Model-Based Composition of Embedded Software and Systems

Janos Sztipanovits
Institute for Software Integrated Systems (ISIS)
Vanderbilt University

Hong Kong Science Park
Shatin, Hong Kong
February 21-22, 2006
Content

- Domain Specific Tool Chains
  - Modeling Languages
  - Challenges in Building Tool Chains
- Metamodeling and Metaprogrammable Tools
  - Metamodeling
  - Model Data Management
  - Model Transformations
  - Tool Integration
- Semantics for Tool Integration
  - Semantic Units
  - Semantic Anchoring
Goal and Approaches

Building increasingly complex networked embedded systems from components
- Naïve “plug-and-play” approach does not work in embedded systems (neither in larger non-embedded systems)
- Model-based software design focuses on the *formal representation, composition, analysis and manipulation of models* during the design process.

Approaches with differences in focus and details
- MDA: Model Driven Architecture
- MDD: Model-Driven Design
- MDE: Model-Driven Engineering
- MIC: Model-Integrated Computing
Composition Concerns in Model-Based Design

System Composition (Product Models)

- Heterogeneous Distributed Embedded Layered

Tool Composition (Design Process Models)

- Composable
- Integrated
- Correct by construction

Single Tools → Customizable Frameworks → Composition Frameworks

- UPAAL
- Rational Rose
- SL/SE
- VS

ECLIPSE TOOLS

ESCHER TOOLS

Workshop on Hybrid and Embedded Systems 2006
| Component Behavior | Modeled on different levels of abstraction:  
|                    | • Transition systems (FSM, Time Automata, Cont. Dynamics, Hybrid), **fundamental role of timed models**  
|                    | • Precise relationship among abstraction levels  
|                    | • Research: **dynamic/adaptive behavior**  
| Structure          | Expressed as a system topology:  
|                    | • Module Interconnection (Nodes, Ports, Connections)  
|                    | • Hierarchy  
|                    | • Research: **dynamic topology**  
| Interaction        | Describes interaction models among components:  
|                    | • Set of well-defined **Models of Computations (MoC)** (SR, SDF, DE,...)  
|                    | • **Heterogeneous**, but precisely defined interactions  
|                    | • Research: **interface theory** (time, resources,...)  
| Scheduling / Resource Allocation | Mapping/deploying components on platforms:  
|                                | • Dynamic Priority  
|                                | • Behavior guarantees  
|                                | • Research: composition of schedulers |
### Examples for Approaches

<table>
<thead>
<tr>
<th>Component Behavior</th>
<th>Process (Hierarchical Timed Automaton)</th>
<th>Metropolis (ASV¹, UCB)</th>
<th>Ptolemy II (Lee, UCB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF (Sifakis, Verimag)</td>
<td>Dynamically Created Channels + Flexible interaction modeling: - blocking - non-blocking</td>
<td>Process (Hierarchical, Active Components)</td>
<td>Java Code/Behavioral Models</td>
</tr>
<tr>
<td>Structure</td>
<td>Netlists (port, interface, connection)</td>
<td>Medium (port, parameter, useport)</td>
<td>Hierarchical Module Interconnection</td>
</tr>
<tr>
<td>Interaction</td>
<td>Dynamic Priorities</td>
<td>Scheduler (port, parameter)</td>
<td>Heterogeneous Models of Computation + Directors</td>
</tr>
<tr>
<td>Scheduling / Resource Allocation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹ ASV: Alberto Sangiovanni

---

1. Alberto Sangiovanni-Vincentelli

---

*Workshop on Hybrid and Embedded Systems 2006*
**Tool Composition Dimension: Core Modeling Layers**

