UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

GUSTAVO CIPRIANO MOTA SOUSA

<Título do Trabalho>

<Subtítulo do Trabalho>

Goiânia
2012

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: <Título do Trabalho> – <Subtítulo do Trabalho>

Autor(a): Gustavo Cipriano Mota Sousa

Goiânia, Dia de Setembro de 2012.

Gustavo Cipriano Mota Sousa – Autor

Fábio Moreira Costa – Orientador

<Nome do Co-orientador> – Co-Orientador

GUSTAVO CIPRIANO MOTA SOUSA

<Título do Trabalho>

<Subtítulo do Trabalho>

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em <Nome do Programa de Pós-Graduação>.

Área de concentração: <Área de Concentração>.

Orientador: Prof. Fábio Moreira Costa

Co-Orientador: Prof. <Nome do Co-orientador>

Goiânia
2012

GUSTAVO CIPRIANO MOTA SOUSA

<Título do Trabalho>

<Subtítulo do Trabalho>

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em <Nome do Programa de Pós-Graduação>, aprovada em Dia de Setembro de 2012, pela Banca Examinadora constituída pelos professores:

Prof. Fábio Moreira Costa
Instituto de Informática – UFG
Presidente da Banca

Prof. <Nome do Co-orientador>
<Nome da Unidade Acadêmica do Co-orientador> – <Sigla da Universidade do Co-orientador>

Prof. Ricardo Couto Antunes da Rocha
Instituto de Informática – UFG

Prof. Nelson Souto Rosa
Centro de Informática – UFPE

Gustavo Cipriano Mota Sousa

<Texto com um perfil resumido do autor do trabalho. Por exemplo: (Graduou–se em Artes Cênicas na UFG - Universidade Federal de Goiás. Durante sua graduação, foi monitor no departamento de Filosofia da UFG e pesquisador do CNPq em um trabalho de iniciação científica no departamento de Biologia. Durante o Mestrado, na USP - Universidade de São Paulo, foi bolsista da FAPESP e desenvolveu um trabalho teórico na resolução do Problema das Torres de Hanói. Atualmente desenvolve soluções para problemas de balanceamento de ração para a pecuária de corte.)>

<Dedicatória do trabalho a alguma pessoa, entidade, etc.>

Agradecimentos

<Texto com agradecimentos àquelas pessoas/entidades que, na opinião do autor, deram alguma contribuíção relevante para o desenvolvimento do trabalho.>

<Epígrafe é uma citação relacionada com o tópico do texto>

<Nome do autor da citação>,
<Título da referência à qual a citação pertence>.

Resumo

Sousa, Gustavo Cipriano Mota. **<Título do Trabalho>**. Goiânia, 2012. 34p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

<Resumo do trabalho>

Palavras–chave

<Palavra chave 1, palavra chave 2, etc.>

Abstract

Sousa, Gustavo Cipriano Mota. **<Work title>**. Goiânia, 2012. 34p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

A sketchy summary of the main points of the text.

Keywords

<Keyword 1, keyword 2, etc.>

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Códigos de Programas

CAPÍTULO 1

Introdução

Em abordagens tradicionais de desenvolvimento, software é comumente implementado por meio de codificação em linguagens de programação de propósito geral. Apesar de amplamente empregadas, estas linguagens se baseiam em conceitos relacionados à plataforma de implementação e não ao problema a ser resolvido. Esta distância semântica entre o problema em questão e as abstrações disponíveis nas linguagens é uma das causas da complexidade envolvida na especificação de uma solução para um determinado problema [6], o que por sua vez exige um grande esforço durante o desenvolvimento de software.

Com o intuito de amenizar esta complexidade, abordagens dirigidas por modelos propõem o uso de modelos como um meio de preencher a lacuna entre o domínio do problema e a plataforma de implementação. Para isto, essas abordagens se apoiam na construção de modelos descritos a partir de abstrações próximas ao domínio do problema, e no processamento automatizado desses modelos em abstrações da plataforma de implementação.

O termo *Model-Driven Engineering* (MDE) é utilizado para descrever esse conjunto de abordagens que utilizam modelos como os principais artefatos no processo de engenharia de software [8]. Nessas abordagens, o uso de modelos não se limita à documentação ou ao projeto do software, mas também se presta ao seu desenvolvimento. Assim sendo, a atividade principal de desenvolvimento passa a ser a de modelagem em termos de construções relativas ao domínio do problema em substituição à de codificação baseada em elementos da plataforma de implementação.

Apesar disso, abordagens de MDE não restringem o uso de modelos à etapa de desenvolvimento dentro do ciclo de vida de software. De forma geral, o principal objetivo dessas abordagens é abstrair dos desenvolvedores as complexidades essencialmente tecnológicas, que guardam mais relação com as capacidades da plataforma de implementação do que com o problema a ser resolvido. Nessa perspectiva, modelos também podem ser empregados para representar aspectos da execução do software, permitindo o emprego de abstrações mais apropriadas para monitorar ou adaptar um sistema em tempo de execução [2].

Devido a essas capacidades, essas abordagens tem sido propostas como uma forma de lidar com a crescente demanda por aplicações complexas, notadamente aquelas que envolvem elementos de computação distribuída em ambientes heterogêneos e em constante mudança, apresentando requisitos de disponibilidade e segurança, adaptabilidade e evolução, entre outros [6] [3]. A construção desse tipo aplicação empregando uma abordagem tradicional exige um grande esforço, e é comumente desempenhada por especialistas em desenvolvimento de software. O emprego de abstrações próximas ao domínio do problema possibilitado pelas tecnologias de MDE abre espaço para um novo cenário, onde usuários especialistas de domínio, ou até mesmo usuários finais podem construir este tipo de aplicações através da manipulação de modelos de alto nível.

