EMSOFT 2005, Jersey City, NJ, September 18-22, 2005

Toward a Semantic Anchoring Infrastructure for Domain-Specific Modeling Languages*

Kai Chen
Institute for Software Integrated Systems
Vanderbilt University,
Nashville, TN, 37205
chenk@isis.vanderbilt.edu

Janos Sztipanovits
Institute for Software Integrated Systems
Vanderbilt University,
Nashville, TN, 37205
janos@isis.vanderbilt.edu

Sandeep Neema
Institute for Software Integrated Systems
Vanderbilt University,
Nashville, TN, 37205
sandeep@isis.vanderbilt.edu

ABSTRACT

Metamodeling facilitates the rapid, inexpensive development of domain-specific modeling languages (DSML-s). However, there are still challenges hindering the wide-scale industrial application of model-based design. One of these unsolved problems is the lack of a practical, effective method for the formal specification of DSML semantics. This problem has negative impact on reusability of DSML-s and analysis tools in domain specific tool chains. To address these issues, we propose a formal well founded methodology with supporting tools to anchor the semantics of DSML-s to precisely defined and validated "semantic units". In our methodology, each of the syntactic and semantic DSML components is defined precisely and completely. The main contribution of our approach is that it moves toward an infrastructure for DSML design that integrates formal methods with practical engineering tools. In this paper we use a mathematical model, Abstract State Machines, a common semantic framework to define the semantic domains of DSML-s.

Categories and Subject Descriptors

D.2 [Software]: Software Engineering; D.3.1 [Software]: Programming Language—*Formal Definitions and Theory*

General Terms

Language, Design

Keywords

domain-specific modeling language, Model-Integrated Computing, abstract syntax, semantic anchoring.

*This research was supported by the NSF ITR Grant CCR-0225610 "Foundations of Hybrid and Embedded Software Systems".

EMSOFT'05, September 19–22, 2005, Jersey City, New Jersey, USA.

1. INTRODUCTION

Model-based design uses models, which are formal, composable and manipulable during the design process [30]. The modeling languages are domain-specific, offering designers modeling concepts and notations that are tailored to characteristics of their application domain. Domain-specific modeling languages (DSML-s) represent the structural and behavioral aspects of embedded software and systems. Their semantics capture concurrency, communication abstractions, temporal and other physical properties. For example, a DSML framework (i.e. a set of related modeling aspects) for embedded systems might represent physical processes using ordinary differential equations, signal processing using dataflow models, decision logic using finite-state machines, and resource management using synchronous models.

DSML-s are convenient tools for the design and implementation of embedded software and systems (ESSs). A well-made DSML captures the concepts, relationships, integrity constraints, and semantics of the application domain and allows users to program declaratively through model construction. Domain experts can easily master a DSML, since the domain concepts with which they are already familiar are incorporated in the modeling language . The applications of DSML-based tools, such as Simulink/Stateflow [5], or metaprogrammable tool chains, such as the Model-Integrated Computing (MIC) tools [4], range from non critical systems (e.g. cell phones) to safety critical systems (e.g. medical systems and drive-by-wire controllers for cars). However, adoption of DSML-s and model-based design has been slowed down by the following concerns:

- The use of DSML-s with tightly integrated analysis tool chains leads to the accumulation of design assets as models defined in a DSML. Consequently, users run high risk of being "locked-in" a particular tool chain.
- Incomplete and informal specification of DSML-s makes precise understanding of their syntax and semantics difficult. While a tightly integrated tool chain seems to relieve users from the need of fully understanding the syntax and semantics of the DSML-s, the cost may be high: the lack of in-depth understanding of created models and analysis methods may prevent the organization from adopting new modeling and model analysis methods.
- The lack of formally specified semantics of DSML-s and analysis tools create major risk in safety critical

applications. Semantic mismatch between design models and modeling languages of analysis tools may result in ambiguity in safety analysis or may produce conflicting results across different tools.

Some of these concerns have been alleviated by the appearance of metamodeling languages [8] representing the abstracts syntax of DSML-s, and metaprogrammable tools [25] that can be adapted easily and inexpensively to different domains. However, abstract syntax metamodeling - although an essential step - does not solve the problem caused by the lack of precise and explicit specification of DSML semantics. There has been much effort in the research community to define semantics of modeling languages by means of informal mathematical text (see e.g. the specification of the Hybrid Interchange Interchange Format, HSIF, [28]) or using formal mathematical notations (see e.g. the operational semantics specification for StateFlow in [20]). In either case, precise specification of semantics requires significant effort, which increases the cost of the specification and adoption of DSML-s. In addition, practical applications of DSML-s require their evolution as the users' understanding of the domain changes.

