Modeling Component-based Systems in BIP

ETR 2007

Nantes, Sept. 2007

Joseph Sifakis
VERI MAG
sifakis@imag.fr

In collaboration with G. Goessler (INRIA),
A. Basu, M. Bozga, H. Nguyen (Verimag)
Component-based construction – Objectives

Develop a rigorous and general basis for real-time system design and implementation

• Concept of component and associated composition operators for incremental description and correctness by construction

• Concept for real-time architecture encompassing heterogeneity, paradigms and styles of computation e.g.
  ▪ Synchronous vs. asynchronous execution
  ▪ Event driven vs. time driven computation
  ▪ Distributed vs. centralized execution

• Automated support for component integration and generation of glue code meeting given requirements
Approches involving components

• Theory such as process algebras and automata

• SW Component frameworks, such as
  - Coordination languages extensions of programming languages: Linda, Javaspaces, TSpaces, Concurrent Fortran, NesC, BPEL
  - Middleware e.g. Corba, Javabeans, .NET
  - Software development environments: PCTE, SWbus, Softbench, Eclipse

• System modeling languages: SystemC, Statecharts, UML, Simulink/Stateflow, Metropolis, Ptolemy

• Architecture Description Languages focusing on non-functional aspects e.g. AADL

Lack of

• frameworks treating interactions and system architecture as first class entities that can be composed and analyzed (usually, interaction by method call)
• rigorous models for behavior and in particular aspects related to time and resources.
Sources of heterogeneity [Henzinger&Sifakis FM06]

Heterogeneity of interaction
- Atomic or non atomic
- Rendezvous or Broadcast
- Binary or n-ary

Heterogeneity of execution
- Synchronous execution
- Asynchronous execution
- Combinations of them

Heterogeneity of abstraction e.g. granularity of execution
Sources of Heterogeneity - Example

A: Atomic interaction
R: Rendezvous
B: Broadcast

Asynchronous Computation

Lotos
CSP

Java
UML

SDL
UML

Synchronous Computation

Matlab/Simulink
VHDL, Statecharts,
Synchronous languages
Overview

• About component-based construction

• Basic Concepts

• Interaction modeling

• Priority modeling

• Implementation

• Modeling systems in BIP

• Discussion
Component-based Construction: The Problem

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

Glue operators

- model mechanisms used for communication and control such as protocols, schedulers, buses
- restrict the behavior of their arguments, that is the projection of the behavior of $gl(C_1,C_2,\ldots,C_n)$ on actions of $C_1$ is contained in the behavior of $C_1$
Component-based Construction: Formal Framework

Operational Semantics: the meaning of a compound component is an atomic component

Algebraic framework:
• Components are terms of an algebra of terms \((C, \cong)\) generated from \(C_0\) by using operators from \(GL\)
• \(\cong\) is a congruence compatible with operational semantics
Component-based Construction: Constructivity – Incremental description

1. Decomposition

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

$\text{gl} (\text{gl}'(C_1, C_2, \ldots, C_n)) \cong \text{gl} \oplus \text{gl}'(C_1, C_2, \ldots, C_n)$
Component-based Construction: Constructivity – Compositionality

Build correct systems from correct components: rules for proving global properties from properties of individual components

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

\[ c_1 \ldots c_n \]

\[ \text{sat } \tilde{gl}(P_1, \ldots, P_n) \]

We need compositionality results about preservation of progress properties such as deadlock-freedom and liveness and extra-functional properties.
Component-based Construction: Constructivity – Composability

Make the new without breaking the old: Rules guaranteeing non interference of solutions

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

Property stability phenomena are poorly understood. We need composability results e.g. feature interaction in middleware, composability of scheduling algorithms, theory for reconfigurable systems.
Overview

• About component-based construction

• Basic Concepts

• Interaction modeling

• Priority modeling

• Implementation

• Modeling systems in BIP

• Discussion
BIP: Basic Concepts

Layered component model

Priorities (Memoryless Controller)

Interactions (Structured Connectors)

