



Precise Behavioral Semantics for Domain Specific Modeling Languages

September 25, 2007
Jacksonville, FL

Sponsored by the OMG and the Model Integrated Computing SIG

[Registration](#)

[Hotel](#)

Model analysis and model-based code generation require the precise specification of Domain Specific Modeling Languages (DSMLs). This is partly achieved by using metamodeling languages, metamodels, and UML profiles describing the abstract syntax (concepts, relationships and well-formedness rules) of DSMLs. While abstract syntax metamodeling has been an important step in model-based design and been used in various model-based frameworks, such as MDA, MIC, Eclipse-based tools, and Software Factories, explicit and formal specification of behavioral semantics has not received much attention, which creates a possibility for semantic mismatch between design models and modeling languages of analysis tools. While this difficulty exists in virtually every domain where DSMLs are employed, it is particularly problematic in safety-critical real-time and embedded systems domain, where semantic ambiguities may produce conflicting results across different tools. The increasing role of DSMLs in software and systems brings in additional challenges due to the following characteristics of model-based design flows:

1. Heterogeneity of tool chains. Tool chains supporting domain-specific design flows integrate modeling, analysis and synthesis tools using DSMLs with overlapping semantics. Explicit representation of their semantics is only a necessary but not sufficient condition for integratability. Designers need to understand precisely the relationship between the semantics of languages to establish consistent design flows.
2. Heterogeneity of systems. Systems are composed from heterogeneous components using heterogeneous interaction mechanisms. Modeling and understanding heterogeneous systems is a significant challenge. Since DSMLs are designed for modeling heterogeneous systems, the specification of their semantics must address these challenges.
3. Validation and verification. Specification of behavioral semantics of DSMLs is not only an exercise in mathematical precision but has practical significance. DSML designers need to validate behavioral semantics via inspecting traces generated by test models. Similarly, semantic accuracy of simulators and code generators must be tested via comparing their behavior with behaviors generated by the "reference semantics".

This workshop will focus on techniques and approaches to the precise and pragmatic definition of behavioral semantics for domain-specific modeling languages. One promising method, called 'semantic anchoring' relies on the use of well-defined 'semantic units' of simple, well-understood constructs (like a finite state machine) and on the use of model transformations that map higher level modeling constructs into configured semantic units. In the workshop we will discuss recent results in semantics specification, using semantic anchoring and other methods.

Program

- 09:00** **Keynote**
Dr. Janos Sztipanovits, Vanderbilt University
- 09:30** **DSML Semantics and Software Producibility**
Dr. Greg Sullivan and Dr. Basil Krikeles, BAE Systems, Advanced Information Technologies
- 10:00** **Domain Specific Models for Safety-Critical Real-Time Systems**
Christian Buckl, Embedded Systems and Robotics Lab, Technische Universität München
- 10:30** **Break**
- 10:45** **Modeling Dynamic Architectures with DSMLs**
Ethan Jackson, Microsoft Research
- 11:15** **Panel Discussion**
- 12:00** **Lunch**
- 13:00** **Graphical Syntactic Extensions of Haskell for Block Diagrams**
Pieter J. Mosterm, The MathWorks, Inc
- 13:30** **Weaving Behavior into Metamodels with Kermet**
Didier Vojtisek, INRIA (Institut National de Recherche en Informatique et Automatique)
- 14:00** **A Language and a Methodology for Precise DSML Behavioral Semantics Modeling**
Dr. Chokri Mraidha, CEA-List
- 14:30** **Break**
- 14:45** **Approaches to Refining the Semantic Unit Library**
Ryan Thibodeaux, Institute for Software Integrated Systems, Vanderbilt University
- 15:15** **Composition Issues for Behavioral Semantics of Domain-Specific Modeling Languages**
Joseph Porter, Institute for Software Integrated Systems, Vanderbilt University
- 15:45** **Panel Discussion**

See the following pages for Detailed Presentation Abstracts.

