2
Gerenciamento de Recursos Dirigido por Modelos (MDRM)

A técnica de gerenciamento de recursos dirigido por modelos (MDRM – *model-driven resource management*) é resultado de diversas contribuições provenientes dos trabalhos do Laboratório TeleMídia da PUC-Rio na área de Engenharia de Serviços de Telecomunicações, ao longo dos últimos 10 anos. Para melhor contextualizar as contribuições que serão apresentadas nesta tese, um breve histórico do laboratório, especificamente no tema Qualidade de Serviço (QoS – *quality of service*), é apresentado na Seção 2.1.

MDRM, como resultado de tal evolução, se dedica a complementar arquiteturas de provisão de QoS, oferecendo técnicas avançadas de engenharia de software especificamente voltadas para o desenvolvimento de modelos de gerenciamento de recursos. MDRM tem como objetivos principais i) alcançar a generalidade necessária para incluir recursos heterogêneos distribuídos nos modelos de gerenciamento de recursos; ii) possibilitar que a construção e manutenção dos modelos de gerenciamento de recursos se tornem um trabalho colaborativo entre os diferentes atores presentes no ambiente; iii) garantir a portabilidade e interoperabilidade por meio da independência de plataforma no desenvolvimento dos modelos e iv) permitir a adaptabilidade dos modelos, tanto no projeto quanto depois de instanciados nas plataformas-alvo. Esses objetivos se juntam a diversos outros aspectos de projeto que devem ser considerados em todas as etapas de especificação de MDRM e são discutidos na Seção 2.2.

Para atingir seus objetivos, MDRM adota conceitos e processos de modelagem e geração de código análogos aos definidos pelo paradigma de desenvolvimento de software orientado por modelos (MDSD – *model-driven software development*). A Seção 2.3 descreve rapidamente o paradigma MDSD e a Seção 2.4, finalmente, introduz a abordagem MDRM.

PUC-Rio - Certificação Digital Nº 0321123/CA

2.1. Histórico do Laboratório TeleMídia

O Laboratório TeleMídia vem dedicando-se a desenvolver trabalhos em Qualidade de Serviço desde 1998, ao iniciar estudos para a definição do Modelo de Composição de Serviços (SCM – *service composition model*) (Colcher, 2000). SCM é um modelo conceitual que provê abstrações genéricas para adaptabilidade em serviços de comunicação, compostos por elementos básicos que podem ser organizados de forma aninhada. SCM foi tomado como base para a definição de frameworks orientados a objetos que especificam de forma genérica os mecanismos necessários para provisão de QoS em ambientes de processamento e comunicação (Gomes, 2001).

Com a percepção de que os mesmos mecanismos básicos de provisão de QoS são utilizados em diversos subsistemas que compõem um serviço de comunicação fim-a-fim, os frameworks seguem uma estrutura recorrente, herdada do SCM. Os frameworks estabelecem pontos de flexibilização (*hot spots*) que devem ser especializados para que se adaptem às idiossincrasias do ambiente, topologia ou plataforma. Ao modelar os mecanismos a partir de um único conjunto de frameworks, suas interfaces se tornam homogêneas, provendo um ambiente simplificado de provisão de QoS fim-a-fim. Resumidamente, os frameworks foram divididos em três subconjuntos:

- Framework para parametrização de serviços: modela as estruturas de dados responsáveis por definir um esquema genérico de parâmetros de caracterização de serviços, que podem ser agrupados em categorias de serviço. O framework permite a construção de uma hierarquia de parâmetros capazes de expressar informações sobre o serviço em vários níveis de abstração.
- Frameworks para compartilhamento de recursos: baseiam-se no conceito de recurso virtual para modelar os mecanismos de compartilhamento e de alocação de recursos. Recursos virtuais são parcelas de utilização de um recurso real alocadas entre os fluxos (de dados, de instruções, etc.) submetidos pelos usuários. A primeira abstração de árvores de recursos virtuais foi concebida em meio à definição desses frameworks, definindo um arranjo hierárquico de escalonadores de recursos virtuais. Parte do

PUC-Rio - Certificação Digital Nº 0321123/CA

presente trabalho, o meta modelo VRT (*virtual resource trees*), é uma evolução do conceito de árvores de recursos virtuais.

- Frameworks de orquestração de recursos: modelam os mecanismos que realizam a divisão da responsabilidade de provisão da QoS entre os subsistemas participantes do serviço. A modelagem da orquestração de recursos se subdivide em dois frameworks distintos. O framework para negociação de QoS modela os mecanismos de admissão, negociação e mapeamento que operam durante as fases de solicitação e estabelecimento de serviços. Já o framework para sintonização de QoS modela os mecanismos de monitoração e sintonização que atuam na fase de manutenção dos serviços.

Os frameworks genéricos foram posteriormente especializados para a modelagem da provisão de QoS na Internet (Mota, 2001), em redes móveis infra-estruturadas (Lima, 2003) e em sistemas operacionais (Moreno, 2003a). Particularmente, a arquitetura QoSOS (Moreno, 2003b) estendeu os frameworks genéricos de QoS para descrever mecanismos de QoS adaptáveis para sistemas operacionais, independentes de sua localização (estações finais, roteadores, etc). Entre outras contribuições, QoSOS introduziu de forma preliminar o framework de Adaptação de Serviços, que descreve meta-mecanismos que automatizam adaptações do sistema e, assim, permitem a provisão de novos serviços ou a modificação dos já existentes.

A primeira instanciação da arquitetura QoSOS resultou no projeto QoSOSLinux v.0.1, desenvolvido parcialmente como prova de conceito por Moreno (2004). QoSOSLinux fazia uso de uma estrutura básica de árvores de recursos virtuais, implementada a partir de escalonadores de recursos providos por outros trabalhos (Nahrstedt, 1998; Almesberger, 1999). Alguns mecanismos ausentes nesses escalonadores foram desenvolvidos para complementá-los e, assim, possibilitar uma validação na prática dos frameworks e do próprio conceito de árvore de recursos virtuais.

Esse trabalho especialmente direcionado ao gerenciamento de recursos em sistemas operacionais desencadeou uma série de estudos para identificar as diversas dificuldades quando estende-se tal problema a um cenário fim-a-fim. A motivação para o aprofundamento nessa investigação parte da percepção de que os trabalhos relacionados a arquiteturas de QoS fim-a-fim demonstram certa

PUC-Rio - Certificação Digital Nº 0321123/CA

incapacidade em tratar importantes requisitos de gerenciamento de recursos. Sob a luz desse estudo, concluiu-se que o gerenciamento de recursos é um subproblema na provisão de QoS fim-a-fim que até então não concentrava os esforços apropriados em questões como harmonização, colaboração, adaptabilidade, entre outras.

2.2. Aspectos de projeto no gerenciamento de recursos com QoS

A provisão de QoS fim-a-fim depende diretamente do gerenciamento adequado dos recursos computacionais presentes em cada subsistema que, de alguma forma, colabora no oferecimento do serviço. Os recursos devem ser compartilhados utilizando-se estratégias de gerenciamento que garantam que os requisitos de desempenho das aplicações sejam atendidos. As estratégias definem, por exemplo, o algoritmo de escalonamento de um recurso, as condições de aceitação para a admissão de um novo fluxo, entre outros.

No entanto, sérias dificuldades são enfrentadas na construção de infra-estruturas de provisão de QoS fim-a-fim, devido principalmente aos fatores distribuição e heterogeneidade de recursos. Para uma adequada provisão de QoS, a infra-estrutura deve ser capaz de lidar com recursos que podem diferir pela sua (i) natureza (e.g. passiva, ativa), (ii) finalidade (e.g. CPU, enlaces de rede, discos), (iii) tecnologia (e.g. enlaces ATM, enlaces Ethernet) e (iv) meio (e.g. fibra, cabo coaxial). As plataformas que abrigam e controlam diretamente os recursos podem variar, por sua vez, em termos de hardware e de software, se são sistemas embutidos ou de propósito geral, entre outras características.

Algumas arquiteturas de QoS procuram amenizar essa complexidade gerenciando o uso dos recursos do ambiente em níveis de abstração mais altos Ao invés de submeterem estratégias de gerenciamento sobre recursos reais como CPU e filas de rede, essas arquiteturas se limitam a controlar a admissão de novos fluxos, considerando, por exemplo, a carga já submetida aos *hosts*. Normalmente, essa abordagem acaba por atribuir uma maior responsabilidade na manutenção das garantias de QoS ao monitoramento de fluxos. Dependendo das características do cenário e das aplicações, esse poder de controle pode ser o suficiente, ou mesmo mandatório para manter certa independência em relação às plataformas utilizadas.

PUC-Rio - Certificação Digital Nº 0321123/CA

Outros cenários, no entanto, podem demandar um controle de QoS mais rígido, em que o gerenciamento de recursos deve atingir uma granularidade tal que parcelas dos recursos reais possam ser reservadas, garantidas por algoritmos de escalonamento e de controle de admissão associados a esses recursos.

