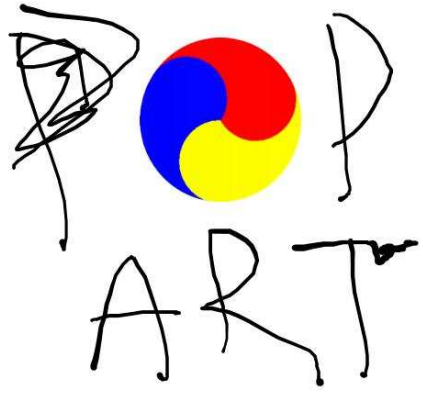
Using discrete controller synthesis for fault-tolerant distributed systems

Alain Girault, Éric Rutten

POP ART, INRIA Rhône-Alpes

Alain.Girault@inrialpes.fr, Eric.Rutten@inrialpes.fr, www.inrialpes.fr/pop-art



Motivations and context

Embedded systems (aeronautics, automotive, ...)

Motivations and context

Embedded systems (aeronautics, automotive, ...)

automatic-control/discrete-event duality:

sampled time iterations, mode switches;

Motivations and context

Embedded systems (aeronautics, automotive, ...)

automatic-control/discrete-event duality:
sampled time iterations, mode switches;

critical real-time: timing constraints;

POP ART

Motivations and context

Embedded systems (aeronautics, automotive, ...)

- automatic-control/discrete-event duality: sampled time iterations, mode switches;
- critical real-time: timing constraints;
- limited resources: computing, memory, power;

Motivations and context

Embedded systems (aeronautics, automotive, ...)

- automatic-control/discrete-event duality: sampled time iterations, mode switches;
- critical real-time: timing constraints;
- limited resources: computing, memory, power;
- distributed and heterogeneous architecture

Motivations and context

Embedded systems (aeronautics, automotive, ...)

- automatic-control/discrete-event duality: sampled time iterations, mode switches;
- critical real-time: timing constraints;
- limited resources: computing, memory, power;
- distributed and heterogeneous architecture

Intrinsically safety-critical systems, requiring

POP ART

Motivations and context

Embedded systems (aeronautics, automotive, ...)

- automatic-control/discrete-event duality: sampled time iterations, mode switches;
- critical real-time: timing constraints;
- limited resources: computing, memory, power;
- distributed and heterogeneous architecture

Intrinsically safety-critical systems, requiring

- safe design using off-line validation

→ need for formal models e.g., transition systems

POP ART

Motivations and context

Embedded systems (aeronautics, automotive, ...)

- automatic-control/discrete-event duality: sampled time iterations, mode switches;
- critical real-time: timing constraints;
- limited resources: computing, memory, power;
- distributed and heterogeneous architecture

Intrinsically safety-critical systems, requiring

- safe design using off-line validation
  → need for **formal models** e.g., transition systems
- safe execution with on-line fault recovery
  → need for **fault tolerance** e.g., recovery

POP ART

Problem statement

Safe design for safe execution

Problem statement

Safe design for safe execution

fault tolerance:

maintain correct functionality, whatever the faults;

Problem statement

Safe design for safe execution

fault tolerance:

maintain correct functionality, whatever the faults;

in a distributed system: upon processor failure:

reconfigure active tasks on remaining ones

POP ART

Problem statement

Safe design for safe execution

fault tolerance:
maintain correct functionality, whatever the faults;

in a distributed system: upon processor failure:
reconfigure active tasks on remaining ones

correctness of the reconfiguration to be validated
w.r.t. properties of fault tolerance

POP ART

Problem statement

Safe design for safe execution

fault tolerance:
maintain correct functionality, whatever the faults;

in a distributed system: upon processor failure:
reconfigure active tasks on remaining ones

correctness of the reconfiguration to be validated
w.r.t. properties of fault tolerance

We apply formal methods to ensure fault tolerance by:

POP ART

Problem statement

Safe design for safe execution

fault tolerance:
maintain correct functionality, whatever the faults;

in a distributed system: upon processor failure:
reconfigure active tasks on remaining ones

correctness of the reconfiguration to be validated
w.r.t. properties of fault tolerance

We apply formal methods to ensure fault tolerance by:

applying controller synthesis: advantages of
correctness of the result, easy modifiability

POP ART

Problem statement

Safe design for safe execution

fault tolerance:
maintain correct functionality, whatever the faults;

in a distributed system: upon processor failure:
reconfigure active tasks on remaining ones

correctness of the reconfiguration to be validated
w.r.t. properties of fault tolerance

We apply formal methods to ensure fault tolerance by:

applying controller synthesis: advantages of
correctness of the result, easy modifiability

producing automatically a controller
enforcing fault-tolerance for a distributed system

POP ART

Using controller synthesis for fault-tolerance

Model of the distributed system:

Using controller synthesis for fault-tolerance

Model of the distributed system:

architecture and environment

processors (fail-silent), fault model (patterns)

Using controller synthesis for fault-tolerance

Model of the distributed system:

architecture and environment

processors (fail-silent), fault model (patterns)

application: configurations

tasks and their placement on the architecture

Using controller synthesis for fault-tolerance

Model of the distributed system:

architecture and environment

processors (fail-silent), fault model (patterns)

application: configurations

tasks and their placement on the architecture

Properties to be enforced:

Using controller synthesis for fault-tolerance

Model of the distributed system:

- architecture and environment
  processors (fail-silent), fault model (patterns)
- application: configurations
  tasks and their placement on the architecture

Properties to be enforced:

- consistent execution: placement constraints

POP ART

Using controller synthesis for fault-tolerance

Model of the distributed system:

- architecture and environment
  processors (fail-silent), fault model (patterns)
- application: configurations
  tasks and their placement on the architecture

Properties to be enforced:

- consistent execution: placement constraints
- functionality fulfillment e.g., reach termination

POP ART

Using controller synthesis for fault-tolerance

Model of the distributed system:

- architecture and environment
  processors (fail-silent), fault model (patterns)
- application: configurations
  tasks and their placement on the architecture

Properties to be enforced:

- consistent execution: placement constraints
- functionality fulfillment e.g., reach termination
- optimization of costs (time, power) and qualities

POP ART

Using controller synthesis for fault-tolerance

Model of the distributed system:

architecture and environment
processors (fail-silent), fault model (patterns)

application: configurations
tasks and their placement on the architecture

Properties to be enforced:

consistent execution: placement constraints

functionality fulfillment e.g., reach termination

optimization of costs (time, power) and qualities

Using controller synthesis: find, if it exists,
the controller of the model enforcing the properties

