Component-based Construction of Real-time Systems in BIP

Joseph Sifakis
in collaboration with
A. Basu, M. Bozga and G. Goessler

Workshop on Foundations and Applications of Component-based Design

Seoul, October 26, 2006
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. data driven computation
  ▪ Distributed vs. centralized execution

• Automated support for component integration and generation of glue code meeting given requirements
Component-based construction – The BIP framework

Layered component model

Priorities (Conflict resolution)

Interaction Model (Collaboration)

B E H A V I O R

Composition (incremental description)

PR1 ⊕ PR2 ⊕ PR12

IM1 ⊗ IM2 ⊗ IM12
An atomic component has

- A set of ports $P$, for interaction with other components
- A set of control states $S$
- 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)
Component-based construction – The BIP framework: Behavior

\[ s1 \xrightarrow{p} g_p f_p \xrightarrow{f_p} s2 \]

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

**Semantics**
- **Enabledness**: \( g_p \) is true and some interaction involving p is possible
- **Execution**: interaction involving p followed by the execution of \( f_p \)
Overview

• Interaction modeling
• Priority modeling
• Implementation
• Modeling systems in BIP
• Discussion
• A **connector** is a set of ports which can be involved in an interaction.

• Port attributes (**complete**, **incomplete**) are used to distinguish between rendezvous and broadcast.

• An **interaction** of a connector is a set of ports such that: either it contains some complete port or it is maximal.

Interactions:

\{\text{tick1, tick2, tick3} \} \{\text{out1} \} \{\text{out1, in2} \} \{\text{out1, in3} \} \{\text{out1, in2, in3} \}
Interaction modeling - Examples

1. CN: \{cl1, cl2\}  CP: \emptyset
   - cl1
   - cl2

2. CN: \{out, in\}  CP: \{out\}
   - out
   - in

3. CN: \{in1, out, in2\}  CP: \{out\}
   - out
   - in1
   - in2
Interaction modeling – Composition

CN[P,C]: {put, get}
CP[P,C]: Ø

CN[P]: {put}, {prod}
CP[P]: {prod}

CN[C]: {get}, {cons}
CP[C]: {cons}

CN: {put, get}, {prod}, {cons}
CP: {prod}, {cons}
**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}`: $x < z + y \rightarrow x := y + z - x$, $y := y + x$, $z := z + x$

- Notice the difference between control flow and data flow (input, output)
- Maximal progress: execute a maximal enabled interaction
Overview

• Interaction modeling

• Priority modeling

• Implementation

• Modeling systems in BIP

• Discussion
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 - Definition

Priority rules

<table>
<thead>
<tr>
<th>Priority rule</th>
<th>Restricted guard $g_1'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true $\rightarrow p_1 \ll p_2$</td>
<td>$g_1' = g_1 \land \neg g_2$</td>
</tr>
<tr>
<td>$C \rightarrow p_1 \ll p_2$</td>
<td>$g_1' = g_1 \land \neg(C \land g_2)$</td>
</tr>
</tbody>
</table>
Priorities – Example: FIFO policy

\[ t_1 \leq t_2 \rightarrow b_1 \prec b_2 \]

\[ t_2 < t_1 \rightarrow b_2 \prec b_1 \]

- sleep1
- a1
- start t1
- wait1
- b1
- e1
- use1

- sleep2
- a2
- start t2
- wait2
- b2
- e2
- use2

#
Priorities – Example: EDF policy

D1-t1 ≤ D2-t2 → b2 ↯ b1

D2-t2 < D1-t1 → b1 ↯ b2

Diagram:

- sleep1
  - a1
  - start t1

- wait1
  - b1
  - e1

- use1

- sleep2
  - a2
  - start t2

- wait2
  - b2
  - t2 ≤ D2

- use2

- t1 ≤ D1

- A green arrow labeled '#' indicates the relationship between use1 and use2.
Priorities – Composition

\[ \begin{array}{c}
\text{pr2} \\
\text{pr1}
\end{array} \quad \neq \quad \begin{array}{c}
\text{pr1} \\
\text{pr2}
\end{array} \]

\[ \begin{array}{c}
\text{pr1} \\
\text{pr2}
\end{array} \]

\[ \begin{array}{c}
a <^1 b \\
b <^2 c \\
a <^1 b
\end{array} \]

\[ \begin{array}{c}
a \\
b \\
a
\end{array} \]

\[ \begin{array}{c}
c \\
c \\
c
\end{array} \]
Priorities – Composition (2)

**Take:**

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

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

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

**Results:**
- The operation \( \oplus \) is partial, associative and commutative
- \( \text{pr1} \oplus \text{pr2}(B) \neq \text{pr1}(\text{pr2}(B)) \)
- \( \text{pr1} \oplus \text{pr2}(B) \text{ refines pr1} \cup \text{pr2}(B) \text{ refines pr1}(\text{pr2}(B)) \)
- Priorities preserve deadlock-freedom
Priorities – Example: Mutual exclusion + FIFO policy

