Model-Based Design

Edited and presented by
Janos Sztonpanovits
Vanderbilt University
Model-based design focuses on the formal representation, composition, and manipulation of models during the design process.
# System Composition Approaches

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

"Model Based Design", J. Sztipanovits
## Tool Composition Approaches

<table>
<thead>
<tr>
<th>Domain-Specific Tools; Design Environments</th>
<th>Domain-Specific Design Flows and Tool Chains:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• ECSL - Automotive</td>
<td>• ECSL - Automotive</td>
</tr>
<tr>
<td>• ESML - Avionics</td>
<td>• ESML - Avionics</td>
</tr>
<tr>
<td>• SPML - Signal Processing</td>
<td>• SPML - Signal Processing</td>
</tr>
<tr>
<td>• CAPE/eLMS</td>
<td>• CAPE/eLMS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metaprogrammable Tools, Integration Frameworks</th>
<th>MIC Metaprogrammable Tool Suite: (mature or in maturation program)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Metamodeling languages</td>
<td>• Metamodeling languages</td>
</tr>
<tr>
<td>• Modeling Tools</td>
<td>• Modeling Tools</td>
</tr>
<tr>
<td>• Model Transformations</td>
<td>• Model Transformations</td>
</tr>
<tr>
<td>• Model Management</td>
<td>• Model Management</td>
</tr>
<tr>
<td>• Design Space Construction and Exploration</td>
<td>• Design Space Construction and Exploration</td>
</tr>
<tr>
<td>• Tool Integration Framework</td>
<td>• Tool Integration Framework</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Semantic Foundation</th>
<th>Semantic Foundations (work in progress):</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Semantic Anchoring Environment (SAE)</td>
</tr>
<tr>
<td></td>
<td>• Verification</td>
</tr>
<tr>
<td></td>
<td>• Semantic Integration</td>
</tr>
</tbody>
</table>
Intersection of System and Tool Composition Dimensions

- Component Behavior
  - Semantic Units and Semantic Anchoring
  - Compositional Semantics
  - Semantic Foundation;

- Interaction
  - Metamodels, Metamodel Composition & Metaprogammable Tool Chain Composition

- Resource Modeling (Schedule)
  - Model Composition in Domain-Specific Design Flows
  - Metaprogammable Tools, Environments
  - Domain-Specific Tools, Tool Chains

"Model Based Design", J. Sztipanovits

ITR Review, Oct. 4, 2006
Domain Specific Design Flows and Tool Chains

• Integration of tools into tool chains
  - ECSL - Control
  - ESML - Avionics
  - SPP - Signal Processing
  - FCS - Networked Embedded Systems
  - SCA - Software Defined Radio

• Integration among tool frameworks:
  Metropolis, Ptolemy II, MIC,
  Simulink/Stateflow, ARIES, CheckMate,…

• www.escherinstitute.org
Intersection of System and Tool Composition Dimensions

Component Behavior

Semantic Units and Semantic Anchoring

Interaction

Compositional Semantics

Resource Modeling (Schedule)

Semantic Foundation;

Metamodel, Metamodel Composition & Metaprogrammable Tool Chain Composition

Model Composition in Domain-Specific Design Flows

Metaprogammable Tools, Environments

Domain-Specific Tools, Tool Chains

"Model Based Design", J. Sztipanovits
Syntactic Layer

- Modeling & Metamodelling
- Model Data Management
- Model Transformation
- Tool Integration
- Design-Space Exploration

Domain models
Interchange Formats

Abstract Syntax
Meta-models

Structural Semantics

Semantic Domain:
Set-Valued

\[ M_S = \{ r \in R \mid r \} \]

\[ M_C \]

"Model Based Design", J. Sztipanovits

ITR Review, Oct. 4, 2006
Metamodeling View of a 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)
Objective: Optimize the SW architecture by selecting a component model and by allocating functions to components. Platform: Heterogeneous Dataflow
Component Model
Tools: GME, GReAT, C Compiler, WCET Analyzer

