Towards the Compositional Specification of Semantics for Heterogeneous DSML-s

Janos Sztipanovits
ISIS, Vanderbilt University

October 26, 2006
Composition Domains

**Computation system composition domains:**
- SW functional components
- MoC abstractions
- Comm. abstractions

**Physical system composition domains:**
- Dynamics
- Power
- Comp. Arch.
- Comm. Arch.

**Physical “instantiation”**
- Detailed physical characteristics of the devices (phys. architecture, speed, bus structure, bandwidth, …)
- Interactions between code and physical behavior (speed, power dissipation)
- Lower layers of code interact with application code (scheduler, memory manager, middleware services, …)
- Interference across modules due to shared physical resources

**Deployment models**

Component-based design with cross-cutting constraints is a very hard problem
Design Aspects Are Not Orthogonal

Controller Dynamics
- Digital Controller
- D/A
- S/H
- Power Amp.
- Plant and Sensors
- A/D

Embedded Software
- App-1
- App-2
- App-3
- Lib-1
- Lib-2
- Lib-3
- Kernel/Services/Hardware

- control law
- tolerated error, stability
- sampling rate
- limit-cycle oscillation
- loop delay
- noise

- HW/SW architecture
- Data types selection
- Scheduling policy,...
- Numeric accuracy*
- Latency
- Jitter

*Quantization, saturation, truncation,...
Design Aspects in a Simplified ES Design Flow

Requirement Specification

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sensor image must be partitioned into chips to extract 1 potential regions of interest</td>
</tr>
<tr>
<td>2</td>
<td>Regions of interest must be matched with target images and classified within 30 ms of arrival</td>
</tr>
</tbody>
</table>

Control Design (dynamics)

Component Design

HW Arch. Design

Software Architecture

System Arch. Design

SW Deployment
Q1: What are the basic concepts for describing components? Several, structured in different design aspects and defined by metamodels.

Q2: What types of component interaction are supported? Several, structured in different design aspects and defined by metamodels and semantics.

Q3: What kind of resources can be modeled and are they first class citizens of the formalism? There are modeling aspects focusing on resources and there are component attributes in other aspects the establish the links.

Q4: How do you think the following models, styles and design principles are interrelated and can be combined:
- synchrony v.s. asynchrony: essential for heterogeneous, networked systems
- event-triggered/data-triggered/time triggered: all needed
- separation of concerns: this is the crucial point
Approach-1: Components Are Single Artifacts

Component: locus of activities interacting with other components via well defined interfaces

- Enriched interfaces
- Precise interaction models
- Heterogeneity

The component concept is extended with more elaborate interfaces to enable system composition along multiple design aspects.


Approach-2: Components Are Design Spaces

Component: is a design space defined by a set of interacting modeling aspects

Interaction among modeling aspects is defined by some transformation (e.g. $T = (R_{Y_1} \times R_{Y_2}) \rightarrow R_{Y_3}$) or by constraints over the design space (e.g. $D'(Y', C) = \{ r \in (R_{Y_1} \times R_{Y_2} \times R_{Y_3}) | r \models C \}$).


Design Space Specification and Composition Requires DSML Composition

Deployment Example: SW Architecture Model needs to be composed with System Architecture Model by allocating SW components to OSEK Tasks and Communication Channels.

**Tools:** GME, AIRES (schedulability), CANoe (Bus emulator)
Structural Semantics is Important

\[ L = \left\langle Y, R_Y, C, \left\langle \prod_{i \in J} l_i \right\rangle \right\rangle \]

\[ \models: R_Y \leftrightarrow R_Y. \]

Structural Interpretation:

\[
\begin{align*}
\models r = \{\text{true}\} & \iff (r \models C) \\
(r \neq C) & \iff (\models r = \{\text{false}\}).
\end{align*}
\]

Jackson, Sztipanovits
EMSOFT’06

- DSML Composition (metamodel composition) methods in the Generic Modeling Environment (GME):
  - Class Merge
  - Metamodel Interfacing
  - Class Refinement
  - Template Instantiation
  - Metamodel Transformations

- Analysis Tools:
  - OCL constraint checker
  - FORMULA (Jackson)
Structural Composition is Supported by Metamodelling Abstractions

Metamodel Interfacing

Class Refinement

Class Merge
There Are Many Interesting Applications
Composition of Behavioral Semantics

- Given a DSML
  \[ L = \langle Y, R_Y, C, (\downarrow \downarrow)_{i \in I} \rangle \]
  the transformational interpretation \( \downarrow \downarrow \) is a mapping:
  \[ \downarrow \downarrow^T : R_Y \leftrightarrow R_Y \]

- Behavioral semantics will be defined by specifying the transformation of the DSML models to models with operational semantics.