Para a construção destes modelos são empregadas linguagens especificamente projetadas para capturar os conceitos de um determinado domínio. Os modelos, por sua vez, são transformados de forma automatizada em construções da plataforma de implementação. No entanto, para que o processamento desses modelos seja realizado com um alto grau de automatização é necessário que essas transformações incorporem conhecimento específico de domínio [6]. Portanto, para se obter os benefícios previstos pelas abordagens de MDE, é preciso construir linguagens e mecanismos de processamento específicos de domínio para uma determinada classe de aplicações. Essa infraestrutura possibilita que modelos abstratos, descritos diretamente pelos usuários, sejam executados de forma automatizada.

Apesar dessa abordagem trazer um grande benefício para a construção de aplicações em um determinado domínio de negócio, um grande esforço ainda é exigido na construção dos mecanismos de transformação ou plataformas para execução de modelos específicos desse domínio. Devido à necessidade de incorporar conhecimento específico do domínio a ser tratado, esses mecanismos são comumente descritos em linguagens de programação de propósito geral ou em linguagens de transformação genéricas, o que aumenta a complexidade na sua construção.

A despeito disso, diferentes classes de aplicações em domínios diversos podem apresentar necessidades similares de processamento. O processamento de modelos de alto nível centrados no usuário usualmente envolve operações de transformação e negociação de modelos, adaptação em tempo de execução, além de outras específicas do domínio de negócio. Logo, a aplicação de uma abordagem dirigida por modelos à construção destes mecanismos específicos de domínio que compartilham algumas características pode trazer benefícios interessantes. Ao se aplicar tal abordagem, estendemos o uso de modelos para além da descrição de aplicações, possibilitando também a descrição da plataforma para execução dessas aplicações. Além de simplificar a construção dessas plataformas, esta abordagem também facilita sua evolução com o intuito de atender ao surgimento de novos requisitos, à mudanças na plataforma de implementação, ou à evolução da linguagem

específica de domínio correspondente.

Uma categoria interessante de aplicações que podemos destacar está relacionada ao provimento de serviços a partir de um conjunto heterogêneo de recursos. Uma aplicação desta categoria conta com um conjunto de recursos que apresentam diferentes características, e tem como objetivo fornecer um serviço de alto nível ao usuário, que abstraia detalhes dos recursos utilizados para efetivamente realizar o serviço desejado. Assim sendo, tais aplicações precisam ser capazes de gerenciar completamente os recursos, abstraindo do usuário detalhes da escolha e configuração de recursos, bem como seu monitoramento e eventual substituição.

A Máquina Virtual de Comunicação (*Communication Virtual Machine* - CVM) é um exemplo de plataforma para construção e execução de aplicações que se enquadra nessa categoria. A CVM é uma plataforma de *middleware* específica de domínio que emprega uma abordagem dirigida por modelos para possibilitar a criação de serviços de comunicação a partir de modelos de alto nível definidos pelo usuário.

Como uma plataforma dirigida por modelos, a CVM atua a partir do processamento de modelos descritos em uma linguagem específica de domínio chamada Linguagem de Modelagem de Comunicação (*Communication Modeling Language* - CML) [5]. Um modelo em CML descreve um cenário de comunicação, incluindo participantes, tipos de dados, e de mídia de um serviço de comunicação a ser realizado. Detalhes sobre os dispositivos ou tecnologias de comunicação que serão utilizados internamente para estabelecer a comunicação não precisam ser descritos em um modelo em CML.

A CVM é considerada uma plataforma de *middleware* de comunicação centrada no usuário devido ao alto nível de abstração de seus modelos, que podem ser facilmente construídos por usuários finais [4]. A partir de um determinado modelo descrito em CML, a CVM é capaz de fornecer o serviço desejado de forma automatizada. Além disso, um modelo em CML pode ser modificado durante o curso de uma sessão de comunicação, desencadeando adaptações da CVM para atender ao cenário atualizado descrito pelo modelo modificado. Isso qualifica os modelos em CML como modelos de desenvolvimento e de tempo de execução [2].

Para realizar um serviço de comunicação de forma automatizada e transparente ao usuário a CVM precisa realizar diversas operações desde transformações e negociação de modelos até a seleção e gerenciamento de provedores de comunicação. A CVM conta com uma arquitetura em camadas que possuem responsabilidades bem definidas e encapsulam os principais aspectos envolvidos na realização da comunicação.

Essa mesma estratégia também é empregada na área de micro-grids [1] para possibilitar a construção de aplicações de gerenciamento dos elementos de uma rede elétrica inteligente local diretamente pelos usuários finais. Isto demonstra o potencial dessa estratégia para a construção de plataformas para provimento de serviços descritos

por modelos a partir de um conjunto heterogêneo de recursos.

Neste trabalho, nos apoiamos nas soluções empregadas pela CVM para propor uma abordagem dirigida por modelos para o desenvolvimento de plataformas específicas de domínio voltadas para a construção e execução de aplicações que fornecem um serviço descrito por meio de um modelo centrado no usuário a partir de um conjunto heterogêneo de recursos. Ao fazê-lo, buscamos uma forma sistematizada de capturar as soluções empregadas pela CVM de forma que possam ser reutilizadas seguindo uma abordagem baseada em modelos. Assim sendo, a abordagem proposta prevê que uma plataforma seja realizada através de uma máquina virtual com uma arquitetura de camadas, semelhante à exibida pela CVM.