<table>
<thead>
<tr>
<th>Domain-Specific Tools; Design Environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Domain Specific Tool Chains</td>
</tr>
<tr>
<td>- Adopt domain specific modeling concepts</td>
</tr>
<tr>
<td>- Support domain specific design-flows</td>
</tr>
<tr>
<td>- Reflect salient design concerns of the domain</td>
</tr>
<tr>
<td>- Include relevant platforms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metamodeling and Metaprogrammable Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Metaprogrammable Tool Suites provide reusable tools and tool integration frameworks for creating domain specific tool chains</td>
</tr>
<tr>
<td>- Model builders</td>
</tr>
<tr>
<td>- Model transformation tools</td>
</tr>
<tr>
<td>- Model data management tools</td>
</tr>
<tr>
<td>- Design-space exploration tools</td>
</tr>
<tr>
<td>- Analysis tools</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Semantic Foundation</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Semantic Foundations (work in progress):</td>
</tr>
<tr>
<td>- Semantic “Units”</td>
</tr>
<tr>
<td>- Semantic Anchoring Environment (SAE)</td>
</tr>
</tbody>
</table>
Content

- Domain Specific Tool Chains
  - Modeling Languages
  - Challenges in Building Tool Chains

- Metamodeling and Metaprogrammable Tools
  - Metamodeling
  - Model Data Management
  - Model Transformations
  - Tool Integration

- Semantics for Tool Integration
  - Semantic Units
  - Semantic Anchoring
Model-Integrated Computing Approach (ISIS-VU)

Modeling Domain Specific Design Flows: Examples in MIC:
- ECSL - Automotive
- ESML - Avionics
- SPML - Signal Processing
- CAPE/eLMS - Learning Technology

Metamodeling and Metaprogrammable Tools: (mature or in maturation program)
- GME (Generic Model Editor)
- GReAT (Model Transformation)
- OTIF (Tool Integration Framework)
- UDM (Universal Data Model)
- DESERT (Design Space Exploration)

Modeling Semantics (work in progress):
- Semantic “Units”
- Semantic Anchoring

Workshop on Hybrid and Embedded Systems 2006
Modeling Formalisms Are Different

```
medium IntM implements IntWriter, IntReader, IW, IR, IC, IS, IN {
    int storage, space, n;
    IntM() { space = 1; n = 0; }
    update void writeInt(int data) {
        await (space>0; this.IW, this.IS; this.IW)
        await (true; this.IC; this.IS, this.IN; this.IC) {
            space = 0; n = 1;
            storage = data;
        }
    }
    update int readInt() {
        await (n>0; this.IR, this.IB; this.IB)
        await (true; this.IC, this.IS, this.IN; this.IC) {
            space = 1; n = 0;
            return storage;
        }
    }
    
    process Y {
        port IntReader port0;
        port IntReader port1;
        port IntWriter port2;
        ...
        void thread() {
            int x;
            while(true) {
                await {
                    (port0.x>0 && port1.x>0);
                    port0.IntReader, port1.IntReader;
                    port0.IntReader, port1.IntReader
                } {
                    z = foo(port0.readInt(), port1.readInt());
                }
                port2.writeInt(z);
            }
            int foo(int x, int y) {
            ...
        }
    }
```

Ptolemy II

Metropolis

Workshop on Hybrid and Embedded Systems 2006
Emergence of Modeling Language Standards

- SySML

- Others (UML-2; RT-UML, SLML, AADL, ...)

SysML Partners [www.sysml.org](http://www.sysml.org)

Workshop on Hybrid and Embedded Systems 2006
Current Status of System/SW Modeling Languages

- The number of new standards is growing driven by competing consortiums and .org-s
- Intended scope ranges from “unified” to “specific”.
- Many views them as programming languages
  - Wait for the “Unified One” to ensure reusability of tools
  - Slow down deployment because of the lack of standards
  - Wait for executable models

- Modeling and analysis tools are not integratable (closed camps emerge protected by a “standard”).
- Semantics is largely neglected or left to undocumented interpretations of tool developers.
Trends in Modeling Languages

- Increasing acceptance of metamodeling and Domain-Specific Modeling Languages based on standard metamodels (Meta Object Facility, MOF)
- Emergence of metaprogrammable tools
- Desire for solving the “semantics problem”
- Better understanding of the role of precise model transformations in model-based generators and in building domain-specific tool chains from reusable tools
Example Tool Chain: Vehicle Control Platform (VCP)