Observa-se, portanto, que diferentes demandas por representações de necessidades de QoS podem se manifestar, de acordo com certas características do cenário distribuído. Cada representação, ou especificação de QoS, possui notação adequada ao seu correspondente nível de abstração, o qual determina o conjunto de parâmetros a serem usados na descrição dos requisitos. O grau de controle de QoS, exemplificado nos dois parágrafos anteriores, é um fator que pode ditar qual é o nível de abstração de QoS mais baixo necessário para satisfazer o controle requerido.

Outros fatores também influenciam na escolha dos níveis de abstração mais adequados, como o conjunto de atores presentes no cenário. Por exemplo, o usuário da aplicação precisa expressar sua solicitação no mais alto nível possível, em termos da finalidade daquela aplicação; o operador de telecomunicações e o administrador de redes, pelo contrário, desejam manipular diretamente as reservas de recursos reais; o desenvolvedor de aplicações deseja, na maioria das vezes, se ater às necessidades de comunicação fim-a-fim e de processamento e armazenamento finais; e o projetista de serviços de telecomunicações deseja prever todas essas possibilidades para disponibilizá-las em seu produto. Como, em geral, há a coexistência de diferentes atores[1] em vários cenários de gerenciamento a infra-estrutura de QoS deveria prover, idealmente, mais de um nível de abstração. Além disso, ainda outras representações de QoS são internamente necessárias, demandadas pelas camadas de software que podem compor a infra-estrutura, tais como os protocolos de negociação, middleware, bibliotecas e sistemas operacionais.

Boa parte das soluções de infra-estrutura de QoS existentes permite a especificação de requisitos de QoS em níveis de abstração mais altos e que podem ser mapeados para outros níveis mais baixos até que se atinja parâmetros

[1] Quando nas discussões desta tese faz-se referência a diferentes atores, não significa que eles devam ser pessoas realmente distintas, pois há muitos casos em que um ator assume outros papéis, como o desenvolvedor da aplicação que, também, projeta seu serviço. Mesmo quando os nomes dos atores não se encaixam ao cenário, pode ser que suas funções são ali atribuídas a outrem.

PUC-Rio - Certificação Digital Nº 0321123/CA

relacionados ao subsistema de gerenciamento de recursos internamente modelado. Porém, devido ao fato de que cada solução possui seu próprio modelo de gerenciamento de recursos, a participação de atores presentes no cenário distribuído, mas não considerados pela infra-estrutura de QoS, torna-se impossível. Além disso, não se prevê a possibilidade de compartilhamento do gerenciamento de recursos com outras infra-estruturas. Na pior das hipóteses, o uso concomitante de mais de uma infra-estrutura de QoS sobre um mesmo ambiente poderia levar a uma contabilidade incorreta de recursos, já que um subsistema de gerenciamento não consideraria as alocações do outro.

Além disso, na prática, usuários e desenvolvedores têm à disposição um conjunto de abstrações de QoS totalmente desconexas daquelas entregues aos operadores de serviços e administradores de redes. Isso pode levar a dificuldades na realização de ações de manutenção e adaptação dos serviços. Em certos cenários, os operadores e administradores não têm noção de quais são as garantias de QoS já atribuídas a algumas aplicações, porque a infra-estrutura de QoS não as expressa em níveis de abstração ou interfaces acessíveis ou de interesse deles.

A busca por soluções para esses problemas pode ser iniciada pela compreensão do conceito básico de recurso, já que sua heterogeneidade e distribuição são pontos-chave na complexidade de provisão de QoS. Recurso refere-se a um elemento ou componente usado no intuito de auxiliar na realização de uma tarefa, ou que simplesmente provê algo útil a quem dele faz uso. Imediatamente, nota-se que o conceito é abstrato o bastante para assumir diferentes níveis de representação em ambientes distribuídos. Um recurso pode ser a fila de pacotes de uma interface de rede em um roteador, como também pode ser a rede como um todo ou mesmo toda a infra-estrutura de um serviço.

Sem perda de generalidade, pode-se propor que a aceitação de uma solicitação de QoS, expressa em um certo nível de abstração, culmine sempre na representação de um recurso correspondente, possivelmente abstrato, que possui como características exatamente os requisitos descritos na especificação. Por exemplo, a solicitação de um usuário para um “serviço de vídeo sob demanda com qualidade de TV”, caso aceito, gera a representação de um recurso abstrato no subsistema de gerenciamento, denominado “vídeo sob demanda”, com a característica “qualidade de TV”. A solicitação preparada pelo desenvolvedor para sua aplicação descrevendo “serviço de comunicação fim-a-fim com taxa média de

PUC-Rio - Certificação Digital Nº 0321123/CA

4Mbps e retardo máximo de 30ms", quando efetivada, vai gerar um recurso denominado "canal de rede fim-a-fim", com características "r=4Mbps e s=30ms". Operadores de serviços de telecomunicações poderiam fazer o mesmo, criando a representação de recursos no subsistema de gerenciamento, como um enlace de 1Gbps que interliga as estações A e B.

Com esse procedimento, obtém-se uma representação de recurso para cada especificação de QoS, em cada nível de abstração correspondente. A capacidade de expressão de recursos em múltiplos níveis é fundamental para o suporte aos diversos atores. Essa noção inicial que associa cada nível de especificação de QoS a uma abstração de recurso correspondente é um primeiro passo para a uniformização desejada no gerenciamento de recursos fim-a-fim.

Com a uniformização do gerenciamento de recursos, todos os atores podem utilizar e realizar manutenções naqueles níveis a eles autorizados. Se por decisão de projeto, uma abstração de recurso puder ser mapeada em abstrações de níveis inferiores, essa estrutura recorrente poderia ser percorrida pelos atores no intuito de facilitar, em qualquer nível, a modificação de garantias de QoS já estabelecidas (sintonização de QoS) e a alteração do comportamento de estratégias de gerenciamento existentes (adaptação de serviços). Evidentemente, a criação de novos serviços ficaria extremamente facilitada, contanto que projetistas e operadores possam realizar a criação e a configuração de novas abstrações de recursos.

A proposta de uniformização não surtirá efeito caso o subsistema de gerenciamento de recursos continue sendo definido internamente às infra-estruturas de QoS. Por isso, deve-se caminhar para uma solução independente de plataforma e independente de infra-estrutura de QoS. A Figura 1 apresenta uma arquitetura simplificada para a proposta de uniformização de recursos e de tratamento do gerenciamento como subproblema à parte da provisão de QoS.

No entanto, várias dificuldades devem ser contornadas para que tal solução seja viável. É necessário o emprego de técnicas avançadas de engenharia de software que preencham os diversos requisitos de formalização e representação das abstrações de recursos, e que permitam manter o código independente de plataforma, tanto quanto possível e for de interesse.

PUC-Rio - Certificação Digital Nº 0321123/CA

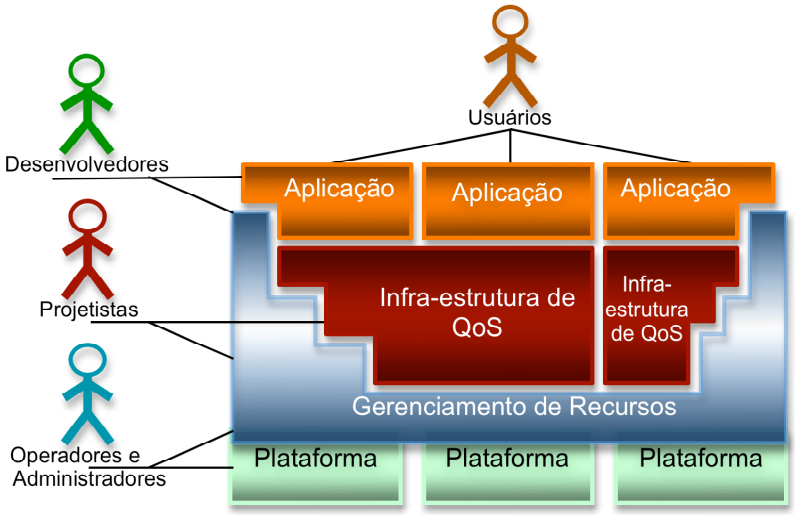

Figura 1 – Gerenciamento de recursos com abstrações multiníveis.

O Quadro 1, a seguir, resume os principais aspectos de projeto que podem ser considerados na busca por uma abordagem mais adequada para o gerenciamento de recursos fim-a-fim com suporte a QoS. Em meio a esses aspectos, capturados ao longo das discussões apresentadas nesta seção, estão os objetivos principais da proposta deste trabalho: i) alcançar a generalidade necessária para incluir recursos heterogêneos distribuídos em um subsistema de gerenciamento de recursos; ii) possibilitar que a configuração e manutenção de recursos se tornem um trabalho colaborativo entre os diferentes atores presentes no ambiente; iii) garantir a portabilidade e interoperabilidade por meio da independência de plataforma e iv) permitir a adaptabilidade dos serviços, tanto em tempo de projeto quanto em tempo de operação.