Para realizar a abordagem proposta, prevemos a construção de meta-modelos que permitam descrever as camadas existentes na arquitetura proposta. Estes meta-modelos devem possuir construções relativas às responsabilidades de suas respectivas camadas. Para demonstrar a viabilidade desta abordagem, construímos um meta-modelo que permite a descrição da camada mais inferior desta arquitetura, que tem como responsabilidade gerenciar os recursos disponíveis. Além disso, demonstramos a utilização do meta-modelo por meio da construção e processamento automatizado de uma instância que equivale à camada correspondente na CVM.

1.1 Objetivos

O objetivo geral deste trabalho é propor uma abordagem para a construção dirigida por modelos, de máquinas virtuais capazes de prover serviços a partir de um conjunto heterogêneo de recursos. As máquinas virtuais construídas a partir desta abordagem são projetadas para processarem modelos de alto nível descritos em uma linguagem específica de domínio. A abordagem proposta tem como objetivo empregar técnicas de MDE para capturar as soluções existentes na CVM de forma sistematizada.

Além disso, com o intuito de demonstrar a viabilidade de tal proposta, este trabalho tem os seguintes objetivos específicos:

- Projetar um meta-modelo para definição da camada de gerenciamento de recursos dentro da abordagem proposta.
- Construir uma instância do meta-modelo proposto que represente a implementação da CVM para esta camada com o objetivo de demonstrar a abordagem proposta.
- Desenvolver um ambiente de execução que possibilite a execução de instâncias do meta-modelo construído.
- Demonstrar a genericidade da abordagem proposta (o que pode ser feito extrapolando conclusões obtidas com a NCB para outro domínio, e.g., microgrid)???

Observação 1.1 *Coloquei na estrutura de tópicos.*
São esses mesmos os objetivos?

1.2 Contribuições

- Proposta de uma forma sistematizada (baseada em modelos) de reutilizar as soluções existentes na CVM (arquitetura em camadas/transformações/negociação) em outros domínios

1.3 Metodologia

Com o intuito de atingir os objetivos estabelecidos, a seguinte metodologia de pesquisa é adotada:

- Revisão do estado de arte de abordagens dirigidas por modelo, seu emprego em plataformas de *middleware*, e sua relação com outras abordagens.
- Estudo das diferentes abordagens dirigidas por modelos, incluindo conceitos e técnicas associadas.
- Estudo da plataforma CVM e sua interpretação à luz das abordagens dirigidas por modelo.
- Proposta de uma abordagem para modelagem comportamental de um serviço cuja a estrutura é descrita através de modelos de alto nível. (ainda não encontrei como explicar isso)

1.4 Contribuições

Aplicar a abordagem dirigida por modelos à construção da plataforma que executa modelos... Aplicando a arquitetura em camadas...

1.5 Organização da dissertação

CAPÍTULO 2

Referencial teórico

2.1 Model driven engineering

Visão geral de MDE

2.1.1 Modelos

Dimensões de modelagem

2.1.2 Meta-modelagem

Linguagens

2.1.3 Technical spaces

MOF/EMF/...

2.2 CVM

camadas

2.3 Reflexão computacional

2.4 Computação autônoma

CAPÍTULO 3

Uma abordagem dirigida por modelos para construção de máquinas virtuais dirigidas por modelos

Este capítulo descreve a abordagem proposta por este trabalho para a construção dirigida por modelos de máquinas virtuais capazes de executar aplicações descritas através de modelos de alto nível. A seção 3.1 descreve de forma geral a abordagem proposta e o cenário onde ela se posiciona. A seção 3.2, por sua vez, detalha parte do escopo da abordagem proposta.

3.1 Visão geral

Aplicações que integram diversos recursos com o intuito de prover uma funcionalidade de alto nível de forma transparente ao usuário precisam ser capazes de tratar adequadamente a seleção, configuração, adaptação e gerenciamento dos recursos disponíveis. O desenvolvimento deste tipo de aplicação, utilizando abordagens tradicionais de programação exige a codificação (em linguagem de propósito geral) do comportamento desejado para todos estes aspectos, além dos requisitos específicos do domínio de negócio da aplicação.

O emprego de técnicas de MDE possibilita reduzir significativamente a complexidade envolvida na construção de aplicações com estas características. A concepção de uma linguagem de modelagem específica de domínio e mecanismos capazes de processá-la possibilita que aplicações dentro de um dado domínio sejam descritas a partir de modelos de alto nível baseados em conceitos próprios do domínio em questão.

Para tornar isto possível, o conhecimento do domínio é incorporado à linguagem e aos mecanismos responsáveis pelo seu processamento, tornando desnecessária a descrição desse conhecimento durante a concepção de um modelo que represente uma determinada aplicação. A CML, como um exemplo de linguagem de modelagem específica do domínio de comunicação, incorpora em seu projeto elementos diretamente relacionados

à descrição de cenários de comunicação entre pessoas. A CVM por sua vez é a plataforma capaz de processar esta linguagem e, para isso, também precisa incorporar todas as operações envolvidas na realização de um serviço de comunicação já apresentadas.

Assim sendo, a aplicação dessa abordagem ao domínio de comunicação permite a descrição dos requisitos e estrutura de uma comunicação através de construções relativas a este domínio. No entanto, o comportamento envolvido na realização do serviço, incorporado à plataforma, ainda é codificado em uma linguagem de propósito geral. Logo, ajustar ou evoluir esse comportamento para atender novos cenários ou necessidades exige um grande esforço de codificação.