Common Semantic Domain: Hybrid Automata
Abstract Syntax and Transformations: Meta-Models
Domain Models and Tool Interchange Formats: Tool Chains

Vehicle Control Platform (VCP)

Behavior Model
Component Structure
Component Interaction
Schedulability Analysis
Behavior Simulation

Simulink Stateflow
ECSL-DP
OSEK/Code
DESSERT

(Experimental tool chain integrated using OTIF)

Workshop on Hybrid and Embedded Systems 2006
Constructing Tool Chains: Modeling and Transformations

- Large influence of concrete syntax
- No clear role of semantics
- It is not clear what are we doing?
Content

- Domain Specific Tool Chains
  - Modeling Languages
  - Challenges in Building Tool Chains
- Metamodelling and Metaprogrammable Tools
  - Metamodelling
  - Model Data Management
  - Model Transformations
  - Tool Integration
- Semantics for Tool Integration
  - Semantic Units
  - Semantic Anchoring
Metamodeling Layer Objectives

- Metamodelling
- Model Data Management
- Model Transformation
- Tool Integration
Metamodeling and Domain Specific Modeling Languages

Domain Specific Modeling Language (DSML)

\[ L = < C, A, S, M_S, M_C> \]

- **Model**: precise representation of artifacts in a modeling language \( L \)
- **Modeling language**: defined by the notation (C), concepts/relations and integrity constraints (A), the semantic domain (S) and mapping among these.
- **Metamodel**: formal (i.e. precise) representation of the modeling language \( L \) using a metamodelling language \( L_{\text{M}} \).
Modeling Example: Metamodel and Models

**Metamodel:**
- Defines the set of admissible models
- “Metaprogramms” tool

**Model:**
- Describes states and transitions
- Modeling tool enforces constraints

*Workshop on Hybrid and Embedded Systems 2006*
Metaprogrammable Modeling Tool: GME

- 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#
Content

- **Domain Specific Tool Chains**
  - Modeling Languages
  - Challenges in Building Tool Chains

- **Metamodeling and Metaprogrammable Tools**
  - Metamodeling
  - Model Data Management
  - Model Transformations
  - Tool Integration

- **Semantics for Tool Integration**
  - Semantic Units
  - Semantic Anchoring
Model Data Management: The UDM Goals

- To have a conceptual view of data/metadata that is independent of the storage format.
- Such a conceptual view should be based on standards such as UML.
- Have uniform access to data/metadata such that storage formats can be changed seamlessly at either design time or run time.
- Generate a metadata/paradigm specific API to access a particular class of data.
Model Data Management:
The UDM Tool Suite
Content

- Domain Specific Tool Chains
  - Modeling Languages
  - Challenges in Building Tool Chains
- Metamodeling and Metaprogrammable Tools
  - Metamodeling
  - Model Data Management
  - Model Transformations
  - Tool Integration
- Semantics for Tool Integration
  - Semantic Units
  - Semantic Anchoring
Model Transformation: The “Workhorse” of MIC

Relevant Use of Model Transformations:
- Building integrated models by extracting information from separate model databases
- Generating models for simulation and analysis tools
- Defining semantics for DSML-s

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.
Model Transformation: The GReAT Tool Suite

Tools: UMT Language, GRE (engine), C/G, GR-DEBUG
Content

- Domain Specific Tool Chains
  - Modeling Languages
  - Challenges in Building Tool Chains
- Metamodeling and Metaprogrammable Tools
  - Metamodeling
  - Model Data Management
  - Model Transformations
  - Tool Integration
- Semantics for Tool Integration
  - Semantic Unites
  - Semantic Anchoring
Open Tool Integration Framework: OTIF

