Towards Systematic Model-Based Development of Embedded Systems

Janos Sztipanovits
ISIS, Vanderbilt University

DATE 07
Nice, France
April 20, 2007
• Introduction
  – Basic concepts: platforms, abstractions and DSML-s

• Model Integrated Computing
  – Structural and Behavioral Semantics
  – Metamodel composition
  – MIC Tool Suite

• Making Behavioral Semantics Explicit
  – Semantic anchoring
  – Remarks on composition
Content

Introduction
  – Basic concepts: platforms, abstractions and DSML-s

• Model Integrated Computing
  – Structural and Behavioral Semantics
  – Metamodel composition
  – MIC Tool Suite

• Making Behavioral Semantics Explicit
  – Semantic anchoring
  – Remarks on composition
Changing Platforms

Emerging new platforms continue changing requirements for embedded systems design.

The changes in platforms are fundamental:
- parallel architectures,
- customizable HW

Cell Processor Platform
- Power processor element
- 8 Synergistic processing elements
- Element Interconnect bus

Architecture Constraints
- Single PPE
- Elements must share memory
- Elements must share interconnect bus

Xilinx DSP Hardware Platform
- Programmable arithmetic fabric
- Hundreds of Xtreme DSP slices
- 100+ multipliers

Flexible Architecture
- Custom HW
- Parameterizable HW IP
- Open HW data format

Kees Vissers, Xilinx, MPSoC 2005
Changing Applications

- The share of value of embedded computing components in the application areas expected to reach 30%-40% value by 2010
- Applications:
  - Automotive Systems
    - Light and heavy automobiles, trucks, buses
  - Aerospace Systems
    - Airplanes, space systems
  - Consumer electronics
    - Mobile phones, office electronics, digital appliances
  - Health/Medical Equipment
    - Patient monitoring, pumps, artificial organs
  - Industrial Automation
    - Supervisory Control and Data Acquisition (SCADA) systems, chemical and power plants
    - Manufacturing systems
  - Defense
    - Source of superiority in all weapon systems
- National Health Information Network, Electronic Patient Record
  - Md l d t it f i
- Applications:
  - Automotive Systems
  - Light and heavy automobiles, trucks, buses
  - Aerospace Systems
  - Airplanes, space systems
  - Consumer electronics
  - Mobile phones, office electronics, digital appliances
  - Health/Medical Equipment
  - Patient monitoring, pumps, artificial organs
  - Industrial Automation
  - Supervisory Control and Data Acquisition (SCADA) systems, chemical and power plants
  - Manufacturing systems
  - Defense
  - Source of superiority in all weapon systems
- Operating Room of the Future (Goldman)
  - Closed loop monitoring and control; multiple treatment stations, plug and play devices; robotic microsurgery
  - System coordination challenge
- Progress in personalized medicine: systems biology; disease dynamics, control
Platforms Are Layers of Abstractions

Key Idea: Manage design complexity by creating layers of abstractions in the design flow.
(Platform-based design: Alberto Sangiovanni-Vincentelli)

Abstraction layers define platforms.

Abstractions are linked through refinement relations.

Abstraction layers allow the verification of different properties.

Software architecture defines the composition of functions such that a least fixed point exists and is unique.

Hardware architecture defines a set of concurrent functional units, where the software architecture can be deployed.

Behavior models define a set of timed automata with local clocks and broadcast. Models can be analyzed with TCTL.
A Basic Pattern in Design Flows

Behavioral libraries
- Capture behavior
- Verify behavior

Architecture libraries
- Capture Architecture
- Verify Architecture

Map behavior to architecture
- Verify uP performance

Refine HW/SW uP architecture
- Link to uP Architecture ver.

Link to HW/SW implementation

Platform 1
- Platform 2

Platform 3

High-Level Design Flow for Embedded Systems: Alberto Sangiovanni-Vincentelli
Vehicle Control Platform (VCP) is a heterogeneous experimental tool chain for automotive.
Content

• Introduction
  – Basic concepts: platforms, abstractions and DSML-s

• Model Integrated Computing
  – Structural and Behavioral Semantics
  – Metamodel composition
  – MIC Tool Suite

• Making Behavioral Semantics Explicit
  – Semantic anchoring
  – Remarks on composition
Key Idea: Capture intrinsic domain concepts with domain-specific modeling languages (DSML-s) and partition DSML-s into structural and behavioral semantics.

- The **structural semantics** views a model as a structure, and provides a means for calculating which structures are well-formed.
  - No operator was provided for composition of values, so this merge model is semantically meaningless in this domain.