$\longrightarrow$ synthesis of the correct reconfiguration controller

POP ART

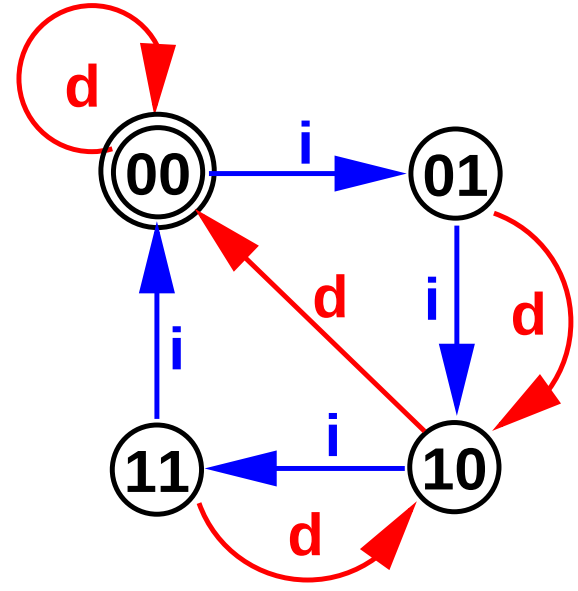
Discrete control synthesis

Purpose: *make* a property hold in the controlled system*!*

transition system:

all possible behaviours

(incl. bad ones)



Discrete control synthesis

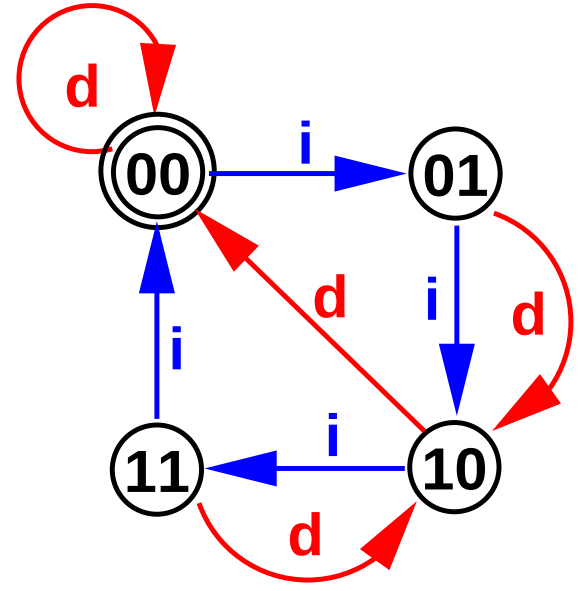
Purpose: *make* a property hold in the controlled system*!*

transition system:

all possible behaviours (incl. bad ones)

events: uncontrollable, and controllable: *to be constrained*

e.g., i *controllable,* d *not*



Discrete control synthesis

Purpose: *make* a property hold in the controlled system*!*

transition system:

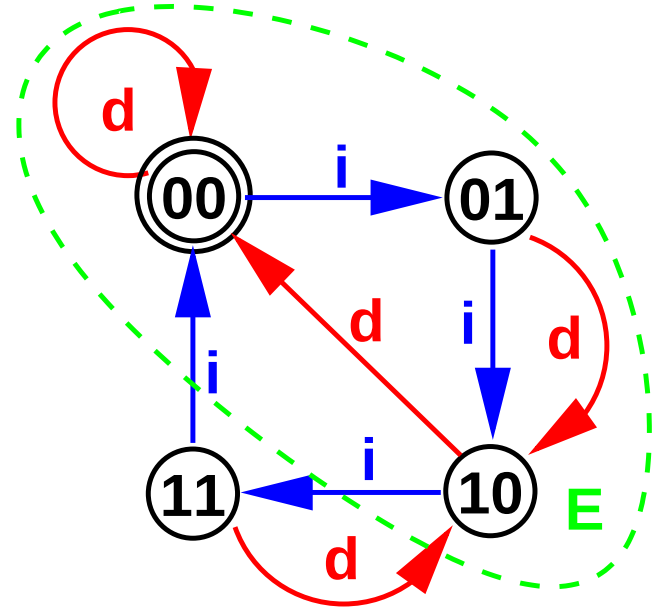
all possible behaviours (incl. bad ones)

events: uncontrollable, and controllable: *to be constrained*

e.g., i *controllable,* d *not*

objectives: properties

e.g., *make invariant* w.r.t. E s.t. not (s1 and s2)



Discrete control synthesis

Purpose: *make* a property hold in the controlled system*!*

transition system:

all possible behaviours (incl. bad ones)

events: uncontrollable, and controllable: *to be constrained*

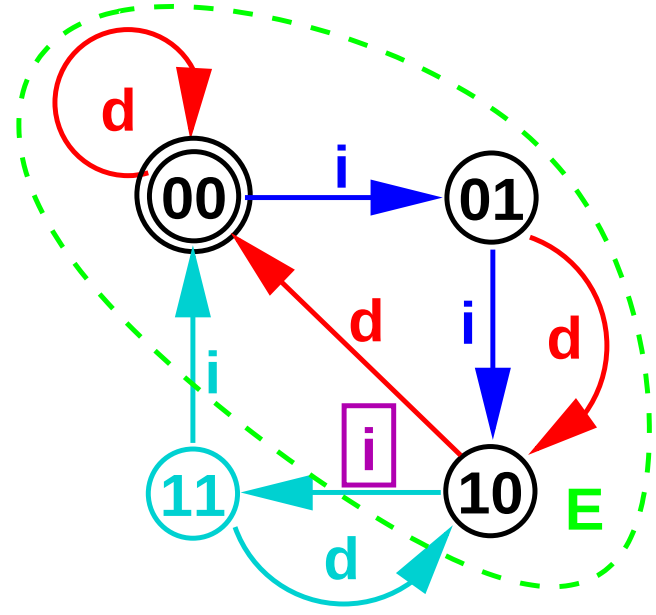
e.g., i *controllable,* d *not*

objectives: properties

e.g., *make invariant* w.r.t. E s.t. not (s1 and s2)

controller $\{ctrl\}=f(state, unctrl)$

e.g., *inhibit event* i *from state* 10



Property enforcing layers

Mixed imperative/declarative descriptions [ESOP03]