Composition (incremental description)
BIP: Basic Concepts

Priorities: $\emptyset$

Interactions: sr1r2r3

Rendezvous
BIP: Basic Concepts

Priorities: $x \prec xy$ for $x,y=\text{interactions}$

Interactions: $s, sr1, sr2, sr3, sr1r2, sr2r3, sr1r3, sr1r2r3$

- **Sender**
- **Receiver1**
- **Receiver2**
- **Receiver3**

Broadcast
BIP: Basic Concepts

Priorities: $x \prec xy$ for $x,y=\text{interactions}$

Interactions: $s, sr_1r_2r_3$

- $s$
- $r_1$
- $r_2$
- $r_3$

Sender

Receiver1

Receiver2

Receiver3

Atomic Broadcast
BIP: Basic Concepts

Priorities: $x < xy$ for $x, y =$ interactions

Interactions: $s, sr1, sr1r2, sr1r2r3$

Causal Chain
BIP: Basic Concepts - Semantics for Interactions

Given

• a set of atomic components \( \{B_i\}_{i=1..n} \) where \( B_i = (Q_i, P_i, \rightarrow_i) \)

• a set of interactions \( \gamma \subseteq \mathcal{P} \) where \( P = \bigcup_{i=1..n} P_i \)

Define \( \gamma(B_1,.., B_n)=(Q, P, \rightarrow_{\gamma}) \) where \( Q = \times_{i=1..n} Q_i \) by the rule

\[
\begin{align*}
\text{Given: } & \quad a = \{p_i\}_{i \in I} \in \gamma \land \forall i \in I \ (p_i \in P_i \land q_i - p_i \rightarrow_i q'_i) \\
\text{Then: } & \quad (q_1,.., q_n) - a \rightarrow_{\gamma} (q'_1,.., q'_n) \text{ where } q'_i = q_i \text{ if } i \notin I
\end{align*}
\]
An atomic component has

- A set of ports $P$
- A set of control locations $S$
- A set of variables $V$
- A set of transitions of the form
  - $p$ is a port
  - $g_p$ is a guard, boolean expression on $V$
  - $f_p$ is a function on $V$ (block of code)
**BIP: Basic Concepts - Behavior Modeling**

- **p**: a port through which interaction is sought
- **gp**: a pre-condition for interaction through p
- **fp**: a computation (local state transformation)

**Semantics**: interaction followed by computation
- A transition is enabled if gp is true and some interaction involving p is possible
- The execution of the enabled transition involves the execution of an interaction involving p followed by the execution of fp
Overview

• About component-based construction

• Basic Concepts

• Interaction modeling

• Priority modeling

• Implementation

• Modeling systems in BIP

• Discussion
Interaction Modeling: Connectors

• A **connector** is a set of ports which can be involved in an interaction
• Port attributes (**trigger** ▼, **synchron** ○) are used to model rendezvous and broadcast.
• An **interaction** of a connector is a set of ports such that: either it contains some trigger or it is maximal.

Interactions:

```
tick1 tick2 tick3, out1, out1 in2, out1 in3, out1 in2 in3
```
Interaction Modeling: Connectors
**Interaction Modeling: Hierarchical Connectors**

- **Rendezvous:** $abc$

- **Broadcast:**
  
  $$a(1+b)(1+c) = a + ab + ac + abc$$

- **Atomic Broadcast:**
  
  $$a(1 + bcd) = a + abcd$$

- **Causal chain:**
  
  $$a(1+b(1+c(1+d))) = a + ab + abc + abcd$$
Interaction Modeling: The Algebra of Connectors

**Syntax:** For P be a set of ports, such that \( 0,1 \notin P \), \( AC(P) \) is defined by

\[
\begin{align*}
s & ::= [0] | [1] | [p] | [x] \quad \text{(synchrons)} \\
t & ::= [0]' | [1]' | [p]' | [x]' \quad \text{(triggers)} \\
x & ::= s \mid t \mid x.x \mid x + x
\end{align*}
\]

where

- \([ ]\), \([ ]'\) are unary *typing* operators
- \(\cdot\) is the *fusion* operator which is associative, commutative, \([1]\) is an identity element and \([0]\) is an absorbing element
- \(\oplus\) is the *union* operator.
Interaction Modeling: Hierarchical Connectors

Rendezvous: abc

Broadcast: 
\[a'bc = a(1+b)(1+c)\]

Atomic Broadcast: 
\[a'[bcd] = a(1+bcd)\]

Causal chain: 
\[a'[b'[c'd]]=a(1+b(1+c(1+d)))\]
For two connectors $\gamma_1 = a'.b$ and $\gamma_2 = c'.d$

$$[\gamma_1]' \cdot [\gamma_2] = [a'b]' \cdot [c'd]$$
Interaction Modeling:
The Algebra of Connectors - Equivalence vs. Congruence

Definition: $\gamma_1 \approx \gamma_2$ if $\gamma_1$ and $\gamma_2$ represent the same set of interactions

