Component-based Modeling of Real-Time Systems

ARTIST2 Summer School
Nässlingen, 2005

Joseph Sifakis
VERIMAG
sifakis@imag.fr

In collaboration with
Gregor Goessler (INRIA)
Ananda Basu, Marius Bozga, Susanne Graf (Verimag)
Motivation

Modeling plays a central role in systems engineering

• Can profitably replace experimentation on actual systems
• Can provide a basis for rigorous system development and implementation (model-based approaches).

Modeling real-time systems

• Raises hard problems about concepts, languages and their semantics e.g. What is an architecture? What is a scheduler? How synchronous and asynchronous systems are related?

• Requires a deep understanding of basic system design issues such as development methodologies (combination of techniques and tools, refinement) and architecture design principles.
Model-based Development

Move from physical prototypes to virtual prototypes (models) with obvious advantages: minimize costs, flexibility, genericity, formal validation is a possibility.

Modeling and validation environments for complex real-time systems

- Libraries of Components
  - ex. HW, SW, Models of continuous dynamic systems
- Languages and tools for assembling components

Synthesize embedded software from domain-specific models
ex. Matlab, SystemC, UML, SDL.
Objectives

Provide a rigorous and general framework for modeling,

- Based on a general concept of architecture as a means to organize computation (behavior, interaction, control)
- Encompassing heterogeneous description, specific styles and paradigms, e.g.
  - synchronous and asynchronous execution
  - heterogeneous interaction (strong, weak, event-driven, state-driven)
  - architecture styles e.g. client-server, blackboard architecture
- Equipped with rules for correctness-by-construction wrt. generic properties such as deadlock-freedom, liveness, safety.
- Providing a basis for automated support for component integration and generation of glue code meeting given requirements
Overview

- Modeling real-time systems
- Component-based construction
- Interaction Models
- Property enforcement by controllers
- Priority systems
- Implementation of the BIP framework
- Discussion
Modeling real-time systems - our approach

Thesis:
A Timed Model of a RT system can be obtained by “composing” its application SW with timing constraints induced by both its execution and its external environment.
Modeling real-time systems - our approach

<table>
<thead>
<tr>
<th></th>
<th>Application SW</th>
<th>Timed model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DESCRIPTION</strong></td>
<td>Reactive machine (untimed)</td>
<td>Reactive machine + External Environment + Execution Platform</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>Reference to physical (external) time</td>
<td>Quantitative (internal) time Consistency pbs - timelocks</td>
</tr>
<tr>
<td><strong>TRIGGERING</strong></td>
<td>Timeouts to control waiting times</td>
<td>Timing constraints on interactions</td>
</tr>
<tr>
<td><strong>ACTIONS</strong></td>
<td>No assumption about Execution Times Platform-independence</td>
<td>Assumptions about Execution Times Platform-dependence</td>
</tr>
</tbody>
</table>
Modeling real-time systems - our approach

- Application SW
- Platform Timed Model
- Environment Timed Model
- User Requirements

Composition/Synthesis

- Code Generation
- System Timed Model

System Analysis
- Implementation
- Diagnostics

Component-based modeling
Heterogeneity - Abstraction Levels

Model (requirements)

Application Software

System

Execution Platform
Heterogeneity - Abstraction Levels

Model
(requirements)

Application
Software

System

Integration

Execution Platform1

Execution Platform2

Joseph Sifakis  ARTIST2 Summer School  Sept. 29-Oct. 2  2005
Heterogeneity - from application SW to implementation

Application SW

Matlab/Simulink
Lustre
Esterel
ADA
SDL
RT- Java
UML
C
C++

Implementation

DSP
μcontroller

TTA
CAN
RTOS
CORBA
OSEK
Heterogeneity - from application SW to implementation

Functional properties - logical abstract time
High level structuring constructs and primitives
Simplifying synchrony assumptions wrt environment

Non functional properties, involving time and quantities
Task coordination, scheduling, resource management,
Execution times, interaction delays, latency
Heterogeneity - synchronous vs. asynchronous execution

**Synchronous**
- Lustre, Esterel
- Statecharts
- Non interruptible execution steps
- Usually, single task, single processor
- «Everybody gets something »

