Component-based Construction of Heterogeneous Real-time Systems in BIP

FOSE 2010
Zurich, November 22-23, 2010

Joseph Sifakis
VERIMAG Laboratory

in collaboration with
A. Basu, S. Bensalem, M. Bozga, M. Jaber, J. Quilbeuf
Electronic components integrate **software and hardware** jointly and specifically designed to provide given functionalities, which are often **critical**.
Mixed SW/HW system design is different from pure SW design.

New trends break with traditional Computing Systems Engineering. It is hard to jointly meet technical requirements such as:

- **Reactivity**: responding within known and guaranteed delay
  Ex: flight controller

- **Autonomy**: provide continuous service without human intervention
  Ex: no manual start, optimal power management

- **Dependability**: guaranteed minimal service in any case
  Ex: attacks, hardware failures, software execution errors

- **Scalability**: at runtime or evolutionary growth (linear performance increase with resources)
  Ex: reconfiguration, scalable services

...and also take into account economic requirements for optimal cost/quality

**Technological challenge**: Capacity to build systems of guaranteed functionality and quality, at an acceptable cost.
## System Design – State of the Art

### TODAY
- We master – at a high cost – two types of systems which are difficult to integrate:
  - Safety and/or security critical systems of low complexity
    - Flight controller, smart card
  - Complex « best effort » systems
    - Telecommunication systems, web-based applications

### TOMORROW
- We need:
  - Affordable critical systems
    - Ex : transport, health, energy management
  - Successful integration of heterogeneous systems of systems
    - Internet of Things
    - Automated Highways
    - New generation air traffic control
    - "Ambient Intelligence"
Traditional systems engineering disciplines are based on solid theory for building artefacts with predictable behaviour over their life-time.

Computing systems engineering lacks similar constructivity results

- only partial answers to particular design problems
- predictability is hard to guarantee at design time
- *a posteriori* validation remains essential for ensuring correctness
Rigorous System Design
- Model-based Design
- Component-based Design
- Correct-by-construction Design

The BIP Component Framework
- The BIP Language
- SW Componentization
- Expressiveness
- Distributed Implementation

Discussion
Correctness

- This means that the designed system meets its requirements.
- Ensuring correctness requires that the design flow relies on the use of models with well-defined semantics.
- The models should consistently encompass system description at different levels of abstraction from application software to its implementation.
- Properties met at some step of the design flow, should be preserved in all the subsequent steps

Productivity

- using high level domain-specific languages for ease of expression and natural expression of data parallelism and functional parallelism
- allowing reuse of components and the development of component-based solutions
- tools for programming, validation and code generation
System Design – Essential Properties

Performance

- guaranteeing extra-functional properties regarding optimal resource management.
- resources such as memory, time and energy must be first class concepts encompassed by rigorous models.
- possibility to analyze and evaluate efficiency in using resources as early as possible along the design flow.

Parsimony

- Design choices are only implied by requirements - system designers privilege specific programming models or implementation principles that a priori exclude efficient solutions.
- Design choices such as
  - reducing parallelism (through mapping on the same processor)
  - reducing non determinism (through scheduling)
  - fixing parameters (precision, frequency, voltage)
- are resolved so as to determine appropriate implementations
A rigorous system design flow allows guaranteeing that the designed system meets some essential requirements. It is

- **model-based**: the software and system descriptions used along the design flow should be based on a single semantic model. This is essential for the overall coherency and efficiency.
  - Relate system descriptions and their properties for different abstraction levels and purposes (validation, performance evaluation, code generation).

- **component-based**: it provides primitives for building composite components as the composition of simpler heterogeneous components.

- **correct-by-construction**: it rely on tractable theory for guaranteeing at design time essential properties so as to avoid as much as possible monolithic *a posteriori* validation.
Rigorous System Design

- Model-based Design
- Component-based Design
- Correct-by-construction Design

The BIP Component Framework

- The BIP Language
- SW Componentization
- Expressiveness
- Distributed Implementation