Property enforcing layers

Mixed imperative/declarative descriptions [ESOP03]

local constraints of components: set of automata

Property enforcing layers

Mixed imperative/declarative descriptions [ESOP03]

- local constraints of components: set of automata
- global constraints on interactions: properties

Property enforcing layers

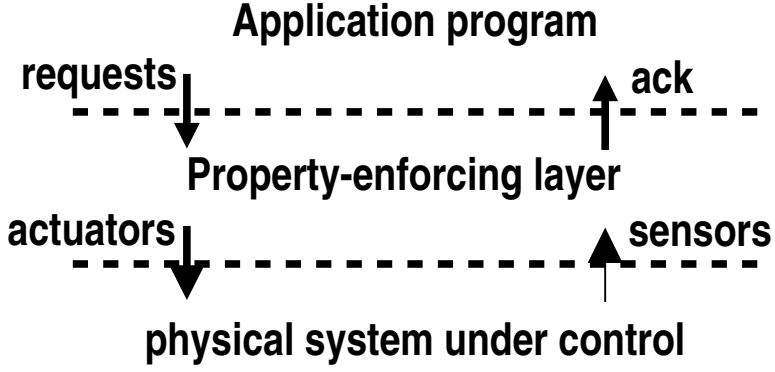
Mixed imperative/declarative descriptions [ESOP03]

- local constraints of components: set of automata
- global constraints on interactions: properties
- combination by control synthesis as compilation

Property enforcing layers

Mixed imperative/declarative descriptions [ESOP03]

- local constraints of components: set of automata
- global constraints on interactions: properties
- combination by control synthesis as compilation

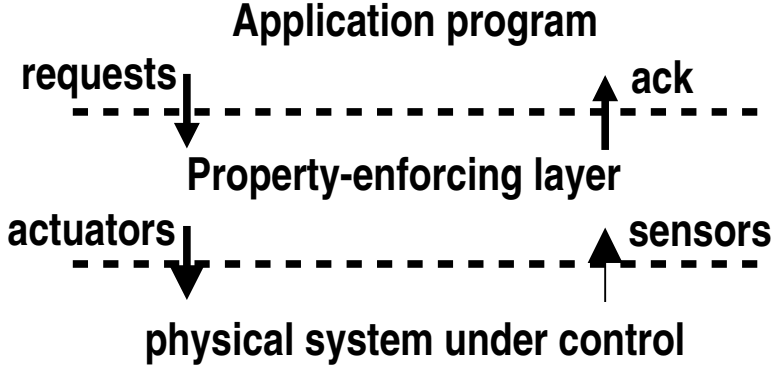


Automatic generation of property enforcing layers

Property enforcing layers

Mixed imperative/declarative descriptions [ESOP03]

- local constraints of components: set of automata
- global constraints on interactions: properties
- combination by control synthesis as compilation



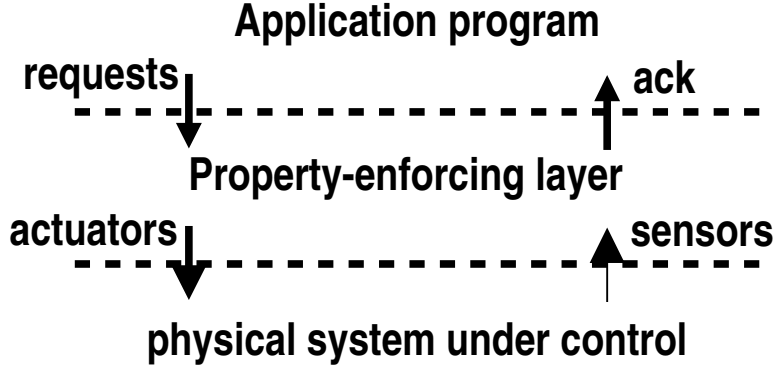
Automatic generation of property enforcing layers

- correct control not just monitoring

Property enforcing layers

Mixed imperative/declarative descriptions [ESOP03]

- local constraints of components: set of automata
- global constraints on interactions: properties
- combination by control synthesis as compilation



Automatic generation of property enforcing layers

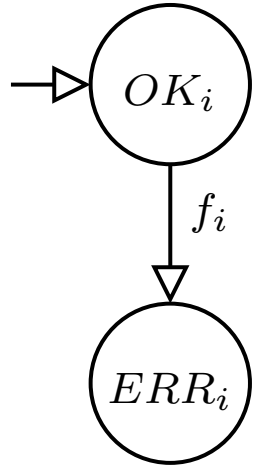
- correct control not just monitoring
- efficient synthesis (relatively) on prepared model

Architecture model

Local processor: fail-silent, permanent failure

multiple tasks, time-sharing; load are additive

quantitative bounds b_i (e.g., power, CPU load)

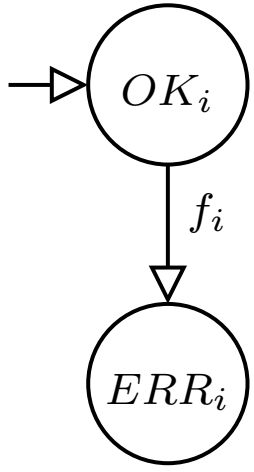


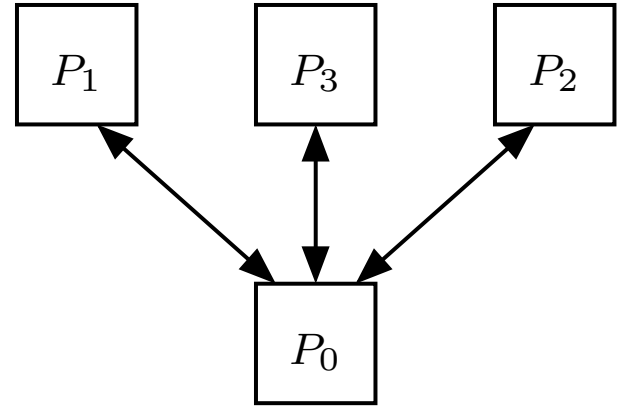
Architecture model

Local processor: fail-silent, permanent failure

multiple tasks, time-sharing; load are additive

quantitative bounds b_i (e.g., power, CPU load)





$\mathcal{S} = \{P_1, P_2, P_3\}$

Network model: *heterogeneous*

processor P_0 dedicated for control, failless

fully connected network, no communication failure

Environment or fault model

What failures can occur in the system?

