Rigorous Component-based System Design Using the BIP Framework

Saddek Bensalem

Join work with
A. Basu, M. Bozga, P. Bougos, J. Sifakis

VERIMAG Laboratory (Grenoble, France)

Fifth Annual Layered Assurance Workshop

December 5-6, 2011 Orlando, Florida
• Introduction

• The BIP Framework
  – Basic Concepts and Results
  – The BIP Language and the associated tools

• The Rigorous System Design Flow

• Discussion
We master, at high cost:
- critical systems of low to medium complexity
  - ex: flight controllers
- complex best effort systems
  - ex: telecommunication systems

We need
- affordable critical systems
  - ex: active safety, health, robots
- integration of systems of systems
  - ex: internet of things, smart grids, ambient intelligence

Today

A long way to go ...

Complexity: mainly for building systems by integration of existing components

Design Approaches:
- empirical and based on the expertise of the teams
- reuse/extend/improve solutions that have been proven efficient and robust

Lack of constructivity results:
correctness cannot be guaranteed by design, validation is mandatory

Tomorrow
System design is the process leading to a mixed software-hardware system meeting given requirements. The expected behavior of the system to be designed with respect to its potential users and its environment. Executable platform-independent model meeting the requirements. System composed of HW and SW – the HW platform may be given.
System Design vs Software Design

**Programs and Algorithms**
- Terminating
- Deterministic
- Behaviour: relations independent from physical resources needed for their execution
- Correctness independent in the dynamic characteristics of the execution platform

**Systems**
- Non-terminating
- Non-deterministic
- Behaviour: relations between histories of inputs and histories of outputs
- Correctness dependent on the dynamic characteristics of the execution platform
Trends in System Design

ES must jointly meet technical requirements

- **Reactivity:** responding within a known and bounded delay.
- **Autonomy:** providing continuous service without human intervention.
- **Dependability:** invulnerability to threats including attacks, hardware failures, software execution errors
- **Scalability:** performance increase is commensurable with the increase of resources

In addition

- ES must meet requirements for **optimal/quality** as they are integrated in **mass-market** products
What is needed?

Foundations for a rigorous system design
Rigorous System Design

Three grand Challenges

1. Marrying Physicality and Computation
   • theory and models encompassing continuous and discrete dynamic to predict the global behavior of a system interacting with its physical environment.

2. Component based Design
   • theory, models and tools for the cost-effective building of complex systems by assembling heterogeneous components

3. Adaptivity
   • systems must provide a service meeting given requirements in interaction with uncertain environments.
Rigorous System Design – Essential Properties: Productivity

Efficiency of the design process

Skills

Components

Tools
Rigorous System Design – Essential Properties: Performance

- Languages for describing feasible (correct) design solutions

- Optimal use of resources through design space exploration to resolve choices such as
  - reducing parallelism (through mapping on the same processor)
  - reducing non-determinism (through scheduling)
  - fixing parameters (quality, frequency, voltage)
System Design - Essential Properties: Correctness

- Avoid design errors or eliminate them as early as possible
- Incremental construction and validation – scalability
- Traceability between application software and implementation
Our Approach

Develop the BIP framework:
• model-based and component-based design
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

$$gl(C_1, C_2, \ldots, C_n) \mid A_1 \text{ refines } C_1$$
Develop the BIP framework:
• model-based and component-based design
• expressive enough to encompass heterogeneity of
  – execution: synchronous and asynchronous components
  – interaction: function call, broadcast, rendez-vous
  – abstraction levels: hardware, middleware, application software
Our Approach

Develop the BIP framework:

• model-based and component-based design
• expressive enough to encompass heterogeneity of
  – execution: synchronous and asynchronous components
  – interaction: function call, broadcast, rendez-vous
  – abstraction levels: hardware, middleware, application software
• using a minimal set of constructs and principles for guaranteeing correctness by construction.
Component-Based Construction: Incremental Description

1. Decomposition

\[ gl \cong D_1 \oplus D_2 \]

