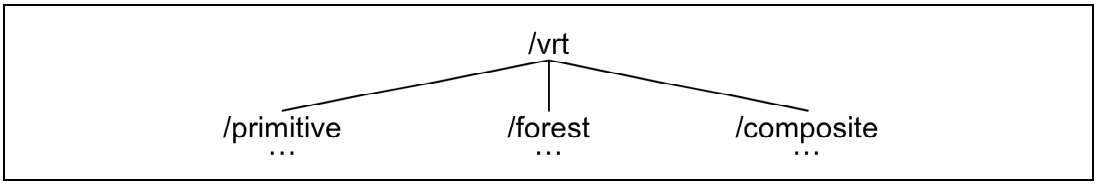

Figura 16 – Estrutura de primeiro nível de VRT-FS

6.1. VRTs Primitivas

As VRTs primitivas relacionadas a recursos reais gerenciados pelo sistema local são representadas sob o diretório `/vrt/primitive`, ilustrado pela Figura 17 em uma estruturação típica. Devido ao fato de que recursos reais podem ser escalonados tanto no tempo quanto no espaço, ele possui dois subdiretórios: `./temporal` e `./spatial`. Ambos possuem a mesma estrutura: seus diretórios filhos representam exatamente os recursos virtuais raízes das VRTs de cada recurso gerenciado pelo sistema e devem ser criados automaticamente durante a inicialização de VRT-FS.

O recurso virtual raiz de uma VRT primitiva é representado pelo diretório `./[dimensão]/nome_do_recursoX`, onde `X` é um inteiro usado somente quando é necessário diferenciar nomes de recursos de um mesmo tipo. Dessa forma, na Figura 17, `./spatial/mem` é o subdiretório que representa a raiz da VRT que gerencia o espaço alocado em memória principal; `./spatial/hd1` é a raiz da VRT para o espaço no primeiro disco; `./temporal/nic1` é a raiz da VRT de uma interface de rede; `./temporal/cpu` é a raiz da VRT da(s) CPU(s); e `./temporal/hd1` é a raiz da VRT da fila de acesso ao primeiro disco.

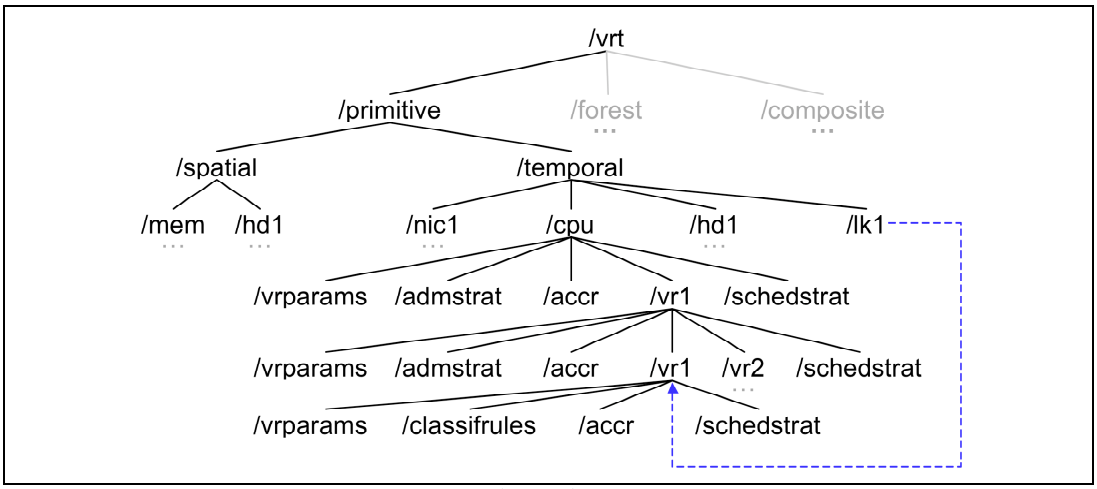

Figura 17 – Estrutura típica do diretório `/vrt/primitive`

Ainda na Figura 17, o foco é dado a uma CPU VRT, sendo sua estrutura expandida como exemplo e para acompanhamento das próximas definições. Todos os subdiretórios sob uma raiz de VRT representam recursos virtuais filhos dessa raiz. Com a representação de recursos virtuais por meio de diretórios, a operação *Traverse* de recursos virtuais é trivialmente mapeada para instruções `chdir` (`cd`). Na figura, `./vr1`, `./vr1/vr1` e `./vr1/vr2` são exemplos de diretórios de recursos virtuais filhos da raiz `cpu`.

