A Framework for Component-based Construction

Workshop on Distributed Embedded Systems
Leiden, 21-24 November 2005

Joseph Sifakis
VERIMAG and ARTIST2 NoE
sifakis@imag.fr

In collaboration with
Ananda Basu, Marius Bozga
Motivation

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

• Study the concept of architecture as a means to organize computation (behavior, interaction, control)
• Define a meta-model for real-time architectures, encompassing specific styles, paradigms, e.g. modeling
  - Synchronous and asynchronous execution
  - Event driven and state driven interaction
  - Distributed computation
  - Architecture styles such as client-server, blackboard architecture
• Provide automated support for component integration and generation of glue code meeting given requirements
Overview

• Component-based construction – the notion of glue
  • Interaction Models
  • Priorities
  • The BIP framework
  • Discussion
Component-based construction - components

Build systems by composition of components

Atomic 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

![Diagram of atomic components with put, prod, and put labels.]

![Diagram of a transition system with x=1, y:=1, x=1, y:=0, in, x, out, and y labels.]

Joseph Sifakis  Distributed Embedded Systems  Nov. 21-24 2005
Component-based construction – formal framework

Pb: Build a component \( C \) satisfying a given property \( P \), from:
- \( \mathcal{C}_0 \) a set of atomic components
- \( \mathcal{G}\mathcal{L} = \{gl_1, \ldots, gl_i, \ldots\} \) a set of glue operators on components

\[ g_{l12} \]
\[ c_1 \quad c'_1 \quad c_2 \quad c'_2 \]

- Components are terms of an algebra of terms (\( \mathcal{C}, \cong \))
  generated from \( \mathcal{C}_0 \) by using operators from \( \mathcal{G}\mathcal{L} \)
- \( \cong \) 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
  \[ gl(C_1, C_2, \ldots, C_n) \mid A_1 \text{ refines } C_1 \]
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 $gl$

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

2. Flattening of terms

\[ gl_1 = C_1 \oplus gl_2 \]

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

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

Property stability phenomena are poorly understood
• feature interaction
• non composability of scheduling algorithms

Joseph Sifakis    Distributed Embedded Systems    Nov. 21-24 2005
Component-based construction - compositionality vs. composability
Component-based modeling – The BIP framework

Layered component model

Priorities (Memoryless Controller)

Interaction Model (Typed Connectors)

BEHAVIOR

Composition (incremental description)

PR1 ⊕ PR2 ⊕ PR12

IM1 ⊗ IM2 ⊗ IM12
Overview

- Component-based construction – the notion of glue
- Interaction Models
- Priorities
- The BIP framework
- Discussion
Interaction models

- A **connector** is a maximal set of compatible actions.
- An **interaction** is a non-empty subset of a connector.

- Action types *(complete ▼, incomplete ●)* are used to define which subsets are interactions.
- Interactions either contain some complete action or are maximal.

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

- **cl1, cl2**
  - CN: \{cl1, cl2\}
  - MCI: \emptyset

- **out, in**
  - CN: \{out, in\}
  - MCI: \{out\}

- **in1, out, in2**
  - CN: \{in1, out, in\}
  - MCI: \{out\}
Interaction models – operational semantics

CN: \{\text{put, get}, \text{prod}, \text{cons}\}
MCI: \{\text{prod}, \text{cons}\}

\begin{itemize}
  \item put \rightarrow \text{prod}
  \item get \rightarrow \text{cons}
\end{itemize}
Interaction models - composition

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

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

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

prod \rightarrow put

get \rightarrow cons

CN: \{put, get\}, \{prod\}, \{cons\}
MCI: \{prod\}, \{cons\}
Interaction models – results [Goessler Sifakis 2003]

Incremental commutative composition encompassing blocking and non-blocking interaction
Overview

