Information and Software Technology 62 (2015) 21–41

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

An adaptive middleware design to support the dynamic interpretation of domain-specific models

Karl A. Morris [a,*], Mark Allison [b], Fábio M. Costa [c], Jinpeng Wei [d], Peter J. Clarke [d]

[a] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
[b] School of Computer Science, Engineering, and Physics, University of Michigan-Flint, Flint, MI 48502, USA
[c] Instituto de Informática, Universidade Federal de Goiás, CEP 74690-815, Goiânia, GO, Brazil
[d] School of Computing and Information Sciences, Florida International University, Miami, FL 33199, USA

ARTICLE INFO

Article history:
Received 7 November 2013
Received in revised form 7 February 2015
Accepted 9 February 2015
Available online 15 February 2015

Keywords:
Models at runtime
Adaptable middleware
Domain independence
Domain specific classifier

ABSTRACT

Context: As the use of Domain-Specific Modeling Languages (DSMLs) continues to gain popularity, we have developed new ways to execute DSML models. The most popular approach is to execute code resulting from a model-to-code transformation. An alternative approach is to directly execute these models using a semantic-rich execution engine – Domain-Specific Virtual Machine (DSVM). The DSVM includes a middleware layer responsible for the delivery of services in a given domain.
Objective: We will investigate an approach that performs the dynamic combination of constructs in the middleware layer of DSVMs to support the delivery of domain-specific services. This middleware should provide: (a) a model of execution (MoE) that dynamically integrates decoupled domain-specific knowledge (DSK) for service delivery, (b) runtime adaptability based on context and available resources, and (c) the same level of operational assurance as any DSVM middleware.
Method: Our approach will involve (1) defining a framework that supports the dynamic combination of MoE and DSK and (2) demonstrating the applicability of our framework in the DSVM middleware for user-centric communication. We will measure the overhead of our approach and provide a cost-benefit analysis factoring in its runtime adaptability using appropriate experimentation.
Results: Our experiments show that combining the DSK and MoE for a DSVM middleware allow us to realize efficient specialization while maintaining the required operability. We also show that the overhead introduced by adaptation is not necessarily deleterious to overall performance in a domain as it may result in more efficient operation selection.
Conclusion: The approach defined for the DSVM middleware allows for greater flexibility in service delivery while reducing the complexity of application development for the user. These benefits are achieved at the expense of increased execution times, however this increase may be negligible depending on the domain.

1. Introduction

Model Driven Software Development (MDSD) has become a widely used paradigm in the area of software engineering with its growth increasing in recent years [20,43]. As a result of the growth of MDSD there has also been much interest in Domain-Specific Modeling Languages (DSMLs), particularly, the graphical version of DSMLs [19,26,27]. Conventional approaches to using DSMLs focus on model transformation utilized in other areas of software engineering, where models in one language are translated into another language prior to execution, e.g., models created in UML are translated into Java [31,34]. A developing trend in this area is to remove the steps involved in conventional model translation, and to instead execute the models directly. This approach requires a semantically rich environment which is able to interpret models at this level of abstraction.

A class of DSMLs that supports model execution using a semantically rich execution engine is referred to as *Interpreted Domain-Specific Modeling Languages* (i-DSMLs) [11]. An i-DSML execution engine is one approch that provides a facility for the direct execution of models by using a 4-layered architecture, where each layer receives and performs operations on an increasingly granular view of the model, before passing the transformed version of the model

* Corresponding author.
E-mail addresses: karl.morris@temple.edu (K.A. Morris), markalli@umflint.edu (M. Allison), fmc@inf.ufg.br (F.M. Costa), weijp@cis.fiu.edu (J. Wei), clarkep@cis.fiu.edu (P.J. Clarke).

http://dx.doi.org/10.1016/j.infsof.2015.02.003
0950-5849/© 2015 Elsevier B.V. All rights reserved.

to the next layer in the stack. We will refer to an i-DSML execution engine as a *Domain-Specific Virtual Machine* (DSVM). The first DSVM to use the 4-layered architecture was the *Communication Virtual Machine* (CVM), which interprets *Communication Modeling Language* (CML) models in the user-centric communication domain [15,54]. Another DSVM, currently under construction, that uses a similar architecture is the *Microgrid Virtual Machine* (MGridVM), which interprets *Microgrid Modeling Language* (MGridML) models in the energy management for smart microgrid domain [1–3].

The four layers in the DSVM are: (1) *user interface* – allows users to declaratively specify models using an i-DSML; (2) *synthesis engine* – takes models as input at runtime and, based on the changes between the current runtime model and new model, generates control scripts to be processed by the next layer (model synthesis); (3) *middleware* – executes the controls scripts to manage and coordinate the delivery of domain services; and (4) *broker* – provides an API to the middleware and interfaces with the underlying platform to realize the services required.