Como mencionado no Capítulo 3, um recurso virtual pode ser destacado de uma VRT para facilitar a gerência, como quando uma parcela de CPU (e sua administração) é atribuída a um usuário do sistema operacional. VRT-FS lança mão da abstração de elo para representar uma raiz de VRT originada a partir de um recurso virtual. Elos abaixo de `/vrt/primitive/*` podem apontar apenas para diretórios de recursos virtuais e podem ter qualquer nome. Na Figura 17, `/vrt/primitive/temporal/lk1` exemplifica a promoção do recurso virtual `./vr1/vr1` a raiz. O uso de elo para uma raiz promovida permite que as operações sobre a nova VRT sejam refletidas na VRT onde se localiza o recurso virtual de origem, porém de forma transparente, já que o caminho enxergado na gerência da nova VRT passa pelo elo.

Todo diretório de recurso virtual possui um arquivo chamado `vrparams`, que se dispõe a ser uma interface para a manutenção da alocação concedida ao recurso virtual. A instrução de leitura (`read`) sobre `vrparams` retorna a capacidade alocada ao recurso virtual representado pelo diretório que o contém. A instrução de escrita (`write`) altera a capacidade alocada, o que demanda um novo processo de admissão, detalhado mais a frente. Note que `write` sobre `vrparams` é permitida apenas em diretórios não-raízes, uma vez que raízes de VRTs representam a alocação total de um recurso ou grupo de recursos. E quando se tratar de uma raiz representada por um elo, essa permissão é naturalmente herdada do diretório apontado, dependendo, então, se ele é raiz ou não.

Diretórios de recursos virtuais podem possuir outros cinco arquivos, que representam as interfaces para manutenção de cada uma das estratégias de gerenciamento associadas ao recurso virtual: `schedstrat` - estratégia de escalonamento; `admstrat` - estratégia de admissão; e `accr` - regras de controle de acesso.

O conteúdo desses arquivos depende da forma pela qual a instanciação do modelo implementa tais estratégias de gerenciamento nos sistemas operacionais envolvidos. Algumas possibilidades são: descrições de regras (Kotsovinos, 2006), parâmetros de algoritmos dinâmicos (Barria, 2001) e código de programa dinamicamente carregável, como módulos de núcleo (Salzman, 2007), serviços de micronúcleo e componentes de software. A instrução `read` sobre esses arquivos retorna o conteúdo do arquivo carregado, ao passo que a instrução `write` dispara a inclusão (ou substituição) da nova estratégia de gerenciamento no sistema (operação *Adapt*). Recursos virtuais não-folhas possuem os arquivos `schedstrat`, `admstrat` e `accr`.

Recursos virtuais folhas possuem somente os arquivos `schedstrat` e `accr`, uma vez que não possuem um controlador de admissão. Por serem os nós que especificam as regras de classificação, recursos virtuais folha contam ainda com o arquivo `classifrules`., sobre o qual a operação `read` retorna a lista de regras e a operação `write` atualiza todo o conjunto de regras.

Recursos virtuais são criados (operação *Split*) por meio da instrução `mkdir`, permitida apenas sob diretórios que possuem o arquivo `admstrat`. O nome do novo diretório é livre. O sistema deve disponibilizar automaticamente os arquivos `vrparams` (nulo); `schedstrat` (com alguma estratégia padrão, como FIFO, round robin...); `accr` (com regras de controle de acesso padrão) sob o novo diretório. Note que o processo de admissão ainda não se iniciou, pois ainda não foram passados os parâmetros de QoS desejados para o novo recurso virtual. Por isso, `read` sobre `vrparams` retorna nulo até que o processo de admissão seja bem sucedido. Os parâmetros de QoS são fornecidos como conteúdo de um `write` sobre `vrparams` e devem ser compatíveis com aqueles esperados pela estratégia de admissão do nó pai. Caso contrário, `write` retornará erro, como se a admissão fosse negada. A instrução `write` somente retornará com sucesso se o controlador de admissão aprovar a requisição.

Veja que o novo recurso virtual ainda não está criado, pois o controlador de admissão do nó pai apenas respondeu de forma afirmativa ao pedido de admissão. De fato, o processo de controle de admissão é composto de duas fases: pedido (`admit`) e confirmação (`commit`). Após o retorno bem sucedido de `write`, o arquivo `vrparams` deve ser alvo de novo `write`, desta vez contendo o caractere

único ‘0’ (zero), para que a confirmação seja efetuada. No intervalo entre pedido e confirmação, `read` em `vrparams` continuará retornando conteúdo nulo. A confirmação é permitida apenas dentro de um certo limite de tempo. Se `vrparams` não for reescrito até lá, o diretório do recurso virtual em questão será removido, automaticamente. O processo de admissão baseado em dois passos é muito útil em alguns esquemas de negociação de QoS distribuída, evitando ocorrências de impasses (*deadlocks*), possíveis em reservas concorrentes de recursos (Nahrstedt, 1998).