2. Flattening

Flattening can be achieved by using a (partial) associative operation \( \oplus \) on GL
Building correct systems
from correct components

$c_i \text{ sat } P_i$ implies $\forall gl \exists gl$ gl

$c_1 \cdots c_n$ sat $\tilde{gl}(P_1, \ldots, P_n)$

Component-Based Construction: Constructivity – Compositionality
Make the new without breaking the old: Rules guaranteeing non interference of solutions

Property stability phenomena are poorly understood. We need composability results e.g. feature interaction in middleware, composability of scheduling algorithms, theory for reconfigurable systems.
Our Approach

Develop the BIP component framework:

• model-based and component-based design
• expressive enough to encompass heterogeneity of
  – execution: synchronous and asynchronous components
  – interaction: function call, broadcast, rendez-vous
  – abstraction levels: hardware, middleware, application software
• using a minimal set of constructs and principles for guaranteeing correctness by construction.
• treating interaction and system architectures as first class entities that can be composed and analyzed independently of the behavior of individual components
Our Approach

Develop the BIP component framework:

• model-based and component-based design
• expressive enough to encompass heterogeneity of
  – execution: synchronous and asynchronous components
  – interaction: function call, broadcast, rendez-vous
  – abstraction levels: hardware, middleware, application software
• using a minimal set of constructs and principles for guaranteeing correctness by construction
• treating interaction and system architectures 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
• providing automated support for validation and performance analysis
• Introduction

• The BIP Framework
  – Basic Concepts and Results
  – The BIP Language and the associated tools

• The Rigorous System Design Flow

• Discussion
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 C code)
The BIP Framework: Interaction Model

- 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{tick}_1, \text{tick}_2, \text{tick}_3\} \{\text{out}_1\} \{\text{out}_1, \text{in}_2\} \{\text{out}_1, \text{in}_3\} \{\text{out}_1, \text{in}_2, \text{in}_3\}
The BIP Framework: Interaction Model
BIP Construction Space

- **separation of concerns:** between behavior and architecture (interaction and priority)
- **semantic unification:** heterogeneous components can be unified through transformation in the construction space
- **correctness by construction:** basis for study of preservation of properties under architecture or behavior transformations
• Introduction

• The BIP Framework
  – Basic Concepts and Results
  – The BIP Language and the associated tools

• The Rigorous System Design Flow

• Discussion
The BIP Framework: An example

Priorities
Glue

Interactions
Glue

Behavior

\[ z_4 > 0 \]

\[ p_{123} < r_3 r_4 \]

\[ u := \max(x_1, x_2) \]

\[ u, x_3 := v \]

\[ y_1 := x_1 / 2 \]

\[ x_1 \text{++} \]

\[ y_1 \]

\[ q_1 \]

\[ p_1 \]

\[ p_2 \]

\[ p_3 \]

\[ p_4 \]

\[ x_4 \]

\[ y_2 := f_2(x_2) \]

\[ q_2 \]

\[ r_3 \]

\[ r_4 \]

\[ z_3 \]

\[ z_4 \]

\[ [y_1 < y_2] \]

\[ q_{123} \]

\[ r_{34} \]

\[ y_1 \]

\[ q_1 \]

\[ q_2 \]

\[ q_3 \]

\[ q_4 \]

\[ v \]

\[ \uparrow u := \max(x_1, x_2) \]

\[ \downarrow u, x_3 := v \]

\[ \uparrow v := \max(u, x_3) \]

\[ x_1, x_2 := u \]

\[ x_1, x_2 := u \]

\[ x_3 \]

\[ x_4 \]

\[ g_3(x_3) \]

\[ \text{Behavior} \]
The BIP Language

// atomic component definition

atomic type Atom(int p, int q, ...
  data int x, y, z, ...
  data DataType u, v, w, ...
  port MyPort p1(x)
  port TypePort2 p2(y, u)

place s1, s2, s3, s4, ...

initial to s1
do { /* initialization code */}
on p1 from s1 to s2
  provided guard1
do { /* transition code */}
on p2 from s2 to s3
  provided x < y
do { /* plain C code */}
export port MyPort p1 is r1
end