In this paper we argue that we can reach a reliable, safe and affordable technology for model-based design by developing an infrastructure for *semantic anchoring* of DSML-s via completing the following agenda:

1. Development of precise specification for a set of well-defined "*semantic units*" that capture the semantics of basic models of computations in a formal framework.

2. Development of modeling language front-end for the semantic units and specification of their abstract syntax by using metamodeling.

3. Development of an infrastructure for the transformational specification of DSML semantics by defining the mapping between the abstract syntax metamodels of DSML-s and that of the semantic units.

The proposed approach has direct relationship to and builds on the following model-based design concepts:

1. In platform-based design [29] the concept of *common semantic domain* plays essential role in mapping functional models to architecture platforms. A semantic unit (or an integrated group of semantic units) forms a common semantic domain for those DSML-s, which are anchored to it.

2. *Abstract semantics* [6] represents the common semantic features of families of models of computation, and realize models of computation through specialization of this abstract semantics. We define semantic units such that they can be concretized into a family related models of computations.

3. *Multiple-aspect modeling* [23] enables the managing of complexity of DSML-s by composing them into interrelated aspects. Semantic anchoring is used for the transformational specification of different DSML aspects.

The primary contribution of this paper is the description of key steps of the semantic anchoring process using finite state automata modeling as an example. We use in the process existing MIC tools: the Generic Model Environment (GME) [4] for metamodeling, the Graph Rewriting and Transformation (GReAT) tool [2] for model transformation, the Abstract State Machines (ASM) [11, 18], as a common semantic framework to define the semantic domain of DSML-s, and AsmL [1] - a high-level executable specification language based on the concepts of ASM.

The organization of this paper proceeds as follows: Section 2 describes our methodology to support the DSML design process. The concept of Semantic units and semantic anchoring are defined in Section 3. In Section 4, we use a simple DSML capturing the finite state machine domain from Ptolemy [6] as a case study to demonstrate key steps in the semantic anchoring process. Our conclusions and future work appear in Section 5.

2. BACKGROUND: DSML SPECIFICATION

Formally, a DSML is a five-tuple of concrete syntax (C), abstract syntax (A), semantic domain (S) and semantic and syntactic mappings (M_S, and M_C):

$$L = \{C, A, S, M_S, M_C\}$$

The concrete syntax C defines the specific notation used to express models, which may be graphical, textual or mixed. The abstract syntax A defines the concepts, relationships, and integrity constraints available in the language. The semantic domain S is usually defined in some formal framework in terms of which the meaning of the models is explained. The syntactic mapping $M_C : C \rightarrow A$ mapping assigns syntactic constructs (graphical, textual or both) to the elements of the abstract syntax. The semantic mapping $M_S : A \rightarrow S$ semantic mapping relates syntactic concepts to those of the semantic domain.

The languages that are used for defining components of DSML-s are called *metalanguages* and the formal specifications of DSML-s are called *metamodels*. The specification of the abstract syntax of DSML-s requires a meta-language that can express concepts, relationships, and integrity constraints. The specification of the semantic domain and semantic mapping is more complicated, because models might have different interesting interpretations; therefore DSML-s might have several semantic domains and semantic mappings associated with them. For example, the *structural semantics* of a modeling language describes the meaning of the models in terms of the structure of model instances: all of the possible sets of components and their relationships, which are consistent with the well-formedness rules is defined by the abstract syntax. Accordingly, the semantic domain for structural semantics is defined by a *set-valued semantics*. The behavioral semantics may describe the evolution of the state of the modeled artifact along some time model. Hence, the behavioral semantics is formally captured by a mathematical framework representing the appropriate form of dynamics. (It is interesting to note that since DSML-s specify structural and behavioral invariants, which will be satisfied by all domain-specific models (DSMs), the concept of DSML - or more accurately its metamodel - is equivalent to the concept of domain architecture.)

The fact that DSML-s may have several semantic domains underlines the differences between defining semantics for programming languages and for DSML-s. Still, approaching the issues in the conceptual framework of "languages"

instead of the conceptual framework "domain architectures" has the advantage of drawing from the rich theoretical and engineering background of language design and specification.

3. SEMANTIC ANCHORING

Our approach to the construction of an infrastructure for semantic anchoring of DSML-s is based on the following observations:

1. DSML-s are opportunistically created according to the needs of domains.

2. There is a well-defined, finite set of Models of Computations (MoCs) [29], which describe canonical interaction patterns among physical and computational components of embedded systems. These MoCs can be defined by a set of : $L_i = \{C_i, A_i, S_i, M_{Si}, M_{Ci}\}$ of minimal languages, where the intuitive meaning of "minimality" is the simplest modeling language required to describe a selected type of behavior.