Por convenção, todo novo recurso virtual é folha, por isso também o arquivo `classifrules` é automaticamente criado junto a criação do recurso. Para que um recurso virtual se habilite a ser pai de outros, basta a criação do arquivo `admstrat` com alguma estratégia de admissão codificada no seu conteúdo (no mínimo, uma estratégia que sempre responda afirmativamente). Nesse caso, o arquivo `classifrules` será removido, automaticamente.

A instrução `write` sobre o arquivo `vrparams` de um recurso virtual já alocado implementa a operação *Tune*. Essa operação se traduz em uma nova admissão, que também segue dois passos, exatamente como a alocação inicial descrita nos parágrafos anteriores. A diferença está no fato de que, entre pedido e confirmação, o recurso virtual continua detentor da parcela alocada anteriormente, que somente será alterada se a nova admissão for bem sucedida. Caso contrário, a alocação anterior permanecerá valendo.

A instrução `rmdir` (operação *Release*) pode ser realizada em diretórios de recursos virtuais folhas. Veja que esses diretórios possuem dois arquivos não removíveis (`vrparams`, `classifrules`, `accr`, e `schedstrat`) e, por isso, a instrução `rmdir` possui uma semântica diferente em VRT-FS: ela é permitida sobre diretórios que na prática não estão vazios, por possuírem esses, e somente esses, arquivos. Por outro lado, para liberar um recurso virtual intermediário (que não uma raiz, por não ser permitido), todos os seus diretórios filhos devem ser previamente removidos, assim como seu arquivo `admstrat`. Esse seria o processo para transformar um nó intermediário em uma folha.

6.2. Florestas

As florestas administradas a partir do sistema local são representadas sob o diretório `/vrt/forest`, conforme ilustrado pela Figura 18. Dentre essas florestas, podem ser encontradas: (i) a floresta formada pelas VRTs primitivas de recursos locais, ou seja, a floresta do sistema operacional; e (ii) qualquer outro tipo de floresta, na qual toda requisição de orquestração deve ser encaminhada a este sistema local, por ser ele o administrador. Esse cenário pode ser exemplificado pelos *resource brokers* (Nahrstedt, 1998), comuns em protocolos de gerenciamento centralizado de recursos.

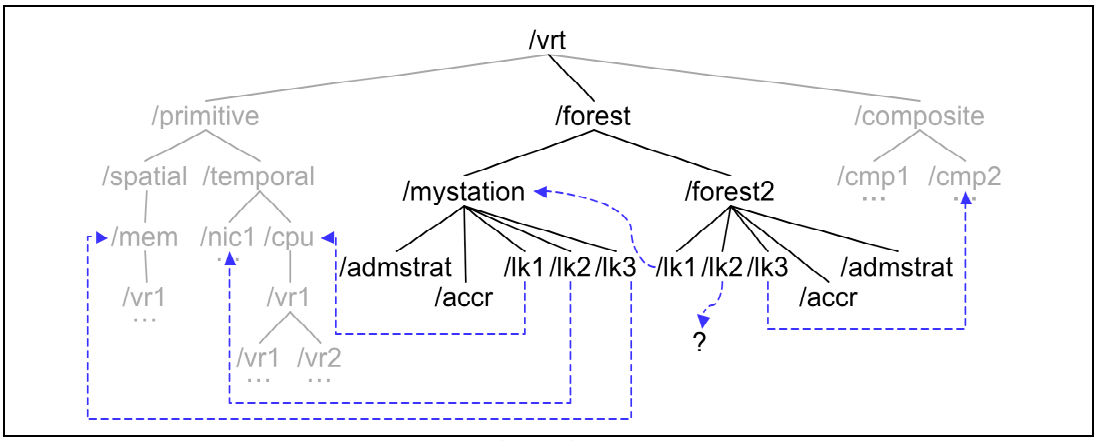

Figura 18 – Estrutura típica do diretório `/vrt/forest`

O diretório da floresta do sistema operacional é automaticamente criado durante a inicialização de VRT-FS, e possui estrutura semelhante a `/vrt/forest/mystation` da Figura 18. Outras florestas podem ser construídas pelo administrador do sistema por meio da instrução `mkdir` sob `/vrt/forest`, criando um diretório cujo nome é livre e que representa a floresta em si. A instrução `mkdir` automaticamente cria os arquivos `admstrat` e `accr` com estratégias de gerenciamento padrão (como aquelas que sempre respondem afirmativamente), que são associadas aos controladores de admissão e de acesso da floresta, devendo ser personalizadas em seguida. Esses são os únicos arquivos presentes em um diretório de floresta e possuem a mesma semântica para leitura e escrita que seus homônimos em diretórios de recursos virtuais em VRTs primitivas.