Uniformização

A generalidade da solução e o tratamento da distribuição e heterogeneidade dos recursos dependem da formulação de abstrações capazes de uniformizar os mecanismos de reserva e manutenção de recursos. A uniformização dos recursos aliada à representação em diferentes níveis de abstração vêm a facilitar o suporte a outros aspectos desejados, tais como, a composição de reservas fim-a-fim, o gerenciamento colaborativo, a adaptabilidade, a portabilidade e a interoperabilidade.

Provisão de QoS fim-a-fim

A solução de gerenciamento de recursos deve oferecer mecanismos para reserva e manutenção dos recursos de cada subsistema relevante em cenários distribuídos, possibilitando a composição de garantias de QoS fim-a-fim, por parte de infra-estruturas de provisão de QoS.

PUC-Rio - Certificação Digital Nº 0321123/CA

Gerenciamento colaborativo de recursos

Além dos componentes de software de infra-estruturas de QoS (protocolos, middleware, bibliotecas, etc.), a solução de gerenciamento de recursos deve permitir a interação de outros atores possivelmente presentes em cenários distribuídos, desde o usuário da aplicação ao administrador de redes. Todos eles, atuando em conjunto, formam um esforço de colaboração na manutenção dos recursos que traz vantagens em termos de personalização e evolução dos serviços.

Portabilidade e Interoperabilidade

A solução de gerenciamento de recursos deve oferecer métodos de operação que mantenham independência das plataformas tanto quanto possível. Se houver necessidade de métodos específicos de plataforma, estes devem ser bem definidos de forma a serem implementados em funcionalidade equivalente e interoperável.

Coexistência de estratégias de gerenciamento

A diversidade de aplicações e de seus requisitos impõe o oferecimento simultâneo de diferentes tipos de serviços, cada qual melhor regido por diferentes estratégias de gerenciamento de recursos. É necessário que a solução possibilite o uso concomitante de diferentes técnicas de compartilhamento sobre um mesmo recurso, habilitando a disponibilização de uma ampla gama de serviços.

Adaptabilidade

A solução de gerenciamento de recursos deve permitir que o comportamento dos mecanismos de compartilhamento de cada recurso, definido pelas estratégias de gerenciamento, seja modificado em tempo de operação, para uma rápida implantação ou modificação de serviços. Adaptações são ações necessárias em resposta ao surgimento de novos tipos de demanda, reformulação de políticas de gerenciamento, aperfeiçoamento de sistemas, entre outros.

Aplicabilidade independente do grau de controle de QoS

Os graus de controle estabelecidos pelas infra-estruturas de QoS devem poder ser expressos e controlados por meio de alocações de recursos nos níveis de abstração necessários. A solução deve evitar que a uniformização do gerenciamento de recursos interfira na capacidade de expressão das necessidades de uso de cada recurso.

Aplicabilidade independente do grau de especialização do ambiente

A solução deve considerar seu funcionamento para os diversos propósitos de ambientes distribuídos. Ou seja, devem ser oferecidas abstrações adequadas para infra-estruturas de QoS voltadas a ambientes com acoplamento de software forte (grades e aglomerados) ou fraco (aplicações cliente-servidor), sejam envolvendo sistemas de propósito geral ou sistemas embutidos, ou ainda, construídas sobre arquiteturas de middleware ou não.

PUC-Rio - Certificação Digital Nº 0321123/CA

Escalabilidade

A solução de gerenciamento de recursos deve evitar que a carga adicional (overhead) inerente ao gerenciamento uniforme e distribuído torne tanto impraticável a participação de dispositivos com limitado poder de processamento, quanto limitado o poder de crescimento do número de nós envolvidos no sistema distribuído.

Segurança

A solução deve impor barreiras de segurança sobre as operações críticas oferecidas, principalmente sobre os mecanismos de adaptação de serviços. Se mal intencionadas ou mesmo erroneamente programadas, adaptações podem ser capazes de expor dados privados, danificar ou interromper os serviços.

Ferramentas de configuração eficientes e dinâmicas

A solução deve disponibilizar um conjunto de ferramentas de arquitetura e manutenção de serviços que explorem seu poder de configuração, uniformidade e adaptabilidade, e que permitam a implantação fácil e rápida de novos serviços.

Quadro 1 – Aspectos de projeto no gerenciamento de recursos

PUC-Rio - Certificação Digital Nº 0321123/CA

2.3. Desenvolvimento de Software Dirigido por Modelos

Entre as diversas técnicas de engenharia de software que melhor se adéquam aos objetivos do presente trabalho, está o desenvolvimento de software dirigido por modelos (*MDSD – Model-driven software development*) (Stahl, 2006). MDSD vem ganhando espaço rapidamente nos últimos anos, principalmente após a padronização da arquitetura dirigida por modelos (*MDA – Model-driven Architecture*) (Miller, 2003) pelo OMG (*Object Management Group*). De fato, MDSD é uma proposta de generalização de MDA, motivada pela identificação de certas limitações associadas aos objetivos focados pelo OMG quando de sua padronização. Esta seção se dedica a uma rápida apresentação de MDSD, iniciando pelos conceitos herdados de MDA, com o intuito de introduzir o leitor a esse paradigma e identificar possíveis vantagens de aplicação dos mesmos conceitos no processo de gerenciamento de recursos.

MDA foi concebido com dois objetivos claros: permitir a interoperabilidade e a portabilidade no desenvolvimento de software. Descrevendo de forma simplificada, os projetistas utilizam uma linguagem formal para especificar modelos abstratos de sistemas de software, de forma independente de plataforma.

Os modelos sofrem transformações recorrentes, podendo, inclusive, gerar outros modelos, até a geração de código específico de plataforma.

Nota-se que UML (*Unified Modeling Language*) (ISO/IEC, 2005) alcança a mesma independência de plataforma, porém restrita à modelagem. Até a introdução de MDA, as ferramentas baseadas em UML eram muito limitadas, capazes de gerar apenas porções mínimas de código incompleto (*skeletons*) na linguagem (plataforma) desejada. O programador deveria completar o código, já na plataforma desejada, para descrever todas as funcionalidades do sistema. Para portar o mesmo modelo para outras plataformas, era necessária a reprogramação de todas as funcionalidades. Além disso, manutenções no modelo tinham de ser espelhadas manualmente em cada plataforma, tornando-se um processo muito custoso. As funções de maior uso das ferramentas de desenvolvimento com suporte a UML estavam relacionadas a documentação e engenharia reversa. UML apresenta essas deficiências principalmente devido ao seu fraco formalismo e modelo de relacionamento. UML, por si, se mostra uma ótima ferramenta para documentação de código, porém inadequada para um processo de desenvolvimento em que a modelagem vem a desempenhar um papel central.

Um desenvolvimento centrado na modelagem independente de plataforma pode, de fato, oferecer facilidades para a instanciação de produtos em diversas plataformas, com alto grau de automatização. As ações de manutenção de software também podem ser beneficiadas, pois modelos que descrevem a lógica (e não só a estrutura) de uma aplicação podem ser alterados e reinstanciados nas plataformas-alvo, obtendo-se, então, revisões de documentação e código simultaneamente.

Em MDA, a descrição de um modelo independente de plataforma (PIM – *Platform-independent model*) é feita utilizando-se um perfil UML. Um perfil UML é uma linguagem que adapta ou estende UML para melhor acondicioná-la ao domínio do sistema a ser modelado. Um perfil UML faz uso de construtos básicos de UML, como classes, associações, estereótipos e restrições para definir uma linguagem formal de construção de modelos em um domínio específico. Por isso, pode-se dizer que um dado perfil UML é um meta modelo para a construção de modelos no domínio a que se dedica.

A partir de UML 2.0, incluída em MDA, tanto a própria UML quanto os perfis UML gerados são definidos por construtos básicos formalizados em um

PUC-Rio - Certificação Digital Nº 0321123/CA

framework denominado MOF (*Modeling Object Facility*) (OMG, 2002). Por ser uma especificação dedicada a descrever meta modelos, MOF é dito ser um meta meta modelo em MDA. As ferramentas de modelagem, transformadores e geradores de código devem ter MOF como representação interna de modelos descritos sobre perfis UML.

O motivo de se definir um modelo independente de plataforma (PIM) se deve ao fato de que conceitos são mais estáveis que tecnologias, e se os conceitos são modelados por meio de especificações formais, mais facilmente obtém-se ferramentas de transformações automáticas. Essas transformações podem gerar, a partir de um PIM, outros modelos, também descritos em perfis UML, denominados modelos específicos de plataforma (*PSM – Platform-specific model*). Os PSM adaptam o PIM às idiossincrasias da plataforma, descrevendo, por exemplo, como uma certa classe ou estereótipo deve se especializar na linguagem da plataforma.

Podem haver, ainda, transformações entre PSMs, dependendo dos níveis de abstração de plataformas desejados. Por exemplo, um PIM pode ser transformado em um PSM dedicado a um framework, que, por sua vez, pode ser transformado em um PSM dedicado a uma linguagem de programação. Finalmente, os PSMs (ou mesmo o próprio PIM) devem sofrer uma última transformação, para a geração do código na plataforma-alvo. A Figura 2 ilustra as sucessivas transformações modelo-para-modelo e modelo-para-código típicas em MDA.