Presentation Abstracts

DSML Semantics and Software Producibility

We define Software Producibility as the fitness of software systems to survive and evolve as part of today's complex ecosystems of interacting components and technologies and to be robust with respect to changing requirements. We have proposed that standards need to be developed for a model-based methodology that enables the design and implementation of quantitative metrics that are useful predictors of Software Producibility. We intend to pursue these producibility issues in the context of our work on the DARPA DMT program, in collaboration with Vanderbilt, MIT, and other parts of BAE Systems. The focus of our work on DMT will be tool evolution, in which case a crucial question is whether the semantics of post-evolution models are consistent with the original model semantics. This question is a simplified version of the general question of whether two DSML tools use comparable behavioral models and thus whether the results of the tools are consistent in a formal sense. We are particularly interested in dynamic, "operational" methods for verifying semantic consistency, as opposed to purely denotational methods. For example, analyzing model-generated traces, as abstractions of runtime behavior, in a "challenge-response" framework, enables dynamic coupling of self-describing DSML-derived components. It is generally agreed that DSMLs aid the overall maintainability of a system. We believe that it is important the DSMLs can exist within a heterogeneous and evolving toolchain, and thus semantics for DSMLs is critically important. In this briefing we will explore the close relationship between DSML and Software Producibility. Clearly, in a dynamic environment of tool and system evolution, if the semantic ambiguities that may arise are not properly handled Software Producibility will suffer. Conversely, how can we ensure that the overall system of tool-chains and software systems can evolve within the additional constraint of maintaining an acceptable level of Software Producibility?

BIOGRAPHY:

Greg Sullivan: Ph.D. in computer science, specializing in programming language semantics, dynamic languages, verification and validation, model-based programming, aspect-oriented programming. Related interests in AI planning and scheduling.

Basil Krikeles is Chief Architect for Intelligent Systems at BAE Systems, Advanced Information Technologies. He holds a Ph.D. in mathematics from Yale University. Since 1997 he has been involved in a number projects addressing issues of large scale software development, distributed computing, generative programming and model-driven software development

Domain Specific Models for Safety-Critical Real-Time Systems

In this presentation we are addressing the issue of model-based development and code generation for fault-tolerant embedded systems. The possibility to generate code automatically from the model accelerates the development process and is therefore very attracting. Particularly for the domain of safety-critical applications, where the developers are typically application domain experts with less background in programming fault-tolerant real-time systems, the possibility to provide extensive code generation is crucial. Existing tools focus predominantly on the functional aspects of the applications like the control functions. However, major parts especially of fault-tolerant embedded systems are related to system aspects. We understand by the term system aspects all non-functional aspects related to the distribution of the embedded system and the need for fault-tolerance: process management, scheduling, inter-process communication, communication within the distributed system and the fault-tolerance mechanisms. These aspects are in general not addressed by existing tools and have to be implemented manually.

We will describe an approach to use model-based development and code generation of system aspects for fault-tolerant embedded systems. By using an adequate domain specific language with explicit execution semantics this approach becomes feasible. Applying the concept of logical execution time, it is possible to design a model that is deterministic in the execution and guarantees replica determinism. By separating the

functional behavior from the state of the system the foundation to the automatic realization of voting and synchronization mechanisms was laid.

To separate the different aspects of the system and to simplify the modeling process, we split up the model into distinct sub models to describe the hardware, the software, the fault model and the fault-tolerance mechanisms separately.

For illustration purpose, we use the approach to implement an elevator control using a hot-standby system. The developer has only to implement the control function and a fault-detection function for the used micro-controller. The rest of the required code, including the temporal correct execution of the system, the communication within the distributed system and the fault-tolerance mechanisms, was automatically generated by our tool.

BIOGRAPHY:

Christian Buckl received his diploma degree in 2004 at the Technische Universität München. Since then, he is working in the lab for Embedded Systems and Robotics.

Since 2006, he is heading the research group for model-based development of embedded systems.