**Asynchronous**
- ADA, SDL
- Event triggered
- Multi-tasking - RTOS
- Usually, static Priorities
- «Winner takes all »

Application SW
Component-based approaches

Implementation
Heterogeneity - example

A: Atomic interaction

B: Blocking interaction

Asynchronous Computation

Synchronous Computation

Java
UML

SDL
UML

Lotos
CSP

Lotus
CSP

A B

nonA B

A nonB

nonA nonB

Matlab/Simulink
VHDL/SystemC
Statecharts
Overview

• Modeling real-time systems

• Component-based construction

• Interaction Models

• Property enforcement by controllers

• Priority systems

• Implementation of the BIP framework

• Discussion
Component-based construction - components

Build systems by composition of components

Components are building blocks composed of behavior and interface

Behavior is a transition system

Interface hides irrelevant internal behavior and provides some adequate abstraction for composition and re-use, e.g. set of action names (ports) and associated variables
Component-based construction – formal framework

Pb: Build a component $C$ satisfying a given property $P$, from

- $C_0$: a set of atomic components
- $\mathcal{GL} = \{gl_1, \ldots, gl_i, \ldots\}$ a set of glue operators on components

- Components are terms of an algebra of terms $(\mathcal{E}, \cong)$ generated from $C_0$ by using operators from $\mathcal{GL}$
- $\cong$ is a congruence compatible with operational semantics
Component-based construction – formal framework

Glue operators transform sets of components into components

Glue operators
• model mechanisms used for communication and control such as protocols, controllers, buses
• restrict the behavior of their arguments, that is $\text{gl}(C_1, C_2, \ldots, C_n) | A_1 \text{ refines } C_1$
Component-based construction - requirements

Examples of existing frameworks:
- Sequential functions with logical operators and delay operators for building circuits
- Process algebras
- Distributed algorithms define generic $gl$ for a given property $P$ e.g. token ring, clock synchronization …

Pb: Find a set of glue operators meeting the following requirements:
- Expressiveness (discussed later)
- Incremental description
- Correctness-by-construction
Component-based construction – incremental description

1. Decomposition of $gl$

\[ gl \cong C_1 \oplus C_2 \oplus \ldots \oplus C_n \]

2. Flattening of terms

\[ gl_1 \oplus gl_2 \cong gl \]

Flattening can be achieved by introducing an idempotent operation $\oplus$ such that $(GL, \oplus)$ is a commutative monoid and

\[ gl(gl'(C_1, C_2, \ldots, C_n)) \cong gl \oplus gl'(C_1, C_2, \ldots, C_n) \]
Component-based construction - Correctness by construction: compositionality

Build correct systems from correct components

$c_i \text{ sat } P_i$ implies $\forall gl \exists \tilde{gl}$

$\begin{cases} c_1 \\ \vdots \\ c_n \end{cases}$ sat $\tilde{gl}(P_1, \ldots, P_n)$

We need compositionality results about preservation of progress properties such as deadlock-freedom and liveness.
Component-based construction - Correctness by construction: composability

Make the new without breaking the old

\[
\text{sat } P \quad \text{and} \quad \text{sat } P' \\
\text{implies} \quad \text{sat } P \land P'
\]

Property stability phenomena are poorly understood
- feature interaction
- non composability of scheduling algorithms
Component-based construction - compositionality vs. composability
Overview

- Modeling real-time systems
- Component-based construction
- Interaction Models
- Property enforcement by controllers
- Priority systems
- Implementation of the BIP framework
- Discussion
Component-based modeling – The BIP framework

Layered component model

Scheduler: dynamic priority rules

Interaction Model: Connectors + Interactions

Composition (incremental description)
**Interaction models**

**Connectors** are maximal sets of compatible actions.

**Interactions** are subsets of connectors; they are defined by using typing \((\text{complete} \ ▽ \ , \ \text{incomplete} \ \bullet)\): either they are maximal or they contain some complete interaction.