Remark: $\approx$ is not a congruence as it is not preserved by fusion, for example $a'b \approx a+ab$ and $a'bc \not\approx ac+abc$

Semantics in terms of causality trees preserving the causality relation induced by triggers
CN: BUS={send, rec1, rec2}
  send: true $\rightarrow$ skip
  send.rec1: x<y $\rightarrow$ x:=y-x, y:=y+x
  send.rec2: x<z $\rightarrow$ x:=z-x, z:=z+x
  send.rec1.rec2: x<z+y $\rightarrow$ x:=y+z-x, y:=y+x, z:=z+x

Maximal progress: execute a maximal enabled interaction
Interaction Modeling: Composition - Semantics

CN: BUS = \{p_1, p_2, \ldots , p_n, \ldots , p_s\}

\[ \alpha = p_1 p_2, \ldots , p_n : G_\alpha \rightarrow F_\alpha \]

\[ g_\alpha = g_{p_1} \land g_{p_2} \land \ldots \land g_{p_n} \land G_\alpha \]

\[ f_\alpha = F_\alpha ; f_{p_1} , f_{p_2} , \ldots , f_{p_n} \]
Interaction Modeling: Producer-Consumer

Producer

Resource

Consumer

put(item, buffer)

Item := first(buffer)

Producer

Resource

Consumer

PrReq

PrRel

PrReq

PrRel

PrReq

PrRel

CnRel

CnReq

CnReq

CnRel

CnReq

CnRel

Pr_item

Active

Idle

Pr_item

Active

Idle

Cn_item

Active

Idle

Item := first(buffer)

PrRel

PrReq

CnRel

CnReq

PrRel

PrReq

CnRel

CnReq

PrRel

PrReq

CnRel

CnReq

buffer

Item

Item

Interaction Modeling: Checking for Deadlock-freedom

For K1,K2,K3 deadlock-free components

\[
en(p_1) \land \neg en(p_2) \land
\neg en(q_1) \land en(q_2)
\]

\[
en(p_1) \land \neg en(p_2) \land
en(q_2) \land \neg en(q_3) \land en(r_3) \land \neg en(r_1)
\]
Overview

• About component-based construction

• Basic Concepts

• Interaction modeling

• Priority modeling

• Implementation

• Modeling systems in BIP

• Discussion
Priorities

Priorities are a powerful tool for restricting non-determinism

• they allow straightforward modeling of urgency and scheduling policies for real-time systems

• run to completion and synchronous execution can be modeled by assigning priorities to threads

• they can advantageously replace (static) restriction of process algebras
Priorities: Priorities as Controllers

A controller restricts the non determinism of system S to enforce a property P

Controller for P

interaction → state

Interactions

Results [Goessler&Sifakis, FMCO2003]:

- Restrictions induced by controllers enforcing deadlock-free state invariants can be described by dynamic priorities

- Conversely, for any restriction induced by dynamic priorities there exists a controller enforcing a deadlock-free control invariant
Priorities: Definition

<table>
<thead>
<tr>
<th>Priority rule</th>
<th>Restricted guard g₁'</th>
</tr>
</thead>
<tbody>
<tr>
<td>true → p₁ ∩ p₂</td>
<td>g₁' = g₁ ∧ ¬g₂</td>
</tr>
<tr>
<td>C → p₁ ∩ p₂</td>
<td>g₁' = g₁ ∧ ¬(C ∧ g₂)</td>
</tr>
</tbody>
</table>
Priorities: Definition

A *priority order* is a strict partial order $\subseteq \text{Int} \times \text{Int}$

A set of *priority rules*, $PR = \{ C_i \rightarrow \sigma_i \}$, where $\{C_i\}_i$ is a set of disjoint state predicates.

**Operational Semantics**

$$g'_k = g_k \land \bigwedge C \rightarrow \sigma \in PR \left( C \Rightarrow \bigwedge p_k (p_i \rightarrow g_i) \right)$$
Priorities: FIFO policy

PR : \( t_1 \leq t_2 \rightarrow b_1 \bowtie b_2 \)  \( t_2 < t_1 \rightarrow b_2 \bowtie b_1 \)
Priorities: EDF policy

PR: D1-t1 ≤ D2-t2 → b2( b1
D2-t2 < D1-t1 → b1( b2

idle1

a1

start t1

ready1

b1

t1 ≤ D1

f1

exec1

idle2

a2

start t2

ready2

b2

t2 ≤ D2

f2

exec2

#
Priorities: Composition

PR1
PR2

PR1 ≠ PR2

a \langle 1^b
b \langle 2^c
a \langle 1^b

Priority modeling– Composition (2)