Semantic anchoring of an arbitrary $L = \{C, A, S, M_S, M_C\}$ modeling language to an $L_i = \{C_i, A_i, S_i, M_{Si}, M_{Ci}\}$ model of computation means specifying the $M_A : A \rightarrow A_i$ mapping. The $M_S : A \rightarrow S$ semantic: mapping of L is defined by the $M_S = M_{Si} \circ M_A$ composition, which means that the semantics of L is anchored to the S_i semantic domain of the L_i model of computation.

To develop an infrastructure for semantic anchoring requires the completion of the following tasks:

1. Selection of formal framework(s) for the mathematically precise specification of the ingredients of the languages (abstract syntax, etc.).

2. Selection of the canonic MoC-s and their specifications in the formal frameworks.

3. Development of methods and tools for specifying mapping between modeling languages.

Figure 1 shows our experimental tool architecture that supports the semantic anchoring of DSML-s. The GME tool suit is used to define the abstract syntax, A, for a $L = \{C, A, S, M_S, M_C\}$ DSML using UML Class Diagrams and OCL as metalanguage [9]. The $L_i = \{C_i, A_i, S_i, M_{Si}, M_{Ci}\}$ MoC is defined as an AsmL specification. In order to emphasize the central role of these fully specified modeling languages capturing fundamental MoC-s, we will call them semantic units. In this paper we specify their operational semantics: the semantic unit is defined in terms of (a) an AsmL Abstract Data Model (which corresponds to the A_i, abstract syntax specification of the modeling language defining the semantic unit in the AsmL framework), (b) the S_i, semantic domain (which is implicitly defined by the ASM mathematical framework), and (c) the M_{Si}, semantic mapping, defined as a model interpreter written in AsmL.

The $M_A : A \rightarrow A_i$ semantic anchoring of L to L_i is defined as a model transformation using the GReAT tool suite. The abstract syntax A and A_i are expressed as metamodels. Connection between the GME-based metamodeling environment and the AsmL environment is provided by a syntax conversion. Since the GReAT tool suit generates a model translator from the metalevel specification of the model transformation, any domain model written in the DSML can be directly translated into AsmL and can be simulated using the AsmL simulator. In the followings, we give detailed explanation about our methodology and the involved tools.

3.1 Abstract Syntax Modeling

The Generic Modeling Environment (GME) [4, 25] is a meta-programmable tool that supports the OMG four-layer metamodeling architecture. GME allows users both to design and to model with domain-specific modeling environments. The GME modeling environments may be created by using GME itself through a process of metamodeling. The GME metamodel is based on UML class diagrams [8] and OCL.

3.2 Specification of Semantic Units

Semantic anchoring requires the specification of semantic units in a formal framework using a formal language, which not only precise but also manipulable. The formal framework must be general enough to represent all three components of the $M_S : A \rightarrow S$ specification; the abstract syntax, A, with set-valued semantics, the S semantic domain to represent the dynamic behavior and the mapping between them. Examples for possible formal frameworks are the following:

- TLA^+ is a formal specification language developed by Lamport, which is based on the Temporal Logic of Actions [24]. TLA was designed to specify a wide class of systems, ranging from discrete to hybrid dynamics. Because of this generality and because TLA^+ is a complete language with a precise syntax and formal semantics, it is a good candidate for our purpose.

- The tagged signal model [27] developed by Lee and Sangiovanni-Vincentelli represents behavioral properties of concurrent systems using set-valued semantics. It clarifies, for example, the relationship between the synchronous models of computation used in synchronous languages and asynchronous models such as process networks and dataflow. It has been applied to construct a formal semantics for discrete-event systems.

- Reactive Modules are a formal model developed by Alur and Henzinger [10] for concurrent systems. The model is able to represent synchronous and asynchronous component interactions in a unified framework and supports compositional verification.

- Abstract State Machine (ASM), formerly called Evolving Algebras [18], is a general, flexible and executable modeling structure with well-defined semantics. General forms of behavioral semantics can be encoded as (and simulated by) an abstract state machine [11]. ASM is able to cover a wide variety of domains: sequential, parallel, and distributed systems, abstract-time and real-time systems, and finite- and infinite-state domains. ASM has been successfully used to specify the semantics of numerous languages, such as C [19], Java [13], SDL [16] and VHDL [12]. In particular, the International Telecommunication Union adopted an ASM-based formal semantics definition of SDL as part of SDL language definition [7].