// connector type definition

connector type Bus (PortType1 p1, PortType2 p2)
define port-expression
data int y
...on interaction1 provided guard
down { /* interaction code */}
...on p1 p2 provided p1.x > 0
  up { y = p1.x + p2.x }  
down { p1.x = p2.x = y; #}
...export port PortType p0(y)
end

// compound component type definition

compound type Compo(int p, ...
compound component CompType_1 c1(p, ...)
  component CompType_2 c2(p, ...
  component CompType_n cn

compound component CompType_2 c2(p, ...
  connector ConType_1 x1( c1.p, ...
  connector ConType_k xk( x1.p0, cn.r )

priority pri01
  provided guard
  xi:interaction1 < xj:interaction2
...export port PortType1 c1.p is p
export port PortTypek xk.p0 is q
...end
Sequential Implementation

• The reference implementation for BIP models
• Separate compilation of component’s code and coordination code

• The sequential engine runs one execution trace according to the BIP semantics
• The sequential engine provides extra-functionality for run-time verification and model-checking
Sequential Implementation

• Execution of the Engine

Diagram:
- **init**
  - execute initial transitions
- **stable**
  - compute feasible interactions
  - execute interacting transitions
- **feasible**
  - filter using priorities
  - choose a maximal interaction
- **choose**
  - execute chosen interaction data transfer
- **execute**
  - execute transitions
The BIP Toolbox
• Introduction

• The BIP Framework
  – Basic Concepts and Results
  – The BIP Language and the associated tools

• The Rigorous System Design Flow

• Discussion
BIP System Design

- Application Software (Programming Models)
  - Translation
  - Software Model - BIP
    - Functional Validation
      - Invariant Generation
      - Deadlock Detection
      - Functional Simulation
    - Performance Analysis
      - Timed Simulation
    - Extended System Model - BIP
      - Profiling/Calibration
  - System Model - BIP
    - Refines
    - Refines
      - Model Transformation (I)
      - Model Transformation (II)
      - SR Software Model - BIP
      - Code Generation
        - Deployed Software
          - Middleware
            - (Multi-Core) Platform

Use BIP as a unifying semantics model for various programming models

Translations defined so far:
- (Discretized) Timed and Hybrid Systems
- Synchronous Systems (Lustre, MATLAB/Simulink, Scicos, ...)
- Architecture Description Languages (AADL)
- Domain Specific Languages and MoCs
  - Autonomous Robotic Applications (GeNoM)
  - Wireless Sensor Network Applications (TinyOs + nesC)
  - Process Networks in the Distributed Operation Layer (DOL)
- ...

Systematic approach based on two level translation into BIP:
- structural translation of the language constructs, the programmers view
- structural translation of the language operational semantics, the execution model view
A framework from ETHZ for programming parallel applications and specifying their mapping onto multicore architectures.

**Application model:**
- Process network of sw-processes, sw-channels, and connections (XML).
- Process behavior (C file per process).

**Architecture model:**
- Computation resources interconnected according to communication paths (XML description).
- Communication paths specify communication between CPU's using a common memory.

