Component-based Construction of Real-Time Systems

Joseph Sifakis
VERIMAG

ARTIST2 MOTIVES School
Trento, February 19-23, 2007
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.
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

nonA R

A R

nonA R

nonA B

nonA B

Synchronous Computation

Matlab/Simulink
VHDL/SystemC
Synchronous languages
Overview

- About component-based construction
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion
Build a component $C$ satisfying a given property $P$, from

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

Glue operators

- model mechanisms used for communication and control such as protocols, controllers, buses.
- restrict the behavior of their arguments, that is

$$gl(C_1, C_2, \ldots, C_n) | A_1 \text{ refines } C_1$$
Component-based construction – Formal framework

Semantics:
• Atomic components $\rightarrow$ behavior
• Glue operators transform sets of components into components

The process algebra paradigm
• Components are terms of an algebra of terms $(\mathcal{C}, \cong)$ generated from $\mathcal{C}_0$ by using operators from $\mathcal{GL}$
• $\cong$ is a congruence compatible with semantics
Find sets of glue operators meeting the following requirements:

1. Incremental description
2. Correctness-by-construction
3. Expressiveness (discussed later)
Component-based construction – Incremental description

1. Decomposition

\[ C_1, C_2, \ldots, C_n \]

\[ gl_1 \]

\[ \cong \]

\[ C_1, C_2, \ldots, C_n \]

\[ gl_2 \]

2. Flattening

Flattening can be achieved by using a (partial) associative operation \( \oplus \) on GL
Component-based construction – Correctness by construction : Compositionality

Building correct systems from correct components

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

Integrated components preserve essential properties

Composability means non interference of properties of integrated components. Lack of results for guaranteeing property stability e.g.

- non composability of scheduling algorithms
- feature interaction
Component-based construction – The BIP framework

Layered component model

Priorities (Conflict resolution)

Interaction Model (Collaboration)

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

$s_1 \xrightarrow{p \ g_p \ f_p} s_2$

$p$: 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

- About component-based construction
- 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. **cl1, cl2**
   - CN: \{cl1, cl2\}
   - CP: \emptyset

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

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

Atomic Broadcast:
send + send rec1 rec2

Causal chain:
a+ab+abc+abcd

≈
Interaction modeling – Operational semantics

CN: \{put, get\}, \{prod\}, \{cons\}
CP: \{prod\}, \{cons\}

Operational Semantics

Semantics

prod \rightarrow put \rightarrow prod
get \rightarrow cons \rightarrow get
Interaction modeling – Composition

CN[P,C]: {put, get}
CP[P,C]: ∅

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

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

CN: {put, get}, {prod}, {cons}
CP: {prod}, {cons}
Interaction modeling – Composition: Results [Goessler&SifakisSCP2005]

Incremental commutative composition directly encompassing rendezvous and broadcast
**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
Interaction modeling – Checking for deadlock-freedom \cite{Goessler&Sifakis:FSTTCS2003}

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** $E$
  - $(\alpha, p, k) \in E$ if $\alpha$ is an interaction, $p \in \alpha$ is an incomplete port of $k$
  - $(k1, p1, \alpha) \in E$ if $p1 \in \alpha$ is a port of $k1$

**Blocking condition for an incomplete port $p$:**

\[ Bl(p) = g_p \land \neg (g_{p1} \land g_{p2} \land g_{p3}) \]
Possibility of deadlock for the components of circuits $\omega$
such that $Bl(\omega) = \land_{p \in \omega} Inc(\omega) \land Bl(p) = false$

where $Inc(\omega) = \land_{k \in \omega} Inc(k)$ with $Inc(k)$ the set of the control states of $k$ offering only incomplete ports.
Overview

• About component-based construction
• 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
A controller restricts the behavior (non determinism) of system $S$ to enforce a property $P$.

**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.
**Priority rules**

<table>
<thead>
<tr>
<th>Priority rule</th>
<th>Restricted guard $g_1'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$true \rightarrow p_1 \perp p_2$</td>
<td>$g_1' = g_1 \land \neg g_2$</td>
</tr>
<tr>
<td>$C \rightarrow p_1 \perp p_2$</td>
<td>$g_1' = g_1 \land \neg (C \land g_2)$</td>
</tr>
</tbody>
</table>
Priorities – Definition (2)

\[ \text{pr} = \{ C_i \rightarrow \langle \rangle_i \} \text{ is a set of priority rules, where} \]

• \( \{C_i\}_i \) is a set of disjoint state predicates

• \( \langle \rangle_i \subseteq \text{Interactions} \times \text{Interactions} \) is a strict partial order

\[ g'_k = g_k \land \bigwedge C \rightarrow \langle \epsilon \text{pr} \ (C \Rightarrow \bigwedge p_k \langle p_i \rightarrow g_i \rangle) \]
Priorities – Example: FIFO policy

t₁ ≤ t₂ → b₁ ∓ b₂
t₂ < t₁ → b₂ ∓ b₁

sleep₁

a₁

start t₁

wait₁

b₁

e₁

use₁

#

e₂

use₂

start t₂

a₂

wait₂

b₂

sleep₂
Priorities – Example: EDF policy

D1-t1 ≤ D2-t2 → b2 ∥ b1

D2-t2 < D1-t1 → b1 ∥ b2

Diagram:
- Node `sleep1` with transitions `a1`, `start t1`, `b1`, `t1 ≤ D1`, `e1`
- Node `wait1` with transition `b1`
- Node `use1` with transition `#` and edges to `sleep1` and `wait1`
- Node `sleep2` with transition `a2`, `start t2`, `b2`, `t2 ≤ D2`
- Node `wait2` with transition `b2`
- Node `use2` with transition `#` and edges to `sleep2` and `wait2`
Priorities – Composition