Para agrupar as VRTs primitivas, VRTs compostas e outras florestas pertencentes à nova floresta, a abstração de elos é novamente utilizada, agora com

o objetivo de criar ponteiros para os diretórios de VRTs e demais florestas participantes. Nomes para esses elos também são de livre escolha. Na Figura 18, os elos estão representados por nomes arbitrários `lkX` e cada um deles tem seu alvo indicado por uma linha tracejada. É importante enfatizar que elos em um diretório de floresta somente apontam para diretórios de raízes de VRTs ou diretórios de outras florestas.

Nota-se que VRTs e florestas administradas por sistemas remotos podem fazer parte de uma floresta administrada pelo sistema local. Para isso, um protocolo de sistema de arquivos distribuído é necessário. Na Figura 18, o elo `/vrt/forest/frt2/lk2` representa um ponteiro para uma VRT remota. Com esse esquema, através do próprio sistema de arquivos, é possível a identificação dos recursos de uma floresta disponíveis para possíveis orquestrações. Outro ponto relevante está no fato de que apenas o diretório `/vrt/forest` precisa estar compartilhado pelo sistema de arquivos distribuído para que as orquestrações e conseqüentes criações de VRTs compostas, descritas a seguir, sejam possíveis. Essa solução é ótima para gerenciamento de *clusters*, porém, por questões de desempenho e escala, seria desaconselhável para sistemas distribuídos fracamente acoplados.

6.3. VRTs Compostas

Uma VRT composta é formada a partir da orquestração e reserva de recursos em outras VRTs, primitivas ou compostas, de uma mesma floresta. A orquestração de recursos não é um processo disparado a partir de alguma interação com VRT-FS, mas sim, por um plano de negociação de QoS que utiliza negociadores e controladores de admissão de alto nível conforme descrito no Capítulo 3. Porém, as estratégias de gerenciamento associadas às florestas e VRTs orquestradas devem ser buscadas em VRT-FS. Essa abordagem, que separa os mecanismos de alocação e escalonamento dos mecanismos de negociação, tem a vantagem de permitir que diferentes protocolos de negociação sejam adotados, conforme o legado e as necessidades característicos de cada floresta.

VRTs compostas são criadas sob o diretório `/vrt/composite` sempre que orquestrações sobre florestas administradas pelo sistema local são requisitadas e,

claro, bem sucedidas. Ao contrário das VRTs primitivas, que têm suas raízes automaticamente criadas durante a iniciação do sistema, VRTs compostas são criadas sob demanda, após o sucesso de uma orquestração. A Figura 19 ilustra exemplos desses diretórios.

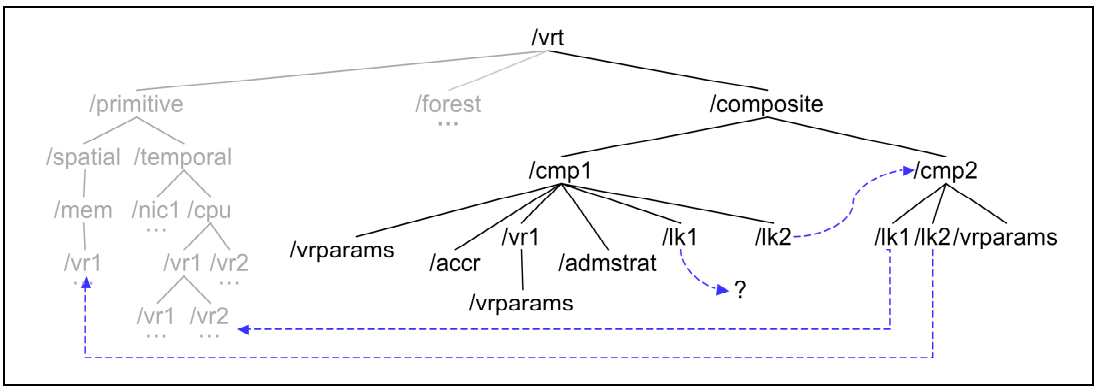

Figura 19 – Estrutura típica de do diretório `/vrt/composite`

A instrução `mkdir` sob `/vrt/composite` cria um diretório de nome livre que representará a raiz de uma nova VRT composta, e somente pode ser realizada por um processo de orquestração ou de gerência manual. Assim como em VRTs primitivas, convenciona-se que novos diretórios são sempre folhas. O novo diretório é disponibilizado apenas com o arquivo `vrparams`, porque o arquivo `schedstrat` pode não ser necessário, já que a grande maioria das árvores compostas não lidam com aspectos de escalonamento. De qualquer forma, as instruções `read` e `write` possuem a mesma semântica que apresentada para os mesmos arquivos em VRTs primitivas.