Environment or fault model

What failures can occur in the system?

all processors can fail: no tolerance whatsoever

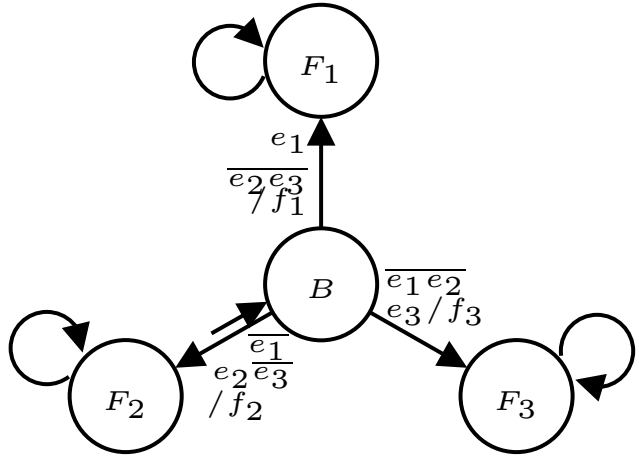
Environment or fault model

What failures can occur in the system?

all processors can fail: no tolerance whatsoever

(a) only one failure

(a)



Environment or fault model

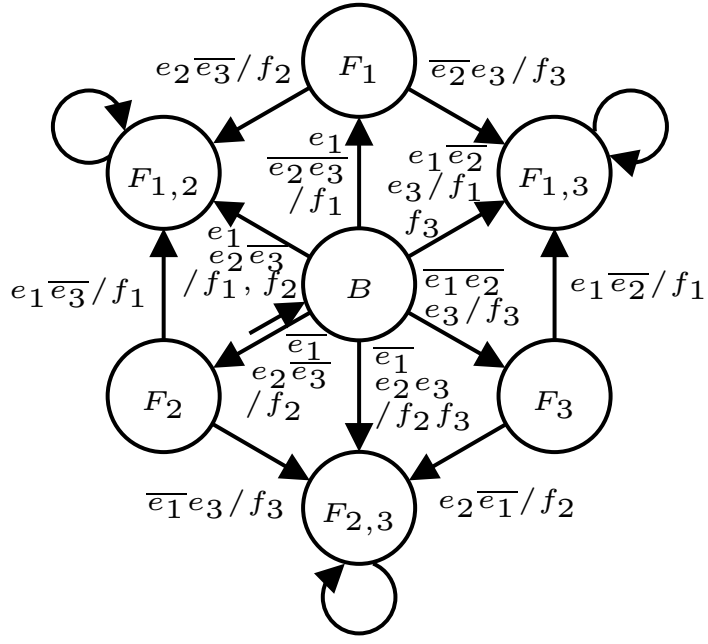
What failures can occur in the system?

all processors can fail: no tolerance whatsoever

(a) only one failure

(b) two failures possibly simultaneously

(b)



Environment or fault model

What failures can occur in the system?

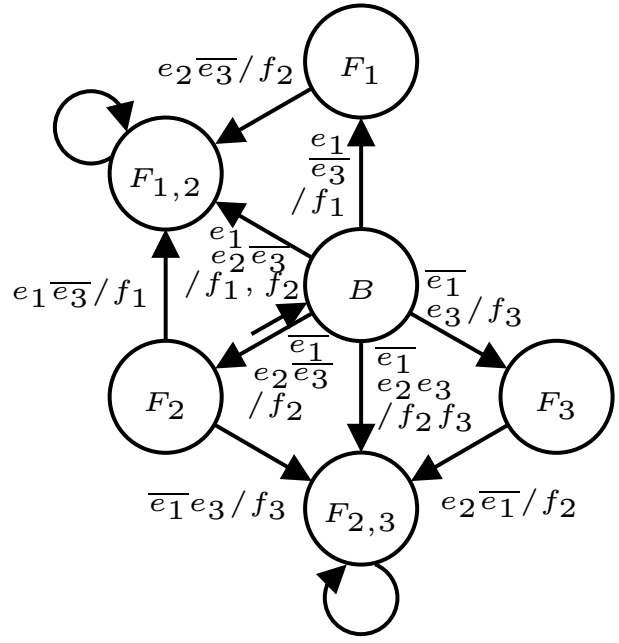
all processors can fail: no tolerance whatsoever

(a) only one failure

(b) two failures possibly simultaneously

(c) other patterns e.g., not 1 and 3 together

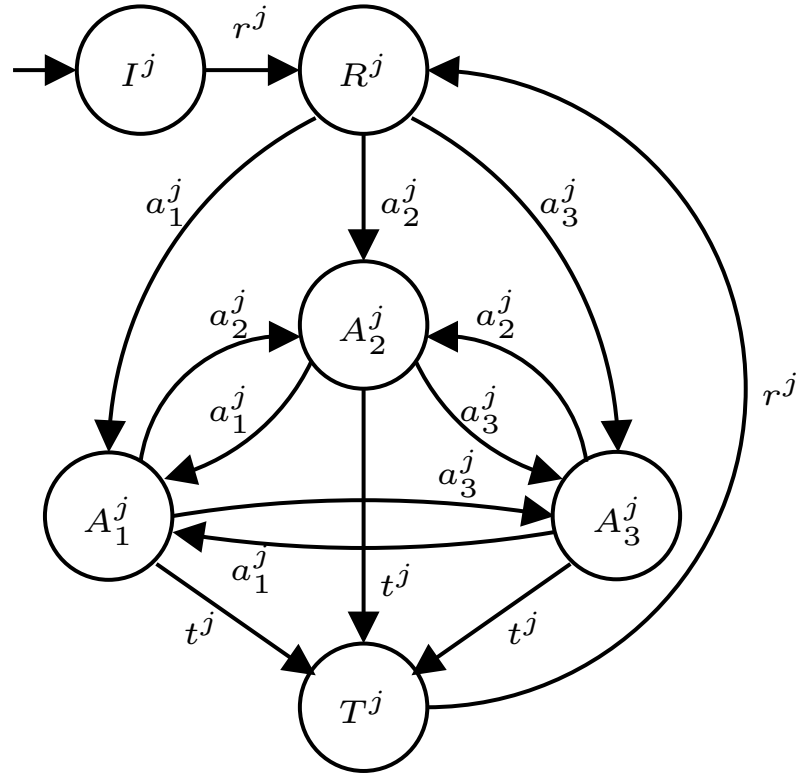
(c)



Task model (i)

Basic control structure pattern

task j, executable on 3 procs.