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Take:

\[ pr1 \ominus pr2 \]

\[ = \]

\[ pr1 \oplus pr2 \]

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

Results:

• The operation \( \ominus \) 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
Priorities – Example: Mutual exclusion + FIFO policy

<table>
<thead>
<tr>
<th>t1 ≤ t2 → b1 ↪ b2</th>
<th>t2 &lt; t1 → b2 ↪ b1</th>
</tr>
</thead>
<tbody>
<tr>
<td>true → b1 ↪ e2</td>
<td>true → b2 ↪ e1</td>
</tr>
</tbody>
</table>

Diagram:
- **sleep1**
  - **a1**
  - **start t1**
- **wait1**
  - **b1**
  - **e1**
- **use1**
- **sleep2**
  - **a2**
  - **start t2**
- **wait2**
  - **b2**
- **use2**
  - **e2**
Priorities – Checking for deadlock-freedom: Example

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

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

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

- About component-based construction
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion
Implementation – the BIP toolset

Graphic language
AADL or UML

BIP language

THINK

C++

BIP Platform

IF

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

Component Meta-model

Interaction Meta-model

Priority Meta-model

Engine

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
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
connector BUS= \{p, p’, … , \}
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

priority PR
  if C_1 (\alpha_1 < \alpha_2), (\alpha_3 < \alpha_4) , ...
  if C_2 (\alpha < ...), (\alpha < ...) , ...
  ...
  if C_n (\alpha <...), (\alpha <...) , ...
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

• About component-based construction
• Interaction modeling
• Priority modeling
• Implementation
• Modeling systems in BIP
• Discussion
Modeling in BIP—Other frameworks 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
Modeling in BIP– Model construction space

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
Modeling in BIP – 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

Timed architecture

PR: red_guards $\rightarrow$ tick $\langle$ all_other_ports $\rangle$

Modeling in BIP – Timed systems: Example

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

CPU1
T1
WCED = 8

CPU2
T2
WCED = 4

CPU3
T3
WCED = 1

End-to-end Delay?

Source: http://www.tik.ee.ethz.ch/~leiden05
Workshop on Distributed Embedded Systems, Leiden, November 21-24, 2005
Modeling in BIP – Timed systems: Example (2)

Task Component

- get
- count++
- tick
- delay++
- get
- count++
- tick

READY

- start, (count>0)
- count--, delay:=0
- [delay<= WCET ]

EXEC

- finish, [delay<= WCET ]

finish

52
Modeling in BIP – Timed systems: Example (3)

System architecture

PR: tick \{ EvntT1, T1T2, T2T3, T3.Finish \}
Modeling in BIP – Synchronous systems

Synchronous component

PR: syn\ all_other_ports

Synchronous architecture
Modelling in BIP – Synchronous mod2 counter

Modulo-2 counter

in: X
out: Y

Zero
syn → Zero'
flip
g_{flip}: X=1
f_{flip}: Y:=0

Zero'
syn ← Zero

One'
syn ← Zero

One
syn ← One'

flip
g_{flip}: X=1
f_{flip}: Y:=1

syn
Modeling in BIP – Synchronous mod8 counter

PR: syn⟨flip_0, syn ⟩ ⟨flip_1, syn ⟩ ⟨flip_2

CN: syn={syn_0, syn_1, syn_2}, f_{syn}: X_1 := Y_0; X_2 := Y_1 \land Y_0
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 into (W*H)/256 macro blocks, outputs one at a time.

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

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` in
- `c = MAX; c := 0`
- `grabMacroBlock(); c := c + 1`
- `out` out

**Reconstruction**

- `f_out` in
- `c = MAX; c := c + 1`
- `reconstruction()`
- `out` out

**Generic Functional component**

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

Additional notes:

- `MAX = (W * 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

• About component-based construction

• 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
- Correctness-by-construction techniques for deadlock-freedom and liveness, based (mainly) on sufficient conditions on the structure
Discussion – The BIP framework: Work directions

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 $\forall P \exists g \in GL_2 \ g(B_1, \ldots, B_n) \ sat \ P \Rightarrow \exists \ g \in GL_1. \ g(B_1, \ldots, B_n) \ sat \ P$

• Model transformations
  ▪ relating classes of systems
  ▪ preserving properties

• Distributed implementations of BIP
Methodology
- Using BIP as a programming model
- Reference architectures in BIP

BIP toolset Implementation
- 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
More about BIP:


- Email to Joseph.Sifakis@imag.fr

THANK YOU