Durante a orquestração de recursos que originou a nova VRT composta, cada VRT participante teve um recurso virtual folha criado e dedicado à composição. Para representar esses recursos virtuais sob o diretório da VRT composta, elos são novamente utilizados. Os diretórios apontados devem ser sempre folhas de VRTs, e não podem se tornar intermediários. Por isso, é necessário um esquema para verificar, antes de uma operação *Split*, se um recurso virtual participa ou não de uma composição.

Apesar de ter sido ilustrado na Figura 19 por uma linha tracejada direta ao alvo, o caminho apontado por um elo que compõe uma VRT composta passa, necessariamente, por `/vrt/forest`, de forma a indicar a floresta orquestrada. Por sinal, endereçar um recurso virtual de uma VRT localizada remotamente somente é possível por meio desse diretório. Por exemplo, o caminho apontado

por `./cmp2/lk2` deve ser `/vrt/forest/mystation/lk1/vr1/vr2`, considerando a estrutura apresentada na Figura 18.

Para cada nível de abstração envolvido na negociação, uma VRT composta deve ser criada e representada em VRT-FS. VRTs compostas criadas em um processo de orquestração para criação de uma VRT de nível de abstração superior possuem apenas o nó raiz. Note que na Figura 19 o diretório `./cmp2` representa a raiz de um VRT composta desse tipo. Uma instrução `mkdir` sob `./cmp2` retornaria erro, já que ele é uma folha e não possui um controlador de admissão associado. Por sua vez, `./cmp1` é a raiz de uma VRT composta que possui uma estratégia de admissão e, por isso, foi possível a criação de um recurso virtual filho (`./cmp1/vr1`). Os arquivos de estratégias de gerenciamento são exatamente os mesmos que em VRTs primitivas: admstrat, accr, e, schedstrat. As instruções `read` e `write` sobre cada um desses arquivos têm a mesma semântica que seus pares em VRTs primitivas.

6.4. VRT-FS Linux

O trabalho de implementação de VRT-FS deve ser antecedido por uma avaliação de como seus aspectos fundamentais, principalmente (i) sua solução baseada em sistemas de arquivos especiais, (ii) sua capacidade de adaptação e (iii) seu relacionamento com recursos reais podem ser atingidos no sistema operacional alvo. Nota-se que existem diferentes alternativas para implementar VRT-FS sobre um mesmo sistema operacional, uma vez que pode-se optar por modificações mais ou menos intrusivas, do ponto de vista do sistema. Isso depende diretamente de como os subsistemas de gerenciamento de cada recurso serão reimplementados seguindo o modelo VRT. VRT-FS é capaz de interagir com escalonadores e controladores de admissão que podem estar definidos no núcleo do sistema ou em processos no espaço do usuário.

Os requisitos de VRT-FS merecem especial atenção quando se trata das VRTs primitivas, que por lidarem diretamente com as filas de acesso de cada um dos recursos de interesse, precisam ser eficientes. De fato, o maior desafio em uma implementação de VRT-FS é adotar uma solução de projeto capaz de integrar

ao sistema operacional as funcionalidades de VRTs primitivas e o suporte a adaptabilidade, com um bom desempenho.

Como mencionado no Capítulo 2, o meta modelo VRT é uma evolução do modelo de gerenciamento de recursos definido pela arquitetura QoSOS, prototipada naquela época sobre um núcleo Linux, dando origem ao projeto QoSOSLinux (Moreno, 2004). Naturalmente, como continuidade ao trabalho, VRT-FS também foi prototipado em Linux, sob o codinome VRT-FS Linux.

O núcleo Linux (Love, 2005) possui características interessantes que vão ao encontro dos requisitos de VRT-FS. Primeiro, sua licença é de software livre, o que significa que seu código está disponível e pode ser modificado a contento. Segundo, sua API para o desenvolvimento de sistemas de arquivos permite uma rápida geração de *drivers* de novos sistemas de arquivos especiais, sem intervenção sobre o código do núcleo. Terceiro, seu núcleo monolítico é extensível, o que lhe confere a capacidade de receber ou se desfazer de partes de código (módulos de núcleo) dinamicamente em tempo de execução. Isso permite que, em VRT-FS, o driver do sistema de arquivos e todas as estratégias de gerenciamento configuráveis em tempo de execução possam ser encapsuladas em módulos de núcleo, pelo menos para as VRTs primitivas. Finalmente, as estruturas de dados internas ao núcleo que tratam o conteúdo de sistemas de arquivos são facilmente navegáveis e extensíveis por outras partes do próprio núcleo. Por todos esses atributos, o protótipo VRT-FS Linux está codificado no espaço do núcleo, o que levou a pequenas modificações no código original, com o uso massivo de módulos de núcleo. A Figura 20 apresenta, em um diagrama de blocos, a organização de VRT-FS Linux.