Transformações modelo-para-modelo descrevem, geralmente, como os construtos do meta modelo de origem são mapeados nos construtos do meta modelo de destino. Já as transformações modelo-para-código não consideram um meta modelo de destino, pois geram artefatos baseados na plataforma. Normalmente, trata-se de substituições de textos, que podem ser definidas, por exemplo, por meio de *templates* (Stahl, 2006).

Para que as transformações consigam gerar a maior porção de código possível de uma aplicação, os meta modelos devem ser completados por especificações do comportamento algorítmico dos seus elementos. Os perfis UML, por si somente, não são capazes de expressar toda lógica de uma aplicação, mas sim suas estruturas de dados, relacionamentos entre classes, restrições, etc.

PUC-Rio - Certificação Digital Nº 0321123/CA

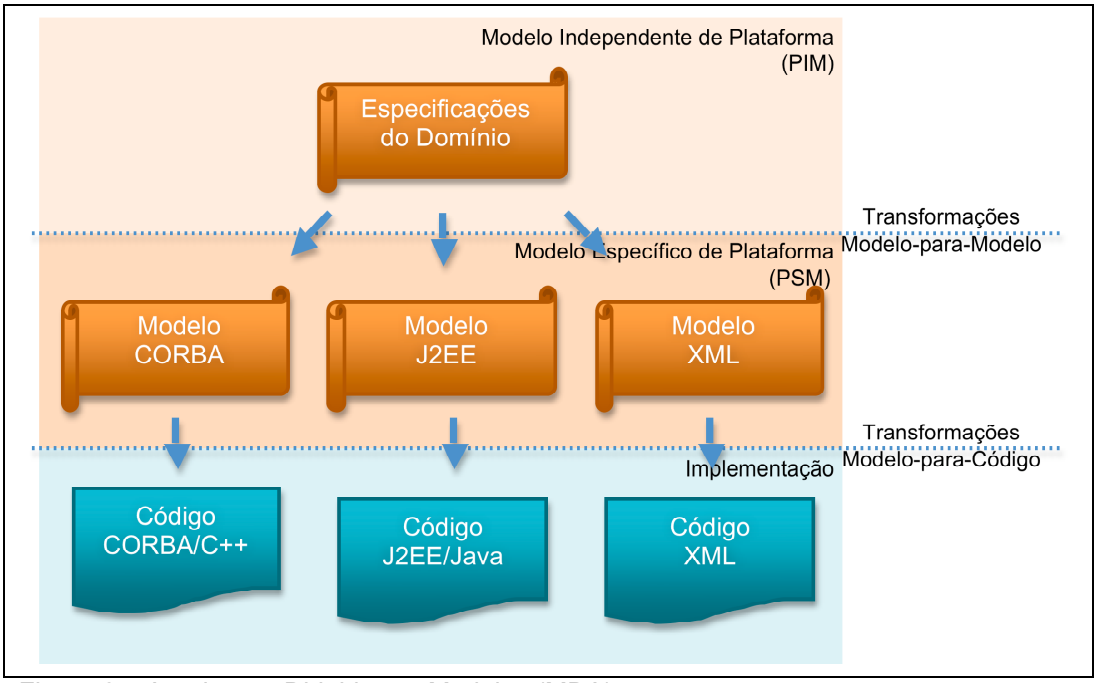

Figura 2 – Arquitetura Dirigida por Modelos (MDA)

Em UML 2.0, o OMG incluiu semântica de ações, descritas verbalmente, a serem padronizadas como linguagens textuais e gráficas pelos próprios desenvolvedores de ferramentas. De um modo geral, pode-se expressar variáveis, operações lógicas e aritméticas, e várias outras funcionalidades básicas de linguagens de programação imperativas. Fica a cargo dos desenvolvedores, também, a implementação das regras de tradução das semânticas de ações para a plataforma-alvo. Nota-se que MDA estabelece que a descrição da estrutura da aplicação, baseada nos meta modelos, é realizada em separado da especificação de seu comportamento procedimental.

MDA representa um esforço de padronização de técnicas já em uso há algum tempo, porém em uma escala restrita e sem uma formalização de processo de desenvolvimento. Programação generativa, modelagem de domínio específico e geração automática de código vinham sendo adotadas individualmente em diversas soluções arquiteturais. Com MDA, esses mecanismos entram em linhas de produção de software de forma integrada, definindo um processo de desenvolvimento ancorado por um padrão para interoperabilidade e portabilidade.

No entanto, o foco dessa padronização pode levar a limitações quando em um processo de desenvolvimento de software deseja-se incluir outros objetivos que não somente interoperabilidade e portabilidade. O uso recorrente de especificações em torno de MOF e UML 2.0 acabam por tornar MDA incapaz de

PUC-Rio - Certificação Digital Nº 0321123/CA

satisfazer requisitos de maior reuso e melhor adequação a domínios. Por isso, os desenvolvedores que buscam esses requisitos adotam conceitos análogos a MDA e criam as linguagens e processos mais especializados aos seus domínios.

Surge, então, a necessidade de harmonização dos conceitos derivados de MDA para definir uma terminologia e requisitos básicos que podem ser adotados por desenvolvedores independentemente. A proposta MDSD (Stahl, 2006) tem exatamente esse objetivo. MDA passa a ser, assim, um caso específico de MDSD em que a interoperabilidade advinda da padronização é seu grande atrativo. Como esperado, MDSD não impõe o uso de meta meta modelos ou linguagens específicos. De fato, a abordagem se restringe a descrever os conceitos e terminologia do processo de desenvolvimento, em boa parte herdados de MDA.

O conceito a partir do qual se inicia o desenvolvimento em MDSD é o de domínio, que delimita o escopo de interesse ou conhecimento da aplicação a ser modelada. Em sistemas complexos, pode ser interessante o particionamento do domínio em subdomínios, de forma a facilitar a modelagem limitando-a em escopos menores. Subdomínios podem ser interessantes quando torna-se apropriada a definição de uma linguagem de modelagem especializada ou simplesmente para redução dos princípios modelados. A estruturação de subdomínios constitui, sem dúvida, uma forma de separação de interesses (*separation of concerns*) tal qual a provida pelo desenvolvimento de software orientado a aspectos (AOSD – *Aspect-oriented software development*) (Aksit, 2004).

A formalização da estrutura do domínio (ou de sua parte relevante) é definida por um meta modelo, usando uma sintaxe abstrata e critérios (semântica estática) que definem se um modelo criado é válido ou não (bem formado). Um meta modelo é a instância de um meta meta modelo, que, por sua vez, define os conceitos básicos disponíveis para a criação do meta modelo. A definição de um meta meta modelo em comum é muito útil na integração de ferramentas dedicadas à meta modelagem ou à modelagem sobre quaisquer meta modelos, pois são neles em que se baseia a representação interna de um modelo criado.

Uma linguagem de domínio específico (DSL – Domain-specific language) permite tornar os aspectos relevantes de um domínio modeláveis, ou seja, expressáveis formalmente. A construção de uma DSL se baseia em um meta modelo, na semântica estática do meta modelo e em uma sintaxe concreta que

PUC-Rio - Certificação Digital Nº 0321123/CA

representa o meta modelo. Ao final, uma DSL, também chamada de linguagem de modelagem, define a semântica dinâmica necessária para dar significado aos construtos do meta modelo. DSLs são construídas pressupondo que devem ser intuitivamente claras a quem delas faz uso, no caso o modelador que tem total conhecimento sobre o domínio. Para isso, DSLs mapeiam conceitos presentes no espaço do problema para uma representação direta e facilmente notável pelo especialista no domínio. Para oferecer maiores facilidades, várias DSLs oferecem maior agilidade de especificação ao definirem seus construtos graficamente, enquanto outras procuram permitir um maior poder de expressão textualmente.

Modelos formais são sentenças formuladas a partir de uma DSL para a representação abstrata de uma aplicação contextualizada no domínio da DSL. Conceitualmente, essa especificação, descrita por meio da sintaxe concreta de uma DSL, representa a instanciação do meta modelo correspondente. Conforme Stahl & Völter (2006), a palavra "abstrata" em MDSD não deve ser tomada no sentido de indefinição ou imprecisão de significado, mas sim, redução da representação à essência.

Sobre os modelos formais são aplicadas transformações automatizadas que podem gerar outros modelos (transformações modelo-para-modelo) ou código específico para uma plataforma (transformações modelo-para-código). O conceito de plataforma é amplo o suficiente para abrigar desde arquiteturas de componentes a linguagens de programação de propósito geral. Aliás, quanto mais especializada for a plataforma, menos árdua será a transformação, uma vez que o código gerado pode contar com APIs de alto nível e de maior qualidade. Exemplos de plataformas especializadas são os frameworks, conjuntos de super classes, componentes, e, até mesmo, interpretadores de código.