RFP is Discussed at MIC PSIG OMG

Share models using Publish/Subscribe Metaphor

Status:
- Completed, tested in several tool chains
- Protocols in OMG/CORBA
- CORBA as a transport layer
- Integration with ECLIPSE is in progress
Integrated MIC Tool Suite

Modeling
- GME
- Model Data Management
- UDM
- OTIF
- Best of Breed
  - Modeling Tools
  - Simulators
  - Verifiers
  - Model Checkers

Design Space Exploration
- DESERT
- Meta Models
- Design Space Exploration
- Design Space Pruning
- Design Space Modeling

Model Transformation
- GReAT
- Model Data Management
- Open Tool Integration Framework
- ESCHER Quality Controlled Repository: http://escher.isis.vanderbilt.edu

Workshop on Hybrid and Embedded Systems 2006
Content

- Domain Specific Tool Chains
  - Modeling Languages
  - Challenges in Building Tool Chains
- Metamodelling and Metaprogrammable Tools
  - Metamodelling
  - Model Data Management
  - Model Transformations
  - Tool Integration
  - Tool Integration Example
- Semantics for Tool Integration
  - Semantic Units
  - Semantic Anchoring
“Front-End View” of the VCP Tool Chain

Vehicle Control Platform (VCP)

Behavior Model
Simulink
Stateflow

Component Structure
ECSL-DP
GME

Component Interaction
DESERT

Schedulability Analysis
vector

Behavior Simulation

(Experimental tool chain integrated using OTIF)

Workshop on Hybrid and Embedded Systems 2006
"Backplane View" of the VCP Tool Chain
VCP Models

**Requirement Specification**

1. The window has to be fully opened and fully closed within 4 [s].

2. If the up or down command is issued for at least 200 [ms] and at most 1 [s], the window has to be automatically opened or closed completely, respectively.

3. The position of the window has to be shown in centimeters.

**Functional Design**

- Forwarder Ignition Clamp
- Forwarder Power Mode
- RWD displayed Controller
- Voltage Surveillance

**HW Config. Design**

- Battery ECU
- Air Conditioning ECU
- CAN bus connector
- CAN bus interface

**Software Architecture Design**

- SW Deployment
- SW Deployment

**System Config. Design**

- T_SPO
- T_Fwd
- T_HST
- T_Fwd

---

Sztipanovits
Objective: Optimize the design of controller to meet requirement specification
Platform: SL/SF modeling language
Tools: SL/SF Model Builder+Simulator

Requirement Specification

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The window has to be fully opened and fully closed within 4 [s].</td>
</tr>
<tr>
<td>2</td>
<td>If the up or down command is issued for at least 200 [ms] and at most 1 [s], the window has to be automatically opened or closed completely, respectively.</td>
</tr>
<tr>
<td>3</td>
<td>The force to detect when an obstacle is present should be less than 100 [N].</td>
</tr>
<tr>
<td>4</td>
<td>When an obstacle is present, the window should be lowered by approximately 10 [cm].</td>
</tr>
</tbody>
</table>
Embedded SW Architecture Design

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

Functional blocks - SW Component Mapping
Objective: Design System configuration that meets cost/reliability/power requirements.
Platform: CAN-Bus; OSEK, ECU
Tools: GME, Firmware and OSEK config. tools
Objective: Optimize System architecture by allocating SW components to OSEK Tasks and Communication Channels. 
Platform: OSEK Model 
Tools: GME, AIRES (schedulability), CANoe (Bus emulator)
VCP Tool Chain Output

Functional Code

OS/Firmware Glue Files

OIL File

DBC File (CAN-Bus)
VCP Design Flow

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The window has to be fully opened and fully closed within 4 [s].</td>
</tr>
<tr>
<td>2</td>
<td>If the up or down command is issued for at least 200 [ms] and at most 1 [s], the window has to be automatically opened or closed completely, respectively.</td>
</tr>
</tbody>
</table>

- **Requirement Specification**
- **Functional Design**
- **Embedded Software Architecture**
- **System Config.**
- **HW Config. Design**
- **SW Deployment**
Design Flow & Tool Chain