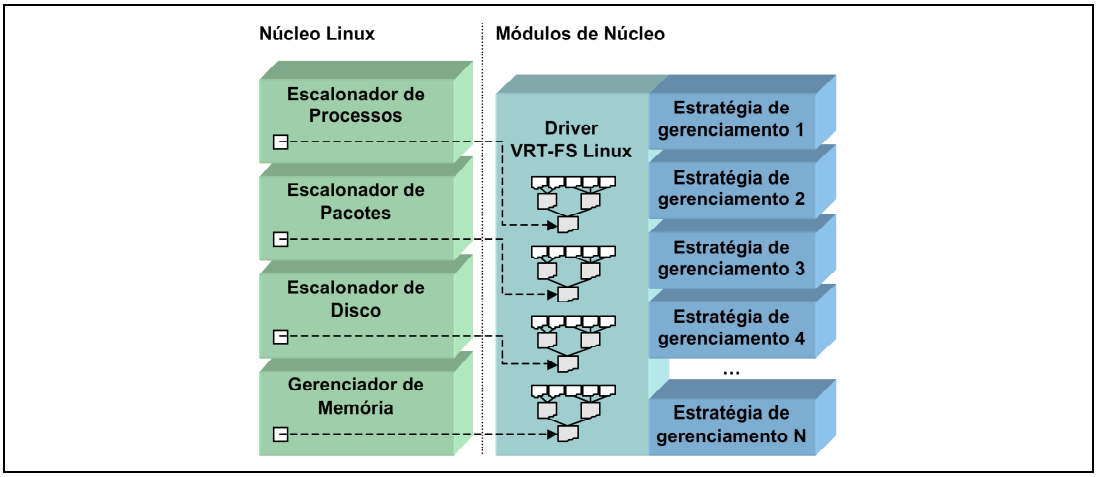

Figura 20 – Arquitetura de VRT-FS em Linux

Ao ser carregado o *driver* VRT-FS e montado o sistema de arquivos, cada escalonador de recurso do núcleo Linux que será gerenciado por VRTs deve ser desviado para o escalonador da raiz da VRT do recurso. Note que, para isso, ganchos (*hooks*) devem ser incluídos no código do núcleo para permitir tais desvios. Alguns subsistemas do núcleo Linux possuem ganchos predefinidos (e.g. pilha de protocolos), exatamente para a captura de algum processamento interno, possibilitando seu monitoramento ou mesmo a mudança de seu comportamento. Porém, esse não é o caso dos escalonadores de recursos, e por isso uma intervenção direta no código do núcleo é necessária. A programação de ganchos nos escalonadores é a única modificação obrigatória no código do núcleo para a implementação de VRT-FS Linux.

Após o desvio, o escalonamento hierárquico do modelo VRT toma o controle. Cada estratégia de escalonamento é encapsulada em um módulo de núcleo, podendo ser reusada por diversos recursos virtuais, na mesma ou em outras VRTs. De fato, toda estratégia de gerenciamento deve poder ser reusada, o que torna necessário um gerenciamento especial de dados locais de estratégias em cada recurso virtual, que será alvo de discussões a seguir. Antes, porém, é necessário conhecer as estruturas internas de sistemas de arquivos especiais em Linux e como, a partir delas, foi possível incluir as operações de VRT-FS.

Resumidamente, um sistema de arquivos em Linux é representado no núcleo por um superbloco, um conjunto de *inodes* e um conjunto de *dentries*. O superbloco descreve características gerais do sistema de arquivos, tais como o tamanho dos blocos de informação utilizados nas instruções e qual seu diretório raiz (representado por um *dentry*). Um *inode* é a estrutura que traz as informações sobre um arquivo ou diretório independentemente das várias instâncias (nomes e localizações) em que eles podem estar representados no sistema de arquivos. Assim, os *inodes* provêem uma indexação planificada do conteúdo dos arquivos e diretórios. Entre as informações por eles armazenadas estão as permissões de acesso, datas de criação/modificação/acesso, endereçamento do conteúdo etc. O encadeamento lógico das entradas de um sistema de arquivos é representado por uma lista de *dentries* (entradas de diretório) cacheadas em memória, trazendo informações sobre qual o *inode* associado, o nome do arquivo/diretório representado, seu caminho inferido a partir do encadeamento etc.