**Mapping:**
- Allocation of sw-process to hw-processor, sw-channels to hw-memories (XML description).

**DOL (Distributed Operation Layer) generator**

```
<process name="square">
  <port type="input" name="1"/>
  <port type="output" name="2"/>
  <source type="c" location="square.c"/>
</process>
```

```
<!-- sw_channels -->
<sw_channel type="fifo" size="10" name="C1">
  <port type="input" name="0"/>
  <port type="output" name="1"/>
</sw_channel>
```

```
<!-- arm core 1 -->
<processor name="ARM1" type="RISC">
  <configuration name="memory" value="pm1"/>
</processor>
```

```
<!-- distributed external memory -->
<memory name="sh_mem" type="RAM">
  <configuration name="cycles" value="1"/>
  <configuration name="cycles" value="6"/>
</memory>
```

```
<!-- bus -->
<hw_channel name="ahb" type="BUS">
  <configuration name="frequency" value="100000000"/>
  <configuration name="bytespercyle" value="4"/>
</hw_channel>
```

```
<!-- process bindings -->
<binding name="binding_generator" xsi:type="computation">
  <process name="generator"/>
  <processor name="ARM1"/>
</binding>
```

```
<!-- channel bindings -->
<binding name="binding_fifoC1" xsi:type="communication">
  <sw_channel name="C1"/>
  <writepath name="pm1ahbsh_mem"/>
  <readpath name="sh_memahbpm2"/>
</binding>
```

```
<!-- connection -->
<connection name="g-c">
  <origin name="generator">
    <port name="1"/>
  </origin>
  <target name="C1">
    <port name="0"/>
  </target>
</connection>
```
Every process and every SW-channel are independently translated to atomic components in BIP.

Connectors are generated from connections of the process network.

---

Generation of Application SW Model

```c
#define IN 1
#define OUT 2
typedef struct _local_states {
  int index;
  int len;
} Square_state;
void square_init(DOLProcess *p) {
  p->local->index = 0;
  p->local->len = LENGTH;
}
int square_fire(DOLProcess *p) {
  float I;
  if (p->local->index < p->local->len) {
    DOL_read((void*)IN, &i, sizeof(float), p);
    i = i * i;
    DOL_write((void*)OUT, &i, sizeof(float), p);
    p->local->index++;
  }
  if (p->local->index >= p->local->len) {
    DOL_detach(p);
    return -1;
  }
  return 0;
}
```

---

```
var: index, len, i, address, size;
recv [count<N]
buff[i]=x;count++; i=(i+1)%N;

send [count>0]
y=buff[j];count--; j=(j+1)%N;
```

---

```
IN
recv

OUT
send
```

---

```
IN
recv

OUT
send
```

---

```
IN
recv
```
Construction of HW Template

- **Collection of HW-processor, memory and bus components connected as defined in the architecture**
  - HW-processor and memory are placeholder
  - uses HW component library
Mapping: Fill up the HW Templates

- Transformation on sw model:
  - Splitting FIFO channel
  - Breaking atomic read/write
  - Adding interactions with CPU-Scheduler
  - FIFO buffers mapped to memory
- Transformations fully preserve functional behavior
- Uses HdS component library
System Model Construction

• System model generated by applying a fixed number of transformations on the software model
  – splitting software channels
  – breaking atomicity of read/write operations in processes
  – inserting HdS components
  – ...

• Transformations fully preserve functional behavior
  – ensure correctness-by-construction!

• No deadlocks are introduced

• Using a given set of BIP library components
  (characterized by the HW architecture, OS)
BIP System Design

Application Software (Programming Models)

Translation

Software Model - BIP

Functional Validation
- Invariant Generation
- Deadlock Detection,
- Functional Simulation,
...

Model Transformation (I)

System Model - BIP

Profileing/
Calibration

Extend
System Model - BIP

Mapping

Model Transformation (II)

SR
Software Model - BIP

Code generation

Deployed Software

Middleware

(Multi-Core) Platform

refines

updates

Performance Analysis
- Timed Simulation
...

refines
Compositional Verification

- Compositional rule for proving state invariants:

\[
\begin{align*}
(B_i |= \Box \Phi_i)_{i=1,n} & \quad \Psi \in \forall \gamma(B_1, \ldots, B_n), \Phi_1, \ldots, \Phi_n \\
\wedge_{i=1,n} \Phi_i \wedge \Psi & \Rightarrow P \\
\gamma(B_1, \ldots, B_n) |\Box P
\end{align*}
\]

- Combine two categories of particular invariants:
  - Component Invariants \((\Phi_i)_{i=1,n}\)
  - Interaction Invariants \(\Psi\)

automatically generated from BIP models!
Compositional Verification

- Compositional rule for proving state invariants:

- Combine two categories of particular invariants:
  - Component Invariants \((\Phi_i)_{i=1,n}\)
  - Interaction Invariants \(\Psi\)

automatically generated from BIP models!
Compositional Verification

- Compositional rule for proving safety properties:

\[
\begin{align*}
( B_i \models □ \Phi_i )_{i=1,n} & \quad \Psi \in \mathbb{M}( γ(B_1, ..., B_n), Φ_1, ..., Φ_n ) \\
\land_{i=1,n} Φ_i \land Ψ & ⇒ P
\end{align*}
\]

\[γ(B_1, ..., B_n) \models □ P\]

Component Invariants:

- over-approximations of the set of reachable states of atomic components
- computed using static analysis of behavior
Compositional Verification

• Compositional rule for proving safety properties:

\[
\begin{align*}
(B_i \models \Box \Phi_i)_{i=1,n} & \quad \Psi \in \mathcal{I}\mathcal{I}(\gamma(B_1, \ldots, B_n), \Phi_1, \ldots, \Phi_n) \\
\wedge_{i=1,n} \Phi_i \wedge \Psi & \Rightarrow P \\
\gamma(B_1, \ldots, B_n) \models \Box P
\end{align*}
\]

Interaction Invariants:

- characterize constraints on the global state space induced by synchronizations between components.

- computed by static analysis of interaction structures
Implementation: D-Finder
## Results on deadlock-freedom checking of all the modules

<table>
<thead>
<tr>
<th>Modules</th>
<th>Components</th>
<th>Locations</th>
<th>Interactions</th>
<th>States</th>
<th>LOC</th>
<th>Minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LaserRF</td>
<td>43</td>
<td>213</td>
<td>202</td>
<td>$2^{20} \times 3^{29} \times 34$</td>
<td>4353</td>
<td>1:22</td>
</tr>
<tr>
<td>Aspect</td>
<td>29</td>
<td>160</td>
<td>117</td>
<td>$2^{17} \times 3^{23}$</td>
<td>3029</td>
<td>0:39</td>
</tr>
<tr>
<td>NDD</td>
<td>27</td>
<td>152</td>
<td>117</td>
<td>$2^{22} \times 3^{14} \times 5$</td>
<td>4013</td>
<td>8:16</td>
</tr>
<tr>
<td>RFLEX</td>
<td>56</td>
<td>308</td>
<td>227</td>
<td>$2^{34} \times 3^{35} \times 1045$</td>
<td>8244</td>
<td>9:39</td>
</tr>
<tr>
<td>Antenna</td>
<td>20</td>
<td>97</td>
<td>73</td>
<td>$2^{12} \times 3^{9} \times 13$</td>
<td>1645</td>
<td>0:14</td>
</tr>
<tr>
<td>Battery</td>
<td>30</td>
<td>176</td>
<td>138</td>
<td>$2^{22} \times 3^{17} \times 5$</td>
<td>3898</td>
<td>0:26</td>
</tr>
<tr>
<td>Heating</td>
<td>26</td>
<td>149</td>
<td>116</td>
<td>$2^{17} \times 3^{14} \times 145$</td>
<td>2453</td>
<td>0:17</td>
</tr>
<tr>
<td>PTU</td>
<td>37</td>
<td>174</td>
<td>151</td>
<td>$2^{19} \times 3^{22} \times 35$</td>
<td>8669</td>
<td>0:59</td>
</tr>
<tr>
<td>Hueblob</td>
<td>28</td>
<td>187</td>
<td>156</td>
<td>$2^{12} \times 3^{10} \times 35$</td>
<td>3170</td>
<td>5:42</td>
</tr>
<tr>
<td>VIAM</td>
<td>41</td>
<td>227</td>
<td>231</td>
<td>$2^{10} \times 3^{6} \times 665$</td>
<td>5099</td>
<td>4:14</td>
</tr>
<tr>
<td>DTM</td>
<td>34</td>
<td>198</td>
<td>201</td>
<td>$2^{28} \times 3^{20} \times 95$</td>
<td>4160</td>
<td>13:42</td>
</tr>
<tr>
<td>Stereo</td>
<td>33</td>
<td>196</td>
<td>199</td>
<td>$2^{27} \times 3^{20} \times 95$</td>
<td>3591</td>
<td>13:20</td>
</tr>
<tr>
<td>P3D</td>
<td>50</td>
<td>254</td>
<td>219</td>
<td>$2^{13} \times 3^{5} \times 5^{4} \times 629$</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>$2^{58} \times 3^{66} \times 85$</td>
<td>11395</td>
<td>40:57</td>
</tr>
<tr>
<td>NDD+RFLEX</td>
<td>82</td>
<td>459</td>
<td>344</td>
<td>$2^{56} \times 3^{49} \times 5^{2} \times 209$</td>
<td>12257</td>
<td>73:43</td>
</tr>
</tbody>
</table>
Distributed Implementation