Interactions:
- \{\text{tick1}, \text{tick2}, \text{tick3}\}, \{\text{out1}\}, \{\text{out1}, \text{in2}\}, \{\text{out1}, \text{in3}\}, \{\text{out1}, \text{in2}, \text{in3}\}
Interaction models - examples

- **cl1, cl2**
  - **CN**: {cl1, cl2}
  - **MI**: ∅

- **out, in**
  - **CN**: {out, in}
  - **MI**: {out}

- **in1, out, in2**
  - **CN**: {in1, out, in}
  - **MI**: {out}
Interaction models - definition

Given a set of atomic components $\mathbf{K}$ with disjoint action vocabularies $A_i$ for $i \in \mathbf{K}$,

- A connector $\gamma$ is a non empty subset of $\bigcup_{i \in \mathbf{K}} A_i$ such that $|\gamma \cap A_i| \leq 1$
- **Interactions** are non empty subsets of connectors
- An interaction model $\text{im}$ is a pair $\text{im} = (\Gamma, \Delta)$ such that
  - $\Gamma$ is a set of non comparable connectors
  - $\Delta$ is a set of **minimal complete** interactions with $\forall \delta \in \Delta \exists \gamma \in \Gamma. \delta \subseteq \gamma$.

The interactions of $\text{im} = \Gamma \cup \{\alpha | \exists \delta \in \Delta. \delta \subseteq \alpha \subseteq \gamma\}$
Interaction models – operational semantics

CN: {put,get},{prod},{cons}
MI: {prod},{cons}

Operational Semantics
{put, get}
put prod get
putget
×
×

Semantics
Operational

Joseph Sifakis
ARTIST2 Summer School
Sept. 29-Oct. 2
2005
Interaction models - composition

CN[P,C]: \{put, get\}
MI[P,C]: \emptyset

CN[P]: \{put\}, \{prod\}
MI[P]: \{prod\}

CN[C]: \{get\}, \{cons\}
MI[C]: \{cons\}

prod \rightarrow put

get \rightarrow cons

CN: \{put, get\}, \{prod\}, \{cons\}
MI: \{prod\}, \{cons\}
Interaction models - composition

IM[K_1,K_2]:
CN[K_1,K_2] : \{a_1, a_2, a_3, a_4\}, \{a_{11}, a_{12}\}
MI[K_1,K_2] : \{a_1,a_2,a_3,a_4\}, \{a_{11}\}

IM[K_1]:
CN[K_1] : \{a_1, a_2\}, \{a_5, a_9\}, \{a_6, a_9\}
MI[K_1] : a_5, a_6, a_{11}

IM[K_2]:
CN[K_2] : \{a_3, a_4\}, \{a_7, a_{10}\}, \{a_8, a_{10}\}
MI[K_2] : a_{10}
Interaction models – composition (2)

IM[K₁,K₂]:
CN[K₁,K₂] : {a₁, a₂, a₃, a₄}, {a₁₁, a₁₂}
MI[K₁,K₂] : {a₁,a₂,a₃,a₄}, {a₁₁}

IM[K₁]:
CN[K₁] : {a₁, a₂}, {a₅, a₉},{a₆, a₉}
MI[K₁] : a₅, a₆, a₁₁

IM[K₂]:
CN[K₂] : {a₃, a₄}, {a₇, a₁₀}, {a₈, a₁₀}
MI[K₂] : a₁₀

K₁

K₂

CN[K₁ ∪ K₂] = max CN[K₁] ∪ CN[K₂] ∪ CN[K₁,K₂]
MI[K₁ ∪ K₂]  = min MI[K₁] ∪ MI[K₂] ∪ MI[K₁,K₂] }
Interaction models – results [Goessler Sifakis]

Incremental commutative composition encompassing blocking and non-blocking interaction
Interaction models - mod8 counter

a0, a1

b0, b1

c0, c1

a1, b0

b1, c0

a1, c0

b1, c1

a1, b1, c0

a1, b1, c1

a0, a1, b0

b1, c0, c1

Joseph Sifakis  ARTIST2 Summer School  Sept. 29-Oct. 2  2005
Interaction models-mod8 counter(2)
Interaction models - checking for deadlock-freedom

For a given system (set of components + interaction model), its **dependency graph** is a bipartite labeled graph with

Nodes \( N = \text{Set of components} \cup \text{Set of minimal interactions} \)

Edges \( E \)

- \((\alpha, a, k) \in E \) if \( \alpha \) is an interaction, \( a \in \alpha \) is an incomplete action of \( k \)
- \((k1, a1, \alpha) \in E \) if \( a1 \in \alpha \) is an action of \( k1 \)