We take:

PR1 ⊕ PR2 is the least priority containing PR1 ∪ PR2

Results:

• The operation ⊕ is partial, associative and commutative
• PR1(PR2(B)) ≠ PR2(PR1(B))
• PR1 ⊕ PR2(B) refines PR1 ∪ PR2(B) refines PR1(PR2(B))
• Priorities preserve deadlock-freedom
Priorities: Mutual Exclusion + FIFO policy

\[
\begin{align*}
\text{t1} \leq \text{t2} & \rightarrow \ b1 \prec b2 \\
\text{t2} < \text{t1} & \rightarrow \ b2 \prec b1 \\
\text{true} & \rightarrow \ b1 \prec f2 \\
\text{true} & \rightarrow \ b2 \prec f1
\end{align*}
\]

Diagram:
- idle1
- a1
- start t1
- ready1
- b1
- f1
- exec1
- idle2
- a2
- start t2
- ready2
- b2
- f2
- exec2
Priorities: Mutual Exclusion - Example

PR : \( b_1 \prec f_2 \) \( b_2 \prec \{ f_1, b_1' \} \) (mutex on R)

PR' : \( b_2' \prec f_1 \) \( b_1' \prec \{ f_2, b_2 \} \) (mutex on R')

Risk of deadlock: \( PR \oplus PR' \) is not defined
Overview

• About component-based construction

• Basic Concepts

• Interaction modeling

• Priority modeling

• Implementation

• Modeling systems in BIP

• Discussion
The BIP Framework: Model Construction Space - Related Approaches

Vanderbilt’s Approach

Semantic Unit Meta-model
- Composition Operators
- Behavior

Operational Semantics
- ASML

Metropolis

Semantic Domain
- Quantity Manager
- Media
- Behavior

Operational Semantics
- Platform

PTOLEMY

MoC (Model of Computation)
- Director
- Channels
- Behavior

Operational Semantics
- Platform
A system is defined as a point of the 3-dimensional space.
Separation of concerns: any combination of coordinates defines a system.
The BIP Framework: Model Construction Space

The model construction space for PTOLEMY
The BIP Framework: Model Construction Space – Property Preservation

- Deadlock-free
- Invariant

Characterize relations between classes by elementary model transformations:
- Untimed-timed
- Synchronous – asynchronous
- Event triggered – data triggered
The BIP Framework: Implementation

BIP Program

BIP MetaModel

BIP Model

compiler

deadlock detection
invariant generation

structural analysis

code generation

BIP C++ Code

BIP/Linux Platform

centralized/distributed
execution,
guided/exhaustive
simulation

BIP/C Code

BIP/Think Platform

centralized execution
(on bare machines)

JAHUEL

FXML
The BIP Framework: Implementation - generation of C++ code

- Component Meta-model
- Interaction Meta-model
- Dynamic priorities Meta-model
- Execution Engine

BIP model

The BIP Framework: Implementation - The Execution Platform

Interaction model

Priorities

Execution Engine

Platform
The BIP Framework: Implementation – The BIP Execution Engine

```
<table>
<thead>
<tr>
<th>init</th>
<th>Launch atom’s threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop</td>
<td>Wait all atoms</td>
</tr>
<tr>
<td></td>
<td>Compute legal interactions</td>
</tr>
<tr>
<td>execute</td>
<td>Notify involved atoms</td>
</tr>
<tr>
<td></td>
<td>Execute chosen interaction transfer</td>
</tr>
<tr>
<td>choose</td>
<td>Choose among maximal</td>
</tr>
<tr>
<td>filter</td>
<td>Filter w.r.t. priorities</td>
</tr>
<tr>
<td>ready</td>
<td></td>
</tr>
</tbody>
</table>
```

component C
port complete: p1, ... ; incomplete: p2, ...
data {# int x, float y, bool z, .... #}
init {# z=false; #}
behavior
  state s1
    on p1 provided g1 do f1 to s1'
    ......................
    on pn provided gn do fn to sn'
  state s2
    on ..... 
    ....
  state sn
    on .... 
end
The BIP Framework: The Language – Atomic Components

component CallingUnit(int px, int py)

complete port request // request port
port release // release port
port tick // time synchronization port

data int x // x coordinate of the calling unit
data int y // y coordinate of the calling unit
data int t // local clock