Modeling Dynamic Architectures with DSMLs

Model-based design provides strategies for designing, verifying, and deploying distributed and heterogeneous systems. One popular incarnation of model-based design uses domain-specific modeling languages (DSMLs) to capture the structural invariants and precise semantics of a problem domain. Model-based design via DSMLs proceeds as follows: First, the essential components of the problem domain are represented as modeling primitives, with which large models can be built. The rules, or structural invariants, for proper juxtaposition of modeling primitives are also defined. Second, the modeling primitives are assigned precise semantics via a “semantic anchoring”. At this point a DSML for the problem domain is well-defined. Next, the engineer constructs point designs (models) using the DSML, and then simulates and/or verifies these models for behavioral correctness. Finally, an implementation is synthesized from the models, which can then be deployed. Often times multiple DSMLs are employed for the purposes of evolving a high-level specification towards the final implementation, thus iterating the above procedure.

This design procedure yields an implementation that is a reflection of the original high-level models and remains a reflection of the models throughout its run-time. Interestingly, there is an expanding set of problem domains where the very architecture of the system evolves throughout its run-time. These, so-called, dynamic architectures are not easily modeled by an immutable set of high-level models, but instead transition through many different high-level models. For example, applications implemented with service oriented architectures (SOA) on ad hoc networks will have a deployment that evolves as services/nodes enter and exit the network. Systems with fault mitigation capabilities must re-architect themselves on the fly in order to correct faults. Adaptive systems may make similar run-time modifications to their architectures. In this work we show how advances in model-based design and DSML semantics can be used to design systems with dynamics architectures.

Dynamic architectures can be implemented with the tools and techniques of standard model-based design. In particular, we restrict our attention to systems where the degree of re-architecting can be bounded or enumerated by a set of scenarios. This notion of bounded-ness is commonly found in fault mitigating, adaptive, and modal systems. Adaptation is brought into the modeling process by constructing a set of high-level models that capture the possible adaptation scenarios. Using this set of scenarios and the semantic anchor, we produce an implementation that contains the adaptation scenarios such that each scenario faithfully reflects the DSML semantics. Advances in partial evaluation can be used to generate dynamic architectures using just the basic code generators available for the DSML. Importantly, dynamic architectures can be modeled and generated using the same model-based tools as those for static architectures.

Once an implementation has been instrumented with adaptive capabilities, we must also provide an interface to access these capabilities. We have shown that DSML specifications can also be useful for this purpose. The DSML provides a structural description of models, and this structural description can be

converted into a communication protocol. Component A can demand component B to adapt to a particular scenario by sending a structural description of that scenario to B. Using the DSML structural invariants, an arbitrary adaptation request from component A can be converted into one of the legal scenarios of component B in a precise and robust manner. Thus, we conclude that the model-based approach has significant utility for the methodological implementation of dynamic architectures.

BIOGRAPHY:

Ethan Jackson is currently at Microsoft Corporation. He received his B.S. in Computer Engineering from the University of Pittsburgh in Pittsburgh, PA, and his PhD in Computer Science at Vanderbilt University in Nashville, TN. His research focuses on model-based approaches to the design of correct and reliable systems. His work has ranged from novel tool architectures for embedded system design to the mathematical foundations of domain specific modeling languages and metaprogrammable tools. His latest work aims to extend model-based formal methods to adaptive software systems, and further develop compositional formal semantics for domain-specific modeling languages.

Graphical Syntactic Extensions of Haskell for Block Diagrams

Graphical formalisms in general and block diagrams specifically are often preferred for modeling of computation such as, for example, the control law of an embedded control system. Precise semantic specifications, on the other hand, are often preferred to be given in a textual language. In this work, block diagrams are treated as a syntactic extension to a textual language, Haskell.

Ellner and Taha provide a good introduction to the unneeded gap between block diagram and textual languages. They also provide a promising way to formally bridge this gap with a visual multi-stage calculus called PreVIEW. This work aims to bridge this gap in a different but complementary way. Instead of trying to provide a graphical language that is equivalent to a textual one, a textual language (Haskell) is extended with a few graphical features. Furthermore, this work does not address multi-stage programming, and it addresses a full language (Haskell) rather than a minimal calculus suitable for theoretical work.

The particular idea of implementing a block diagram language as syntactic sugar on top of Haskell is discussed and partially implemented by Reekie. This is closely related to the approach of implementing a domain-specific language by embedding it in a general-purpose language.