**Blocking condition for an incomplete action** \( a \):

\[ Bl(a) = \text{en}(a) \land \neg (\text{en}(a1) \land \text{en}(a2) \land \text{en}(a3) ) \]
Interaction models - checking for deadlock-freedom (2)

Theorem 1: A system is deadlock-free if its atomic components have no deadlocks and its dependency graph has a backward closed subgraph such that for all its circuits $\omega$

$$Bl(\omega) = \bigwedge_{a \in \omega} Inc(\omega) \land Bl(a) = false$$

where $Inc(\omega) = \bigwedge_{k \in \omega} Inc(k)$ with $Inc(k)$ the set of the states of $k$ from which only incomplete actions can be executed.
Interaction models - checking for deadlock-freedom: example

producer  put

get₁  consumer₁

CN: \{put, get₁, get₂\}
MI: \{put, get₁\}, \{put, get₂\}

get₂  consumer₂
Interaction models - checking for deadlock-freedom: example

\[ \omega_1 = (\text{producer, } n_1, \text{consumer}_2, n_2) \quad \text{Bl}(\omega_1) = \text{false} \]

\[ \omega_2 = (\text{producer, } n_2, \text{consumer}_1, n_1) \quad \text{Bl}(\omega_2) = \text{false} \]

\[ \omega_3 = (\text{consumer}_1, n_1, \text{consumer}_2, n_2,) \]

\[ \text{Bl}(\omega_3) = \text{Inc}(\omega_3) \land \text{en}(\text{get}_1) \land \neg (\text{en}(\text{get}_2) \land \text{en}(\text{put})) \land \text{en}(\text{get}_2) \land \neg (\text{en}(\text{get}_1) \land \text{en}(\text{put})) \]

= \text{Inc}(\omega_3) \land \text{en}(\text{get}_1) \land \text{en}(\text{get}_2) \land \neg \text{en}(\text{put})

Deadlock-freedom if \( \text{Inc(producer)} \land \neg \text{en(put)} = \text{false} \)
Interaction models - checking for individual deadlock-freedom

Definition: A component of a system is individually deadlock-free if it can always perform some action.

Theorem 2: Sufficient condition for individual deadlock-freedom of a component $k$

- $k$ belongs to a backward closed subgraph of a dependency graph satisfying conditions of Theorem 1;
- In any circuit of this subgraph, all its components are controllable with respect to their outputs i.e. it is always possible by executing complete interactions, to reach states enabling all the output actions of the component;
- All the $n$-ary interactions for $n>2$ are strong synchronizations.
Overview

• Modeling real-time systems

• Component-based construction

• Interaction Models

• Property enforcement by controllers

• Priority systems

• Implementation of the BIP framework

• Discussion
Property enforcement - the role of controllers

A controller restricts access to controllable (critical) interactions of $S$ so as to respect a constraint (state predicate) $K_0$.
Property enforcement - the role of controllers

Controller for
\[ K_0 = (w \leq 1) \land \neg((1 \leq r) \land (1 \leq w)) = (w=0) \lor (r=0) \land (w\leq1) \]

where \( w = \#\text{writers}, \ r = \#\text{readers} \)
Property enforcement - control invariants

A control invariant $K \Rightarrow K_0$

- Control invariants are preserved by uncontrollable actions
- It is possible to maintain the system in $K$ by executing controllable actions
Property enforcement – the notion of restriction

The *restriction* of $S$ by a constraint $K$ is a system $S/K$

Note that in $S/K$, $K$ holds right before and right after the execution of any *controllable* action.

*If $K$ is a control invariant of $S$ then $S/K$ is the controlled system.*
Property enforcement – the notion of restriction

Controller for
\[ K_0 = (w \leq 1) \land \neg((1 \leq r) \land (1 \leq w)) = (w=0) \lor (r=0) \land (w \leq 1) \]

where \( w \) = #writers, \( r \) = #readers
Property enforcement – relating deadlock-free restrictions and priorities [Goessler Sifakis 2003]

If $K$ is a constraint characterizing a set of deadlock-free states of $S$ then there exists a set of priority rules $pr$ such that $pr(S)$ preserves $K$

For any control invariant $K$ of $S$ there exists a set of dynamic priority rules $pr$ such that the scheduled system $S/K = pr(S)$