Discussion
Model-based – Marry Physicality and Computation

**Processor constraints:**
- CPU speed
- memory
- power
- failure rates
- temperature

**Environment constraints:**
- Performance (deadlines, jitter, throughput)

**Computing:**
- algorithms
- protocols
- architectures

**EMBEDDED SYSTEM**
Model-based – Marry Physicality and Computation

Embedded SW Design cannot ignore HW design

**Processor constraints:**
- CPU speed
- memory
- power
- failure rates
- temperature

**Environment constraints:**
- Performance (deadlines, jitter, throughput)

**Computing:**
- algorithms
- protocols
- architectures
Model-based – Marry Physicality and Computation

**Processor constraints:**
- CPU speed
- memory
- power
- failure rates
- temperature

**Environment constraints:**
- Performance (deadlines, jitter, throughput)

**Computing:**
- algorithms
- protocols
- architectures

Embedded SW Design cannot ignore control design
Model-based – Marry Physicality and Computation

Embedded SW Design coherently integrates all these

**Processor constraints:**
- CPU speed
- memory
- power
- failure rates
- temperature

**Environment constraints:**
- Performance (deadlines, jitter, throughput)

**Computing:**
- algorithms
- protocols
- architectures

We need to revisit and revise computing to integrate methods from EE and Control
Physical Systems Engineering

Analytic Models
Component: transfer function
Composition: parallel
Connection: data flow

Computing Systems Engineering

Computational Models
Component: subroutine
Composition: sequential
Connection: control flow
Model-based – Marry Physicality and Computation

Matlab/Simulink Model
Model-based – Marry Physicality and Computation

UML Model (Rational Rose)

Start(H0_time) / begin
  clock.set(298900);
  H0.set(H0_time)
end

Wait_Ignition_Time

timeout(clock) / begin
  clock.set(TimeConstants_MS_100);
  current_is_ok:=EVBO.Open()
end

[ current_is_ok = true ]

Open_EVBO

timeout(clock) / begin
  current_is_ok:=EVVP.Open()
end

[ current_is_ok = true ]

Stop1

timeout(clock) / current_is_ok:=EVVP.Close()

Wait_Close_EVBO

Abort

Stop2

timeout(clock) / begin
  current_is_ok:=EVBO.Close();
  CyclicsAnomaly(); AcyclicsAnomaly(); Guidance_TaskAnomaly(); EAPIAnomaly(); Thrust_MonitorAnomaly()
end

Wait_Close_EVBO

[ current_is_ok = false ] / clock.reset()
Rigorous System Design
  - Model-based Design
  - Component-based Design
  - Correct-by-construction Design

The BIP Component Framework
  - The BIP Language
  - SW Componentization
  - Expressiveness
  - Distributed Implementation

Discussion
System designers deal with a large variety of components, each having different characteristics, from a large variety of viewpoints, each highlighting different dimensions of a system.
Component-based– Heterogeneity

Develop a **framework** for model-based and component-based design

- **expressive** enough to directly encompass heterogeneity of
  - Execution: synchronous and asynchronous components
  - Interaction: function call, broadcast, rendezvous, monitors
  - Abstraction levels: hardware, execution platform, application software

- using a **minimal set of constructs** and principles

- treating interaction and system architecture as **first class entities** that can be composed and analyzed - independently of the behavior of individual components

- providing automated support for efficient implementation on given platforms
Component-based– Heterogeneity

Thread-based programming

Actor-based programming

Software Engineering

Systems Engineering
Build a component $C$ satisfying a given property $P$, from

- $C_0$, a set of **atomic** components described by their behavior
- $\mathcal{G} = \{gl_1, \ldots, gl_i, \ldots\}$ a set of **glue operators** on components

- Move from frameworks based on single composition operators to frameworks based on families of composition operators
- Glue operators allow modeling coordination mechanisms such as such as protocols, schedulers, buses
Component-based – Glue Operators

Use operational semantics to define the meaning of a composite component – glue operators are “behavior transformers”

Glue Operators
- build interactions of composite components from the actions of the atomic components e.g. parallel composition operators
- can be specified by using a family of derivation rules (the Universal Glue)
Rigorous System Design
- Model-based Design
- Component-based Design
- Correct-by-construction Design

The BIP Component Framework
- The BIP Language
- SW Componentization
- Expressiveness
- Distributed Implementation

Discussion
Plasticity: Glue is a first class concept independent from behavior

1. Decomposition

2. Composition
Compositionality: Build correct systems from correct components: rules for proving global properties from properties of individual components

$$c_i \text{ sat } P_i \implies \forall gl \exists \text{ sat } gl(P_1, \ldots, P_n)$$

We need compositionality results for the preservation of progress properties such as deadlock-freedom and liveness as well as extra-functional properties.
Composability: Essential properties of components are preserved when they are integrated.