<table>
<thead>
<tr>
<th>Condition</th>
<th>Event 1</th>
<th>Event 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 ≤ t2</td>
<td>b1 &lt; b2</td>
<td>t2 &lt; t1</td>
</tr>
<tr>
<td>true</td>
<td>b1 &lt; e2</td>
<td>true</td>
</tr>
</tbody>
</table>

- **Sleep** events: sleep1, sleep2
- **Wait** events: wait1, wait2
- **Use** events: use1, use2

- start t1
- start t2

**Example:**

睡1

等待1

使用1

t1 ≤ t2 → b1 < b2

true → b1 < e2

等待2

使用2

t2 < t1 → b2 < b1

true → b2 < e1
Priorities – Checking for deadlock-freedom: Example

Mutex on $R'$: $b_1 \langle f_2 \rangle \space b_2 \langle \{ f_1, b_1' \} \rangle$

Mutex on $R$: $b_1' \langle \{ f_2, b_2 \} \rangle \space b_2' \langle f_1 \rangle$

Risk of deadlock: $b_1' \langle b_2 \rangle$ and $b_2 \langle b_1' \rangle$
Overview

- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion
Implementation – the BIP toolset

Graphic language
AADL or UML

BIP language

C++

BIP Platform

THINK

IF

IF Platform
Implementation – C++ code generation for the BIP platform

Component Meta-model

Interaction Meta-model

Priority Meta-model

Engine

BIP Platform

BIP model

C → a(b)
Implementation – The BIP platform

- Code execution and state space exploration features
- Implementation in C++ on Linux using POSIX threads
Implementation – The BIP platform: The engine

- **init**: Launch atom’s threads
- **loop**: Wait all atoms, Compute legal interactions
- **execute**: Execute chosen interaction transfer
- **choose**: Choose among maximal
- **filter**: Filter w.r.t. priorities
- **stable**: Notify involved atoms
Overview

- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion
Modeling in BIP – Other approaches encompassing heterogeneity

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. Full separation of concerns: any combination of coordinates defines a system.
Modeling in BIP – Model construction space (2)

Model construction space for PTOLEMY
The BIP framework – Relating classes of components

Study transformations characterizing relations between classes of systems:
• Untimed – timed
• Synchronous – asynchronous
• Event triggered – data triggered
Modeling in BIP – Timed systems

Timed Component

PR: red guards \rightarrow tick \langle all\_other\_ports

Timed architecture
Modeling in BIP – Synchronous systems

Synchronous component

PR: syn\(\langle\) all_other_ports

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

Reconstruction: regenerates the encoded frame from the encoded macro blocks.
Modeling in BIP – Video encoder: Atomic components

**GrabMacroBlock**

- **f_{in}**
  - `c = MAX` $c := 0$
  - `c < MAX`
  - `grabMacroBlock()`, $c := c + 1$
- **out**

**Reconstruction**

- **f_{in}**
  - `in c < MAX` $c := c + 1$
- **f_{out}**
  - `f_{out}`
  - `c = MAX`
  - `c := 0`
- **reconstruction()**

**Generic Functional component**

- **in**
- **out**
- `fn()`

MAX = $\frac{W \times H}{256}$

W = width of frame

H = height of frame
Modeling in BIP – Video encoder: The BIP 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.
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

- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion
Discussion – The BIP framework: summary

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

Clear separation between structure (interaction + priority) and behavior

- Structure is a first class entity
- Layered description => separation of concerns => incrementality
- Correct-by-construction techniques for deadlock-freedom and liveness, based (mainly) on sufficient conditions on the structure
Methodology
• Modeling: BIP as a programming model, reference architectures in BIP
• Implementation techniques

BIP toolset
• Generation of BIP models from system description languages such as SysML (IST/SPEEDS project), AADL and SystemC (ITEA/Spices project)

• Model transformation techniques in particular for code optimization

• Validation techniques
  ▪ connection to Verimag’s IF simulation/validation environment
  ▪ specific techniques e.g. checking conditions for correctness by construction
Theory

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

• Study notions of **expressiveness** characterizing structure: Given two component algebras defined on the same set of atomic components,
  \[
  \text{CA1 is more expressive than CA2}
  \]
  if \( \forall P \ \exists gl_2 \in GL_2 \ \text{gl}_2(B_1, \ldots, B_n) \sat P \Rightarrow \exists gl_1 \in GL_1. \ \text{gl}_1(B_1, \ldots, B_n) \sat P \)

• Model transformations
  - relating classes of systems
  - preserving properties

• Distributed implementations of BIP
More about BIP:

- Email to Joseph.Sifakis@imag.fr

THANK YOU
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
end
Implementation – the BIP language: connectors and priorities

**connector** BUS = \{ p, p', \ldots , \}

**complete()**

**behavior**

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

**priority** PR

- **if** $C_1 (\alpha_1 < \alpha_2), (\alpha_3 < \alpha_4), \ldots$
- **if** $C_2 (\alpha < \ldots), (\alpha < \ldots), \ldots$
- 
- 
- **if** $C_n (\alpha < \ldots), (\alpha < \ldots), \ldots$
Implementation – the BIP language: compound component

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