Neste trabalho, propomos que esse comportamento também seja definido através de modelos, de forma a reduzir a complexidade envolvida em sua especificação e manipulação. Desta forma, o comportamento envolvido no processamento de um modelo na linguagem específica de domínio que se encontrava incorporado ao código da plataforma passa a ser descrito a partir de um modelo. Portanto, a abordagem dirigida por modelos é aplicada não só à descrição de aplicações, mas também à descrição de plataformas projetadas para executá-las.

Ao fazê-lo, buscamos uma abordagem dirigida por modelos para a construção de plataformas para definição e execução de aplicações descritas através de modelos específicos de um determinado domínio de negócio. Esta abordagem visa a descrição de plataformas como a CVM, capazes de fornecer um serviço de alto nível descrito através de um modelo centrado no usuário a partir de um conjunto heterogêneo de recursos.

Uma linguagem, como a CML, tem como principal elemento a sua sintaxe abstrata que é comumente descrita através de um meta-modelo. Isto possibilita que a CML seja modificada através da manipulação do seu meta-modelo, que também é um modelo. Ao se integrar o meta-modelo de uma linguagem específica de domínio à um modelo que descreve uma máquina virtual capaz de executá-la temos uma linguagem e plataforma para desenvolvimento de aplicações em um determinado domínio que podem ser ajustados e evoluídos através da manipulação de modelos.

No entanto, para que seja vantajoso o emprego de uma linguagem de modelagem para a descrição de uma máquina virtual esta linguagem deve possuir uma abrangência suficiente para que possa ser utilizada em diferentes domínios de negócio. Ao mesmo tempo, tal linguagem não deve ser muito genérica, de forma que não torne a manipulação de modelos tão complexa quanto a codificação em uma linguagem de propósito geral.

Considerando isto, essa linguagem deve conter construções capazes de descrever os aspectos comportamentais que independem do domínio de negócio e que estão envolvidos no fornecimento de serviços de alto nível descritos a partir de modelos construídos diretamente pelo usuário. Esses aspectos, que englobam tarefas como transformações de modelos, negociação de configurações apropriadas, monitoramento do ambiente, seleção

e configuração de recursos, entre outros, podem ser tratados como um domínio técnico. Deste modo, tal linguagem pode ser considerada uma linguagem específica de um domínio que abrange os aspectos técnicos envolvidos na realização de serviços transparentes de alto nível a partir de um conjunto heterogêneo de recursos.

Tendo em vista estas necessidades, propomos uma arquitetura genérica para a definição dessa categoria de máquinas virtuais dirigidas por modelos centrados no usuário. Assim como a CVM, tal arquitetura também se baseia em camadas com responsabilidades bem definidas. Na abordagem proposta, as camadas desta arquitetura são definidas através de modelos cujas construções estão relacionadas às responsabilidades relativas à cada camada. A Figura 3.1 ilustra esta arquitetura de Máquinas Virtuais centradas no Usuário (*User-centric Virtual Machine* - UVM) onde temos as seguintes camadas:

- Interface com o Usuário (*User Interface* - UI), que provê uma interface externa para utilização da plataforma. Além disso, esta camada possibilita a definição e gerenciamento de modelos;
- Mecanismo de Síntese (*Synthesis Engine* - SE), que possui como principal responsabilidade a transformação de um modelo declarativo fornecido pela UI em uma representação algorítmica a ser executada pela camada inferior;
- *Middleware* Centrado no Usuário (*User-centric middleware* - UM), que além de executar as requisições geradas pelo Mecanismo de Síntese, também gerencia os serviços providos pela máquina virtual e as tarefas em execução. Essa também é a camada responsável pela aplicação de restrições de segurança, qualidade de serviço, entre outras específicas do domínio de negócio.
- Intermediador de Serviço (*Service Broker* - SB), que é a camada responsável pelo gerenciamento dos recursos. Assim sendo, essa camada tem como objetivo prover uma interface de acesso aos recursos de forma independente da tecnologia empregada por estes, provendo um serviço transparente à camada UM.

Ao imitar a arquitetura da CVM, nos aproveitamos do conhecimento adquirido em relação à separação de responsabilidades necessárias para a realização de serviços no domínio técnico identificado. No entanto, apesar de apresentar a mesma estrutura, a arquitetura proposta se difere por ser independente do domínio de negócio. Além disso, na abordagem proposta, cada camada é descrita através de um modelo. Este modelo, por sua vez, é construído em conformidade com um meta-modelo da camada que contém elementos associados às responsabilidades da camada. Assim sendo, a abordagem dirigida por modelos é empregada para a definição e manipulação de cada camada da plataforma.