Implementing a new language by sugaring or embedding can be seen as part of Landin's 40-year-old program to avoid reinvention of general-purpose language capabilities when inventing a new language. This may not always be possible and desirable, but it appears to hold for block diagrams.

One could say that the value of block diagrams is that they allow the graphical structure of a program to be expressed in a more direct form. Textual syntaxes have to represent the program (term graph) mostly as a tree, and further, have to represent that tree as a sequence of lexemes. They represent a graph mostly as a tree by adding some context-sensitive constraints to a context-free grammar. They represent a tree as a sequence through mechanisms like parenthesis and precedence. This is not to say that the more direct, graphical form of expression offered by block diagrams is always preferable. Most textual programs can be seen as compact, elegant representations of what would be a rather convoluted graph. But there are times when it would be clearer to see the graph directly, and this is what block diagrams offer.

In general, block diagram languages are never full languages; they are always combined with some other language that defines what individual blocks mean. This meaning is typically defined by a name inside the block, possibly combined with the use of a non-block shape instead of a block, (e.g. the use of a triangle instead of a block, to indicate a gain, with the amount of gain indicated by a numeral like "-1" inside). Thus block diagrams always rely on some other language to provide the symbolic environment in which blocks are defined, so it is a short leap to make this environment a scope in Haskell.

BdHas is Haskell plus syntactic sugar for block diagrams. A block diagram can be used inside a Haskell expression, and a Haskell expression can be used inside a block. To execute this BdHas program, it is passed to a program that de-sugars it into pure Haskell. The meaning of a BdHas program is the Haskell code that results from translating all of its block diagrams to pure Haskell.

BIOGRAPHY:

Pieter J. Mosterman is a senior research scientist at The MathWorks, Inc. in Natick, MA. Before, he was a research associate at the German Aerospace Center (DLR) in Oberpfaffenhofen. He has a Ph.D. degree in Electrical and Computer Engineering from Vanderbilt University in Nashville, TN, and a M.Sc. degree in Electrical Engineering from the University of Twente, Netherlands. His primary research interests are in Computer Automated Multiparadigm Modeling (CAMPAM). He designed the Electronics Laboratory Simulator, nominated for The Computerworld Smithsonian Award by Microsoft Corporation, and HyBrSim, a paper on which was awarded the IMechE Donald Julius Groen Prize. Dr. Mosterman is currently Editor-in-Chief of Simulation: Transactions of The Society for Modeling and Simulation International for the Methodology section, and Associate Editor of IEEE Transactions on Control System Technology and of Applied Intelligence. He was Guest Editor of special issues of ACM Transactions on Modeling and Computer Simulation and IEEE Transactions on Control Systems Technology on the topic of CAMPAM.

Weaving behavior into Metamodels with Kermeta

The Kermeta workbench is a powerful open source environment based on an object-oriented DSL (Domain Specific Language) optimized for metamodel engineering. It allows rapid design and integration of semantically rich metamodels in a unified way. Kermeta workbench has been developed using MDE principles, to provide precision in both structural and dynamic specification at the metamodel level. It leverages the capabilities of EMOF2.0 so that all aspects of DSLs can be specified natively with metamodels. It integrates a classical object-oriented approach with static type checking and genericity into its own reflectively available metamodel.

Kermeta makes it possible to weave the various aspects (such as abstract syntax, static semantics, operational semantics) of a DSML (Domain specific Modeling Language) into an executable metamodel that can be used for advanced uses like operational semantics validation. One of Kermeta strength is its ability to weave these aspects directly from their original formats: typically the DSML metamodel can be defined using either Ecore, EMOF or Kermeta, the static constraints and well formedness rules can be defined in OCL or Kermeta, the operational semantics can be described in Kermeta or Java.

Technically, the Kermeta language consists of an extension to the EMOF 2.0 (Essential Meta-Object Facilities) to support behavior definition. It provides an imperative object-oriented action language to specify the body of operations in metamodels.

Kermeta provides support for many use cases, including:

- implementation of operations directly in metamodels,
- execution of simulation of metamodel behavior for semantics prototyping,
- transformation and weaving of models,
- verification and validation of models against metamodels (as given by a set of static and dynamic constraints),
- for building new DSLs under the shape of metamodels,
- for building any model-driven tools, including tools that generate tools (generative programming).