Em sistemas de arquivos especiais, arquivos não são, necessariamente, conteúdos a serem mantidos em algum dispositivo de armazenamento. Eles podem representar alguma informação a ser obtida/fornecida do/pelo núcleo do sistema em tarefas que vão desde o armazenamento simples de dados até ações de monitoramento e gerenciamento do sistema operacional. No caso de VRT-FS Linux, as informações a serem fornecidas/retornadas pelas instruções são mantidas internamente pelo código do *driver*. Por isso, VRT-FS Linux (assim como boa parte dos sistemas de arquivos especiais) utilizam *inodes* para representar um arquivo/diretório, mas não usam o endereçamento de conteúdo a partir dos *inodes*, típico em sistemas de arquivos normais. Assim, os *inodes* são aqui estruturas mantidas em segundo plano, ao contrário dos *dentries* a eles associados, que por serem estruturas flexíveis e que representam em memória a hierarquia de arquivos e diretórios, permitem sua extensão para representar os nós de VRTs (diretórios), suas estratégias de gerenciamento (arquivos) e outras interfaces (arquivo `vrparams`).

Além das informações já citadas, um *dentry* possui, por concepção, um ponteiro genérico para extensões específicas de cada sistema de arquivos (`void *fs_data`). As descrições daqui em diante podem ser acompanhadas pela Figura 21, que apresenta o relacionamento das estruturas existentes no núcleo Linux (na parte inferior da figura) com as estruturas de VRT-FS Linux (na parte superior).

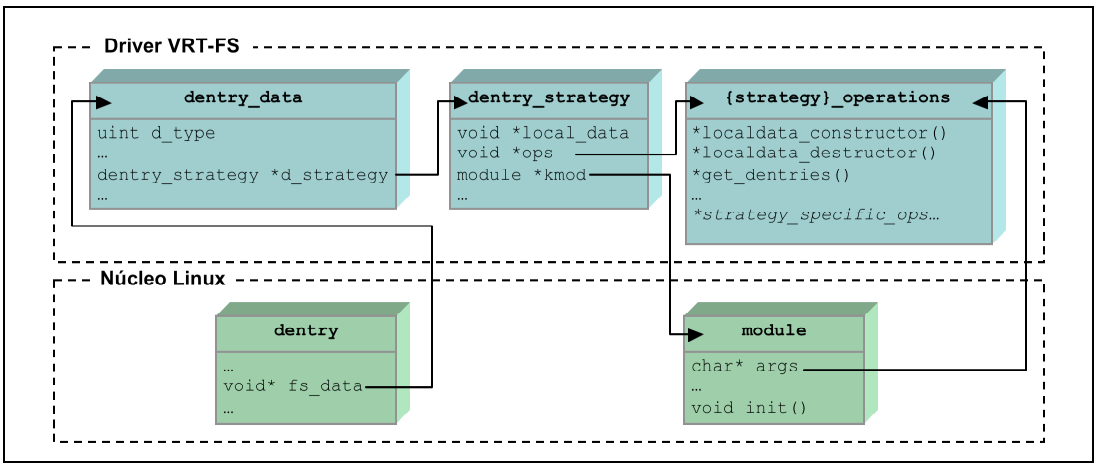

Figura 21 – Relacionamento entre estruturas do núcleo Linux e do *driver* VRT-FS

VRT-FS Linux utiliza o ponteiro `fs_data` de um `dentry` para manter informações complementares em uma nova estrutura, chamada `dentry_data`. Além de informações como o tipo de entrada (recurso virtual, estratégia de

admissão, etc.), ela possui um ponteiro que leva a uma segunda estrutura, chamada `dentry_strategy`, utilizada somente nos casos em que o `dentry` representa um arquivo de estratégia de gerenciamento.

A estrutura `dentry_strategy` armazena um ponteiro para uma área de trabalho (`local_data`) a ser utilizada pela estratégia de gerenciamento para armazenar seu estado de operação referente ao recurso virtual que a contém. Um módulo de núcleo que implementa uma estratégia de escalonamento, por exemplo, precisa manter um conjunto de dados exclusivo, que forma seu estado, para cada recurso virtual ao qual está associado. Permite-se, assim, que um mesmo código inserido via módulo seja reusado em suas diferentes instâncias no sistema de arquivos VRT-FS. Os dois outros ponteiros principais de `dentry_strategy` são responsáveis por manter a ligação entre o `dentry` e o módulo de núcleo carregado onde está a implementação da estratégia de gerenciamento.

O primeiro ponteiro, `kmod`, leva a uma instância da estrutura module, utilizada pelo núcleo para representar um módulo dinamicamente carregado. Para cada `dentry` utilizando uma mesma estratégia de gerenciamento, o contador de referência do respectivo módulo de núcleo deve ser incrementado, caracterizando-o como "em uso". Quando desassociado de um `dentry`, o mesmo contador deve ser decrementado e, uma vez chegando a zero, VRT-FS pode automaticamente remover o módulo por não mais estar em uso. Nota-se que essas são algumas das ações a serem tomadas quando uma instrução `write` sobre um arquivo de estratégia de gerenciamento insere, reinsere ou substitui um módulo. No caso de VRT-FS Linux, o conteúdo dessas instruções `write` são exatamente o código objeto do módulo a ser (re)inserido. A possibilidade de reuso é detectada por VRT-FS quando módulos de mesmo nome interno são fornecidos em sucessivos `writes`. Uma verificação mais apurada poderia aliar ao nome do módulo uma verificação cíclica de redundância (CRC) ou outras funções *hash* sobre seu conteúdo.