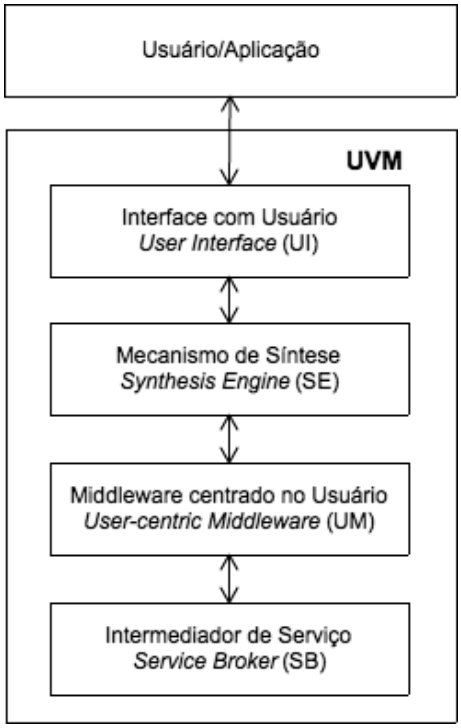

Figura 3.1: *Arquitetura proposta para esta categoria de Máquinas Virtuais centradas no Usuário.*

3.2 Abordagem dirigida por modelos para definição do Intermediador de Serviço

O Intermediador de Serviço é a camada da arquitetura proposta que é responsável pelo gerenciamento dos recursos que serão efetivamente utilizados para prover o serviço solicitado. Cabe à esta camada disponibilizar uma interface de serviços que abstraia as especificidades dos recursos gerenciados para a camada superior. Além disso, o SB deve possuir capacidade de auto gerenciamento, ocultando os detalhes envolvidos na seleção, monitoramento e preparação dos recursos sob sua gerência.

Como uma interface de serviços, o SB se comunica com o UM através de chamadas que podem ser invocadas e eventos que podem ser gerados sinalizando alguma situação. Assim sendo, ao definirmos o comportamento dessa camada, precisamos descrever como serão tratadas as chamadas realizadas pela camada superior, e em que situações serão gerados determinados eventos. Mas além disso, também precisamos descrever como outras tarefas serão realizadas, incluindo o monitoramento e seleção de recursos, manutenção de informações, adaptação da camada, entre outros.

Com isto em mente, neste trabalho, construímos um meta-modelo que possibilita a modelagem do comportamento necessário para atender as responsabilidades definidas para a camada de intermediação de serviço da arquitetura proposta. O meta-modelo em questão contempla a descrição dos seguintes aspectos envolvidos no cumprimento destas

responsabilidades:

- Tratamento de chamadas e eventos.
- Gerenciamento de recursos
- Gerenciamento de estado
- Computação autonômica
- Políticas

Além do meta-modelo, também construímos um ambiente de execução para auxiliar na execução de suas instâncias. Assim sendo, esse ambiente implementa a semântica operacional da linguagem para descrição da camada de intermediação de serviço. O ambiente de execução possibilita que um modelo definido a partir do meta-modelo proposto seja executado.

Desta forma, o meta-modelo e o ambiente de execução construídos permitem que a camada de intermediação de serviço da arquitetura proposta seja definida através de um modelo baseado em construções relacionadas às suas responsabilidades, e possa ser executada e utilizada.

CAPÍTULO 4

Metamodelo do Intermediador de Serviços

A camada de intermediação de serviços é a responsável pelo gerenciamento dos recursos que serão efetivamente utilizados para provisão de serviços. Um intermediador de serviços deve prover uma interface uniforme sobre um conjunto heterogêneo de recursos gerenciados. Essa interface deve abstrair da camada superior as diferenças entre os recursos e a dinâmica envolvida em sua utilização.

Além de disponibilizar uma interface uniforme sobre os recursos, a camada de intermediação de serviços deve apresentar um certo grau de auto gerenciamento. O auto gerenciamento da camada é essencial para abstrair da camada superior as operações de manutenção dos recursos. Estas incluem a seleção de recursos apropriados, bem como sua preparação para atender às solicitações recebidas. Além disso, uma camada auto gerenciável é capaz de monitorar seus recursos e adaptar sua configuração interna para atender à essas solicitações. Isto possibilita à camada otimizar a utilização de seus recursos, bem como a identificação e recuperação de falhas.

Observação 4.1 *A ser colocado no início dessa seção:*

- Como mencionado anteriormente, o Intermediador de Serviços é a camada que possui a responsabilidade de prover uma interface uniforme que abstraia a heterogeneidade dos recursos disponíveis e a dinâmica envolvida em sua utilização.

- Assim sendo, ao definirmos o comportamento dessa camada, precisamos descrever como serão tratadas as chamadas realizadas pela camada superior, e em que situações serão gerados determinados eventos. Mas além disso, também precisamos descrever como outras tarefas serão realizadas, incluindo o monitoramento e seleção de recursos, manutenção de informações, adaptação da camada, entre outros.

descrever a idéia por trás do metamodelo: - definir como serão tratados os eventos/chamadas; - definir quais os recursos que serão gerenciados e suas características; - definir quais os tipos de dados serão manipulados; - definir como serão monitorados os recursos de forma autonômica; - definir como serão escolhidos recursos de acordo com políticas;

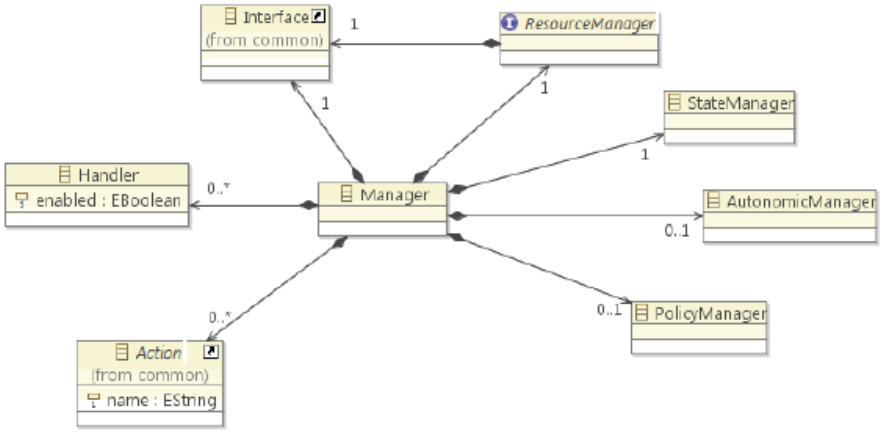

Figura 4.1: *Principais elementos do metamodelo do intermediador de serviços.*

Com o intuito de permitir a instanciação do metamodelo e sua execução - construído o metamodelo usando EMF; - restrições de semântica estática; - construído ambiente de execução que define a semântica operacional;