- **RA**: Requirement Specification
- **FD**: Functional Design
- **SwA**: Embedded Software Architecture
- **HwA**: HW Config. Design
- **DPL**: SW Deployment
- **SY**: System Config.
- **SL/SF**: SW Deployment

**Tools and Technologies**:
- **GME**: GME, GReAT, C Comp.; WCET
- **AIRES**: GME, AIRES, CANoe
- **OSEK**: GME, OSEK Firmware Config tools

*Workshop on Hybrid and Embedded Systems 2006*
Content

- Domain Specific Tool Chains
  - Modeling Languages
  - Challenges in Building Tool Chains

- Metamodelling and Metaprogrammable Tools
  - Metamodelling
  - Model Data Management
  - Model Transformations
  - Tool Integration
  - Tool Integration Example

- Semantics for Tool Integration
  - Semantic Units
  - Semantic Anchoring
How About Semantics?

Transformational Specification of Behavioral Semantics

\[ M_{S1} = M_{S2} \circ M_{12} \]
The "Semantic Units" are specified in a formal framework such as MoC-s. DSML-s or their aspects are anchored to the common semantics using transformations.
Semantic Anchoring Infrastructure

- **Semantic Unit**
  - A well-defined operational semantics for core Models of Computation and Behaviors (e.g. FSM).
- **Semantic Anchoring**
  - Define the semantics a DSML through specifying the transformation specification to a semantic unit.
FSM Model
Metamodel for AsmL Abstract Data Model

Workshop on Hybrid and Embedded Systems 2006
AsmL Abstract Data Model

```plaintext
structure State
  id as String
  initial as Boolean

structure Transition
  id as String

structure Event
  id as String

class StateAutomaton
  S as Set of State
  T as Set of Transition
  E as Set of Event
  Connections as Map of <Transition, (State,State)>
  TriggerEvent as Map of <Transition, Event?>
  OutputEvent as Map of <Transition, Event?>
  var CurrentState as State
  var OutputEvents as Seq of Event = []

GetOutTransitions (s as State) as Set of Transition
  let trans = {t | t in T where Connections(t).First = s}
  return trans

GetEnabledTransitions (e as Event) as Set of Transition
  let trans = GetOutTransitions(CurrentState)
  let enabledTrans = {t | t in trans where TriggerEvent(t) = e or TriggerEvent(t) = null}
  return enabledTrans

DoTransition(t as Transition)
  step
    if OutputEvent(t) <> null then
      OutputEvents := OutputEvents + [OutputEvent(t)]
      WriteLine("OutputEvent " + OutputEvent(t).id)
    CurrentState := Connections(t).Second
    step WriteLine("Do transition " + t.id)

React(e as Event)
  step until fixpoint
    let trans = GetEnabledTransitions(e)
    if Size(trans) <> 0 then
      choose t in trans
      DoTransition(t)
```

Sztipanovits
AsmL Behavioral Semantic Specifications

Behavior in Terms of Abstract Model

React(e as Event)
  step until fixpoint
  let trans = GetEnabledTransitions(e)
  if size(trans) <> 0 then
    choose t in trans
    DoTransition(t)

GetOutTransitions (s as State) as Set of Transition
  let trans = \{ t | t in T where Connections(t).First = s \}
  return trans

GetEnabledTransitions (e as Event) as Set of Transition
  let trans = GetOutTransitions(CurrentState)
  let enabledTrans = \{ t | t in trans where TriggerEvent(t) = e or TriggerEvent(t) = null \}
  return enabledTrans

DoTransition(t as Transition)
  step
    if OutputEvent(t) <> null then
      OutputEvents := OutputEvents + [OutputEvent(t)]
      WriteLine ("OutputEvent " + OutputEvent(t).id)
    step CurrentState := Connections(t).Second
    step WriteLine ("Do transition " + t.id)