The work presented in this article focuses on the middleware layer of the DSVM, which has responsibility for managing and coordinating the delivery of domain services. To achieve this objective the middleware interprets control scripts received from the synthesis engine layer and events received from the broker layer in the DSVM, by loading macros and executing them. These control scripts, and more specifically the commands found therein, are tightly coupled to the domain of the instantiated DSVM and the associated i-DSML. The initial design of the middleware for the CVM, the first DSVM, is presented by Deng et al. [15], however this work provides few, if any details on the implementation of the control scripts and macros used to realize a communication scenario. Wu et al. [53] provide additional details regarding the control scripts and macros used in the CVM middleware.

The use of DSVMs to interpret i-DSML models has applications in several domains, as previously mentioned, and therefore their efficient instantiation is a desired property in order to reduce engineering time and effort, and programming errors. In order to achieve the efficient instantiation of a DSVM in our approach, we must (1) define the domain-specific knowledge (DSK) and the model of execution (MoE) for each layer of the DSVM and (2) and efficient method of combining the DSK and MoE during instantiation of a DSVM. This approach would reduce the engineering time required for the instantiation of a new DSVM instance by allowing the reuse of domain-independent artifacts. More specifically in the context of our work, the DSK would include the domain-specific commands that comprise the control scripts, and the macros associated with each of these commands.

In this article we present a design for the middleware layer that supports the dynamic integration of DSK and MoE to realize domain-specific applications using a DSVM. This work builds on previously published work by Morris et al. [33] in the following areas: (1) design of a complete execution model for the middleware in DSVMs, (2) definition of a complete implementation of the middleware including model generation, selection and execution, and (3) the demonstration of the architecture's applicability in the user-centric communication domain. Our major contributions with respect to designing a DSVM middleware are as follows:

1. A mechanism and necessary artifacts for the proper representation of the domain-specific knowledge (DSK) and model of execution (MoE) for a given domain.
2. A method to dynamically combine domain-specific artifacts to realize the semantics of operations within a domain.
3. A study to determine the impact on execution times of dynamically combined functional components in the middleware layer of the CVM.

Section 2 describes background knowledge relevant to the work presented in this article. Section 3 details the motivation for this work and the problems we address. Section 4 presents an overview of our approach. Section 5 defines several concepts key to the dynamic execution of constructs to realize domain semantics. Section 6 describes the design of the DSVM middleware. Section 7 presents our findings based on experiments conducted using the DSVM middleware prototype. Sections 8 and 9 contains the related work and conclusion, respectively.

2. Background

In this section we introduce an approach to the execution of domain-specific models that is based on changes to models at runtime. These domain-specific models capture the end-users requirements for an application using various concrete syntaxes, including graphical models, text models, and models captured in a user-friendly interface. To execute these models a DSVM is used as an interpreter which consists of a four-layered architecture. One of the layers in the DSVM is the middleware which will be the focus of this paper. After introducing the execution of domain-specific models we will provide an overview of middleware, specifically, the adaptive nature of middleware.

2.1. Execution of domain-specific models

DSMLs allow end-users to easily generate solutions for problems in their respective domains since the solutions are created using abstractions closer to the problem space [19,26,27]. Conventional approaches to realize DSML models usually requires these models to be converted into source code in a traditional high-level language (HLL) using a series of model-to-model and model-to-text transformations. This source code must then be compiled and executed. An alternative approach is to execute these domain-specific models directly using an execution engine. We refer to the languages used to create these models as *Interpreted Domain-Specific Modeling Languages* (i-DSMLs) [11] and the execution engine as a *Domain-Specific Virtual Machine* (DSVM).

In our opinion, the execution engines used to interpret i-DSML models can be considered as a virtual machine based on the taxonomy of virtual machines presented by Smith et al. [42]. DSVMs can be classified as dynamic translators i.e., HLL VMs. Note however, we are moving to a higher level of abstraction, from HLLs to domain-specific models. In the subsequent subsections we introduce i-DSMLs and DSVMs. As previously stated our research team has worked on two DSVMs, the *Communication Virtual Machine* (CVM) in the user-centric communication domain [15,54], and the *Microgrid Virtual Machine* (MGridVM) in the energy management for smart microgrids domain [1–3]. In this article we will focus on the CVM.

2.1.1. Interpreted domain-specific modeling languages

An i-DSML can be described as a five-tuple, similar to a DSML [8], consisting of: a concrete syntax, e.g., graphical models; abstract syntax that defines the language syntax and integrity constraints; semantic domain, containing the domain-specific knowledge; a mapping that assigns syntactic constructs to elements in the abstract syntax; and a semantic mapping that relates abstract syntactic concepts to the semantic domain. The main difference between the traditional DSMLs and i-DSMLs is that the semantics of traditional DSMLs describe how to transform models into source code for a given HLL. The semantics for i-DSMLs define how the application captured by the model is executed to realize the intent of the user requirements without first transforming the model into an HLL.