O metamodelo projetado se estrutura em torno de um elemento principal denominado Manager. Uma instância dessa classe, aqui tratado como gerenciador, define um escopo para o gerenciamento de recursos e agrupa outros elementos que definem atribuições específicas da camada de intermediação de serviços.

A figura 4.1 ilustra os principais elementos desse metamodelo, onde é possível identificar o Manager que agrupa os seguintes elementos:

- Interface que define as chamadas disponibilizadas pela camada e os eventos que podem ser gerados por esta
- Action/Handler que define como as chamadas realizadas a camada e os eventos gerados pelos recursos serão tratados pela camada
- ResourceManager que define os recursos que serão gerenciados por um determinado Manager, incluindo suas interfaces, e como são obtidos.
- StateManager que define os tipos de dados que precisam ser mantidos pela camada para prover seus serviços.
- AutonomicManager que define o auto gerenciamento da camada.
- PolicyManager que define as políticas para seleção de recursos e quando estas devem ser avaliadas

Uma vez construída uma instância da classe Manager, esta pode ser utilizada como um recurso para outra instância de Manager. Isto possibilita a construção de uma hierarquia de gerenciadores que podem ser empregados para sucessivamente prover serviços a partir de outros recursos e gerenciadores de recursos. Além disso, essa característica permite a modularização da camada e a reutilização de gerenciadores de recursos na

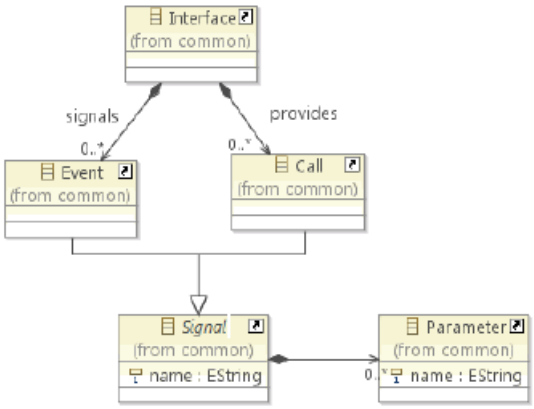

Figura 4.2: *Elementos do metamodelo para descrição de interfaces.*

construção de outros gerenciadores de mais alto nível. A construção da camada de intermediação de serviços se dá a partir da indicação de qual será o seu gerenciador principal, que por sua vez pode empregar outros gerenciadores como recursos subjacentes.

As seções abaixo descrevem em detalhes os principais elementos do metamodelo identificados acima.

4.1 Interface

No metamodelo proposto, a interface para utilização de um gerenciador é definida por meio de chamadas que são providas, e eventos que podem ser sinalizados. Este tipo de interface segue a mesma abordagem empregada pela CVM para comunicação entre camadas [5]. Dessa forma, a utilização de um gerenciador se dá através da realização de chamadas disponíveis e tratamento de eventos gerados.

A interface de uma camada de intermediação de serviços, por sua vez, é definida pela interface de seu gerenciador principal e, portanto, também interage com a camada superior da forma descrita acima. Além disso, as interfaces para utilização dos recursos gerenciados também é descrita da mesma forma, o que possibilita que um gerenciador seja tratado como um recurso.

A figura 4.2 ilustra as classes do metamodelo relacionadas à descrição de interfaces. A classe Interface é utilizada para descrever a interface de um gerenciador ou recurso. Essa classe agrupa um conjunto de chamadas disponibilizadas e eventos que podem ser sinalizados. Chamadas e eventos, denominados sinais, são representados respectivamente através das classes Call e Event que apresentam como característica comum o fato de possuírem um nome e um conjunto de parâmetros, e por isso herdam estes atributos da classe Signal. Os parâmetros de um sinal, por sua vez, são definidos por meio da classe Parameter.

4.2 Tratamento de sinais

O comportamento de uma camada de intermediação de serviços é definido pela forma como esta reage às chamadas realizadas pela camada superior e aos eventos sinalizados pelos recursos. O metamodelo proposto disponibiliza construções que permitem definir uma ação a ser realizada como resposta à um determinado sinal. Esta estratégia se inspira em uma architetura dirigida por eventos [].

O tratamento de sinais na camada de intermediação é definido por meio das classes Signal, Handler e Action. Enquanto os sinais a serem tratados por um gerenciador são descritos como parte da interface do gerenciador e dos recursos gerenciados, os últimos são diretamente agrupados no gerenciador. A figura **??** mostra esses elementos estão organizados no metamodelo.

Uma ação representa uma operação que pode ser executada por um gerenciador, e é definida por meio da classe abstrata Action, que possui as seguintes subclasses:

- MacroAction: permite ao usuário definir uma classe Java que implementa a ação desejada
- CallAction: define uma chamada a ser enfileirada ou executada nos recursos gerenciados ou no próprio gerenciador
- EventAction: define um evento a ser gerado para a camada superior
- SequenceAction: combina uma lista de ações a ser executada em sequência

Uma ação também define um contexto necessário para a sua execução por meio de um conjunto de parâmetros. Os parâmetros de uma ação, descritos através da classe ActionParameter, representam o conjunto de dados necessários para a execução daquela ação.