Need for Metamodel Composition:

Objective: Optimize the SW architecture by selecting a component model and by allocating functions to components. Platform: Heterogeneous Dataflow
Component Model
Tools: GME, GReAT, C Compiler, WCET Analyzer

Functional blocks - SW Component Mapping

"Model Based Design", J. Sztipanovits
Solutions for Compositional Metamodelling

• **Goal:** Composing modeling languages (not models)

• **Metamodel composition methods in the Generic Modeling Environment (GME):**
  - *Class Merge*
  - *Metamodel Interfacing*
  - *Class Refinement*
  - *Template Instantiation*
  - *Metamodel Transformations*
Complex model transformations can be formally specified in the form of executable graph transformation rules.

G/T semantics is very powerful but the implementation needs to be tailored for efficiency.

GReAT is an open source, metamodel-based model transformation language supported by tools: modeling tool, rewriting engine, code generator and debugger. It is based on attributed/typed graph matching, multi-domain rewriting rules, and explicitly sequenced rewriting operators.

Highlights of GReAT extensions: shared spaces, sorting of match results, cross-products of matches, higher-order operators (groups).

Applications of GReAT:
- Simulink/Stateflow verifying code generator
- Several model transformation tools in embedded system toolchains
- Semantic anchoring of domain-specific modeling languages

"Model Based Design", J. Sztipanovits

(Karsai et al, 2005-2006)
Major Applications of Model Transformations

Model Transformations for Schedule Generation:

Implicit Platform Modeling for Analysis:

Explicit Platform Modeling Language:

(Karsai et al, 2005-2006)
Structural Semantics of Models and Metamodels

We followed a formal logic approach to structural semantics. A metamodel is mapped to a set of n-ary function symbols and constraints over an associated Herbrand Universe.

Primitives of DSP Domain

\[ \gamma = \{ \]
\[ \text{insig}(X): X \text{ is an input signal} \]
\[ \text{outsig}(X): X \text{ is an output signal} \]
\[ \text{prim}(X): X \text{ is a basic DSP operation} \]
\[ \text{iport}(X, Y): X \text{ has an input port } Y \]
\[ \text{oport}(X, Y): X \text{ has an output port } Y \]
\[ \text{inst}(X, Y): X \text{ is the DSP operation } Y \]
\[ \text{flow}(X_1, Y_1, X_2, Y_2): \text{Data goes from oport } Y_1 \text{ on } X_1 \text{ to iport } Y_2 \text{ on } X_2 \]

Some Constraints with NAF

1. Instances must use primitives that are defined:
   \[ \text{inst}(x, \text{prim}(y)) \land \neg \text{prim}(y) \Rightarrow \text{malform}(x), \]
2. Ports are placed on defined primitives:
   \[ \text{iport}(\text{prim}(x), y) \land \neg \text{prim}(x) \Rightarrow \text{malform}(y), \]
3. Dataflow connections must start on defined ports:
   \[ \text{flow}(x, \text{oport}(\text{prim}(y), z), u, w) \land \neg \text{oport}(\text{prim}(y), z) \Rightarrow \text{malform}(z). \]

These are the function symbols and some constraints for the example metamodel.

We use an inference procedure to prove well-formedness or malformedness. This inference mechanism is well-defined and tool independent.

We have constructed an automatic theorem prover that answers questions about structural semantics (see poster).

(Jackson, Sztipanovits 2006)
Intersection of System and Tool Composition Dimensions

- **Component Behavior**
  - Semantic Units and Semantic Anchoring
  - Compositional Semantics
  - Semantic Foundation

- **Interaction**
  - Metamodels, Metamodel Composition & Metaprogmmable Tool Chain Composition

- **Resource Modeling (Schedule)**
  - Model Composition in Domain-Specific Design Flows
  - Metaprogmmable Tools, Environments
  - Domain-Specific Tools, Tool Chains

"Model Based Design", J. Sztipanovits
Semantic Anchoring of DSML-s

- **Step 1**
  - Specify the DSML \(<A, C, M_C>\) by using MOF-based metamodels.