- Goal: Semantically robust design environment for composing DSML-s
Step 1
- Specify the DSML \(<A, C, M_c>\) by using MOF-based metamodels.

Step 2
- Select appropriate semantic units \(L = <A, C, M_{C_i}, S_i, M_{S_i}>\) 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

Metamodeling and Model Transformation Tools

- **GME Toolset**: Provide a MOF-based metamodeling and modeling environment.
- **GReAT Tool**: Build on GME for metamodel to metamodel transformation.

Formal Framework for Semantic Units Specification

- **Semantic Unit Spec.**: Abstract Data Model + Operational Semantics Spec.
- **Domain Model (C)**
- **Transformation Engine**
- **Instance**
- **XSLT**

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: Synchronous Data Flow

Abstract Data Model

```
structure Value
  case int as Integer
  case double as Double
  case bool as Boolean
//Data Token, it may contain a value or a null data
structure Token
  value as Value?
//Data Port, when exist is true, the port has an effective data token
class Port
  id as String
  var token as Token = Token(null)
  var exist as Boolean = false
//Data Channel connecting two data ports
class Channel
  id as String
  srcPort as Port
  dstPort as Port
//A Node is a basic unit is the Data Flow. It may be an action or a Guard
abstract class Node
  id as String
  abstract property inputPorts as Seq of Port get
  abstract property outputPorts as Seq of Port get
//The Run method takes tokens from its input ports, do actions and set output
//tokens in the output ports
abstract Fire()
```

Model Interpreter

```
Run (n as Node)
  require n in me.EnabledNodes()
  step
    n.Fire()
    step
    if exists p in n.inputPorts where p.exist then
      error ("After the firing of a node, all input tokens should be consumed by the node.")
    step
    if exists p in n.outputPorts where not p.exist then
      error ("After the firing of a node, each of its output port should have one output token.")
    step
    for all c in me.channels where c.srcPort.exist
      if c.dstPort.exist then
        error ("A input port receives more than one token.")
      else
        WriteLine ("Channel " + c.id + " is sending data tokens.")
        c.dstPort.token := c.srcPort.token
        c.dstPort.exist := true
        c.srcPort.exist := false
  //Return all nodes in the SDF that have all its required data tokens to fire.
  EnabledNodes () as Set of Node
  return {n | n in me.nodes where forall p in n.inputPorts where p.exist}
Initialize ()
  forall p in me.inputPorts where p.exist
  forall c in me.channels where p.id = c.srcPort.id
    c.dstPort.token := c.srcPort.token
  forall c in me.channels where p.id = c.srcPort.id
    c.dstPort.exist := false
    c.dstPort.exist := true
ClearPorts ()
  forall c in me.channels
    if c.srcPort.exist then
      c.srcPort.exist := false
    if c.dstPort.exist then
      c.dstPort.exist := false
```
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.

Data Model
- XSLT

ASM Semantic Framework
Example: HFSML => FSM-SU

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 enabled as Set of Transition = {t | t in outTransitions (CS) where e.eventType = triggerEventType (t)}
   step
   if Size (enabled) = 1 then
     choose t in enabled
     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 (enabled) > 1 then
     error ("NON-DETERMINISM ERROR!")
   else
     return null

HFSML Metamodel (A)

Model Trans. Rules (Ma)

 FSM Metamodel (Ai)

Domain Model (C)

Transformation Engine

Abstract Data Model

Operational Semantics Spec.

Data Model

ASM Semantic Framework

GME Toolset

GReAT Tool

FSM-SU Specification

Instance

XSLT
Example: HFSML $\Rightarrow$ FSM-SU

GME Toolset

HFSML Metamodel (A)

Domain Model (C)

GReAT Tool

Model Trans. Rules ($M_a$)

Transformation Engine

FSM Metamodel (A')

FSM Model (C')

XSLT

ASM Semantic Framework

FSM-SU Specification

Abstract Data Model +
Operational Semantics Spec.

Instance

Data Model

Initial State:
State("S1", true)

State("S2", false)

Transition("T1", true)

Event("E1")

Connections = [T1 $\rightarrow$ (S1, S2)]

Trigger Event = [T1 $\rightarrow$ E1]

Output Events = [T1 $\rightarrow$ E1]

Initial State = S1

return new StateMachine(States, Events, Connections, Trigger Events, Output Events, Initial State)
The semantics of a heterogeneous DSML is probably not captured by a single predefined semantic unit.

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.
### Compositional Specification of Semantics

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

- **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.
Road Ahead

- Continue in deepening the theory and expanding the scope of the compositional specification of semantics.
- Extend the semantic anchoring tools toward becoming a DSML Design Tool Suite.
- Further research on design space composition.