Dependendo da abrangência do domínio da aplicação, da expressividade das DSLs e da especialização da plataforma, as transformações modelo-para-código podem se aproximar ou atingir 100% do código final gerado automaticamente. Muitas vezes, isso se torna impossível e por isso porções avulsas de código são escritas sobre a plataforma de forma a complementar a funcionalidade desejada, com a qual a transformação pode contar. A Figura 3 ilustra de forma simplificada o relacionamento entre os principais conceitos de MDSD.

Stahl e Völter (2006) afirmam que MDSD não somente deve ser usada no desenvolvimento de software nos diversos domínios, mas podem ter seus

PUC-Rio - Certificação Digital Nº 0321123/CA

conceitos adaptados e enriquecidos com novos procedimentos em outros tipos de processos. O maior objetivo de MDSD é generalizar e harmonizar os conceitos relacionados ao desenvolvimento dirigido por modelos, de forma que as variantes de uso adotem os mesmos fundamentos, acelerando o entendimento dos processos. A próxima seção descreve de que forma uma adaptação de MDSD pode trazer vantagens como solução para o gerenciamento de recursos fim-a-fim com suporte a QoS.

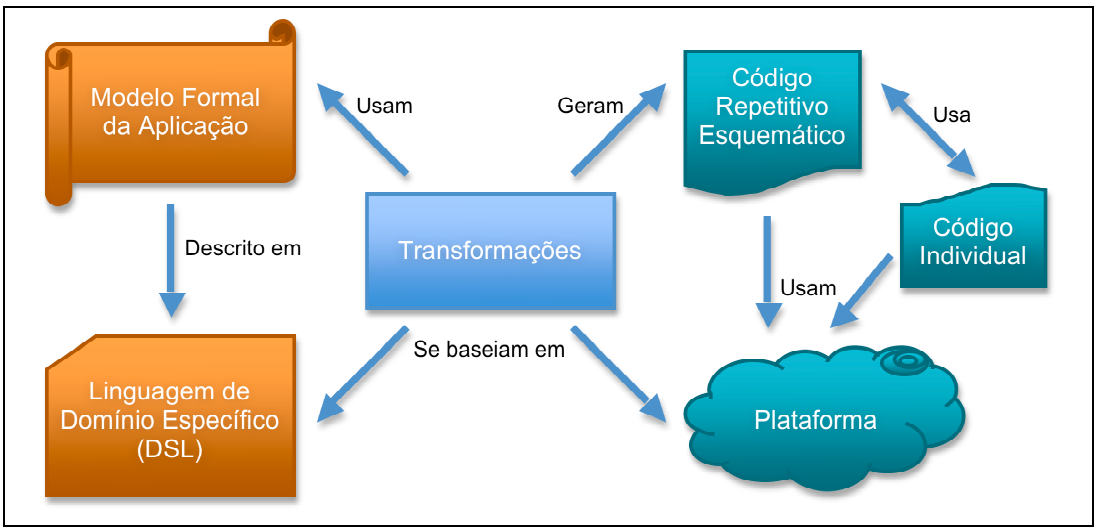

Figura 3 – Desenvolvimento de Software Dirigido por Modelos (MDSD)

2.4. Gerenciamento de Recursos Dirigido por Modelos

A generalidade de MDSD permite que várias alternativas sejam formuladas para o gerenciamento de recursos dirigido por modelos (MDRM). Esse domínio pode ser tomado como um caso de uso específico de MDSD, ou como um processo de especificação arquitetural que incorpora os conceitos de MDSD de forma adaptada.

Conforme mencionado anteriormente, este trabalho é motivado pela falta de uma solução de gerenciamento de recursos fim-a-fim capaz de i) uniformizar o acesso a recursos heterogêneos e em múltiplos níveis de abstração, ii) garantir a interoperabilidade e portabilidade das plataformas, iii) permitir a colaboração entre diversos atores nas tarefas de gerenciamento e iv) prover mecanismos para adaptação de serviços. Esses quatro objetivos em conjunto já representam a

PUC-Rio - Certificação Digital Nº 0321123/CA

motivação necessária para tratar o gerenciamento de recursos como um processo de desenvolvimento diferenciado.

Quanto ao aspecto de colaboração entre atores, nota-se ser de grande importância a análise de como cada ator pretenderia interagir com a solução. Do ponto de vista do desenvolvedor de software, a especificação de termos de QoS para seu produto pode ser tratada como parte da descrição arquitetural da aplicação (Kavimandan, 2007). Particularmente, se as especificações de QoS puderem ser vistas pelo desenvolvedor como a manipulação direta de um recurso de alto nível a ele entregue pela infra-estrutura de QoS, obtêm-se dois planos arquiteturais: um para a especificação da aplicação, que inclusive pode ser regido por MDSD; e um outro para o gerenciamento de recursos com QoS. Outros planos podem ser considerados, porém estão fora do escopo desta discussão.

Outros atores possuem visões diferentes do gerenciamento de recursos. Projetistas de serviços de telecomunicações unem a especificação de topologias, protocolos de rede e categorias de serviços ao provisionamento inicial de recursos necessários para dar suporte à sua infra-estrutura de QoS. Operadores de serviços de telecomunicações e administradores de redes tratam o gerenciamento de recursos como tarefa-chave no seu dia-a-dia, realizando manutenções diretamente sobre o estado corrente do compartilhamento de recursos.

Para tornar o processo de especificação ainda mais complexo, dependendo do tipo de cenário distribuído, pode haver mudanças nos papéis de cada ator. Por exemplo, desenvolvedores podem agir como projetistas, criando ou adaptando serviços personalizados às suas necessidades; projetistas podem agir como operadores, fazendo pequenos ajustes em projetos já instanciados; e operadores podem ter que subir ao nível de desenvolvedores para rastrear toda uma composição fim-a-fim de recursos.

Os distintos níveis de conhecimento técnico dos atores, a demanda para que eles trabalharem colaborativamente e os requisitos diferenciados de manutenção e adaptabilidade, em conjunto, fazem com que MDRM seja concebido como um processo adaptado de MDSD.

Por isso, MDRM não pode estabelecer que a modelagem se refira à lógica de parte de um software (como em MDSD), a despeito da possibilidade de instanciação da estrutura de gerenciamento como tal. Mas sim, o objeto da modelagem deve ser a lógica de compartilhamento (com QoS) dos recursos

PUC-Rio - Certificação Digital Nº 0321123/CA

distribuídos de um ambiente, ancorada em um meta modelo e uma DSL especificamente criados para satisfazer as diferentes demandas e habilidades dos possíveis atores. Se, para uma dada plataforma, o modelo deve ser transformado em código de linguagem de programação ou não, isso pouco importa para a essência de MDRM. Inclusive, devido aos diversos tipos de plataformas envolvidos, passa a ser interessante a livre escolha da forma de suporte para instanciação dos modelos. Contanto que as regras do meta modelo sejam respeitadas, mantém-se a interoperabilidade necessária.

Outro fator de diferenciação entre MDRM e MDSD se evidencia ao considerar-se que em MDRM há a demanda por ações de adaptação de serviços, que serão efetuadas como rotinas de manutenção de modelos já instanciados. Essas manutenções em modelos devem poder ser refletidas imediatamente sobre as plataformas, sem que para isso sejam necessárias interrupções de serviços. Essa característica, especificamente, remete à técnica de programação dirigida por dados (*data-driven programming*), na qual o software em execução pode ser facilmente reconfigurado por meio de alterações em um modelo baseado em estruturas de dados.

De qualquer forma, a adoção da base conceitual de MDSD por MDRM justifica-se por concepção, dado que alguns aspectos de projeto, mencionados na Seção 2.2, são imediatamente beneficiados:

- Uniformização: Fica facilitada, pois as abstrações necessárias para a uniformização no acesso a recursos podem ser definidas no meta modelo, único para todo o processo.
- Gerenciamento colaborativo: A utilização de modelos formais potencialmente intercambiáveis possibilita que vários atores tirem proveito das especificações de recursos, desde que as abstrações da DSL deles se aproximem.
- Portabilidade: Atendida pela descrição de modelos formais em DSL, independentes de plataforma. Transformações podem gerar instanciações específicas para cada plataforma desejada.
- Interoperabilidade: Atendida pela definição de um meta modelo único que todas as ferramentas e plataformas devem suportar internamente.
- Adaptabilidade: Facilitada, pois as adaptações podem ser descritas em alto nível, sob a forma de manutenções sobre os modelos. Porém, deve

PUC-Rio - Certificação Digital Nº 0321123/CA

haver suporte à imediata reconfiguração das instâncias sem interrupções.

- Ferramentas de configuração: A especificação de modelos formais já constitui poderosa ferramenta de desenvolvimento devido às rotinas de validação. Outras funcionalidades podem ser oferecidas, como DSLs gráficas, ambientes de testes e simulação, entre outros.