React(e as Event)
  step until fixpoint
  let trans = GetEnabledTransitions(e)
  if Size(trans) <> 0 then
    choose t in trans
    DoTransition(t)
Transformational Specifications
AsmL Data Model in XML Format

<AsmLADS _id="id988" fileName="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <FSM id="ChecksumMachina" _id="id9d5" initialState="OFF">
    <Children _id="id9f4">
      <LocalEvents _id="id9e5"/>
      <OutputEvents _id="id9e0"/>
    </FSM>
    <Input _id="id9bb">
      <LocalEvent _id="id9bf">
        <State id="OFF" _id="ida17" active="false" master="" initial="true" initialState=""/>
        <OutTransitions _id="ida5b">
          <Slaves _id="ida4d"/>
        </OutTransitions>
      </State>
      <State id="ON" _id="ida18" active="false" master="" initial="false" initialState="ZERO"/>
      <State id="ZERO" _id="ida74" active="false" master="ON" initial="false" initialState="ZERO"/>
      <State id="ONE" _id="ida75" active="false" master="ONE" initial="false" initialState="ZERO"/>
      <Transition id="T11" _id="ida7d" dst="ONE" src="ZERO" guard="true" preemptive="false" outputEvent="" triggerEvent="LocalEvent.start"/>
      <Transition id="T12" _id="idaaf" dst="ZERO" src="ONE" guard="true" preemptive="false" outputEvent="" triggerEvent="LocalEvent.start"/>
      <Transition id="T13" _id="idae0" dst="ZERO" src="ZERO" guard="true" preemptive="false" outputEvent="" triggerEvent="LocalEvent.start"/>
      <Transition id="T14" _id="idae1" dst="ONE" src="ONE" guard="true" preemptive="false" outputEvent="" triggerEvent="LocalEvent.start"/>
      <Transition id="T15" _id="idb0e" dst="ON" src="OFF" guard="true" preemptive="false" outputEvent="" triggerEvent="ModelEvent.stop"/>
      <Transition id="T20" _id="idb0f" dst="OFF" src="ON" guard="true" preemptive="false" outputEvent="" triggerEvent="ModelEvent.reset"/>
      <Transition id="T3" _id="idb10" dst="ON" src="ON" guard="true" preemptive="false" outputEvent="" triggerEvent="ModelEvent.reset"/>
    </Input>
  </AsmLADS>
AsmL Data Model

Instance of the Abstract Model

```plaintext
initStateAutomaton() as StateAutomaton
let S1 = State("S1", true)
let S2 = State("S2", false)
let T1 = Transition("T1")
let e1 = Event("e1")
let S = {S1, S2}
let T = {T1}
let E = {e1}
let Connections = {T1 -> (S1, S2)}
let TriggerEvent = {T1 -> e1}
let OutputEvent = {T1 -> e1}
let InitialState = S1
return new StateAutomaton(S, T, E, Connections, TriggerEvent, OutputEvent, InitialState)
```
Interrelation between System and Tool Composition

Component Behavior
- Transition System (Timed Automata)
- Abstract Syntax Metamodeling + Semantic Anchoring
- Domain-Specific Abstractions Notations

Structure
- Set-Valued Semantics (Ports + Connectors)
- Abstract Syntax Metamodeling + Semantic Anchoring
- Component Architecture Models

Interaction
- I/O Automaton
- COMPOSITION PLATFORMS
- METAPROGRAMMABLE TOOLS
- Domain-Specific Abstractions Notations

Scheduling / Resource Allocation
- Compositional Priority Modeling
- Abstract Syntax Metamodeling
- Semantic Anchoring
- Execution Platforms

Semantic Foundation;
- Metaprogrammable Tools, Environments
- Domain-Specific Tools, Tool Chains

Abstract Syntax Metamodeling + Semantic Anchoring
Domain-Specific Abstractions Notations
“Plug-and-Play” component technology is not sufficient for embedded software of non-trivial size.

Model-based design addresses core issues: it integrates systems and software engineering.

Active research programs in system and tool chain composition have made significant progress in the past five years.