Property stability phenomena are poorly understood. We need composability results e.g. non interaction of features in middleware, composability of scheduling algorithms, of Web services, of aspects.
Rigorous System Design
- Model-based Design
- Component-based Design
- Correct-by-construction Design

The BIP Component Framework
- The BIP Language
- SW Componentization
- Expressiveness
- Distributed Implementation

Discussion
Layered component model

Priorities (conflict resolution)

Interactions (collaboration)

Composition operation parameterized by glue IN12, PR12
BIP – Connectors

Express interactions by combining two protocols: rendezvous and broadcast

- A **connector** is a set of ports that 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.

```
s + sr2 + sr3 +sr2r3
```
Atomic Broadcast: \( a+abc \)

Causality chain: \( a+ab+abc+abcd \)
Verify **global deadlock-freedom** of a system by separate analysis of the components and of the architecture.

Potential deadlock

\[ D = \text{en}(p1) \land \neg \text{en}(p2) \land \text{en}(q2) \land \neg \text{en}(q1) \]

Potential deadlock

\[ D = \text{en}(p1) \land \neg \text{en}(p2) \land \text{en}(q2) \land \neg \text{en}(q3) \land \text{en}(r3) \land \neg \text{en}(r1) \]
Method:
Eliminate potential deadlocks $D$ by computing compositionally global invariants $\chi$ such that $\chi \land D = \text{false}$

\[
B_1 \models \square \phi_1 \quad B_2 \models \square \phi_2 \quad \psi \in \mathcal{I}_I(\gamma(B_1, B_2), \phi_1, \phi_2) \quad \phi_1 \land \phi_2 \land \psi \Rightarrow \chi
\]

\[
\gamma(B_1, B_2) \models \square \chi
\]
BIP – Compositional Verification: D-Finder

The graph shows the verification time (in minutes) for different verification methods as the size of a gas station increases. The size is represented as $N \times 10$ pumps and $100$ customers. The verification methods compared are:

- **Compositional verification**
- **Incremental compositional verification**
- **Monolithic verification: NuSMV**

The graph illustrates how the verification time increases with the size of the gas station, highlighting the efficiency of compositional verification methods compared to monolithic verification.
Rigorous System Design
- Model-based Design
- Component-based Design
- Correct-by-construction Design

The BIP Component Framework
- The BIP Language
  - SW Componentization
    - Expressiveness
    - Distributed Implementation

Discussion
Functional Level ::= Module^∗

Module ::= Service^∗ . Poster

Service ::= Service Controller . Service Task

Service Controller ::= Event Triggered Controller | Cyclic Controller

Cyclic Controller ::= Event Triggered Controller . Cyclic Trigger

Service Task ::= Timed Task | Untimed Task
Event Triggered Controller

**Idle**: the Service is idle

**Ready**: checks the possibility for starting a new Task of the Service

**Exec**: execution of the Task of the Service

**Abort**: Service is aborted
Cyclic Controller ::= Event Triggered Controller . Cyclic Trigger

The Cyclic Trigger starts the Event Triggered Controller every period p.
SW Componentization

Untimed Task

Triggered by request

\[ \text{status} == 1 \]: Task successfully executed

\[ \text{status} == 0 \]: Task aborted
Timed Task
- Obtained from Untimed Task
- Execution time in $[t_1,t_2]$
Untimed Event Triggered Service ::= Event Triggered Controller. Untimed Task

Timed Event Triggered Service ::= Event Triggered Controller. Timed Task

Cyclic Service ::= Cyclic Controller . Timed Task
Module Pom reads and integrates data to provide an estimate of the position of the robot

\[ \text{Pom} ::= \text{SetModel. AddME. SetRefME. Run. SetPos. Poster} \]
## SW Componentization – Results