2.4.1. Adaptação dos conceitos

O domínio de MDRM é primordialmente definido como gerenciamento de recursos fim-a-fim com suporte a QoS. Ele pode ser dividido em subdomínios, uma vez que a especificação da lógica do compartilhamento de recursos envolve não só a estruturação dos recursos, mas também a descrição de certos comportamentos procedimentais, notadamente sob a forma de estratégias de gerenciamento de recursos. Assim, cada tipo de estratégia de gerenciamento (escalonamento, admissão, etc.) é tratado como subdomínio de MDRM, podendo ser desenvolvido sob MDSD, por exemplo.

Especializações do domínio de gerenciamento de recursos poderiam ser sugeridas para acompanhar as variações de cenários distribuídos, como, por exemplo, gerenciamento de recursos em grades, em aglomerados, na Internet ou em sistemas distribuídos embutidos de tempo real (DREs). Porém, deseja-se que as possíveis especializações do domínio não demandem a construção de diferentes meta modelos em MDRM. Se o meta modelo de MDRM for único, abre-se a possibilidade de que diferentes infra-estruturas de provisão QoS utilizem o mesmo subsistema de gerenciamento de recursos concomitantemente. Para ser único, o meta modelo deve, portanto, ser capaz de assimilar as possíveis nuances entre os domínios especializados, permitindo que elas sejam expressas na modelagem de uma instância.

O meta modelo de MDRM é denominado Árvores de Recursos Virtuais (VRT) e permite a definição de estruturas hierárquicas que denotam sucessivos particionamentos na alocação e compartilhamento de um recurso. Boa parte dos objetivos perseguidos neste trabalho, como a uniformização, abstrações de

PUC-Rio - Certificação Digital Nº 0321123/CA

recursos multiníveis e adaptabilidade de serviços são atingidos conceitualmente por meio do meta modelo VRT, que se encontra descrito no Capítulo 3 desta tese.

O fato de o meta modelo ser único dispensa a necessidade de definição de um meta meta modelo, pois as abstrações do meta modelo podem representar por si só os fundamentos necessários para as tarefas de modelagem, validação e transformação. Se fosse preciso, o meta meta modelo poderia descrever as entidades básicas de MDRM, como recurso, parâmetros de QoS, estratégias de gerenciamento, entre outros.

A DSL criada para tornar possível a especificação textual de instâncias do meta modelo VRT é denominada Pan, apresentada no Capítulo 4. Pan é uma linguagem de aplicação XML (Bray, 2006) que se baseia em XML Schema (Thompson, 2004) e Schematron (ISO/IEC, 2006) para expressar o formalismo do meta modelo. Ao adotar tecnologias XML, Pan traz consigo uma série de ferramentas para validação e facilidades de desenvolvimento disponíveis amplamente no mercado. A linguagem Pan oferece construções que permitem aos atores realizarem tanto a especificação de modelos de gerenciamento de recursos, quanto a manutenção de modelos já instanciados. Particularmente, uma manutenção de modelo em Pan pode ser representada pela especificação de operações que representam apenas as diferenças em relação ao modelo original, instanciado. Essas operações são, ainda, acondicionadas em transações para definir a atomicidade e interdependências que ocasionalmente são necessárias.

Transformações de modelos definidos em Pan podem originar outros modelos também em Pan, por exemplo para desmembrá-los em descrições mais concisas, cada qual restrita aos recursos a serem instanciados em uma certa plataforma. Transformações de modelos descritos em Pan para código específico de plataforma representam o passo final do processo de MDRM. Como linguagem de aplicação XML, Pan elege XSLT como principal linguagem para especificar transformações modelo-para-modelo e, dependendo da plataforma, transformações modelo-para-código. A Figura 4 resume os principais conceitos envolvidos em MDRM descritos nesta seção.

A prototipação de ferramentas para MDRM que incluem não somente os transformadores de modelos, como também validadores, editor gráfico e editor textual também é discutida neste trabalho. O Capítulo 5 apresenta a cadeia de modelagem de recursos de MDRM, da especificação à manutenção dos modelos.

PUC-Rio - Certificação Digital Nº 0321123/CA

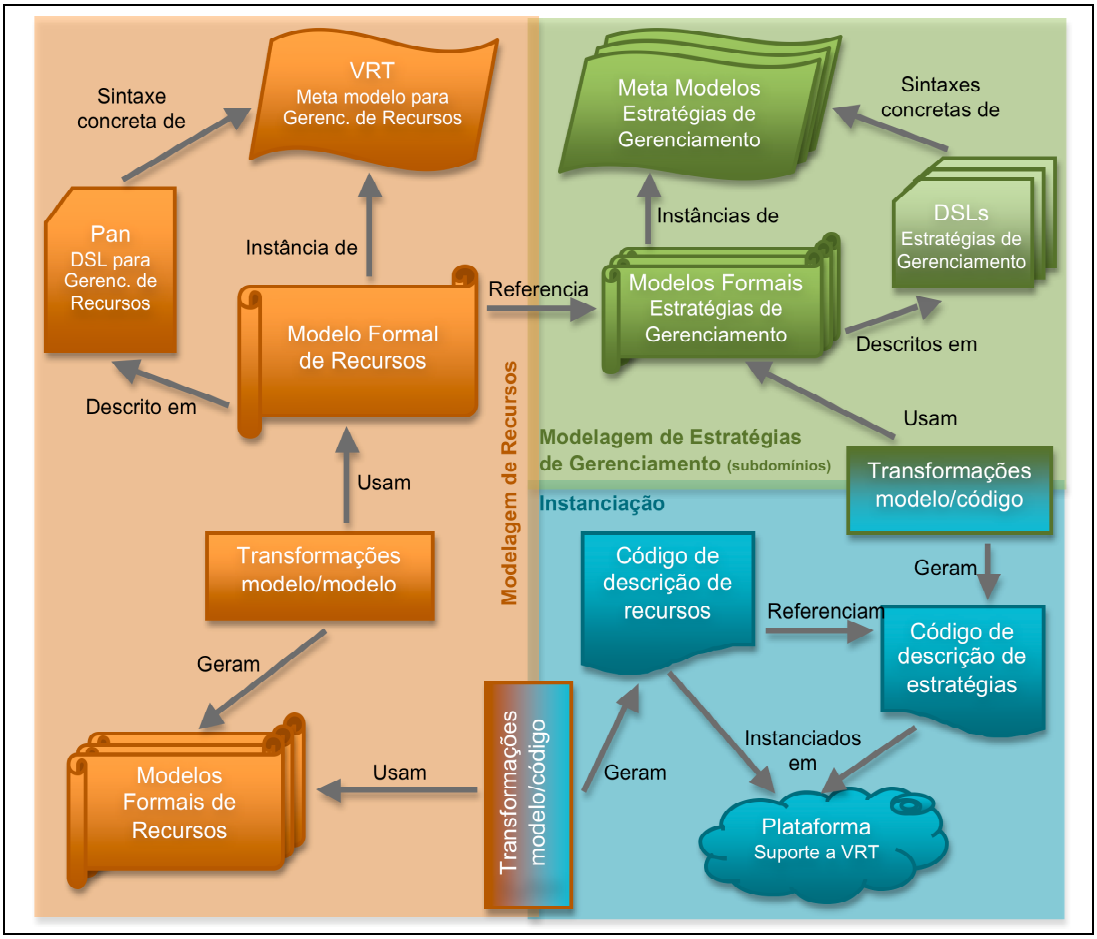

Figura 4 – Gerenciamento de Recursos Dirigido por Modelos (MDRM)

A complexidade do suporte a VRTs nas plataformas-alvo depende diretamente do tipo de recurso a ser gerenciado por cada uma delas. Recursos em níveis de abstração mais altos são representações mais simples, por não envolverem relacionamento estreito com uma certa plataforma. Eles poderiam ser instanciados por meio de repositórios centralizados ou distribuídos, que permitiriam a criação, consulta e manutenção de recursos. Por outro lado, no nível de abstração de recursos reais, aspectos como reserva de recursos e adaptabilidade das estratégias de gerenciamento tornam a instanciação mais complexa. Uma solução de referência denominada VRT-FS (*Virtual Resource Tree File System*) é proposta neste trabalho (Capítulo 6) para o suporte a VRTs em sistemas operacionais de propósito geral. VRT-FS lança mão de abstrações já conhecidas em sistemas operacionais, mais especificamente as providas por sistemas de arquivos, para oferecer uma interface de acesso à configuração e manutenção de VRTs internamente ao núcleo do sistema.

PUC-Rio - Certificação Digital Nº 0321123/CA

2.5.Trabalhos Relacionados

Por ser um problema amplo, o gerenciamento de recursos fim-a-fim com suporte a QoS será analisado partindo do cenário distribuído de gerenciamento, ou seja, arquiteturas relacionadas a middleware, passando pelo suporte a adaptações em sistemas, até os mecanismos de QoS em sistemas operacionais.