Kermeta is composed of a workbench backed up with an innovative metamodeling language. This development environment currently contains the following Eclipse based tools:

- an interpreter and a debugger that allows a model to be executed according to its metamodel operational semantics.
- text and graphical editors, fully integrated within Eclipse, with syntax highlighting, code auto completion, etc.
- an outline view, which allows navigation through the whole model and metamodel.

The Kermeta web site at <http://www.kermeta.org> contains various documents, tutorials (from the installation to the concrete use) and examples. Several scientific papers based on Kermeta have been written in major conferences in the domain of Model-Driven Engineering. (<http://www.kermeta.org/documents/articles/>)

Kermeta has been successfully used in several projects, among which:

- AOSD-Europe, the European network of excellence on AOSD
- Artist2, the European network of excellence on real-time embedded systems
- UsineLogicielle, a System@tic project where Kermeta based operational semantic is associated to functional requirement for test synthesis purposes.
- Speeds, a European FP6 project for aspect-oriented metamodeling of avionics and automotive systems, including operational semantics aspects

Kermeta is INRIA's solution (Institut National de Recherche en Informatique et Automatique) for adding operational semantics to metamodels. The scientific work was initiated by the Triskell team headed by Prof. Jézéquel.

BIOGRAPHY:

Didier Vojtisek is a research engineer at the French research institute INRIA (Institut National de Recherche en Informatique et Automatique). He is Chief Architect of the software development for the major software developed by Triskell team, including Kermeta.

He has participated in the OMG normalization processes, most notably in QVT RFP.

A Language and a Methodology for Precise DSML Behavioral Semantics Modeling

Most systems are built from parts which are designed using various methods: hardware, software, signal processing algorithms, control laws, state machines. The different natures of the parts of a system and the different aspects that must be considered call to different skills and domain specific concepts. Therefore, depending on the domain of interest, engineers are prone to define adequate Domain Specific Modeling Languages (DSML). To reach this purpose they use either meta-modeling techniques or profile based approaches. The current status of both MOF and UML leads engineers to define syntax in a formal manner and semantics in natural language. This results in problems concerning model transformations and formal treatment activities of models because of two main points:

- Natural language becomes easily ambiguous and may lead to different interpretations;
- Variation points may be implicitly interpreted in different manners.

On the one hand, formalizing semantics is crucial to tackle this problem, but on the other hand, engineers are usually not very familiar with the mathematics used for formal semantics. Our goal is to overcome that problem by defining a language dedicated to describe semantics in a manner which is easily usable by software modelers. This language is based on primitives which are dedicated to the description of executions. All those primitives are mathematically grounded. Thus the use of our language to define DSML semantics provides mathematic foundations for those semantics in a transparent manner.

Moreover, modern industrial systems are often component based. This means that they are composed of basic components structured by communication mechanisms. The semantics of those systems is fully characterized by:

- The semantics of the basic components,
- The communication connections between those components,
- The rules that define the nature of communications and their scheduling

Those three items are sufficient to fully characterize the behavioral semantics of component based systems. The rules that define the nature of communications and their scheduling are called a Model of Computation and Communication (MoCC). We define a meta-methodology to specify the semantics of such systems by

providing a generic model of MoCC which takes advantage of the component oriented structure of these systems. This generic model of MoCC can be specialized to express the specific semantics of a given MoCC.

The heterogeneous nature of complex systems leads modelers to choose different DSMLs at the specification phase to describe different parts of the system. Thus, the resulting collection of models does not form a global model of the system: it does not define the way local semantics collaborate at the system level. A way to tackle the problem is to use semantics-gluing-techniques. Our language allows the specification of such glues between semantics using hierarchical MoCC structuring mechanisms.

We have applied this semantic-gluing-technique to hierarchical component-based system modeling in the context of a System@tic cluster project (<http://www.usine-logicielle.org>).