Já o segundo ponteiro, `ops`, leva a uma estrutura com referências para as funções que implementam as operações específicas da estratégia de gerenciamento (`schedstrat_operations`, `admstrat_operations` etc.), tais como: inicialização e destruição da área de trabalho (`localdata_{des,cons}tructor()`); retorno de referências de quais

`dentries` utilizam a estratégia (`get_dentries()`); e operações como `schedule()` em estratégias de escalonamento e `check()` em estratégias de admissão. A referência para tal estrutura está disponível inicialmente no ponteiro `args` da estrutura `module`, alocado no momento do carregamento do módulo. Nota-se que a maioria das operações é dependente do contexto do recurso virtual, por ser necessário o uso de dados locais da estratégia ou de informações de localização na árvore. Com o encadeamento de dados tanto de localização quanto de estratégias a partir da estrutura `dentry`, basta que essas funções recebam como parâmetro um ponteiro para o próprio `dentry` que representa a instância da estratégia, para obterem, assim, as informações de contexto.

O carregamento do módulo cujo conteúdo foi fornecido por meio de uma instrução `write` sobre algum arquivo de estratégia de gerenciamento não é tarefa das mais difíceis, uma vez que o núcleo Linux é capaz de disparar a partir de seu espaço de endereçamento a inserção. No entanto, essa solução que reaproveita o carregador de módulos do núcleo não seria ótima: Ao receber o conteúdo do módulo, a operação `write` deveria copiá-lo para um arquivo em um diretório específico, onde o carregador do núcleo sabe buscar módulos automaticamente (e.g.: `/lib/modules/2.6.xxxxx/`); depois, `write` deve chamar a função `request_module()` para ativar o carregador; na realidade, `request_module()` solicita os serviços de um processo do usuário (*modprobe*) para iniciar a inserção de um módulo; *modprobe* faz diversas verificações de dependências e finalmente solicita a inserção do(s) módulo(s) para o núcleo, novamente, por meio da chamada de sistema `init_module()`; a partir daí o módulo é enfim inserido e o resultado salta de volta para o espaço do usuário (*modprobe*), para o espaço do núcleo (`request_module()`) e finalmente para o contexto da operação `write`. O carregador de módulos do núcleo somente atua dessa forma por dois motivos: i) carregar automaticamente todos os módulos interdependentes, graças às recursivas verificações que compõem o programa *modprobe*; ii) livrar o processamento de tais dependências do espaço do núcleo, o que seria custoso e poderia obrigar a manutenção de várias travas enquanto executando.

Em VRT-FS Linux, contudo, os módulos de estratégias de gerenciamento são auto-contidas, ou seja, não possuem dependências externas. O tempo e

recursos gastos com a adaptação podem ser críticos para a QoS das aplicações já admitidas. O uso do carregador do núcleo exigiria um alto custo para a inserção de um código que simplesmente está "nas mãos" da operação `write`, no espaço do núcleo. O Linux não possui, entranto, funções de inserção direta de módulos disponíveis para outras partes do núcleo. Assim, duas alternativas poderiam trazer melhor desempenho em adaptações: `write` poderia copiar o conteúdo do módulo para um arquivo e depois utilizar o programa *insmod*, que praticamente se resume à chamada de sistema `init_module()`; ou o núcleo pode ser modificado para incluir funções de inserção direta e remoção de módulos. Na versão corrente de VRTF-FS Linux, optou-se por implementar a última solução, que exigiu a tradução da chamada de sistema `init_module()` para uma função global do núcleo, retirando a cópia de dados do espaço do usuário e incluindo outras otimizações de código.

Finalizando, nesta seção, o foco concentrou-se na implementação do sistema de arquivos em si e em sua adaptabilidade. Isso permitiu a análise de toda a infra-estrutura disponibilizada e da complexidade das tarefas de adaptação. Utilizando o encadeamento de `dentries`, escalonadores e controladores de admissão podem percorrer VRTs com grande eficiência através dos ponteiros e, ainda, acessar as estratégias de gerenciamento relevantes para suas tomadas de decisão em cada contexto. A análise de desempenho dos escalonadores, entretanto, é deixada para trabalhos futuros, antecipando-se que seus resultados devem se aproximar, individualmente, aos já alcançados por escalonadores hierárquicos heterogêneos propostos na literatura (Goyal, 1996a; Regehr, 2001a).