Alguns trabalhos recentes se concentraram no uso de técnicas de desenvolvimento dirigido por modelos para atacar algum aspecto relacionado a QoS e gerenciamento de recursos. QUICKER (Kavimandan, 2007) é uma abordagem que usa transformações de modelos para mapear políticas de QoS de uma aplicação para valores de parâmetros de QoS dependentes de plataformas. Esses parâmetros de QoS são usados para configurar o comportamento de sistemas de middleware baseados em componentes com suporte a tempo real. Ferramentas de modelagem baseadas em MDA são usadas para criar modelos de mapeamento de QoS. O uso de ferramentas de validação de modelos permite, nesse caso, que seja verificada a corretude das propriedades nas transformações e que seja favorecida a integração com processos automáticos de síntese dos parâmetros de QoS. QUICKER representa um trabalho que complementa a presente proposta, uma vez que não serão abordadas as questões em torno do mapeamento de parâmetros. A integração dos trabalhos tem grande potencial, não somente devido ao uso de técnicas similares de desenvolvimento, mas também pela possibilidade em MDRM de representação de categorias de serviços e parâmetros em diversos níveis de abstração.

QARMA (Liu, 2004) propõe um gerenciamento de recursos dirigido por modelos também focado em sistemas distribuídos de tempo real. A arquitetura especifica uma ferramenta de modelagem, uma linguagem de especificação e uma arquitetura de sistema implementada em CORBA. Os desenvolvedores constroem modelos de sistemas e automaticamente geram especificações de recursos. O ambiente não possui outros atores, por isso não é considerado o gerenciamento colaborativo. A abordagem de QARMA é a geração de especificação de recursos localizados em hosts para definir sistemas distribuídos e não o gerenciamento desses recursos quanto aos aspectos de compartilhamento e QoS.

Vários trabalhos têm se concentrado na aplicação de técnicas de gerenciamento de recursos em sistemas de middleware existentes, com o objetivo

PUC-Rio - Certificação Digital Nº 0321123/CA

de prover QoS fim-a-fim. Real-time CORBA (OMG, 2000) oferece um esquema de prioridades no gerenciamento de CPU, memória e comunicação, de acordo com as necessidades do usuário especificadas sob a forma de IDL (Interface Definition Language). As prioridades podem ser modificadas por meio de ganchos (hooks) que interceptam as solicitações, para prover sintonização de QoS. OMG (2003) estende vários conceitos introduzidos por Real-time CORBA, para permitir que parâmetros de escalonamento de threads sejam reconfigurados de forma mais dinâmica. Suas threads "distribuíveis" carregam parâmetros de escalonamento (modificáveis) passando pelas fronteiras dos sistemas. Por não possuírem um modelo de recursos, não há uniformidade no gerenciamento e, conseqüentemente, o poder de configuração e reconfiguração fica limitado a alguns mecanismos e recursos.

Cardei (2000) provê mecanismos de negociação e sintonização de QoS no middleware, embutidos em gerenciadores de serviços hierárquicos. Gerenciadores de baixo nível estão associados aos recursos localizados em cada nó do sistema distribuído. Gerenciadores de alto nível coordenam a negociação e sintonização de QoS fim-a-fim. Todos os gerenciadores podem ser dinamicamente carregados permitindo a modificação de comportamento em tempo de execução. A uniformização dos gerenciadores de serviços facilita o gerenciamento de recursos heterogêneos, pelo menos no nível de negociação (com controle de admissão) e sintonização de QoS. Apesar de denominar a hierarquia de gerenciadores como "modelo de recursos", não são mencionadas as políticas de compartilhamento do recurso, tais como estratégias de escalonamento e de admissão, além da adaptabilidade no nível do recurso. As adaptações ocorrem em alto nível, dada a distância entre os gerenciadores de nível mais baixo e os recursos em si.

Chaterjee (1999) define um middleware que inclui gerenciamento de recursos com facilidades para controle de admissão, negociação e sintonização de QoS. Sua implementação se baseia em Java, e inclui um gerenciador do sistema e um conjunto de agentes de recursos e agentes de aplicação. O gerenciador do sistema coordena a criação das aplicações distribuídas, as alocações e sintonizações de QoS. Cada agente de recurso é implementado dentro da máquina virtual Java e controla as threads Java de acordo com os comandos recebidos do gerenciador do sistema. Os agentes de aplicação são um conjunto de métodos Java que monitoram a QoS da aplicação e reportam violações ao seu respectivo agente

PUC-Rio - Certificação Digital Nº 0321123/CA

de recurso. O gerenciador do sistema recebe as notificações e decide sobre as necessidades de sintonização, que são propagadas de volta aos agentes de recurso. Chaterjee (1999) é uma proposta semelhante a Cardei (2000), a não ser pela limitação da hierarquia de gerenciadores de serviços que se resume aos agentes de recursos e ao gerenciador do sistema. As semelhanças se estendem às lacunas descritas no parágrafo anterior, também presentes nesse trabalho.

Pal (2000) apresenta um framework para a especificação de QoS de interações entre objetos CORBA. Uma de suas implementações é combinada com o middleware TAO, que é uma implementação Real-time CORBA capaz de mapear seu esquema de prioridades em serviços DiffServ (Blake, 1998) e IntServ (Braden, 1994). Pal (2000) possui mecanismos de sintonização que modificam as prioridades dinamicamente, de acordo com mudanças nas condições de tráfego na rede. A outra implementação inclui também um mecanismo de reserva de CPU baseado diretamente em um sistema operacional com escalonador de processos periódico. Em ambos os casos, a provisão de QoS pode ser especificada sob aspectos, por meio da linguagem QDL (QoS Description Language). Cada aspecto pode ser especificado por diferentes linguagens, separadamente, tais como uma linguagem para especificação de contratos de QoS, outra para a especificação do comportamento de sintonização de QoS, etc. Apesar da facilidade na programação dos mecanismos de QoS, Pal (2000) não define um modelo de recurso uniforme. Por outro lado, apresenta a opção de um gerenciamento de CPU de alta granularidade ao integrar-se com um sistema operacional de tempo real específico.

Kon (2000) propõe um sistema operacional distribuído, chamado 2k, orientado a objetos, com alto grau de configurabilidade suportado pelo middleware reflexivo DynamicTAO. Diferentemente das outras abordagens, 2K inclui um sistema operacional micronúcleo para melhor integrar as abstrações do middleware com as políticas de compartilhamento dos recursos. Visando uma maior portabilidade, 2K pode também ser implementado apenas como uma camada sobre um middleware CORBA. O modelo de gerenciamento de recursos define gerenciadores de recursos locais que exportam os recursos de hardware de um nó para o sistema distribuído, que por sua vez é observado como um todo por um único gerenciador de recursos global. Os gerenciadores locais são responsáveis por realizar as tarefas de controle de admissão, negociação, reserva e

PUC-Rio - Certificação Digital Nº 0321123/CA

escalonamento em um nó. No trabalho é citado o gerenciador baseado no escalonador de processos de Nahrstedt (1998), no nível do usuário. Não foram especificados outros tipos de recursos além da CPU, nem um modelo de recurso que padronize a organização de políticas de compartilhamento de recursos.

Uma funcionalidade desejável em plataformas com suporte a QoS é a adaptabilidade para a provisão de novos serviços, no intuito de acompanhar a evolução das aplicações e de suas necessidades. A importância da adaptabilidade em sistemas operacionais fica também evidenciada no desenvolvimento de redes programáveis e sua conseqüente demanda por roteadores extensíveis (Gottlieb, 2002).

Adaptações podem ser submetidas a sistemas operacionais sob várias formas, desde parâmetros de configuração do sistema até porções de código dinamicamente ligadas ao código do sistema. O poder de expressividade do comportamento desejado aumenta junto ao grau de permissividade da solução de adaptação. Assim, aspectos de segurança devem ser observados, dado que adaptações mal construídas ou mal intencionadas podem comprometer o funcionamento de todo o sistema.

Sistemas com micronúcleo (microkernel) permitem a adaptabilidade de várias tarefas do sistema operacional facilmente, já que grande parte delas é implementada por processos de nível do usuário. Denominados serviços do micronúcleo, tais processos se comunicam por meio de um canal de troca de mensagens controlado pelo micronúcleo. A baixa velocidade nessa comunicação e a sobrecarga de mensagens no micronúcleo são as principais críticas a esse modelo. Mas, sem dúvida, uma de suas virtudes é a adaptabilidade, já que bastariam o encerramento de um processo e o disparo de um novo para consolidar modificações no comportamento do sistema.

Por sua vez, os sistemas operacionais de propósito geral não foram projetados para suportar adaptações em tempo de operação, já que a maioria deles possui núcleo monolítico. Entretanto, alguns núcleos monolíticos, como Linux (Love, 2005), permitem a inserção de código objeto sob demanda, em tempo de execução. Os módulos de núcleo (Salzman, 2007), como são denominadas as porções de código, têm como objetivo retirar do núcleo partes que podem ou não ser carregadas, de acordo com as configurações da plataforma de hardware. Assim, o núcleo pode ser reduzido, levando a uma economia no uso de memória

PUC-Rio - Certificação Digital Nº 0321123/CA

dedicada ao sistema. Os módulos de núcleo são dedicados à implementação de *drivers* de dispositivos e sistemas de arquivos. Se o código do núcleo do sistema for convenientemente modificado, os módulos podem ser usados em outras funções, como escalonadores de recursos, pilhas de protocolos, etc.