- **Step 2**
  - Select appropriate semantic units \(L = <A_i, C_i, M_{Ci}, S_i, M_{Si}>\) for the behavioral aspects of the DSML.

- **Step 3**
  - Specify the semantic anchoring \(M_A = A \rightarrow A_i\) by using UMT.

"Model-Based Design" J. Sztpanovits
(Chen and Sztipanovits, 2005-2006)
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 Unit Metamodel ($A_i$)
  - Domain Model ($C_i$)

- **Transformation Engine** generates

- **Domain Model (C)**

**Formal Framework for Semantic Units Specification**

- **Semantic Unit Spec.**
  - Abstract Data Model
  - Operational Semantics Spec.

- **Instance**

- **Data Model**

- **ASM Semantic Framework**

**AsmL Tools**

- **Model Checker**
- **Test Case Generator**
- **Model Simulator**

**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; 1/3

"Model Based Design", J. Sztipanovits
Example: HFSML -> FSM-SU; 2/3

```
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>
get

React (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
    WriteLine ("Execute transition: " + t.id)
    CS.active := false
    step
    dstState(t).active := true
    step
    if t in me.outputEventType then
      return Event(outputEventType(t))
    else
      return null
  else
    if Size(enabledTs) > 1 then
      error ("NON-DETERMINISM ERROR!"")
    else
      return null
```
Example: HFSML -> FSM-SU; 3/3
Intersection of System and Tool Composition Dimensions

Component Behavior

Semantic Units and Semantic Anchoring

Interaction

Compositional Semantics

Resource Modeling (Schedule)

Metamodels, Metamodel Composition & Metaprogrammable Tool Chain Composition

Semantic Foundation

Model Composition in Domain-Specific Design Flows

Metaprogrammable Tools, Environments

Domain-Specific Tools, Tool Chains

"Model Based Design", J. Sztipanovits
Component-based Analysis

- Incremental design
  - Associative composition

- Independent implementability
  - No global checks

(Matic and Henzinger, 2006)

"Model Based Design", J. Sztipanovits
Real-time Interface

Assumption
requests bounded by \( a \)
capacity larger than \( c \)

Guarantee
output latency bounded by \( d \)

Output rate function \( i^d(t) = i(t+d) \)

Interface predicate
\( \phi^i \equiv (r \geq c \land i \leq a) \)
\( \phi^0 \equiv o \leq i^d \)

(Matic and Henzinger, 2006)
Interface Algebra

- **Composition operation** $\parallel$

  $F_1 \parallel F_2$

- **Connection operation** $+$

  $F + S$

- **Refinement relation** .

  $F'$ refines $F$ if

  - $S_F \cap S_{F'}$
  - for each port valuation of $F$ there exists a valuation of $F'$:

    $\phi_F \Rightarrow \phi_{F'}$
    $\phi_F^0 \Rightarrow \phi_{F'}^0$

(Matic and Henzinger, 2006)
Algebra Properties

- Incremental design
  - \((F_k G)kH\) is defined
    \[ (F_k G)(kH) \text{ is def. } F_k(GkH) = (F_k G)kH \]
  - \((F_k G)\odot S\) is defined
    \[ (F \odot S)kG \text{ is def. } F_k(GkS) = (F \odot S)kG \]

- Independent refinement
  - \(F_k G\) is defined
    \[ F'_k G \text{ is def. } F_k G = F_k G \]
  - \(F \odot S\) is defined
    \[ F'_k G \text{ is def. } F_k G = F_k G \]
    \[ \forall j=1,\ldots,n: F'_j F_j = E(F'_1,\ldots,F'_n) = E(F_1,\ldots,F_n) \]