Model-Driven Generation of DSML Execution Engines*

Gustavo C. M. Sousa
Fábio M. Costa
Instituto de Informática
Universidade Federal de Goiás
Goiânia-GO, Brazil
{gustavo|fmc}@inf.ufg.br

Peter J. Clarke
School of Computing and
Information Sciences
Florida International University
Miami-FL, USA
clarkep@cis.fiu.edu

ABSTRACT

The combination of domain-specific modeling languages and model-driven engineering techniques hold the promise of a breakthrough in the way applications are developed. By raising the level of abstraction and specializing in building blocks that are familiar in a particular domain, it has the potential to turn domain experts into application developers. Applications are developed as models, which in turn are interpreted at runtime by a specialized execution engine in order to produce the intended behavior. This approach has been successfully applied in different domains, such as communication and smart grid management. However, each time the approach has to be realized in a different domain, substantial re-implementation has to take place in order to put together an execution engine for the respective DSML. In this paper, we present our work towards a generalization of the approach in the form of a meta-model and its respective execution environment, which capture the domain-independent aspects of runtime model interpretation and allow the definition of domain-specific execution engines as instances of the meta-model. We present an initial validation of the approach in the context of the Communication Virtual Machine project, by realizing part of the execution engine architecture in the form of an instance of the proposed meta-model.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*domain-specific architectures, languages*

Keywords

Models at Runtime, Model-Driven Engineering, Domain-Specific Modeling Languages, Metamodeling, Middleware

1. INTRODUCTION

*This work was partly supported by the Capes Foundation, Brazil, Proc. 0759-11-2

Models@Run.Time '12 Innsbruck, Austria

Model driven engineering is being proposed as an approach to deal with the growing complexity involved in the development of modern applications. These applications commonly operate in a distributed and heterogeneous environment and must behave in a dependable manner. These applications also often need to adapt at runtime to comply with changing requirements and environment. In order to achieve these goals the developers need to employ a variety of strategies to

MDE proposes to simplify the development of applications. By employing models built upon abstractions of the problem domain. MDE looks forward reducing the complexity due to the the technical details in the translation from domain concepts to platform concepts.

In order to reduce this complexity, MDE relies on DSMLs and automated processing of models. Models can be either transformed or interpreted.

The use of high level (and maybe graphical) DSMLs? Making it possible for domain experts, or even end users to create applications from their requirements.

Communication Virtual Machine (CVM) and MicroGrid Virtual Machine (MGridVM) are examples of software that employ MDE techniques to allow users to develop applications using DSMLs. Don't forget to mention that its not only applications, but complex applications Besides allowing the development of applications, these software also enables the users to adapt applications during their runtime by manipulating models. These virtual machines are execution engines for DSMLs named CML and MGridML respectively.

to allow the above (user built applications)? DSMLs, along with the mechanisms for their processing embed concepts and semantics of the domain. Once we have a DSML and the mechanism for processing it, users can directly develop applications in that DSML.

While the development of applications in these domains employing their respective DSMLs is quite simple, the same is not true when talking about the execution engine that interprets their languages. These mechanisms are commonly developed in general purpose language and are subject to all the technical complexities involved in the traditional software development.

Despite that, these execution engines share a lot of commonalities. Their applications basically provide a high-level service upon a set of heterogeneous resources. In order to do that, the runtime engine may need to execute many operations related to the processing of models, negotiation of configurations, synthesis of commands, evaluation of policies,

runtime adaptation, and etc. Both CVM and MGridVM are built upon a layered architecture, in which each layer has well defined responsibilities.

Taking that into consideration, we propose a model-driven approach to the construction of execution engines for the provision of high-level services described by DSMLs upon a set of resources. Doing so, we look forward a way to assist the construction of DSML execution engines. Our approach relies on the experience acquired in the development of CVM and MGridVM to propose a generic architecture and the use of modeling as a way to specialize it. We illustrate this approach by presenting a meta-model designed to describe one of the layers of the proposed architecture. Next, we create an instance of this meta-model that describes the same behavior expected from the correspondent layer in the CVM.

2. BACKGROUND

models@runtime são uma forma de autorepresentação com causalidade i.e. reflexão mais próxima do domínio

3. GENERIC ARCHITECTURE OF THE EXECUTION ENGINE

- Like we introduced, in order to get the benefits of MDE (including m@rt) we need DSMLs and their execution engine - While the syntax and static semantics of a DSML can be described by a meta-model, their dynamic semantics is embeded in the execution engine

Overview of the approach: - Integrate a meta-model that describes the abstract syntax and static semantics of a DSML to a model that describes the execution engine for that language (that on its turn encapsulates the static semantics... it was described in the background) - While there are several tools that can be used to define the metamodel and concrete syntaxes for DSMLs, there is little support for defining the dynamic semantics that the execution engine embedded.

Figure showing: DSML meta-model (abstract syntax [mof], static semantics [ocl]) + concrete syntax [gmf] + dynamic semantics [?] composing an environment for the development of applications and show a M@RT being fed into this environment

Dada uma DSML previamente definida, podemos definir um execution engine que vai executar instancias dessa DSML Como benefício temos: - Futuras mudanças no execution engine podem ser para atender mudanças na linguagem (e.g. novas construções) ou para atender requisitos de diferentes ambientes

Even though different domains require different semantics their realization quite often rely on a common set of operations. The processing of runtime models usually requires of the execution engine capabilities of model analysis and negotiation, transformation of models into different representations, adaptation of the internal arrangement of the execution engine among others.

In order to enable the construction and manipulation of models that represent the behavior of a virtual machine we need a specific language (that is, a meta-model) for that. This language needs to be powerful enough to allow the description of virtual machines for different business domains, like the ones described above. Still, this language should not be too generic, as this could make model manipulation as difficult as writing code in a general-purpose language.

Considerando isto, essa linguagem deve conter construções

Figure 1: Arquitetura proposta para esta categoria de Máquinas Virtuais centradas no Usuário.

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.

- based on the solutions employed by CVM/MGridVM, i.e. layered architecture with separation of responsibilities;
- a model based way of specializing this generic architecture i.e. employing a meta-model and instances that specialize

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 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.

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 metamodelo 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:

4. META-MODEL FOR BROKER LAYER

4.1 Interface

No metamodelo proposto, a interface para utilização de um gerenciador é definida através de chamadas providas e eventos que podem ser sinalizados. Este tipo de interface segue a mesma abordagem empregada pela CVM para comunicação entre suas camadas [**?**]. Dessa forma, a utilização da interface 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 possa ser tratado como um recurso.

A classe Interface é utilizada para descrever a interface de um gerenciador ou recurso no metamodelo proposto. Essa classe agrupa um conjunto de chamadas providas e eventos que podem ser sinalizados. Chamadas e eventos 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. A Figura **??** ilustra as classes do metamodelo relacionadas à descrição de interfaces.

4.2 Signal handling

4.3 Resource management

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 **??** ilustra as classes mm do metamodelo envolvidas na descrição de recursos.

4.4 State management

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 **??** ilustra as classes envolvidas na definição dos tipos de dados a serem mantidos pela camada durante sua execução.

4.5 Autonomic computing

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 sele-

cionado 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 Policies

5. EXECUTION ENVIRONMENT

6. EXAMPLE IN THE COMMUNICATION DOMAIN

7. RELATED WORK

Quais as áreas relacionadas, e trabalhos relacionados - Outras abordagens de construção de execution engines de DSMLs

8. CONCLUDING REMARKS

9. REFERENCES