Task model (i)

Basic control structure pattern

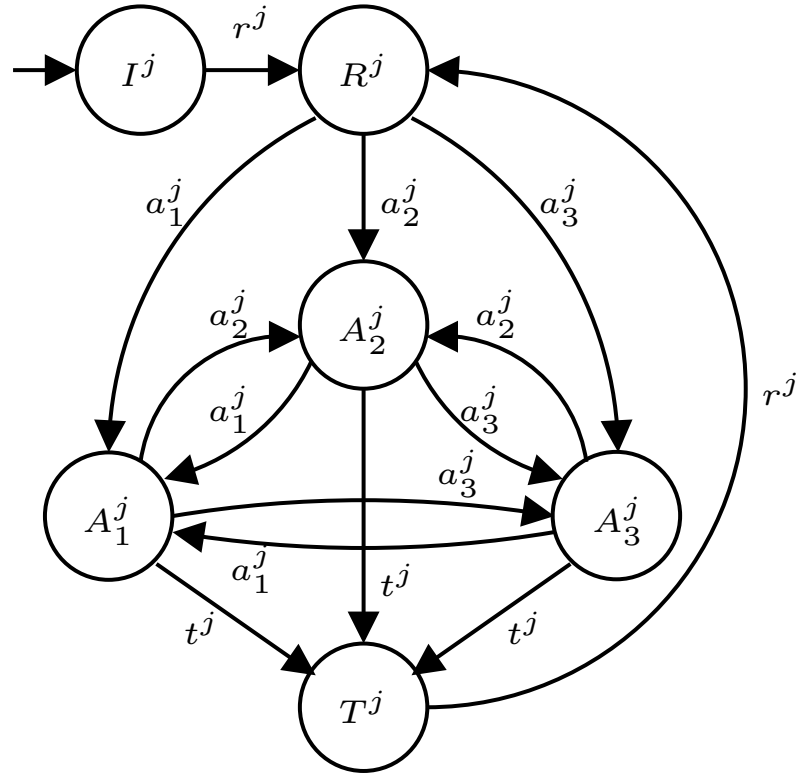
task j, executable on 3 procs.

initially idle in I^j,

upon request r^j: ready R^j

A_i^j : cyclically executed on P_i,

upon termination t_j: ended T^j



Task model (i)

Basic control structure pattern

task j, executable on 3 procs.

initially idle in I^j,

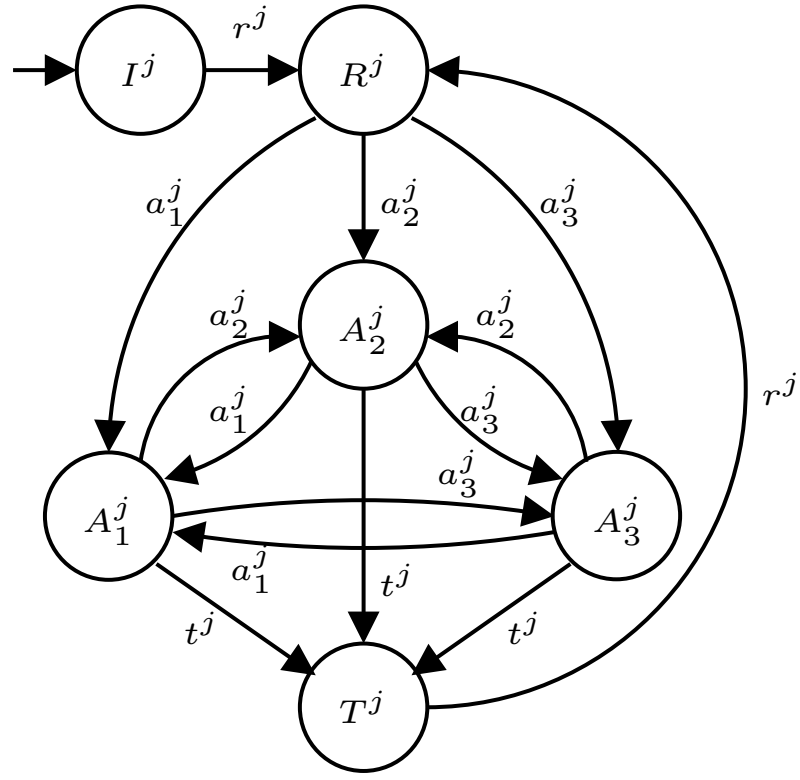
upon request r^j: ready R^j

A_i^j : cyclically executed on P_i,

upon termination t_j: ended T^j

re-configuration: transition

(controllable) from A_i^j to A_k^j



Task model (ii)

Quantitative characteristics: weights associated with states

Task model (ii)

Quantitative characteristics: weights associated with states

Execution time or CPU load required by each task

Task model (ii)

Quantitative characteristics: weights associated with states

- Execution time or CPU load required by each task
- Power consumption on a given processor

Power consumption		processor		
		P_1	P_2	P_3
task	T^1	4	4	2
	T^2	2	2	3
	T^3	2	3	4
bound		5	3	6

Task model (ii)

Quantitative characteristics: weights associated with states

- Execution time or CPU load required by each task
- Power consumption on a given processor
- Quality of the functionality (accuracy, depth of search, algorithm versions, ...)

Power consumption		processor		
		P_1	P_2	P_3
task	T^1	4	4	2
	T^2	2	2	3
	T^3	2	3	4
bound		5	3	6

Task quality		processor		
		P_1	P_2	P_3
task	T^1	3	5	3
	T^2	2	2	5
	T^3	2	2	5

POP ART

Application model

Tasks server: n tasks in parallel

Application model

Tasks server: n tasks in parallel

synchronous composition of behaviours

Application model

Tasks server: n tasks in parallel

synchronous composition of behaviours

composition of costs e.g., addition:

for CPU loads or power: on each P_i: $C_i = \sum_j C_i^j$

for quality: means, or actually sum: $Q = \sum_i \sum_j Q_i^j$

Application model

Tasks server: n tasks in parallel

synchronous composition of behaviours

composition of costs e.g., addition:

for CPU loads or power: on each P_i: $C_i = \sum_j C_i^j$

for quality: means, or actually sum: $Q = \sum_i \sum_j Q_i^j$

Program or scheduler (*not handling distribution)*

Application model

Tasks server: n tasks in parallel

synchronous composition of behaviours

composition of costs e.g., addition:

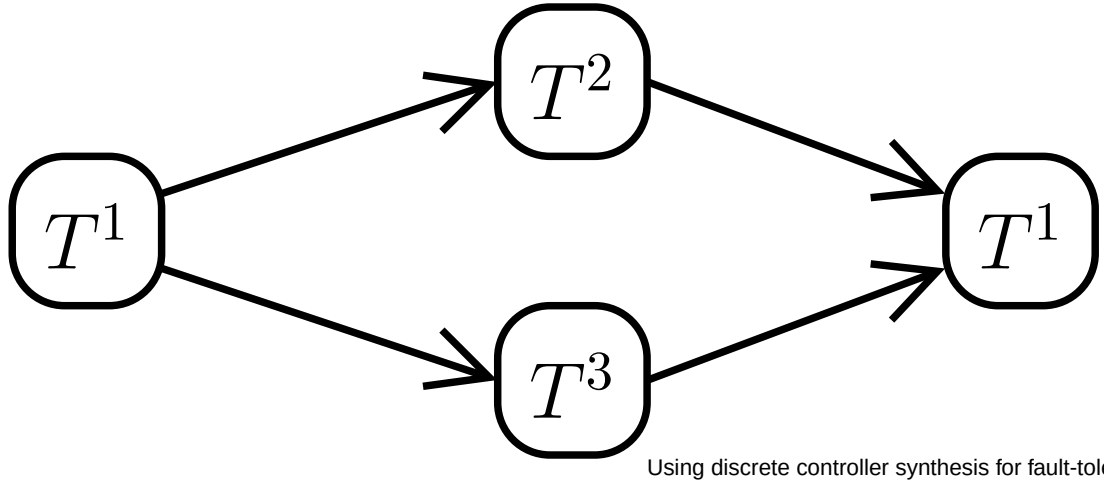
for CPU loads or power: on each P_i: $C_i = \sum_j C_i^j$

for quality: means, or actually sum: $Q = \sum_i \sum_j Q_i^j$

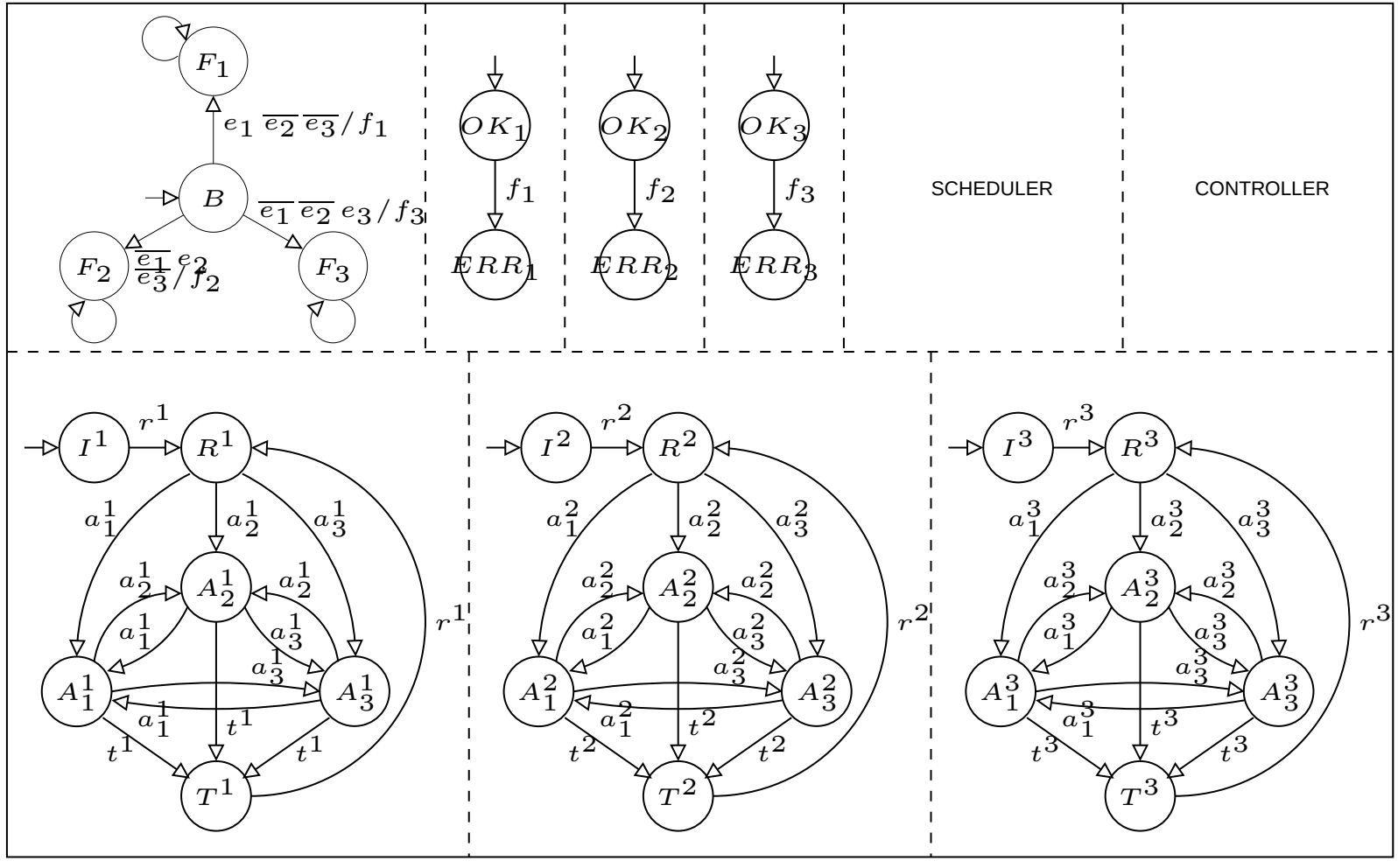
Program or scheduler (*not handling distribution)*

emitting requests in sequence

according to precedence graph



System model



composition of all that → the system to be controlled

Properties and objectives for fault-tolerance

Insuring consistent execution: make it *invariantly true*

Properties and objectives for fault-tolerance

Insuring consistent execution: make it *invariantly true*

No task active on a failed processor $\neg \bigvee_j \bigvee_i (A_i^j \wedge Err_i)$