We provide a UML profile that defines the concepts of our language. This profile allows the specification of the semantics of DSMLs with two back-ends, one for simulation and execution of the models, the other for exploring their possible behaviors and generating test patterns. Our ongoing work deals with the use of this profile for modeling the behavioral semantics of the Basic UML Subset (bUML) of the ongoing OMG standard "Semantics for a Foundational Subset for Executable UML Models", in order to give it a mathematically founded semantics.

Authors:

Supelec: Frédéric Boulanger, Cécile Hardebolle, Dominique Marcadet

CEA-List: Christophe Gaston, Aurélien Ohayon, Arnaud Cuccuru, Chokri Mraidha

BIOGRAPHY:

Chokri Mraidha is a researcher at the CEA-List in the Accord/UML team lead by Sebastien Gerard. He got a master degree in distributed computing in 2001 and a PhD in Computer Science from Evry Val d'Essonne University in 2005. His research interests include real-time and embedded systems model driven development, model executability and heterogeneity.

Approaches to Refining the Semantic Unit Library

The Model-Integrated Computing (MIC) tool suite developed at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University is a metaprogrammable tool infrastructure for the model-based design of real-time and embedded software and systems. Within the MIC infrastructure, Domain-Specific Modeling Languages (DSML-s) capture the structure of systems and model transformation methods are invoked to map models to implementation domains. The MIC semantic anchoring framework specifies semantics for DSML-s via the specification of transformations between domain specific models and a set of semantic units. Previously introduced semantic units for finite state machines and timed automata provided architectural language specifications of the operational semantics of these common behavioral categories using the Abstract State Machine (ASM) formalism; however, ongoing work in refining these semantic units has illustrated the need for developing precise specifications that are both general to a wide-array of common behavioral languages/models and extensible for further refinement to other behavioral categories. Also, applying the semantic anchoring approach has shown that the ASM formalism alone may not be suitable for accurately capturing behavioral models of real-time systems. Current work is investigating the Discrete Event System specification (DEVS) formalism as a possible supplement or extension to the semantic anchoring framework for the concrete specification of componentized software systems that explicitly require the concept of physical time for determining possible behaviors.

BIOGRAPHY:

Ryan Thibodeaux is an Electrical Engineering Ph.D. student and research assistant at the Institute for Software Integrated Systems at Vanderbilt University in Nashville, TN. He received his B.E. in Computer Engineering in 2006 from Vanderbilt University. His research interests include behavioral specifications of domain-specific modeling languages, model-based design tool-chains, and the design of real-time and safety-critical embedded systems.

Composition Issues for Behavioral Semantics of Domain-Specific Modeling Languages

Designers may use semantic units to formally specify behavior in architectural languages. Natural design processes lead to heterogeneous semantics -- for example, component behaviors may be specified as finite automata, while interactions between those components may be specified using data flow concepts.

Composition of semantic units presents a number of challenges for language designers, with commensurate benefits. Many formal modeling frameworks include property verification, simulation, and design exploration capabilities. In order to achieve the benefits of formality, semantic compositions must maintain mathematical consistency. Further, the benefits from resolving the details of the behavioral semantics during composition are offset by the difficulties of creating modeling tools to represent, navigate, and organize those details. Solutions to these problems draw on a number of active research areas in formal language tools, simulation, and mathematical models for semantics.

We aim to provide modeling tools to give designers the power and flexibility to specify heterogeneous behaviors with confidence. Such tools must include means for detecting and resolving behavioral inconsistencies as well as straightforward interfaces to formal verification and simulation tools. We will present a simple case study to demonstrate some of the difficulties and trade-offs that must be resolved to achieve such compositions. The case study features a modeling language that combines static dependencies with formal operational definitions modeled as abstract state machines to describe combined behaviors. This work builds on a preliminary case study presented by the author at the Abstract State Machines conference earlier this year.

BIOGRAPHY:

Joseph Porter is a first-year PhD student with Vanderbilt University's Institute for Software Integrated Systems (ISIS). He received his MSEE and BSEE from the University of Kentucky, and most recently worked as a researcher and developer at Southwest Research Institute in San Antonio, Texas. Experience and interests include model-based design, implementation, and validation for digital systems in control and signal processing.