De qualquer forma, tanto núcleos monolíticos quanto micronúcleos não implementam várias tarefas importantes de gerenciamento de recursos nos módulos ou serviços, mas sim, na parte invariável do núcleo. Outras propostas encontradas na literatura descrevem novas abordagens para a concepção de sistemas operacionais com foco sobre a adaptabilidade, que de certa forma vêm a beneficiar o gerenciamento adaptável de recursos. Engler (1995) propõe uma arquitetura na qual as aplicações utilizam bibliotecas de nível do usuário que implementam as funções do sistema, cabendo ao núcleo mínimo apenas a multiplexação dos dispositivos entre as aplicações. O gerenciamento dos recursos é implementado no nível do usuário e poderia ser facilmente adaptado pelas aplicações. Bershad (1995) descreve um sistema operacional extensível que suporta adaptações escritas em linguagem Modula-3, a qual é fortemente tipada e impõe contratos de interface entre módulos de código, o que leva a proteção de memória. As extensões devem ser assinadas digitalmente por um compilador confiável e poderiam implementar políticas de gerenciamento de recursos específicas por aplicação. Saltzer (1996) também aceita extensões de núcleo, denominadas “enxertos” (*grafts*). Enxertos são arquivos objeto, também gerados por compiladores confiáveis e assinados digitalmente, porém a linguagem utilizada é C++, que não tem uma tipagem segura. A proteção de memória é forçada por meio de técnicas de caixa de areia (sandboxing) e as adaptações são realizadas sob contexto de transações, o que permite a restauração de estados consistentes, caso a execução do módulo seja malsucedida.

Algumas das técnicas apresentadas no parágrafo anterior foram utilizadas em extensões de sistemas operacionais de propósito geral com o objetivo específico de oferecer alguma adaptabilidade em subsistemas relevantes para a QoS. Por exemplo, West (2002) propõe modificações em um núcleo de propósito geral, baseado no Linux, utilizando uma abordagem similar a Bershad (1995). Para o desenvolvimento dos módulos, é utilizada a linguagem Popcorn, que é fortemente tipada e não permite o uso de ponteiros para áreas arbitrárias de memória, promovendo a proteção requerida. Um compilador confiável assina

PUC-Rio - Certificação Digital Nº 0321123/CA

digitalmente o código objeto, que é inserido no sistema como módulo de núcleo. A diferença notável em relação aos demais trabalhos está no fato de que as extensões de West (2002) são voltadas ao suporte a sintonização de QoS e, assim, a infra-estrutura no núcleo é projetada para facilitar adaptações sobre monitores e parcelas de utilização de recursos. Entretanto, não é definido na arquitetura um modelo de gerenciamento de recursos, a não ser pelas interfaces de adaptação de monitores e manipuladores de reservas. Por esse motivo, acaba persistindo o problema da heterogeneidade dos recursos e não é possível realizar adaptações sobre outras políticas de gerenciamento, como os escalonadores de recursos.

Por sua vez, Goyal (1996a) e Regehr (2001a) oferecem adaptações especificamente sobre os escalonadores de processos. Ambos também propõem a modificação do núcleo do sistema, de forma a tirar proveito dos raros mecanismos de adaptação. Em especial, Regehr (2001a) apresenta uma infra-estrutura de suporte a escalonadores hierárquicos dinamicamente carregáveis e uma interface de programação para os escalonadores. O mecanismo de controle de admissão deve ser embutido no escalonador. Nenhuma linguagem especial foi adotada para a implementação dos módulos, e por isso a API não oferece métodos de verificação de segurança, nem do comportamento dos escalonadores. Em Regehr (2001b) o trabalho foi estendido para descrever um modelo de garantias. A garantia provida por um escalonador a seus processos pode ser mapeada em uma outra, genérica, pertencente ao modelo de garantias (e.g.: compartilhamento proporcional, reserva temporal, time sharing...). Isso permite que garantias possam ser convertidas entre si e, curiosamente, que a garantia real, resultado da composição de uma hierarquia de escalonadores, possa ser inferida. Assim, é possível verificar não somente se um escalonador da hierarquia é conveniente para uma aplicação segundo suas necessidades, mas também se toda a hierarquia que leva ao escalonador também o é. Infelizmente, não é possível a análise do código do escalonador para mapeá-lo nas garantias do modelo, o que torna código e associação de garantias independentes entre si.

O desenvolvimento de um módulo para a execução de tarefas de gerenciamento de recursos em espaço do núcleo traz, além do problema de segurança, toda a complexidade das estruturas de um sistema operacional. Os programadores devem ter conhecimento das peculiaridades internas do sistema, o que se torna indesejável mesmo para desenvolvedores de software. No âmbito da

PUC-Rio - Certificação Digital Nº 0321123/CA

provisão de QoS, tais mecanismos seriam desenvolvidos por engenheiros de serviços de telecomunicações e administradores de sistemas, que possuem total domínio do problema do gerenciamento, mas não são programadores especialistas. Por isso, vem sendo cada vez mais comum a adoção de linguagens de domínio específico em vários aspectos dessa área, como especificação de parâmetros de QoS (Jin, 2004), especificação de arquitetura de aplicações com QoS (Duran-Limon, 2004) e especificação de arquiteturas de provisão de QoS (Soares Neto, 2003).

Particularmente, Barreto (2002) apresenta uma DSL para a programação de escalonadores de processos, denominada Bossa, e uma infra-estrutura para a compilação e inserção destes escalonadores em um núcleo Linux. Em Lawall (2005), o trabalho é refinado para descrever como uma hierarquia de escalonadores pode ser construída a partir da Bossa e em Lawall (2006) a DSL é reestruturada em uma arquitetura modular, de forma a organizar as partes comuns no desenvolvimento de sub-famílias de escalonadores (e.g.: periódicos, proporcionais e *time sharing*). Nota-se que com a adoção de uma DSL, as questões acerca da segurança nas adaptações foram minimizadas, já que não há ponteiros de memória e várias verificações de comportamento podem ser realizadas em tempo de compilação. Mas a arquitetura não faz referência ao uso de assinaturas digitais sobre o código. O objetivo do trabalho como um todo não é exatamente o gerenciamento adaptável de CPU com suporte a QoS, mas permitir que usuários e administradores de sistemas operacionais desenvolvam com facilidade novas estratégias de escalonamento, adequadas às suas aplicações. Por isso, não são citados outros mecanismos de gerenciamento de recursos, como controle de admissão, negociação e sintonização de QoS.

A linguagem Bossa possui forte expressividade no domínio do gerenciamento de processos, o que pode ser observado desde a terminologia adotada até os eventos de notificação especificados. Por esse motivo, a linguagem não é genérica o bastante para ser adotada no escalonamento de outros recursos. Há falta de generalidade também na definição de hierarquias, já que um escalonador desenvolvido em Bossa não pode ser colocado em qualquer nível de uma hierarquia. Isso se deve ao fato de que escalonadores intermediários (virtuais) são programados exclusivamente para escalonar outros escalonadores, não podendo ser usados como escalonadores de processos (nós-folha da hierarquia). O

PUC-Rio - Certificação Digital Nº 0321123/CA

mesmo vale para escalonadores de processos, que não podem ser usados como escalonadores virtuais. Finalmente, não é mencionado o suporte a adaptação em tempo de operação pela infra-estrutura implementada, sem que haja a necessidade de reinicialização do sistema.

Almesberger (2002) especifica uma DSL simples, dedicada à configuração do sistema de controle de tráfego do Linux (Almesberger, 1999), que gerencia as filas de pacotes das interfaces de rede. Baseada em XML, ela permite a criação de hierarquias de disciplinas de enfileiramento (escalonadores) previamente disponíveis no sistema, assim como a distribuição da alocação das parcelas do recurso entre cada um dos escalonadores. Filtros de classificação também são configurados por arquivos XML, que, na realidade, vêm a substituir os comandos do programa "tc" (utilitário de configuração do controle de tráfego do Linux). O objetivo é criar uma arquitetura de controle de tráfego que inclui um passo de simulação, capaz de ler os arquivos de configuração em XML e demonstrar os seus resultados antes da implantação.

O algoritmo LDS (Load Dependent Scheduler) (Barria, 2001) é capaz de emular a operação de vários algoritmos de escalonamento, a partir da modificação dos seus próprios parâmetros de operação. Mesmo concebido com o objetivo de se tornar uma ferramenta para análise e comparação do desempenho de algoritmos de escalonamento de pacotes, o LDS é também uma boa solução para adaptação, pois a inserção de um novo escalonador pode ser feita sem a necessidade de programação. Além de trazer benefícios para a segurança, o algoritmo é genérico o bastante para ser adotado como escalonador de outros recursos. Porém, não é capaz de emular algoritmos periódicos, ou outros algoritmos que se baseiam em algum tipo de temporização.

PUC-Rio - Certificação Digital Nº 0321123/CA