Um tratador de sinais, representado pela classe Handler, é utilizado para indicar qual ação deve ser tomada quando um determinado sinal é identificado. Uma instancia de Handler descreve o sinal a ser tratado, se este tratador está habilitado, e a ação a ser tomada.

Além disso, ao definir um tratador, é necessário descrever como serão atribuídos valores aos parâmetros exigidos por uma ação. Esta ligação entre o contexto no qual um sinal é identificado e o contexto exigido por uma ação é definido pela classe ActionExecution. A figura **??** ilustra as classes do metamodelo que estão relacionadas à ligação de contexto.

A classe ActionExecution intermedia a associação entre um tratador e uma ação, e define um conjunto de associações de parâmetros, descritas através classe ParameterBinding. Uma associação de parâmetro define como será atribuído um valor à um parâmetro de uma ação. Um parâmetro pode ser associado à diferentes fontes de valores, que são definidas por subtipos do tipo Value.

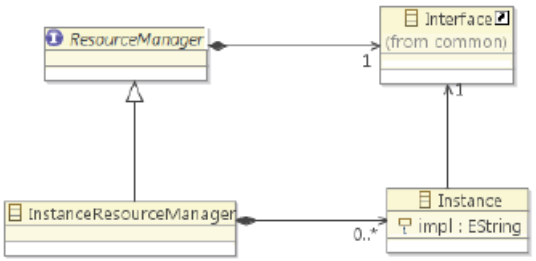

Figura 4.3: *Elementos do metamodelo para descrição dos recursos gerenciados pela camada.*

valor fixo, valor de um parametro que acompanha o sinal, expressao, ou resultado da execução de uma outra ação

A parameter may be bound to a fixed value, a parameter value, an expression, or even the result of a call to another action. This latter option, together with the use of sequence actions, enables the definition of elaborate actions that can be used to handle a signal.

4.3 Recursos

A interface mm ResourceManager do metamodelo é utilizada para descrever as interfaces dos recursos gerenciados pela camada, e como estes serão obtidos. As interfaces dos recursos são descritas através da classe mm Interface, da mesma forma que a interface da camada. A obtenção dos recursos por sua vez, é definida de acordo com a sua natureza, por classes mm que implementam a interface ResourceManager. O metamodelo proposto conta com uma classe mm denominada InstanceResourceManager que permite a descrição de um conjunto fixo de recursos e suas características. Outras classes mm que implementam a interface ResourceManager poderiam, por sua vez, possibilitar a obtenção de recursos de formas mais elaboradas, como por exemplo, através de repositórios de objetos distribuídos.

Uma instância da classe mm InstanceResourceManager agrupa um conjunto de objetos do tipo mm Instance. A classe mm Instance, por sua vez, representa um recurso que é obtido diretamente a partir da instanciação de uma determinada classe rt que implementa o recurso. Além disso, uma instância de Instance também define qual das interfaces descritas pelo ResourceManager correspondente é a interface do recurso. Por fim, a classe mm Instance implementa a interface Annotable, o que permite que metadados sejam associados aos recursos. A Figura 4.3 ilustra as classes mm do metamodelo envolvidas na descrição de recursos.

Durante a execução da camada, um recurso descrito a partir da classe mm Instance tem sua implementação instanciada. Além disso, a interface associada à este objeto

Figura 4.4: *Elementos do metamodelo para descrição dos tipos de dados mantidos pela camada.*

é utilizada para controlar a comunicação com essa implementação recém instanciada. Para possibilitar a integração entre o ambiente de execução e a implementação do recurso, os métodos da classe rt de implementação que representam chamadas disponibilizadas pela interface associada devem incluir a anotação @Call, de forma a estabelecer o relacionamento entre interface e implementação. Além disso, essa mesma classe deve implementar a interface Manageable que possibilita que um recurso sinalize eventos à camada.

4.4 Manutenção de estado

Ao realizar o seu trabalho, a camada de intermediação de serviços recebe chamadas através da sua interface e eventos gerados pelos seus recursos e realiza o processamento correspondente. Muitas vezes, o processamento de um desses sinais recebidos depende de outros sinais já processados anteriormente pela camada. Um sinal pode ter seu processamento diferenciado de acordo com a ocorrência ou não de um evento em um determinado recurso, ou de acordo com o resultado do processamento de uma chamada, etc. Devido a esta característica, é necessário que alguns dados sejam mantidos pela camada entre o tratamento de diferentes ocorrências de sinais.

A classe StateManager do metamodelo tem como função definir os tipos de dados que poderão ser mantidos durante a execução da camada. Os tipos de dados podem ser descritos através de uma estrutura simples baseada em propriedades e subtipos. Além disso, a descrição de um tipo exige a definição de uma propriedade chave que identifique unicamente um registro desse tipo de dados.

Os tipos são definidos através de instâncias da classe State que possui um nome, propriedade chave, e agrupa propriedades e subtipos. Cada propriedade por sua vez é definida por instâncias da classe Property, que por sua vez possuem um nome. Os subtipos são definidos a partir da mesma classe State, o que possibilita a definição de tipos de dados compostos. A Figura 4.4 ilustra as classes envolvidas na definição dos tipos de dados a serem mantidos pela camada durante sua execução.