Any feasible scheduling policy induces a restriction that can be described by priorities
Overview

- Modeling real-time systems
- Component-based construction
- Interaction Models
- Property enforcement by controllers
- Priority systems
- Implementation of the BIP framework
- Discussion
Priority Systems

Priority system = Behavior + A set of dynamic priority rules

Priority rule

<table>
<thead>
<tr>
<th>Priority rule</th>
<th>Restricted guard $g_1'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true → $a_1 \prec a_2$</td>
<td>$g_1' = g_1 \land \neg g_2$</td>
</tr>
<tr>
<td>$C \rightarrow a_1 \prec a_2$</td>
<td>$g_1' = g_1 \land \neg(C \land g_2)$</td>
</tr>
</tbody>
</table>
Priority Systems

A priority order is a strict partial order \( \langle \subseteq A^c \times A \rangle \)

A set of priority rules, \( pr = \{ C_i \rightarrow \langle i \rangle \}_i \) where \( \{C_i\}_i \) is a set of disjoint state predicates

\[
pr = \{ C_i \rightarrow \langle i \rangle \}_i
\]

\[
g'_k = g_k \land \bigwedge C \rightarrow \langle \in pr \} \quad C \Rightarrow \bigwedge \land a_k \langle a_i \rightarrow g_i \rangle
\]

Joseph Sifakis
ARTIST2 Summer School
Sept. 29-Oct. 2 2005

50
Priority Systems - FIFO policy

\[ t_1 \leq t_2 \rightarrow b_1 \prec b_2 \]
\[ t_2 \leq t_1 \rightarrow b_2 \prec b_1 \]
Priority Systems - EDF policy

\[
D_1 \cdot t_1 \leq D_2 \cdot t_2 \rightarrow b_2 \langle b_1 \\
D_2 \cdot t_2 \leq D_1 \cdot t_1 \rightarrow b_1 \langle b_2
\]
Priority Systems - Composition of priorities

\[ pr2 \]
\[ pr1 \]
\[ \cdot \cdot \cdot \]
\[ \neq \]

\[ pr1 \]
\[ pr2 \]
\[ \cdot \cdot \cdot \]
Priority Systems - Composition of priorities

We take:

\[
\begin{align*}
pr1 & \oplus pr2 \\
\cup & \ \\
\end{align*}
\]

\[=\]

\[pr1 \oplus pr2\]

\[pr1 \oplus pr2\] is the least priority containing \[pr1 \cup pr2\]

Results:

- The operation \(\oplus\) is partial, associative and commutative
- \(pr1(pr2(B)) \neq pr1(pr2(B))\)
- \(pr1 \oplus pr2(B) \text{ refines } pr1 \cup pr2(B) \text{ refines } pr1(pr2(B))\)
- Priorities preserve deadlock-freedom
Priority Systems: mutual exclusion + FIFO

\[
t_1 \leq t_2 \rightarrow b_1 \wedge b_2 \quad t_2 \leq t_1 \rightarrow b_2 \wedge b_1
\]

\[
\text{true} \rightarrow b_1 \wedge e_2 \quad \text{true} \rightarrow b_2 \wedge e_1
\]
Priority systems – mutual exclusion

Risk of deadlock: The composition is not a priority order!
The BIP framework - fixed priority preemptive scheduling (1)

\[ b_i \langle b_j, \ r_i \langle r_j, \ r_i \langle b_j, \ b_i \langle r_j \ (\text{access to the resource – priority preserved by composition}) \]
\[ \{b_i, p_j \} \langle f_j, \{r_i, p_j \} \langle f_j, \ n \geq l > j \geq 1 \ (\text{non pre-emption by lower pty tasks}) \]

CN:  \{b_i, p_j \} \{r_i, p_j \} \text{ for } n \geq i, j \geq 1  
MI:  a_i, f_i, b_i \text{ for } n \geq i \geq 1
The BIP framework - fixed priority preemptive scheduling
The BIP framework - fixed priority preemptive scheduling (2)

\[ b_i \prec b_j, \quad r_i \prec r_j, \quad r_i \prec b_j, \quad b_i \prec r_j \] (access to the resource – pty inherited by composition)

\[ p_i \prec f_j, \quad \text{if } w_i \text{ or } e'_i \quad n \geq 1, j \geq 1 \] (non pre-emption by lower pty tasks)

\[ \{b_i, r_i\} \prec \{f_j, p_j\} \quad n \geq 1, j \geq 1 \] (Mutual exclusion)
The BIP framework - traffic light for tramway crossing