behavior
initial do x = px; {# y = py; #} t = 4 to EMPTY
state EMPTY // empty state
  on request provided t >= 4 to OCCUPIED
  on tick do t := t + 1 to EMPTY
state OCCUPIED // occupied state
  on release do t := 0 to EMPTY
  on tick to OCCUPIED
end
end
The BIP Framework: The Language – Connectors and Priorities

connector BUS = \{p, p', \ldots, \}\ncomplete()

behavior
  on \alpha_1 \text{ provided } g_{\alpha_1} \text{ do } f_{\alpha_1}
  on \alpha_2 \text{ provided } g_{\alpha_2} \text{ do } f_{\alpha_2}
end

priority PR
  if C1 (\alpha_1 < \alpha_2), (\alpha_3 < \alpha_4), \ldots
  if C2 (\alpha < \ldots), (\alpha < \ldots), \ldots
  \ldots
  if Cn (\alpha < \ldots), (\alpha < \ldots), \ldots
The BIP Framework: The Language – Connectors and Priorities

connector Bus = A.s, B1.r, B2.r
behavior
  on A.s provided true do A.x = A.x + 1
  on A.s, B1.r provided A.x < B1.y do B1.y = A.x
  on A.s, B2.r provided A.x > B2.y do B2.y = A.x
  on A.s, B1.r, B2.r provided true do A.x = (B1.y + B2.y) / 2
end

priority mutex
  if (C1.t > C2.t) C1.begin < C2.begin
  if (C1.t <= C2.t) C2.begin < C1.begin
end
The BIP Framework: The Language – Compound Components

component name
  contains c_name1 i_name1(par_list)
      ......
  contains c_namen i_namen(par_list)

  connector name1
      ......
  connector namem

  priority name1
      ......
  priority namek
end
component Utopar

    contains CallingUnit c11(1,1)
    contains CallingUnit c12(1,2)
    contains CallingUnit c21(2,1)
    contains CallingUnit c22(2,2)

    contains CentralStation s

    connector request = c11.request, c12.request, c21.request, c22.request, s.request
    ...
    connector release11 = c11.release, s.release
    ...
    connector tick = c11.tick, c12.tick, c21.tick, c22.tick, s.tick
    ...
end
Overview

- About component-based construction
- Basic Concepts
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion
Timed Components – Timed vs. Untimed

PR: red Guards $\rightarrow$ tick $\langle$ all other ports $\rangle$

Timed components - Problem 1

Bursty Event-Stream:
- Period = 10
- Jitter = 50
- Min. Interarrival Dist. = 1

CPU1
- WCED = 8

CPU2
- WCED = 4

CPU3
- WCED = 1
Timed components: Problem 1- Task

![Diagram of a task with states and transitions](image)

- **READY** state:
  - Transition on `get` event: `count++`
  - Transition on `tick` event: `delay++`

- **EXEC** state:
  - Transition on `tick` event: `delay++`
  - Transition on `get` event: `count++`

**Transitions**:
- **start**, `(count>0)`: from READY to EXEC
- **finish**, `[delay<= WCET]`: from EXEC to READY
- **finish**, `delay:=0`: from EXEC to READY
Timed components: Problem 1 - BIP code snippet for Task

```bips
component Task (int wcet)
  port get, start, tick, finish
  data {# int count, delay; #}

  ... init {# count = 0;
    WCET = wcet;
  ...
    #} behavior
  state READY
    on get do {# count++; #} to READY
  on start provided {# count > 0 #} do {# count--; delay = 0; #} to EXEC
  on tick to READY
  state EXEC
    on get do {# count++; #} to EXEC
  on finish when ({# delay <= WCET #}, delayable) to READY
  on tick do {# delay++; #} to EXEC
  end
end
```

Timed components: Problem 1- Bursty Event Stream Generator

Bursty Event-Stream for
Period = T
Jitter = J
Min. Interarrival Dist. = d

\[ a_i \]
\[ a_{i-1} \]
\[ y \]
\[ x \]

\[ i \]
\[ i+1 \]
\[ \ldots \]
\[ i+k \]
Timed components: Problem 1 – Architecture