4.5 Computação Autônoma

Além de abstrair as diferenças de capacidades entre os recursos existentes, a camada de intermediação de serviços também tem como responsabilidade ocultar da camada superior os detalhes relacionados à dinâmica de utilização dos recursos. Assim sendo, ao usuário da camada é indiferente o recurso que está sendo utilizado, como e

quando foi selecionado e todos os detalhes envolvidos em sua preparação realizar as tarefas solicitadas.

Para atender à esta demanda, a camada de intermediação de serviços deve ser capaz de se auto gerenciar, adaptando-se automaticamente para realizar o serviço solicitado dentro das restrições impostas pelo seu contexto. O auto gerenciamento dessa camada envolve o constante monitoramento dos recursos e das solicitações dos usuários para identificar situações que exigem uma ação e escolher a ação apropriada a ser tomada.

Através das abstrações presentes no metamodelo é possível definir o tratamento de eventos gerados pelos recursos e chamadas realizadas através da interface da camada. Estes mecanismos, associados à manutenção de estado possibilitam definir como os recursos e solicitações serão monitorados para identificar cenários que exigem a execução de uma ação.

Apesar de isso ser possível, essas abstrações não são apropriadas para a descrição de situações mais complexas, que podem envolver diversos recursos, o estado da camada, dados de chamadas realizadas, entre outros. Esta limitação ocorre pois as construções para tratamento de sinais associam uma ação diretamente à um evento ou chamada.

Com o intuito de facilitar a definição de como se dará o auto gerenciamento da camada, o metamodelo proposto incorpora um conjunto de abstrações baseadas na arquitetura de computação autônoma proposta pela IBM. Revisão/descrição breve da arquitetura, do loop MAPE-K, sensores e atuadores.

As abstrações presentes no metamodelo proposto permitem descrever regras que determinam a geração e transmissão de conhecimento entre as funções MAPE. Desta forma, ao construir uma camada de intermediação de serviços devem ser descritas regras para identificação de sintomas, solicitações de mudanças e planos de mudança. Em tempo de execução, o monitor utiliza essas regras para identificar a ocorrência de um sintoma. De forma semelhante o analisar e planejador se baseiam nos sintomas identificados e nas solicitações de mudança geradas para realizarem sua tarefa.

A classe AutonomicManager agrupa os elementos relacionados ao gerenciamento autônomo de recursos. Essa classe agrupa elementos que descrevem as regras para geração de sintomas, solicitações de mudanças e planos de mudança. A classe Symptom descreve um sintoma a ser monitorado com o intuito de identificar mudanças no contexto da camada. Um sintoma define um conjunto de condições para que este seja identificado. Em tempo de execução, os recursos e o estado da camada são monitorados e as condições de um sintoma avaliadas. Se todas as condições definidas em um sintoma são atingidas uma ocorrência deste sintoma é gerada e passada ao analisador.

As condições agrupadas pela classe Symptom são descritas a partir de expressões. Além das condições, a classe Symptom também define o contexto em que estas expressões serão avaliadas através da classe Binding que associa um nome (a ser usado

na expressão) à um elemento do tipo Bindable. A interface Bindable, por sua vez, é implementada pelas classes Signal e State. Essas abstrações permitem definir condições que envolvam além de dados de chamadas e eventos o estado mantido pela camada.

A classe ChangeRequest define uma requisição de mudança que deve ser gerada quando um determinado sintoma é identificado. Associado à uma requisição de mudança podemos ter um plano de mudança definido por instâncias da classe ChangePlan. No metamodelo proposto, um plano de mudança define uma ação a ser executada. Uma ação por sua vez pode ser composta de outras ações, conforme visto na seção **??**.

4.6 Políticas para seleção de recursos

Referências Bibliográficas

[1] ALLISON, M.; ALLEN, A. A.; YANG, Z.; CLARKE, P. J. **A software engineering approach to user-driven control of the microgrid.** In: *SEKE*, p. 59–64. Knowledge Systems Institute Graduate School, 2011.

[2] BLAIR, G.; BENCOMO, N.; FRANCE, R. **Models@ run.time**. *Computer*, 42(10):22 –27, oct. 2009.

[3] BÉZIVIN, J. **Model driven engineering: An emerging technical space**. In: Lämmel, R.; Saraiva, J.; Visser, J., editors, *Generative and Transformational Techniques in Software Engineering*, volume 4143 de **Lecture Notes in Computer Science**, p. 36–64. Springer Berlin / Heidelberg, 2006. $10.1007/11877028_2$.

[4] CLARKE, P. J.; HRISTIDIS, V.; WANG, Y.; PRABAKAR, N.; DENG, Y. **A declarative approach for specifying user-centric communication**. *Information Sciences*, p. 89–98, 2006.

[5] DENG, Y.; SADJADI, S. M.; CLARKE, P. J.; HRISTIDIS, V.; RANGASWAMI, R.; WANG, Y. **Cvm - a communication virtual machine**. *Journal of Systems and Software*, 81(10):1640–1662, 2008.

[6] FRANCE, R.; RUMPE, B. **Model-driven development of complex software: A research roadmap**. In: *Future of Software Engineering, 2007. FOSE '07*, p. 37 –54, may 2007.

[7] SCHMIDT, D. **Guest editor's introduction: Model-driven engineering**. *Computer*, 39(2):25 – 31, feb. 2006.

[8] SELIC, B. **The pragmatics of model-driven development**. *Software, IEEE*, 20(5):19 – 25, sept.-oct. 2003.