- The **behavioral semantics** defines what the structures do.
  1. A block $f$ represents an $n$-ary map over some value domain. $f : \mathcal{V}^n \rightarrow \mathcal{V}^m$.
  2. A connection $c$ represents an projection operator: $\pi_{i,m} : \mathcal{V}^m \rightarrow \mathcal{V}$, where $\pi_{i,m}(v_0, v_1, \ldots, v_{m-1}) \mapsto v_i$.
  3. Composition is by function composition: $\text{splitter}(\pi_{0,2} \circ \text{fft}(\text{in1}(t), \text{in2}(t)), \pi_{1,2} \circ \text{fft}(\text{in1}(t), \text{in2}(t)))$. 

---

Model-Integrated Computing
Specification of Structural Semantics of DSML-s

- Metamodels define the structural semantics of DSML-s.

\[
L = \{Y, R_Y, C, (\bigcup_{i \in J} \}) \\
D (Y, C) = \{ r \in R_Y \mid r \models C \}
\]

- GME, the metaprogrammable modeling tool of ISIS, supports rapid construction of metamodels and DSML models.

Metamodeling languages provide structural semantics. Basic metamodeling notation: UML Class Diagram + OCL

Mathematical framework for structural semantics: Terms Algebra (Jackson, Sztipanovits EMSOFT 2006)

A metamodeling language is one of the DSML-s: the same tool can be used for modeling and metamodeling.

MetaGME metamodel of simple statecharts  Model-editor generated from metamodel
Specification of Behavioral Semantics of DSML-s

- Behavioral semantics are defined with model transformations and semantic anchoring.

C++ coding permits complex behavioral semantics, but the “specifications” are cluttered with C++ details.

Graph transformations provide a transparent mechanism to attach semantics. However, not all behavioral semantics can be specified this way.

Semantic anchoring with ASM captures the best of both worlds: Simple graph transformations and simple behavioral specifications.

- Simulation artifacts and test cases can be generated.

Canonical definition of FSM behavioral semantics in ASM

Translation of GME model to ASM data structures
Content

• Introduction
  – Basic concepts: platforms, abstractions and DSML-s

• Model Integrated Computing
  – Structural and Behavioral Semantics
  – Metamodel composition
  – MIC Tool Suite

• Making Behavioral Semantics Explicit
  – Semantic anchoring
  – Remarks on composition
Composing DSML-s

Key Idea: Decompose complex models into aspects. Aspects are composed to create complete specification.

Each view may represent a design concern (separation of concerns).

Design aspects are typically non-orthogonal.

Composition of non-orthogonal DSML-s is a significant challenge for the tool infrastructure.

- An architectural specification is created in each view
- The total system is the composition (synthesis) of all the views

Key Question: If properties hold in a view, do they hold in the entire system?

Key Question: How do engineers keep track of cross-aspect interactions?
DSML Composition is an Essential Practical Tool

Tool support for DSML (metamodel) composition in GME:
- Class merge
- Metamodel interfacing
- Class refinement
- Template instantiation
- Model transformation

Understanding structural and behavioral semantics of composed DSML-s is essential.

Class Refinement
Objective: Optimize the SW architecture by selecting a component model and by allocating functions to components.
Platform: TT Component Model
Tools: GME, GReAT (SL/SF to C generator), C Compiler, WCET Analyzer
VCP Tool-Chain: Tools and Interfaces

Design Space Exploration

- DESERT
- MDL2ECSL

ECSL-DP/GME

ECSL2...

ECSL-Generator/s
GReAT

Platform Code/Configs.

ARIES

DESERT

Testing

CG
VCP Tool Chain Output

Functional Code

OS/Firmware Glue Files

OIL File

DBC File (CAN-Bus)
Content

• Introduction
  – Basic concepts: platforms, abstractions and DSML-s

• Model Integrated Computing
  – Structural and Behavioral Semantics
  – Metamodel composition
  – MIC Tool Suite

Making Behavioral Semantics Explicit
  – Semantic anchoring
  – Remarks on composition
The Generic Modeling Environment (GME) is used for building metamodels and is customized by metamodels (metaprogrammability).

MetaGME combines language constructs for metamodeling and for specifying concrete syntax of DSML-s.

Metacircularity enables changing metalanguages without changing the tool.

The metamodel for MetaGME is the meta-metamodel – defined in MetaGME.

- Configuration through UML and OCL-based metamodels
- Extensible architecture through COM
- Multiple standard backend support (ODBC, XML)
- Multiple language support: C++, VB, Python, Java, C#
Model Data Management: The UDM Tool Suite

The goal of UDM is to have a conceptual view of data/metadata that is independent of the storage format.

UDM provides uniform access to data/metadata such that storage formats can be changed seamlessly at either design time or run time.