PR: \( \text{tick} \langle \{ \text{EvntT1, T1T2, T2T3, T3.Finish} \} \)
Timed components: Problem 1- BIP code snippet for Architecture

component System
contains Launcher eventGenerator(10, 5, 1)
contains Task T1(8), T2(4), T3(1)

connector Tick = eventGenerator.tick, T1.tick, T2.tick, T3.tick
behavior
end
connector EvntT1 = eventGenerator.go, T1.get
behavior
end
...
priority // start < get (no event losses)
getStart1 T1.Start : T1.start < EvntT1 : T1.get
...
priority // finish < get (no event losses)
getFin1 T1T2 : T1.finish < EvntT1 : T1.get
...
priority // tick < get_i (=> tick < finish_i-1 )
getTick2 if (T1.delay == T1.WCET) Tick : T2.tick < T1T2 : T2.get
...
Timed components: Billiards - the Model

PR: tick \( \langle \{\text{shock,flip}\} \rangle \)

CN: shock

\[ g_{\text{shock}} : y_1 = y_2 \land x_1 = x_2 \]

\[ f_{\text{shock}} : v_{x_1}, v_{x_2}, v_{y_1}, v_{y_2} := -v_{x_2}, -v_{x_1}, -v_{y_2}, -v_{y_1} \]

CN: tick

BALL_1

BALL_2
MPEG4 Video Encoder

Transform a monolithic program into a componentized one
  ++ reconfigurability, schedulability
  –– overheads (memory, execution time)

Video encoder characteristics:
  • 12000 lines of C code
  • Encodes one frame at a time:
    – grabPicture() : gets a frame
    – outputPicture() : produces an encoded frame
MPEG4 Video Encoder: Architecture

GrabMacroBlock: splits a frame in \((W*H)/256\) macro blocks, outputs one at a time.

Reconstruction: regenerates the encoded frame from the encoded macro blocks.

GrabMacroBlock:

**Buffered connections**
MPEG4 Video Encoder: Atomic Components

\[ \text{MAX} = \left(\frac{W \times H}{256}\right) \]

\( W = \text{width of frame} \)

\( H = \text{height of frame} \)
MPEG4 Video Encoder: Features

- BIP code describes a control skeleton for the encoder
  - Consists of 20 atomic components and 34 connectors
  - ~ 500 lines of BIP code
  - Functional components call routines from the encoder library

- The generated C++ code from BIP is ~ 2,000 lines

- The size of the BIP binary is 288 Kb compared to 172 Kb of monolithic binary.
MPEG4 Video Encoder: BIP Componentization Overhead

Overhead in execution time wrt monolithic code:

- ~66% due to communication (can be reduced by composing components at compile time)
  - function calls by atomic components to the execution engine for synchronization.

- ~34% due to resolution of non determinism (can be reduced by narrowing the search space at compile time)
  - time spent by engine to evaluate feasible interactions

Problem: Reduce execution time overhead for componentized code
Overview

• About component-based construction

• Basic Concepts

• Interaction modeling

• Priority modeling

• Implementation

• Modeling systems in BIP

• Discussion
Discussion: The BIP Framework - Summary

Framework for component-based modeling encompassing heterogeneity and relying on a **minimal set of constructs and principles**

Clear separation between behavior and architecture
- Architecture = interaction + priority
- Correctness-by-construction techniques for deadlock-freedom and liveness, based on sufficient conditions on architecture

Other 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: The BIP Framework – Towards a Taxonomy of Systems

A component is defined as a point in the space:

$$\text{Behavior} \times \text{Interaction} \times \text{Priority}$$

Classes of components can be obtained by application of simple transformations

- Behavior: Coupling/Decoupling interaction and computation
- Interaction models: synchronous execution
- Priorities: adding/removing priority rules

Basis for property preservation results and correctness by construction
Discussion: The BIP Framework – Expressiveness

Study new notions of expressiveness where architecture is a first class entity. Existing notions consider glue operators as behavior transformers through operational semantics.

Problem: For given $B$, $IM$ and $PR$ which coordination problems can be solved?
Discussion: The BIP Framework – Unification

How?

NesC  NesC2BIP  BIP engine

TinyOS

EC  TC  AC
Discussion: The BIP Framework - Distribution

Centralized Semantics:
Transitions are atomic - Global state
Basic communication primitive: n-ary interaction

Distributed Semantics:
Transitions are non atomic - Partial state
Basic primitive: send/receive
References

Papers available at: http://www-verimag.imag.fr/~sifakis/
- “The Algebra of Connectors – Structuring Interaction in BIP” EmSoft07
- "Modeling Heterogeneous Real-time Systems in BIP“ SEFM06, IEEE.
- “Component-based construction of deadlock-free systems”, FSTTCS03, LNCS 2194.
- “Priority Systems” Proceedings of FMCO’03, LNCS 3188

BIP web page:
http://www-verimag.imag.fr/%7Easync/index.php?view=components