Modeling Heterogeneous Real-time Components in BIP

Joseph Sifakis
in collaboration with Ananda Basu and Marius Bozga
VERIMAG

Composition of Embedded Systems
Scientific and Industrial Issues
Monterey Workshop
Paris, October 16-18, 2006
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. data driven computation
  ▪ Distributed vs. centralized execution

• Automated support for component integration and generation of glue code meeting given requirements
Approaches 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
  ▪ 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.
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

Java  UML
SDL  UML

Lotos  CSP

Asynchronous Computation

Synchronous Computation

Matlab/Simulink
VHDL/SystemC
Synchronous languages
Component-based construction – The BIP framework

Layered component model

Priorities (Conflict resolution)

Interaction Model (Collaboration)

Composition (incremental description)
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

- $p$: 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:**

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

1. \( \text{CN} = \{\text{cl1, cl2}\}, \text{CP} = \emptyset \)

2. \( \text{CN} = \{\text{out, in}\}, \text{CP} = \{\text{out}\} \)

3. \( \text{CN} = \{\text{in1, out, in2}\}, \text{CP} = \{\text{out}\} \)
Interaction modeling – Composition

Interaction modeling Interaction modeling –– Composition

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

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

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

\[ \text{CN}: \{\text{put, get}, \{\text{prod}\}, \{\text{cons}\}\} \]
\[ \text{CP}: \{\text{prod}, \{\text{cons}\}\} \]

prod \rightarrow put

get \rightarrow 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 → $p_1 \langle p_2$</td>
<td>$g_1' = g_1 \land \neg g_2$</td>
</tr>
<tr>
<td>$C \rightarrow p_1 \langle p_2$</td>
<td>$g_1' = g_1 \land \neg(C \land g_2)$</td>
</tr>
</tbody>
</table>
Priorities – Example: FIFO policy

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

Diagram:
- Sleep\(_1\) (a1) → Wait\(_1\) (b1) → Use\(_1\) (e1)
- Sleep\(_2\) (a2) → Wait\(_2\) (b2) → Use\(_2\) (e2)
- Start \(t_1\)
- Start \(t_2\)
- \#
Priorities – Example: EDF policy

D1-t1 ≤ D2-t2 → b2 ∥ b1

D2-t2 < D1-t1 → b1 ∥ b2

Diagram:

- sleep1
- wait1
- use1
- a1
- start t1
- b1
- t1 ≤ D1
- e1

- sleep2
- wait2
- use2
- a2
- start t2
- b2
- t2 ≤ D2
- #
Priorities – Composition

\[ \begin{array}{c}
\text{pr2} \\
\text{pr1}
\end{array} \] \quad \neq \quad \begin{array}{c}
\text{pr1} \\
\text{pr2}
\end{array} \]
Priorities – Composition (2)

Take:

\[ \text{pr1} \oplus \text{pr2} \]

\[ \text{pr1} \cup \text{pr2} \]

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

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

- $t_1 \leq t_2 \rightarrow b_1 \prec b_2$
- $t_2 < t_1 \rightarrow b_2 \prec b_1$

- $\text{true} \rightarrow b_1 \prec e_2$
- $\text{true} \rightarrow b_2 \prec e_1$

**Diagram:**

- Sleep 1: $a_1 \rightarrow \text{start } t_1$
- Wait 1: $b_1 \rightarrow \text{use } 1$
- Use 1: $e_1 \rightarrow \text{sleep } 1$

- Sleep 2: $a_2 \rightarrow \text{start } t_2$
- Wait 2: $b_2 \rightarrow \text{use } 2$
- Use 2: $e_2 \rightarrow \text{sleep } 2$
Priorities – Checking for deadlock-freedom: Example

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

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

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

IF

IF Platform

THINK
Implementation – C++ code generation for the BIP platform

Component Meta-model

Interaction Meta-model

Priority Meta-model

BIP Platform
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**
  - Notify involved atoms
  - Execute chosen interaction transfer

- **choose**
  - Choose among maximal

- **filter**
  - Filter w.r.t. priorities

- **stable**
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'

  state s2
    on ..... 

  state sn
    on ..... 

end
end
connector BUS = \{p, p', \ldots , \}\n
\begin{verbatim}
complete()
  behavior
    on \( \alpha_1 \) provided \( g_{\alpha_1} \) do \( f_{\alpha_1} \)
    ...........
    on \( \alpha_n \) provided \( g_{\alpha_n} \) do \( f_{\alpha_n} \)
\end{verbatim}

priority PR
\begin{verbatim}
  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
\end{verbatim}
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
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
.net

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 → tick \( \langle \text{all\_other\_ports} \rangle \)

Timed architecture
Modeling in BIP – Synchronous systems

Synchronous component

Micro-step

PR: syn\ all_other_ports

Synchronous architecture
Modeling in BIP – MPEG4 Video encoder: Componentization

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
Modeling in BIP – Video encoder: The Encode component

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.
Modeling in BIP – Video encoder: Atomic components

GrabMacroBlock

Reconstruction

Generic Functional component

\[ MAX = \frac{(W \times H)}{256} \]
\[ W = \text{width of frame} \]
\[ H = \text{height of frame} \]
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
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
The BIP framework: Work directions (2)

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, $CA_1$ is more expressive than $CA_2$ if $\forall P \exists g_2 \in GL_2$ $g_2(B_1, \ldots, B_n)$ sat $P \Rightarrow \exists g_1 \in GL_1$. $g_1(B_1, \ldots, B_n)$ 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