UDM generates a metadata/paradigm specific API to access a particular class of data.
MIC Model transformation technology is based on graph transformation semantics.

Model transformations are specified using metamodels and the code is automatically generated from the models.

Models of transformations are expressed in a DSML and built in GME.
The MIC tool suite is metaprogrammable: each component can be customized to domain specific modeling languages by means of metamodels.

All tools are available from a quality controlled open source repository.

ESCHER Quality Controlled Repository: http://escher.isis.vanderbilt.edu
Model-based Software Development

Essential questions:

How do we know that the model transformations (model translator/code generator) are correct?

How do we know that the verified properties of the models are preserved by the generated code running on the selected execution platform?

- Formally specifies the modeling language(s)

METAMODEL

Hand-written Code

Domain Models

MODEL TRANSLATOR

Verification Engine

Code Generator

COMPLILER

Executable components/code

Execution Platform

Implicitly implements the semantics of the modeling language
• Introduction
  – Basic concepts: platforms, abstractions and DSML-s

• Model Integrated Computing
  – Structural and Behavioral Semantics
  – Metamodel composition
  – MIC Tool Suite

• Making Behavioral Semantics Explicit
  – Semantic anchoring
  – Remarks on composition
Specification of Behavioral Semantics of DSML-s

- Behavioral semantics are defined with model transformations and semantic anchoring.

C++ coding permits complex behavioral semantics, but the “specifications” are cluttered with C++ details.

Graph transformations provide a transparent mechanism to attach semantics. However, not all behavioral semantics can be specified this way.

Semantic anchoring with ASM captures the best of both worlds: Simple graph transformations and simple behavioral specifications.

- Simulation artifacts and test cases can be generated
Search for a Formal Framework

- Specification style: Operational semantics
- Solid mathematical foundation
- Tool support for core use cases:
  - Readability (clear syntax and “good-enough” semantics)
  - Validation/exploration of semantics (executable specification)
  - Verification of semantic equivalence (generation of “reference traces”, integratability)

After evaluating several frameworks (Z, TLA+,..) we selected ASM and the AsmL tool suite. AsmL has been developed at MSR (Gurevich)
### Example Specification: FSM

#### Abstract Data Model

```csharp
structure Event
    eventType as String

class State
    initial as Boolean
    var active as Boolean = false

class Transition

abstract class FSM
    abstract property states as Set of State
    abstract property transitions as Set of Transition
    abstract property outTransitions as Map of <State, Set of Transition>
    abstract property dstState as Map of <Transition, State>
    abstract property triggerEventType as Map of <Transition, String>
    abstract property outputEventType as Map of <Transition, String>
```

#### Interpreter

```csharp
abstract class FSM
Run (e as Event) as Event?
    step
        let CS as State = GetCurrentState ()
    step
        let enabledTs as Set of Transition = {t | t in outTransitions (CS) where e.eventType = triggerEventType(t)}
    step
        if Size (enabledTs) >= 1 then
            choose t in enabledTs
            step
                CS.active := false
            step
                dstState(t).active := true
            step
                if t in me.outputEventType then
                    return Event(outputEventType(t))
                else
                    return null
        else
            return null
```

---

**Underlying abstract machine:** ASM  
**Language:** AsmL
Content

• Introduction
  – Basic concepts: platforms, abstractions and DSML-s

• Model Integrated Computing
  – Structural and Behavioral Semantics
  – Metamodel composition
  – MIC Tool Suite

• Making Behavioral Semantics Explicit
  – Semantic anchoring
  – Remarks on composition
DSML Design Through Semantic Anchoring

**Step 1**
- Specify the DSML by using metamodels.

**Step 2**
- Select appropriate semantic units for the behavioral aspects of the DSML.

**Step 3**
- Specify the semantic anchoring $M_A = A \rightarrow A_i$ by using UMT.
Experimental Tool Suite for Semantic Anchoring

- **Metamodelling and Model Transformation Tools**
  - **GME Toolset**
    - DSML Metamodel (A)
    - Domain Model (C)
  - **GReAT Tool**
    - Model Trans. Rules ($M_A$)
    - Transformation Engine
  - **Semantic Model Metamodel (A_i) Model Checker**
  - **Domain Model (C_i)**
  - **XSLT**

- **Formal Framework for Semantic Units Specification**
  - **Semantic Unit Spec.**
    - Abstract Data Model
    - Operational Semantics Spec.