Properties and objectives for fault-tolerance

Insuring consistent execution: make it *invariantly true*

No task active on a failed processor $\neg \bigvee_j \bigvee_i (A_i^j \wedge Err_i)$

Tasks active on a proc. are within capacity $\forall i, C_i \leq b_i$

Properties and objectives for fault-tolerance

Insuring consistent execution: make it *invariantly true*

No task active on a failed processor $\neg \bigvee_j \bigvee_i (A_i^j \wedge Err_i)$

Tasks active on a proc. are within capacity $\forall i, C_i \leq b_i$

Insuring functionality: make that, from all reachable states, the terminal configurations such that $\bigwedge_i T^i$ are *reachable*

Properties and objectives for fault-tolerance

Insuring consistent execution: make it *invariantly true*

No task active on a failed processor $\neg \bigvee_j \bigvee_i (A_i^j \wedge Err_i)$

Tasks active on a proc. are within capacity $\forall i, C_i \leq b_i$

Insuring functionality: make that, from all reachable states, the terminal configurations such that $\bigwedge_i T^i$ are *reachable*

Optimizing costs and qualities among remaining behaviors

Properties and objectives for fault-tolerance

Insuring consistent execution: make it *invariantly true*

No task active on a failed processor $\neg \bigvee_j \bigvee_i (A_i^j \wedge Err_i)$

Tasks active on a proc. are within capacity $\forall i, C_i \leq b_i$

Insuring functionality: make that, from all reachable states, the terminal configurations such that $\bigwedge_i T^i$ are *reachable*

Optimizing costs and qualities among remaining behaviors

maximize global quality varying according to P_i (also giving some progress)

Properties and objectives for fault-tolerance

Insuring consistent execution: make it *invariantly true*

- No task active on a failed processor $\neg \bigvee_j \bigvee_i (A_i^j \wedge Err_i)$
- Tasks active on a proc. are within capacity $\forall i, C_i \leq b_i$

Insuring functionality: make that, from all reachable states, the terminal configurations such that $\bigwedge_i T^i$ are *reachable*

Optimizing costs and qualities among remaining behaviors

- maximize global quality varying according to P_i (also giving some progress)
- minimize global consumption in time or power

POP ART

Properties and objectives for fault-tolerance

Insuring consistent execution: make it *invariantly true*

- No task active on a failed processor $\neg \bigvee_j \bigvee_i (A_i^j \wedge Err_i)$
- Tasks active on a proc. are within capacity $\forall i, C_i \leq b_i$

Insuring functionality: make that, from all reachable states, the terminal configurations such that $\bigwedge_i T^i$ are *reachable*

Optimizing costs and qualities among remaining behaviors

- maximize global quality varying according to P_i (also giving some progress)
- minimize global consumption in time or power

Order of synthesis operations essential: *not commutative*

Illustrative scenarii

Insuring consistent execution:

Illustrative scenarii

Insuring consistent execution:

No task is active on a failed processor

if P_1 goes to ERR_1, any task on P_1 is reconfigured

Illustrative scenarii

Insuring consistent execution:

No task is active on a failed processor

if P_1 goes to ERR_1, any task on P_1 is reconfigured

Tasks active on a proc. are within capacity

when T^1 on P_1, T^2 on P_2, T^3 on P_3, if P_2 goes to ERR_2:
T^2 is forced to migrate to P_1 or P_3, but then overload?
hence forcing migration of both T^1 to P_3 and T^2 to P_1

Illustrative scenarii

Insuring consistent execution:

No task is active on a failed processor

if P_1 goes to ERR_1, any task on P_1 is reconfigured

Tasks active on a proc. are within capacity

when T^1 on P_1, T^2 on P_2, T^3 on P_3, if P_2 goes to ERR_2:
T^2 is forced to migrate to P_1 or P_3, but then overload?
hence forcing migration of both T^1 to P_3 and T^2 to P_1

Insuring functionality avoids staying in R^j

keeping only paths clear and wide enough down to the end

one failure: ok; two failures: no; (c) pattern: ok

Illustrative scenarii

Insuring consistent execution:

No task is active on a failed processor

if P_1 goes to ERR_1, any task on P_1 is reconfigured

Tasks active on a proc. are within capacity

when T^1 on P_1, T^2 on P_2, T^3 on P_3, if P_2 goes to ERR_2:
T^2 is forced to migrate to P_1 or P_3, but then overload?
hence forcing migration of both T^1 to P_3 and T^2 to P_1

Insuring functionality avoids staying in R^j

keeping only paths clear and wide enough down to the end

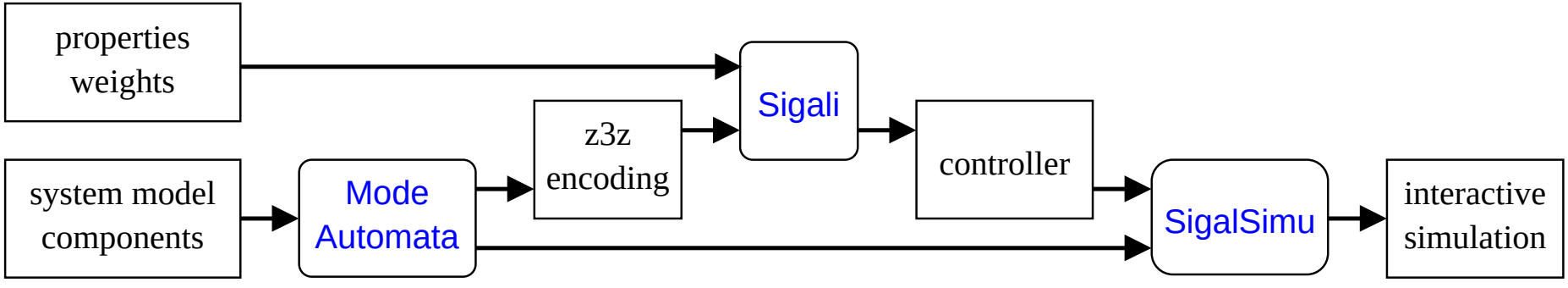
one failure: ok; two failures: no; (c) pattern: ok