• Component-based construction – the notion of glue

• Interaction Models

• Priorities

• The BIP framework

• Discussion
Priorities

Restrict non-determinism by using (dynamic) priority rules

\[
\begin{align*}
\text{Priority rule} & \quad \text{Restricted guard } g_1' \\
\text{true } \rightarrow a_1 \preceq a_2 & \quad g_1' = g_1 \land \neg g_2 \\
C \rightarrow a_1 \preceq a_2 & \quad g_1' = g_1 \land \neg(C \land g_2)
\end{align*}
\]
Priorities

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

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

\[
g'_k = g_k \land \bigwedge C \rightarrow \ll \in pr \left( C \Rightarrow \bigwedge a_k \langle a_i \rightarrow g_i \rangle \right)
\]
Priorities - FIFO policy

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

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

\[ D_1 - t_1 \leq D_2 - t_2 \rightarrow b_2 \prec b_1 \quad D_2 - t_2 \leq D_1 - t_1 \rightarrow b_1 \prec b_2 \]

Diagram:

- sleep1 to wait1 (\(a_1, b_1, t_1 \leq D_1, e_1\))
- wait1 to use1 (\(b_1\))
- use1 to sleep1 (\(e_1, \#\))

- sleep2 to wait2 (\(a_2\))
- wait2 to use2 (\(b_2, t_2 \leq D_2\))
- use2 to sleep2 (\(e_2\))
Priorities - Composition

pr2
pr1

≠

pr1
pr2
Priorities – Composition (2)

We take:

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

= \[
\begin{array}{c}
\quad \\
\quad
\end{array}
\]

pr1⊕pr2 is the least priority containing pr1∪pr2

Results:

• The operation ⊕ is partial, associative and commutative
• pr1(pr2(B)) \neq pr2(pr1(B))
• pr1⊕pr2(B) refines pr1∪pr2(B) refines pr1(pr2(B))
• Priorities preserve deadlock-freedom
Priorities - mutual exclusion + FIFO

\[ t_1 \leq t_2 \rightarrow b_1 \prec b_2 \quad \text{true} \rightarrow b_1 \prec e_2 \]

\[ t_2 \leq t_1 \rightarrow b_2 \prec b_1 \quad \text{true} \rightarrow b_2 \prec e_1 \]

\[
\begin{array}{c}
\text{start } t_1 \\
\text{use1} \\
e_1 \\
\text{wait1} \quad b_1 \\
\text{sleep1} \\
a_1
\end{array}
\quad
\begin{array}{c}
\text{start } t_2 \\
\text{use2} \\
e_2 \\
\text{wait2} \quad b_2 \\
\text{sleep2} \\
a_2
\end{array}
\]
Priorities – mutual exclusion: example

Risk of deadlock: The composition is not a priority order!
Overview

• Component-based construction – the notion of glue

• Interaction Models

• Priorities

• The BIP framework

• Discussion
The BIP framework - fixed priority preemptive scheduling (1)

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

\[ \{ b_i, p_j \} \prec f_j, \{ r_i, p_j \} \prec f_j, \quad n \geq i > j \geq 1 \] (non pre-emption by lower pty tasks)

CN: \( \{ b_i, p_j \} \{ r_i, p_j \} \) for \( n \geq i, j \geq 1 \)

MCI: \( a_i, f_i, b_i \) for \( n \geq i \geq 1 \)
The BIP framework - fixed priority preemptive scheduling (2)

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

\[ p_i \langle f_j, if w_i or e'_i \rangle n \geq i > j \geq 1 \] (non pre-emption by lower pty tasks)

\[ \{b_i, r_i\} \leftrightarrow \{f_j, p_j\} \] \( n \geq i, j \geq 1 \) (Mutual exclusion)
The BIP framework – run to completion

i1 \langle \{o1,i2\} \langle \{o2,i3\} \langle o3

CN: \{o1,i2\}, \{o2,i3\}   MCI: \emptyset

Joseph Sifakis       Distributed Embedded Systems     Nov. 21-24 2005
The BIP framework - modulo-8 counter: atomic component