![Diagram of traffic light for tramway crossing]
The BIP framework – run to completion

i1 {o1,i2} {o2,i3} o3
CN: {o1,i2}, {o2,i3} MI: i1,o3

i1 o1
i2 o2
i3 o3
Overview

• Modeling real-time systems

• Component-based construction

• Interaction Models

• Property enforcement by controllers

• Priority systems

• Implementation of the BIP framework

• Discussion
The execution platform

Interaction model

Priority

Execution kernel

Platform
The execution platform

Component Meta-model

Interaction Meta-model

Dynamic priorities Meta-model

Description in L

C \Rightarrow a < b

L

Execution kernel
Implementation - atomic component: abstract syntax

Component: C
Ports: p1, p2, ...
Data: x, y, z, ....
Access: (p1, {x, y, z}), (p2, {x, u, v}),

Behavior:
    state s1
        on p1 provided g1 do f1 to state s1'
    .................  ......  
    on pn provided gn do fn to state sn'

    state s2
        on .....  
    ........

    state sn
        on .. ..
run() {
    Port* p;
    int state = 1;
    while(true) {
        switch(state) {
            case 1: p = sync(a, g_a, d, g_d);
                if (p == a)
                    f_a; state = 2;
                else
                    f_d; state = 3;
                break;
            case 2: p = sync(b, g_b, e, g_e);
                ... 
            case 3: ... 
        }
    }
}
Implementation - connectors and priorities: abstract syntax

**Connector:** BUS={p, p', ... , }
**complete()**

**Behavior:**
- on $\alpha_1$ provided $g_{\alpha_1}$ do $f_{\alpha_1}$
- on $\alpha_2$ provided $g_{\alpha_2}$ do $f_{\alpha_2}$

**Priorities:** PR
- if C1 then {$(\alpha_1, \alpha_2), (\alpha_3, \alpha_4) , ...$}
- if C2 then {$(\alpha,...), (\alpha,...) , ...$}
- if Cn then {$(\alpha,...), (\alpha,...) , ...$}
The BIP framework - modulo-8 counter: atomic component

- counter

in: X
- flip
  \( g_{\text{flip}}: X = 1 \)
  \( f_{\text{flip}}: Y := 0 \)

out: Y
- flip
  \( g_{\text{flip}}: X = 1 \)
  \( f_{\text{flip}}: Y := 1 \)

Zero
- tick

One
- tick
- flip
  \( g_{\text{flip}}: X = 1 \)
  \( f_{\text{flip}}: Y := 0 \)

Zero'
- tick

One'
- tick
- flip
  \( g_{\text{flip}}: X = 1 \)
  \( f_{\text{flip}}: Y := 1 \)
The BIP framework - modulo-8 counter: architecture
The BIP framework - modulo-8 counter: the model

| tick \langle flip_0 , tick \langle flip_1 , tick \langle flip_2 |
|---|---|---|
| CN: tick={tick_0, tick_1, tick_2} |
| CI: ∅ |
| Transfer : X_1 := Y_0; X_2 := Y_1 ∧ Y_0 |

- CN: tick_0, flip_0
  - CI: flip_0
- CN: tick_1, flip_1
  - CI: flip_1
- CN: tick_2, flip_2
  - CI: flip_2
The BIP framework - Synchronous reactive systems

Data flow partial order

Connector: {s1, s2, s3, s4}
Transfer:
\[ x_2, x_3 := y_1; \]
\[ x_4 := y_2, y_3 \]
Priority: lowest
The BIP framework – implementation: the kernel