Optimizing costs and qualities: different solutions

when minimizing cost first, maximizing quality then

Implementation

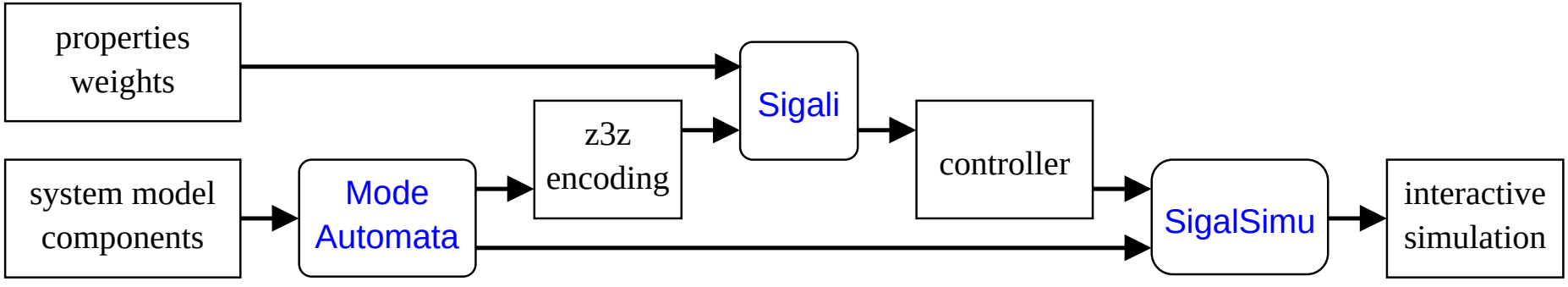
Using synchronous tools



Implementation

Using synchronous tools

behavior specification in Mode Automata (Verimag)

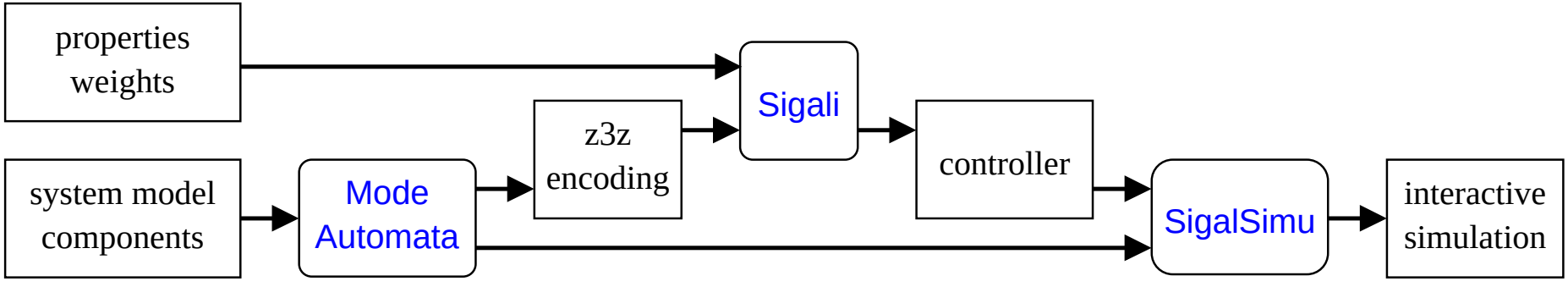


Implementation

Using synchronous tools

behavior specification in Mode Automata (Verimag)

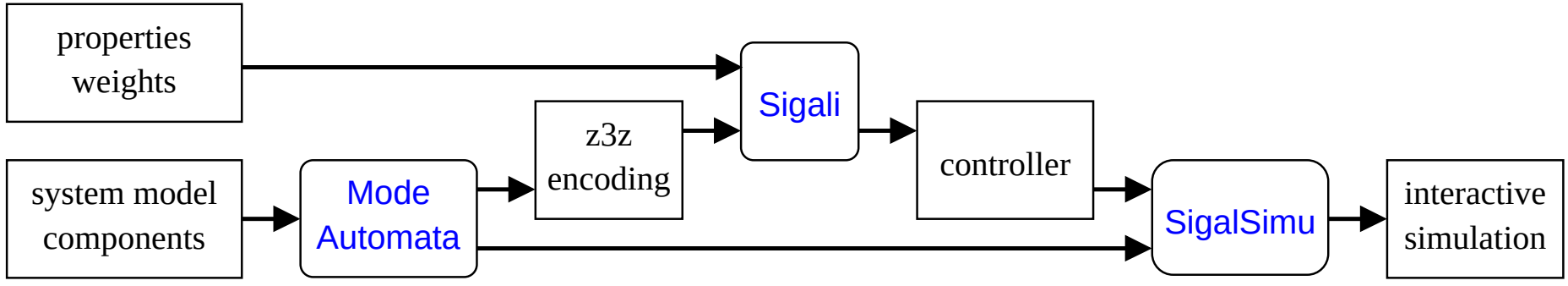
objectives and synthesis with Sigali (IRISA)



Implementation

Using synchronous tools

- behavior specification in Mode Automata (Verimag)
- objectives and synthesis with Sigali (IRISA)
- co-simulation with SigalSimu



Conclusion and perspectives

Results

Conclusion and perspectives

Results

a formal model of a real-time distributed system
processors, faults, tasks, and reconfigurations

Conclusion and perspectives

Results

a formal model of a real-time distributed system
processors, faults, tasks, and reconfigurations

automatic production of a controller
enforcing fault-tolerance by reconfiguration

POP ART

Conclusion and perspectives

Results

a formal model of a real-time distributed system
processors, faults, tasks, and reconfigurations

automatic production of a controller
enforcing fault-tolerance by reconfiguration

Perspectives

POP ART

Conclusion and perspectives

Results

a formal model of a real-time distributed system
processors, faults, tasks, and reconfigurations

automatic production of a controller
enforcing fault-tolerance by reconfiguration

Perspectives

model of tasks with modes, other architectures, ...

POP ART

Conclusion and perspectives

Results

a formal model of a real-time distributed system
processors, faults, tasks, and reconfigurations

automatic production of a controller
enforcing fault-tolerance by reconfiguration

Perspectives

model of tasks with modes, other architectures, ...

properties: exclusions on resources, observers, ...

POP ART

Conclusion and perspectives

Results

- a formal model of a real-time distributed system
  processors, faults, tasks, and reconfigurations
- automatic production of a controller
  enforcing fault-tolerance by reconfiguration

Perspectives

- model of tasks with modes, other architectures, ...
- properties: exclusions on resources, observers, ...
- platform-based design: same system
  used under different control objectives

POP ART