Zero

flip
X=1
Y:=0

mod2 counter

One

X:input

Y:output

Zero'

flip
X=1
Y:=1

One'

tick

tick

tick

tick
The BIP framework - modulo-8 counter: the model

tick \langle \text{flip}_0 \rangle, \text{tick} \langle \text{flip}_1 \rangle, \text{tick} \langle \text{flip}_2 \rangle

CN: tick={tick_0, tick_1, tick_2}
MCI: ∅
Transfer : X_1 := Y_0; X_2 := Y_1 ∧ Y_0

CN: tick_0, flip_0
MCI: flip_0

CN: tick_1, flip_1
MCI: flip_1

CN: tick_2, flip_2
MCI: flip_2
The BIP framework - The execution platform

Interaction model

Priorities

Execution kernel

Platform
The execution platform – the kernel

1. **Init**
   - Launch atom’s threads

2. **Loop**
   - Wait all atoms
   - Notify involved atoms

3. **Execute**
   - Choose among maximal
   - Execute chosen interaction transfer

4. **Choose**
   - Filter w.r.t. priorities

5. **Stable**
   - Compute legal interactions

6. **Ready**
   - Choose among maximal

Diagram:
- **Init** -> **Launch atom’s threads** -> **Loop**
- **Loop** -> **Wait all atoms** -> **Notify involved atoms**
- **Loop** -> **Choose**
- **Choose** -> **Execute chosen interaction transfer** -> **Execute**
- **Execute** -> **Choose among maximal** -> **Choose**
- **Choose** -> **Filter w.r.t. priorities** -> **Filter**
- **Filter** -> **Compute legal interactions** -> **Stable**
- **Stable** -> **Notify involved atoms** -> **Loop**
- **Loop** -> **Wait all atoms** -> **Loop**
The BIP framework - 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 $\{ (\alpha, …), (\alpha, …), … \}$
Overview

- Component-based construction – the notion of glue
- Interaction Models
- Property enforcement by controllers
- Priorities
- The BIP framework
- Discussion
Discussion - Summary

• 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 architecture
  ▪ Architecture is a first class entity
  ▪ Correctness-by-construction techniques for deadlock-freedom and liveness, based on sufficient conditions on architecture (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 – 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
A system is defined as a point of the 3-dimensional space.
Separation of concerns: any combination of coordinates defines a system.
Discussion – construction space

Non Separation of concerns for PTOLEMY
Discussion – construction space: property preservation

Deadlock-free

Invariant

Joseph Sifakis

Distributed Embedded Systems

Nov. 21-24 2005
Discussion - Computational vs. Analytic Approach

\[ y(t) = x(t-1) \]

\[ x(t) \rightarrow \text{Delay}(1) \rightarrow y(t) = x(t-1) \]

\[ y = 0 \quad y = 0 \]
\[ x \uparrow \quad x \downarrow \]
\[ \text{start t} \quad \text{start t} \]
\[ t = 1 \quad t = 1 \]
\[ y = 1 \quad y = 1 \]

x changes at most once within 1 time unit
## Discussion - Computational vs. Analytic Approach

<table>
<thead>
<tr>
<th></th>
<th>Computational</th>
<th>Analytic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non determinism</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Executable</td>
<td>Yes</td>
<td>Maybe</td>
</tr>
<tr>
<td>Probabilities</td>
<td>Badly</td>
<td>Yes</td>
</tr>
<tr>
<td>Analysis</td>
<td>Verification</td>
<td>Averages, Bounds</td>
</tr>
<tr>
<td></td>
<td>Bounds only for finite state systems</td>
<td></td>
</tr>
<tr>
<td>HW design</td>
<td>Maybe</td>
<td>Yes</td>
</tr>
<tr>
<td>SW design</td>
<td>Yes</td>
<td>Maybe</td>
</tr>
</tbody>
</table>
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 – 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