- **Tools for Semantic Unit Specification**
  - **ASM**: A particular kind of mathematical machine, like the Turing machine. (Yuri Gurevich)
  - **AsmL**: A formal specification language based on ASM. (Microsoft Research)
Example: HFSML => FSM-SU
Example: HFSML => FSM-SU

GME Toolset
- HFSML Metamodel (A)
- Domain Model (C)

GReAT Tool
- Model Trans. Rules ($M_A$)
- Transformation Engine
- FSM Metamodel ($A_i$)
- FSM Model ($C_i$)

FSM-SU Specification
- Abstract Data Model
- Operational Semantics Spec.

XSLT
- ASM Semantic Framework

Structure Event
- eventType as String

Class State
- id as String
- initial as Boolean
- var active as Boolean = false

Class Transition
- id as String

Abstract class FSM
- id as String
- abstract property states as Set of State
- get
- abstract property transitions as Set of Transition
- get
- abstract property outTransitions as Map of <State, Set of Transition>
- get
- abstract property dstState as Map of <Transition, State>
- get
- abstract property triggerEventType as Map of <Transition, String>
- get
- abstract property outputEventType as Map of <Transition, String>

React (e as Event) as Event?
- step
  - let CS as State = GetCurrentState()
  - step
    - let enabledT = Set of Transition = {t | t in outTransitions(CS) where e.eventType = triggerEventType(t) and e.isTriggering() = true}
    - step
      - if Size(enabledT) = 1 then
        - choose t in enabledT
        - step
          - // WriteLine("Execute transition: " + t.id)
          - CS.active := false
        - step
          - dstState(t).active := true
        - step
          - if t in me.outPutEvent then
            - return Event(outputEventType(t))
          - else
            - return null
      - else
        - error("NON-DETERMINISM ERROR!")
    - else
      - if Size(enabledT) > 1 then
        - error("NON-DETERMINISM ERROR!")
      - else
        - return null
Example: HFSML => FSM-SU

GME Toolset

HFSML Metamodel (A)

Domain Model (C)

GReAT Tool

Model Trans. Rules (M_a)

Transformation Engine

FSM Metamodel (A_i)

FSM Model (C_i)

FSM-SU Specification

Abstract Data Model + Operational Semantics Spec.

Instance

XSLT

ASM Semantic Framework

Sztipanovits: 35
Content

• Introduction
  – Basic concepts: platforms, abstractions and DSML-s

• Model Integrated Computing
  – Structural and Behavioral Semantics
  – Metamodel composition
  – MIC Tool Suite

• Making Behavioral Semantics Explicit
  – Semantic anchoring
  – Remarks on composition
Heterogeneous DSMLs

• Heterogeneity of systems
  – Complex systems are composed of heterogeneous components using heterogeneous interactions. Modeling and design of heterogeneous systems is a significant challenge.

• Heterogeneity of tool chains
  – Tool chains supporting domain-specific design flows integrate modeling, analysis and synthesis tools using DSMLs with overlapping semantics.

• The semantics of a heterogeneous DSML is probably not captured by a single predefined semantic unit.
Example: EFSM

- EFSM has been developed by General Motors Research to specify vehicle motion control (VMC) software.
- The SEFSM model is a synchronous reactive system including a set of components communicating through event channels and data channels.
- A SEFSM component is an FSM-based model, which integrates a set of stateless computational functions that consume input data and produce output data.
- Events determine which components are to be activated and the order of activations.
Modular Specification of Semantics

**Remark:** The behavioral composition specifies a controller, which restricts the executions of actions. Since the behavior of the embedded semantic units can be described as partial orders on the sets of actions they can perform, the behavioral composition can be modeled mathematically as a composition of the partial orders.

- **Structural Composition** yields the composed Abstract Data Model, \( A =< A_C, A_{SU1}, A_{SU2}, g_1, g_2 > \) where \( g_1, g_2 \) are the partial maps between concepts in \( A_C, A_{SU1}, \) and \( A_{SU2} \).

- **Behavioral composition** is completed by the \( R_C \) set of rules that together with \( R_{SU1} \) and \( R_{SU2} \) form the R rule set for the composed semantics.
The behavior of SEFSM components can be divided into two different behavioral aspects: the FSM-based behavior expressing reactions to events and the SDF-based behavior controlling the execution of computational functions (actions and guards).
A SEFSM system is composed of a set of components, which communicate with each other through event channels and data channels.

The semantics of SEFSM systems is defined as the composition of FSM-SU and SDF-SU.
Summary

• Rapid changes in implementation platforms and heterogeneity of embedded systems applications demand domain specific approaches.
• Recent advancements in model-based design provide reusable infrastructure for building domain specific tool chains or domain specific extensions to established tool frameworks.
• The key requirement is to make structural and behavioral semantics explicit.
• State-based and event-based semantics play important role in this.