<table>
<thead>
<tr>
<th>Modules</th>
<th>Components</th>
<th>Locations</th>
<th>Interactions</th>
<th>States</th>
<th>LOC</th>
<th>Min.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LaserRF</td>
<td>43</td>
<td>213</td>
<td>202</td>
<td>$2^{20}x3^{29}x34$</td>
<td>4353</td>
<td>1:22</td>
</tr>
<tr>
<td>Aspect</td>
<td>29</td>
<td>160</td>
<td>117</td>
<td>$2^{17}x3^{23}$</td>
<td>3029</td>
<td>0:39</td>
</tr>
<tr>
<td>NDD</td>
<td>27</td>
<td>152</td>
<td>117</td>
<td>$2^{22}x3^{14}x5$</td>
<td>4013</td>
<td>8:16</td>
</tr>
<tr>
<td>RFLEX</td>
<td>56</td>
<td>308</td>
<td>227</td>
<td>$2^{34}x3^{35}x1045$</td>
<td>8244</td>
<td>9:39</td>
</tr>
<tr>
<td>Antenna</td>
<td>20</td>
<td>97</td>
<td>73</td>
<td>$2^{12}x3^{9}x13$</td>
<td>1645</td>
<td>0:14</td>
</tr>
<tr>
<td>Battery</td>
<td>30</td>
<td>176</td>
<td>138</td>
<td>$2^{22}x3^{17}x5$</td>
<td>3898</td>
<td>0:26</td>
</tr>
<tr>
<td>Heating</td>
<td>26</td>
<td>149</td>
<td>116</td>
<td>$2^{17}x3^{14}x145$</td>
<td>2453</td>
<td>0:17</td>
</tr>
<tr>
<td>PTU</td>
<td>37</td>
<td>174</td>
<td>151</td>
<td>$2^{19}x3^{22}x35$</td>
<td>8669</td>
<td>0:59</td>
</tr>
<tr>
<td>Hueblob</td>
<td>28</td>
<td>187</td>
<td>156</td>
<td>$2^{12}x3^{10}x35$</td>
<td>3170</td>
<td>5:42</td>
</tr>
<tr>
<td>VIAM</td>
<td>41</td>
<td>227</td>
<td>231</td>
<td>$2^{10}x3^{6}x665$</td>
<td>5099</td>
<td>4:14</td>
</tr>
<tr>
<td>DTM</td>
<td>34</td>
<td>198</td>
<td>201</td>
<td>$2^{28}x3^{20}x95$</td>
<td>4160</td>
<td>13:42</td>
</tr>
<tr>
<td>Stereo</td>
<td>33</td>
<td>196</td>
<td>199</td>
<td>$2^{27}x3^{20}x95$</td>
<td>3591</td>
<td>13:20</td>
</tr>
<tr>
<td>P3D</td>
<td>50</td>
<td>254</td>
<td>219</td>
<td>$2^{13}x3^{5}x5^{4}x629$</td>
<td>6322</td>
<td>3:51</td>
</tr>
<tr>
<td>LaserRF+Aspect+NDD</td>
<td>97</td>
<td>523</td>
<td>438</td>
<td>$2^{58}x3^{66}x85$</td>
<td>11395</td>
<td>40:57</td>
</tr>
<tr>
<td>NDD+RFLEX</td>
<td>82</td>
<td>459</td>
<td>344</td>
<td>$2^{56}x3^{49}x5^{2}x209$</td>
<td>12257</td>
<td>73:43</td>
</tr>
</tbody>
</table>
Rigorous System Design
- Model-based Design
- Component-based Design
- Correct-by-construction Design

The BIP Component Framework
- The BIP Language
- SW Componentization
- Expressiveness
- Distributed Implementation

Discussion
Expressiveness for Component-based Systems

- Different from the usual notion of expressiveness!
- Based on strict separation between glue and behavior

Given two glues $G_1$, $G_2$

$G_2$ is strongly more expressive than $G_1$

if for any component built by using $G_1$ and $C_0$ there exists an equivalent component built by using $G_2$ and $C_0$

$\cong$
Given two glues $G_1, G_2$

$G_2$ is weakly more expressive than $G_1$

if for any component built by using $G_1$ and $C_0$

there exists an equivalent component built by using $G_2$ and $C_0 \cup C$

where $C$ is a finite set of coordination behaviors.
Expressiveness for Component-based Systems

SCCS \text{\ll} IM \text{\ll} BIP

CCS \text{\ll} IM \text{\ll} BIP

CSP \text{\ll} IM \text{\ll} BIP

Universal Glue

[Bliudze&Sifakis, Concur 08]
- Rigorous System Design
  - Model-based Design
  - Component-based Design
  - Correct-by-construction Design

