Foundations for Model-Based Design

Janos Sztipanovits
ISIS, Vanderbilt University
janos.sztipanovits@vanderbilt.edu

MPSOC 2005
Margaux, France
July 11, 2005
Content

• Introduction to model-based design
• System Composition Dimension
  - Layers
  - Approaches
  - Languages
• Tool Composition Dimension
  - Layers
  - Building Tool Chains
• Metamodeling and Metaprogrammable Tools
• Semantics
Goal and Approaches

• Building increasingly complex networked 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
Two Dimensions of MIC

System Composition (Product Models)
- Heterogeneous
- Distributed
- Embedded
- Layered

Tool Composition (Design Process Models)
- Composable
- Integrated
- Correct by construction

- Single Tools
- Customizable Frameworks
- Composition Frameworks

Tools:
- UPAAL
- Rational Rose
- SL/SF
- VS
- ECLIPSE
- ESCHER TOOLS

www.escherinstitute.org
• Introduction to model-based design
• **System Composition Dimension**
  - Layers
  - Approaches
  - Languages
• **Tool Composition Dimension**
  - Layers
  - Building Tool Chains
• Metamodeling and Metaprogrammable Tools
• Semantics
## System Composition Dimension: Core Modeling Aspects

| Component Behavior | Modeled on different levels of abstraction:  
|                    | • Transition systems (FSM, Time Automata, Cont. 
|                    |   Dynamics, Hybrid), **fundamental role of time 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 patterns 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 Research Approaches

<table>
<thead>
<tr>
<th>Component Behavior</th>
<th>Ptolemy II (Lee, UCB)</th>
<th>Metropolis (ASV(^1), UCB)</th>
<th>IF (Sifakis, Verimag)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Java Code/ Behavioral Models</td>
<td>Process (Hierarchical, Active Components)</td>
<td>Process (Hierarchical Timed Automaton)</td>
</tr>
<tr>
<td>Structure</td>
<td>Hierarchical Module Interconnection</td>
<td>Netlists (port, interface, connection)</td>
<td>Dynamically Created Channels</td>
</tr>
<tr>
<td>Interaction</td>
<td>Heterogeneous Models of Computation + Directors</td>
<td>Medium (port, parameter, useport)</td>
<td>Asynchronous Interactions: - P2P - Unicast - Multicast</td>
</tr>
<tr>
<td>Scheduling / Resource Allocation</td>
<td>Scheduler (port, parameter)</td>
<td>Dynamic Priorities</td>
<td></td>
</tr>
</tbody>
</table>

---

1 Alberto Sangiovanni-Vincentelli
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) {
        wait (space>0; this.IW, this.IS; this.IW)
        wait (true; this.IC, this.IS, this.IN; this.IC){
            space = 0; n = 1;
            storage = data;
        }
    }

    update int readInt() {
        wait (n>0; this.IR, this.IN; this.IR)
        wait (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) {
            wait {
                (port0.m()>0 && port1.m()>0;
                port0.IntReader, port1.IntReader;
                port0.IntReader, port1.IntReader)
                { z = foo(port0.readInt().port1.readInt()); } 
            } port2.writeInt(z);
        }
        int foo(int x, int y) { ... }
    }

state send; 
    output stdt(self,m,b) to {receiver}0;
    set t = 10;
    nextstate wait_ack;
    endstate;