Kernel.run

{p₃,q₃,r₁} {q₂,r₂}

{p₁,q₁} {p₂,q₂} {p₃,q₃}

p₁ p₂ p₃
q₁ q₂ q₃
r₁ r₂

P.run
Q.run
R.run
The BIP framework – implementation: the kernel

- init
  - Launch atom’s threads
  - Wait all atoms

- loop
  - Notify involved atoms
  - Wait all atoms

- execute
  - Compute legal interactions
  - Execute chosen interaction transfer

- choose
  - Choose among maximal

- ready
  - Filter w.r.t. priorities

- stable
  - Filter w.r.t. priorities
Overview

• Modeling real-time systems

• Component-based construction

• Interaction Models

• Property enforcement by controllers

• Priority systems

• Implementation of the BIP framework

• Discussion
Discussion - the BIP framework

• Framework for component-based modeling encompassing heterogeneity and relying on a **minimal set of constructs and principles** e.g. interaction models + dynamic priorities

• Clear separation between behavior and structure.
  - Structure is a first class entity
  - Correctness-by-construction techniques for deadlock-freedom and liveness, based on sufficient conditions on structure (mainly)

• Applications at Verimag
  - IF toolset allows layered description of timed systems,
  - Methodology and tool support for generating scheduled code for real-time applications (work by S. Yovine et al.)
Discussion - Interaction models

• The distinction interaction model / behavior is crucial or the model construction methodology. Layered description => separation of concerns => associativity

• Different from other approaches e.g. process calculi, which combine behavior composition operators and restriction/hiding operators at the same level.

\[(P1||P2)\parallel a \parallel P3)\parallel a' \quad \overset{\parallel a \oplus a'}{\rightarrow} \quad P1||P2||P3\]

• Framework encompassing strict and non strict synchronization
Discussion – work dealing with components

- Architecture Description Languages focusing on non-functional aspects or SW Design Description Languages
- Modeling languages: Statecharts, UML, Simulink/Stateflow, Synchronous languages, SystemC, Metropolis, Ptolemy
- Coordination languages (extensions of programming languages): Linda, Javaspaces, TSpaces, Concurrent Fortran, ...
- Middleware standards: IDL, Corba, Javabeans, .NET
- Software development environments: PCTE, SWbuses, Softbench, Eclipse
- Process algebras and automata: Pi-Calculus, ORC, automata-based approaches
Discussion – related approaches

Vanderbilt’s Approach

- Semantic Unit
- Meta-model
- Composition Operators
- Behavior
- Operational Semantics
- ASML
- .net

Metropolis

- Semantic Domains
- Quantity Managers
- Media
- Behavior
- Operational Semantics
- Platform

PTOLEMY

- MoC (Model of Computation)
- Directors
- Connectors
- Behavior
- Operational Semantics
- Platform

Josep Sifakis

ARTIST2 Summer School

Sept. 29-Oct. 2 2005

78
Discussion – separation of concerns

Design Space: a system is defined as a point of the 3-dimensional space
Separation of concerns: any combination of coordinates defines a system
Discussion – future work

Develop a rigorous and general basis for architecture modeling and implementation:

• Study the concept of architecture as a means to organize computation (components, interaction, control)
• Define a meta-model for real-time architectures, encompassing specific styles, paradigms, e.g.
  - Modeling synchronous reactive systems
  - Event driven vs. state driven interaction (distinction event ports / state ports)
  - Hierarchical modeling
  - Timed systems
  - Distributed real-time systems, GALS
  - Architecture styles e.g. client-server, blackboard architecture

• Provide automated support for component integration and generation of glue code meeting given requirements
Discussion – future work: expressiveness

Study Component Algebras \( CA = (B, GL, \oplus, \cong) \)
- \((GL, \oplus)\) is a monoid and \(\oplus\) is idempotent
- \(\cong\) is a congruence compatible with operational semantics

- Study classes of glue operators
- Focus on properties relating \(\oplus\) to \(\cong\)

Study notions of **expressiveness** characterizing structure

Given \( CA_i = (B, GL_i, \oplus_i, \cong_i) \), \( i=1,2 \),

\( CA_1 \) is more expressive than \( CA_2 \) if \( \forall P \)
\[ \exists \, gl_2 \in GL_2 \, gl_2(B_1, \ldots, B_n) \text{ sat } P \Rightarrow \exists \, gl_1 \in GL_1 \, gl_1(B_1, \ldots, B_n) \text{ sat } P \]
Discussion – future work: expressiveness(2)

Example: For given $B$, $IM$ and $PR$ which coordination problems can be solved?

Notion of expressiveness different from existing ones which
- Either completely ignore structure
- or use operators where separation between structure and behavior seems problematic e.g. hiding, restriction