Requirements

- produce efficient decentralized execution models
- allow for concurrent execution of interactions and internal computation of components
- collection of atomic processes/threads intrinsically concurrent – no global state
- point-to-point communication by asynchronous message passing
- ensure correctness-by-construction, that is, the initial model is equivalent to the implementation
Centralized Implementation

Centralized Implementation: one Engine play \textit{all} interactions!
Decentralized Implementation

Decentralized Implementation: dispatch interactions across *multiple* engines!
I1 and I2 are using both sides ports of a choice in a component.

I1 and I2 are conflicting (I1 # I2).

I1 and I2 share a common port.
Distributed Engines Conflict-Free by Construction, by grouping interactions according to the transitive closure of the conflict relation.

1st sol: Conflict-Free Distributed Engines

I1 # I2 # I3
I4 # I5 # I6
Send/Receive BIP

Conflict Resolution Protocol

Interaction Protocol

C1 C2 C3 C4 C5 C6

Protocol

BIP
BIP System Design

Application Software (Programming Models)

Translation

Software Model - BIP

Functional Validation
Invariant Generation
Deadlock Detection,
Functional Simulation,
...

System Model - BIP

Model Transformation (I)

Profiling/Calibration

Extend System Model - BIP

Profiling/Calibration

Mapping

Model Transformation (II)

Code generation

Deployed Software

Middleware

(Multi-Core) Platform
Code Generation: Overview

Abstract System Model - (BIP)

Code Generator

Application Mapping

Functional Code (e.g., C)

Glue Code (e.g., C)

Runtime

Target platform compilation tool

binary image

P2012/MPARM

• Application task: data + thread routine
• Communication: API calls provided by runtime

• Main application routine
• Deployment
  - threads to cores
  - data to memories
• Data allocation: thread stacks; FIFO queues

• API
  - Thread Management
  - Memory allocation
  - Communication
  - Synchronization
Example: P2012 Code Generation

**Functional Code**
- process.map
- channel.map

**Glue Code**
- (Resource Allocation & Thread Creation & Deployment)

**Runtime**
- NPL

```c
#define NUM_PROCESS 14
void* genp_ins_execute(void*);
void* ndpf0_ins_execute(void*);
void* ndpf1_ins_execute(void*);
...
process_map_t process_map[NUM_PROCESS] = {
  { "genp", 0, 0, genp_ins_execute },
  { "ndpf0", 0, 1, ndpf0_ins_execute },
  { "ndpf1", 0, 2, ndpf1_ins_execute },
  ...
};

#define NUM_CHANNEL 24
float FIFO_genp_ndpf0_buffer[45000] L3_SHARED;
float FIFO_genp_ndpf1_buffer[30000] L3_SHARED;
...
qu_handle_t q_handle_FIFO_genp_ndpf0 L3_SHARED;
qu_handle_t q_handle_FIFO_genp_ndpf1 L3_SHARED;
...
channel_map_t channel_map[NUM_CHANNEL] = {
  { "FIFO_genp_ndpf0", (void*)FIFO_genp_ndpf0_buffer, 1, (100*100) *sizeof(float), &q_handle_FIFO_genp_ndpf0 },
  ...
};
```
BIP System Design

Application Software (Programming Models)

Translation

Software Model - BIP

Functional Validation
- Invariant Generation
- Deadlock Detection,
- Functional Simulation,
...

