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;
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
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
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
  \[\rightarrow \text{need for } \text{formal models} \text{ e.g., transition systems}\]

safe execution with on-line fault recovery
  \[\rightarrow \text{need for } \text{fault tolerance} \text{ e.g., recovery}\]
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.
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
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
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:
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
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
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

Properties to be enforced:
- Consistent execution: placement constraints
- Functionality fulfillment: e.g., reach termination
- Optimization of costs: (time, power) and qualities
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:
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
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
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 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

→ synthesis of the correct reconfiguration controller
Discrete control synthesis

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

transition system:

*all possible behaviours* (incl. bad ones)

![Diagram showing a transition system with states 00, 01, 11, and 10, and transitions labeled with 'i' and 'd'.]
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 \( \{\text{ctrl}\} = f(\text{state}, \text{unctrl}) \)
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

Using discrete controller synthesis for fault-tolerant distributed systems – p.6/16
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

Application program
requests
Property-enforcing layer
ack
actuators
sensors
physical system under control

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)

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
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
Environment or fault model

What failures can occur in the system?

(a) only one failure
(b) two failures possibly simultaneously
(c) other patterns e.g., not 1 and 3 together

\[
\begin{align*}
\text{if } e_1 \text{ and } e_3 = f_1 \\
\text{if } e_2 \text{ and } e_3 = f_2 \\
\text{if } e_2 \text{ and } e_1 = f_3 \\
\end{align*}
\]
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 $R^j$

$A^j$: cyclically executed on $P_i$,

upon termination $t^j$: ended $T^j$
Basic control structure pattern

Task model (i)

task $j$, executable on 3 procs.

initially idle in $I^j$,

upon request $r^j$: ready $R^j$

$A^j_i$: cyclically executed on $P_i$,

upon termination $t^j$: ended $T^j$

re-configuration: transition

(controllable) from $A^j_i$ to $A^j_k$
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

<table>
<thead>
<tr>
<th>Power consumption</th>
<th>processor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$P_1$</td>
</tr>
<tr>
<td>$T^1$</td>
<td>4</td>
</tr>
<tr>
<td>$T^2$</td>
<td>2</td>
</tr>
<tr>
<td>$T^3$</td>
<td>2</td>
</tr>
<tr>
<td>bound</td>
<td>5</td>
</tr>
</tbody>
</table>
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, ...)

<table>
<thead>
<tr>
<th>Power consumption</th>
<th>processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>$P_1$</td>
</tr>
<tr>
<td>$T^1$</td>
<td>4</td>
</tr>
<tr>
<td>$T^2$</td>
<td>2</td>
</tr>
<tr>
<td>$T^3$</td>
<td>2</td>
</tr>
<tr>
<td>bound</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task quality</th>
<th>processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>$P_1$</td>
</tr>
<tr>
<td>$T^1$</td>
<td>3</td>
</tr>
<tr>
<td>$T^2$</td>
<td>2</td>
</tr>
<tr>
<td>$T^3$</td>
<td>2</td>
</tr>
</tbody>
</table>
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 $P_i$: $C_i = \sum_j C^j_i$

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

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 \( P_i: C_i = \sum_j C^j_i \)
for quality: means, or actually sum: \( Q = \sum_i \sum_j Q^j_i \)

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 \textit{invariantly true}
Properties and objectives for fault-tolerance

**Insuring consistent execution:** make it *invariantly true*

No task active on a failed processor $\neg \bigvee_j \bigvee_i (A^j_i \land Err_i)$
Properties and objectives for fault-tolerance

Insuring consistent execution: make it \textit{invariantly true}

No task active on a failed processor \(\neg \bigvee_j \bigvee_i (A^j_i \land Err_i)\)

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 \textit{invariantly true}

No task active on a failed processor \( \neg \bigvee_j \bigvee_i (A_{ji}^j \land Err_{ji}) \)

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 \textit{reachable}
Properties and objectives for fault-tolerance

Insuring consistent execution: make it *invariantly* true

No task active on a failed processor \( \neg \bigvee_j \bigvee_i (A^j_i \land Err_i) \)

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 \textit{invariantly true}

\[ \neg \bigvee_j \bigvee_i (A^j_i \land \text{Err}_i) \]

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 \textit{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 \( \neg \bigvee_j \bigvee_i (A^j_i \land \text{Err}_i) \)

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
Properties and objectives for fault-tolerance

Insuring consistent execution: make it \textit{invariantly true}

No task active on a failed processor \( \neg \bigvee_j \bigvee_i (A_{j_i} \land \text{Err}_i) \)

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 \textit{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 \textit{essential}: \textit{not commutative}
Illustrative scenarii

Insuring consistent execution:

No task is active on a failed processor if $P_1$ goes to ERR, 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:

$T_2$ is forced to migrate to $P_1$ or $P_3$, but then overload? hence forcing migration of $T_1$ to $P_3$ and $T_2$ to $P_1$.

Insuring functionality avoids staying in Rj 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.
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

- properties weights
- system model components
- Mode Automata
- z3 encoding
- Sigali
- controller
- SigalSimu
- interactive simulation
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
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
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, ...
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, ...
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