state wait_ack; 
    input ack(sender,c);
    ...
    when 10 <= t<20; 
    ...
    endstate;
```

Ptolemy II

Metropolis

IF
Emergence of Modeling Language Standards

- **SySML**

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

• Introduction to model-based design
• System Composition Dimension
  - Layers
  - Approaches
  - Languages
• Tool Composition Dimension
  - Layers
  - Building Tool Chains
• Metamodelling and Metaprogrammable Tools
• Semantics
Tool Composition Dimension:
Core Modeling Aspects

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

Metamodelling 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)
- GME-MOF/Meta (Metamodeling Env-s)

Modeling Semantics (work in progress):
- Semantic “Units”
- Semantic Anchoring
Interrelation with System Composition

Component Behavior
- State Automaton
- Timed Automaton
- Hybrid Automaton
- ...

Structure
- Set-Valued Semantics

Interaction
- Tagged Signal Model
- State Automaton
- Timed Automaton
- ...

Scheduling / Resource Allocation
- Transition Systems With Priority

Abstract Syntax + Semantic Anchoring

Behavior Modeling View

Abstract Syntax + Semantic Anchoring

Structural Modeling Views

Abstract Syntax + Semantic Anchoring

Interaction Modeling Views

Abstract Syntax + Semantic Anchoring

Resource Access Modeling Views

Semantic Foundation;

Metaprogrammable Tools, Environments

Domain-Specific Tools, Tool Chains

Tools, Compositional Platforms

Metaprogrammable 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
GME

OSEK/Code

DESERT

SL/SF \rightarrow ECSL-DP

EDP

SL/SF \rightarrow DSE

EDP

ECSL-DP \rightarrow AIF
- Large influence of concrete syntax
- No clear role of semantics
- It is not clear what are we doing?
• Introduction to model-based design
• System Composition Dimension
  - Layers
  - Approaches
  - Languages
• Tool Composition Dimension
  - Layers
  - Building Tool Chains
• Metamodeling and Metaprogrammable Tools
• Semantics
Metamodeling Layer Objectives

- Metamodeling
- Model Data Management
- Model Transformation
- Tool Integration
Domain Specific Modeling Language (DSML)

- **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 metamodeling language $L_M$. 

$L = < C, A, S, M_s, M_c >$
Modeling Example: Metamodel and Models

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

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

- GME UML
- GME/UML Interpreter
- XML (Meta) <Uml.xsd>
- GME
- Backends
- XML
- MEM
- CORBA
- MGA
- Binary file
- Network
- UdmCopy
- UDM.exe
- .cpp
- .h
- .xsd
- Validates
- Generic API
- OCL Eval
- UDM Generated code
  - API
  - Meta-objects
- User Program
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
Open Tool Integration Framework: OTIF

- **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

http://www.isis.vanderbilt.edu/Projects/WOTIF/default.html
**MIC Metaprogrammable Tool Suite**

**Generic Model Editor**

GME

**Unified Data Model**

GME, UDM, GREAT, DESERT

Completed tool suite, available through the ESCHER Quality Controlled Repository: http://escher.isis.vanderbilt.edu

**Model Transformation**

**Analysis Tools**

- Simulators
- Verifiers
- Model Checkers

**OTIF**

**Persistency Service**

- Database
- XML
- C++ API

**Design Space Exploration**
"Backplane View" of the VCP Tool Chain

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

Vehicle Control Platform (VCP)
• Introduction to model-based design
• System Composition Dimension
  - Layers
  - Approaches
  - Languages
• Tool Composition Dimension
  - Layers
  - Building Tool Chains
• Metamodeling and Metaprogrammable Tools
• Semantics
How About Semantics?

Transformational Specification of Behavioral Semantics

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

Semantic Anchoring
• 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.
Semantic Integration of Tools

Modeling Tool

Analysis Tool

Common Semantic Domain Simulator

Obligation of DSML Developer

Obligation of Tool Developer
Summary

• “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
• New frontier: explicit semantics
FSM Model
AsmL Abstract Data Model

```
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)
  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)
```
AsmL Behavioral Semantic Specifications

Behavior in Terms of Abstract Model

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

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
<AsmLADS _id="id988" fileName="" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="UDM\AsmL.xsd">
  <FSM id="ChecksumMaching" _id="id9d5" initialState="OFF">
    <Children _id="id9f4">
      <LocalEvents _id="id9e5"/>
      <OutEvents _id="id9e0"/>
    </LocalEvents>
  </FSM>
  <Input _id="id98b">
    <LocalEvent _id="id9bf"/>
    <ModelEvent _id="id9ad">
      <State _id="OFF" _id="ida17" active="false" master="" initial="true" initialState=""/>
      <OutTransitions _id="ida5b">
        <Slaves _id="ida3d"/>
      </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=""/>
      <State _id="ONE" _id="ida75" active="false" master="ON" initial="false" initialState=""/>
      <Transition _id="T11" _id="ida0e" dst="ONE" src="ZERO" guard="true" preemptive="false" outputEvent="">
        triggerEvent="LocalEvent.one"/>
      <Transition _id="T12" _id="idaef" dst="ZERO" src="ONE" guard="true" preemptive="false" outputEvent="">
        triggerEvent="LocalEvent.one"/>
      <Transition _id="T13" _id="ida0f" dst="ZERO" src="ZERO" guard="true" preemptive="false" outputEvent="">
        triggerEvent="LocalEvent.one"/>
      <Transition _id="T14" _id="ida1f" dst="ONE" src="ONE" guard="true" preemptive="false" outputEvent="">
        triggerEvent="LocalEvent.one"/>
      <Transition _id="T1" _id="idb0e" dst="OFF" src="OFF" guard="true" preemptive="false" outputEvent="">
        triggerEvent="ModelEvent.start" triggerEvent=""/>
      <Transition _id="T2" _id="idb0f" dst="OFF" src="ON" guard="true" preemptive="false" outputEvent="">
        triggerEvent="ModelEvent.stop" triggerEvent=""/>
      <Transition _id="T3" _id="idb10" dst="ON" src="ON" guard="true" preemptive="false" outputEvent="">
        triggerEvent="ModelEvent.reset" triggerEvent=""/>
    </Transition>
  </State>
</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)
```