Performance Analysis
- Timed Simulation
...

(...)

System Model - BIP

System Profiling/
Calibration

Extend System Model - BIP

Mapping

Model Transformation (II)

SR Software Model - BIP

Code generation

Deployed Software

Middleware

(Multi-Core) Platform
Our Methodology

1. Build executable model of the overall system
2. Learn probability distribution of key characteristics impacting application
3. Plug distributions and build a stochastic abstract model of the context
4. Apply Statistical model checking on the reduced model
Statistical Model Checking (1)

- Statistical methods to decide if a property is satisfied
- Estimate the probability that a system satisfies a property
- An alternative to avoid exhaustive exploration of state-space of a system
- Results might not be always correct, but possible to bound the probability of making errors
- Accuracy of estimates depends on no. and length of simulations
- Simple to implement and use
- Less memory and time intensive (compared to Model Checking)
What are the questions?

- Qualitative Question: Does $S \models P_{\geq \theta}(\phi)$?
- Quantitative Question: What is the probability for $S$ to satisfy $\phi$?

Principle:

- Reason on a finite set of executions and answer the question
Discussion

- Component framework encompassing heterogeneous composition
  - Separation of concerns 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
  - Correctness-by-construction
    - Source to source transformations
    - Verification based on compositionality, composability and incrementality
  - System-level analysis techniques jointly taking into account
    - application, hardware resources and mapping
    - Component and interaction partitioning

- Applications
  - Software componentization
  - Programming multicore systems
  - Complex systems modeling and analysis e.g., IMA
# BIP Related Projects

<table>
<thead>
<tr>
<th>Project</th>
<th>Domain</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOSE</td>
<td>BGLE</td>
<td>Speculative and Exploratory Design in System Engineering</td>
</tr>
<tr>
<td>ManyCoreLabs</td>
<td>BGLE</td>
<td>Generic Embedded Systems Platform</td>
</tr>
<tr>
<td>SPICES</td>
<td>ITEA</td>
<td>Support for Predictable Integration of Mission Critical Embedded Systems</td>
</tr>
<tr>
<td>COMBEST</td>
<td>FP7</td>
<td>Component-Based Embedded Systems Design Techniques</td>
</tr>
<tr>
<td>PRO3D</td>
<td>FP7</td>
<td>Programming for Future 3D Architectures with Many Cores</td>
</tr>
<tr>
<td>SMECY</td>
<td>ARTEMIS</td>
<td>Smart Multicore Embedded Systems</td>
</tr>
<tr>
<td>ACROSS</td>
<td>ARTEMIS</td>
<td>Artemis Cross-Domain Architecture</td>
</tr>
<tr>
<td>ASCENS</td>
<td>FP7</td>
<td>Autonomic Service Component Ensembles</td>
</tr>
<tr>
<td>CERTAINTY</td>
<td>FP7</td>
<td>Certification of Real Time Applications designed for mixed criticality</td>
</tr>
<tr>
<td>MIND</td>
<td>Minalogic</td>
<td>Technologie d'assemblage des composants logiciels embarques</td>
</tr>
<tr>
<td>CHAPI</td>
<td>ANR</td>
<td>Calcul Embarque Hautes Performances pour des Applications Industrielles</td>
</tr>
<tr>
<td>MARAE</td>
<td>FRAE</td>
<td>Methodes et Architectures Robustes pour l'Autonomie dans l'Espace</td>
</tr>
<tr>
<td>GOAC</td>
<td>ESA</td>
<td>Goal Oriented Autonomous Controller</td>
</tr>
<tr>
<td>SYMPAA</td>
<td></td>
<td>Controleur de Paiement Monetique de type Automate sur Autoroute</td>
</tr>
</tbody>
</table>

[http://www-verimag.imag.fr/Rigorous-Design-of-Component-Based.html](http://www-verimag.imag.fr/Rigorous-Design-of-Component-Based.html)
• Thanks for your attention.

Questions?