- The BIP Component Framework
  - The BIP Language
  - SW Componentization
  - Expressiveness
  - Distributed Implementation

- Discussion
BIP is based on:
- Global state semantics, defined by operational semantics rules, implemented by the BIP Engine
- Atomic multiparty interactions, e.g. by rendezvous or broadcast

Translate BIP models into observationally equivalent S/R-BIP models
- Point to point communication by asynchronous message passing
- Replace the BIP Engine by a set of Engines executing subsets of interactions
- Collection of independent components intrinsically concurrent - No global state
  - Atomicity of transitions is broken by separating interaction from internal computation
- Translation is correct-by-construction
Before reaching a ready state, the set of the enabled ports is sent to the Engine.

From a ready state, await notification from the Engine indicating the selected port.
Distributed Implementation – Multiple Engines

BIP Model

Dispatch interactions across *multiple* engines!
Distributed Implementation – 3-Layer Architecture

[\alpha_1 \alpha_2] [\alpha_3 \alpha_4]

\begin{align*}
\text{C1} & \quad \text{C2} & \quad \text{C3} & \quad \text{C4} & \quad \text{C5} & \quad \text{C6}
\end{align*}

\begin{align*}
\alpha_1 & \quad \alpha_2 & \quad \alpha_3 & \quad \alpha_4
\end{align*}

Conflict Resolution Protocol
- ok
- fail
- reserve

Interaction Protocol
- \alpha_1
- \alpha_2

Conflict Resolution Protocol
- ok
- fail
- reserve

Interaction Protocol
- \alpha_3
- \alpha_4

CR Protocol Model

Protocol
- C1
- C2
- C3
- C4
- C5
- C6
Distributed Implementation – Centralized CRP

Centralized Conflict Resolution Protocol

α1 Interaction Protocol α2

reserve2 ok2 fail2

α3 Interaction Protocol α4

reserve3 ok3 fail3 reserve4 ok4 fail4

C1’ C2’ C3’ C4’ C5’ C6’
Distributed Implementation – Token Ring CRP

Alpha 1

Reserve 2

Ok 2

Fail 2

Alpha 2

Reserve 3

Ok 3

Fail 3

Alpha 3

Reserve 4

Ok 4

Fail 4

Alpha 4

C1' C2' C3' C4' C5' C6'
Distributed Implementation – Detailed Design Flow

Conflict Resolution Protocol

Partitioning of Interactions

Partitioning of Components

Sockets/C++ + Code

Code Generator

MPI/C++ Code

Component 1  129.2.2.1  Core1
Component 2  129.2.2.1  Core3
Component 3  129.2.2.1  Core2
Rigorous System Design
- Model-based Design
- Component-based Design
- Correct-by-construction Design

The BIP Component Framework
- The BIP Language
- SW Componentization
- Expressiveness
- Distributed Implementation

Discussion
Discussion

Component framework encompassing heterogeneous composition

- Clear separation between behavior and architecture (Interaction + Priority) involving a minimal set of constructs and principles
- **Expressiveness**: BIP is as expressive as the universal glue

Rigorous design flow

- Correct-by-construction source-to-source transformations
- Verification techniques based on compositionality, composability and incrementality
- Global analysis techniques jointly taking into account: 1) interaction partitioning; 2) component partitioning; 3) HW resources

Applications:

- software componentization
- programming multicore systems
- complex systems modeling e.g. IMA
A glue operator is a set of derivation rules of the form

\[ \{ q_i \rightarrow a_i \rightarrow_i q'_i \}_{i \in I} \quad \{ \neg q_k \rightarrow a_k \rightarrow_k \}_{k \in K} \]

\[ (q_1, \ldots, q_n) - a \rightarrow (q'_1, \ldots, q'_n) \]

- \( I, K \subseteq \{1, \ldots, n\} \), \( I \neq \emptyset \), \( K \cap I = \emptyset \)
- \( a = \bigcup_{i \in I} a_i \) is an interaction
- \( q'_i = q_i \) for \( i \notin I \)

Notice that, non deterministic choice and sequential composition are not glue operators

A glue is a set of glue operators
Rigorous System Design Flow in BIP

- **Application SW**
  - Translation
  - Code Generation
  - Implementation
  - Platform

- **HW Infrastructure**
  - Transformation
  - System model in BIP
  - Protocols

- **Mapping**
  - Transformation
  - System model in S/R-BIP