Automating the Addition of Fault Tolerance with Discrete Controller Synthesis

Alain Girault and Eric Rutten

22 January 2009
Outline

1. Introduction and context
2. Framework for automating the addition of FT with DCS
3. Specifying the hardware component failures
4. Handling different kinds of failures
5. Advanced DCS features
6. Case studies
7. Related work and conclusion
Outline

1. **Introduction and context**

2. Framework for automating the addition of FT with DCS

3. Specifying the hardware component failures

4. Handling different kinds of failures

5. Advanced DCS features

6. Case studies

7. Related work and conclusion
Problem and motivations

Motivation

Automate the addition of fault-tolerance in embedded systems with Discrete Controller Synthesis (DCS)

Provide a complete framework

- Based on a formal method (DCS) to provide a guaranteed fault-tolerant level
- Encompassing all kinds of hardware devices: processors, communication links, sensors, actuators
- Taking into account all kinds of failures: crashes, value failures, Byzantine
- Providing quantitative results thanks to optimal DCS
Discrete Controller Synthesis (DCS)


Discrete controller synthesis problem

Given the plant $\mathcal{U}$ and the desired system $\mathcal{D}$, find the controller $\mathcal{C}$ s.t.:

$$\mathcal{U} \cap \mathcal{C} \subseteq \mathcal{D}$$

The language $\mathcal{U}$ is built over the set of events $\mathcal{I}$

$\mathcal{I}$ is partitioned into controllable $\mathcal{I}_c$ and uncontrollable $\mathcal{I}_u$

$\mathcal{C}$ can only act on the events in $\mathcal{I}_c$
Closed-loop controlled system and SIGALI

uncontrollable inputs
controller C

controllable inputs

plant \( \mathcal{U} \)

outputs

\( \mathcal{D} \): invariance or reachability property

\( \mathcal{U} \): system model (modular)

SIGALI: [http://www.irisa.fr/vertecs/Logiciels/sigali.html]
Features of our method

- The possibility to try several fault hypotheses on the same specification.
- The possibility to evaluate several fault tolerance requirements.
- In the final program, the guarantee by construction of the fault tolerance level required by the user.

Features specific to DCS

- The failure recovery mechanism is dynamic (hence it does not induce too much redundancy overhead like static methods).
- With a static guarantee on the fault tolerance of the obtained system (unlike dynamic methods).
Formal models for DCS I

Labeled Transition Systems (LTS) $S = \langle Q, q^0, \mathcal{I}, \mathcal{O}, \rightarrow \rangle$

- $Q$ is the set of states and $q^0 \in Q$ is the initial state
- $\mathcal{I}$ is the set of input events, partitioned into $\mathcal{I}_c$ and $\mathcal{I}_u$
- $\mathcal{O}$ is the set of output events
- $\rightarrow \subseteq Q \times \text{Bool}(\mathcal{I}) \times \mathcal{O}^* \times Q$ is the transition relation

Synchronous product $\langle Q_1, q_1^0, \mathcal{I}_1, \mathcal{O}_1, \rightarrow_1 \rangle \parallel \langle Q_2, q_2^0, \mathcal{I}_2, \mathcal{O}_2, \rightarrow_2 \rangle$

$\langle Q_1 \times Q_2, (q_1^0, q_2^0), \mathcal{I}_1 \cup \mathcal{I}_2, \mathcal{O}_1 \cup \mathcal{O}_2, \rightarrow \rangle$

with $(q_1, q_2) \xrightarrow{g_1 \wedge g_2 / a_1 \wedge a_2} (q_1', q_2')$ iff $q_1 \xrightarrow{g_1 / a_1} q_1'$ and $q_2 \xrightarrow{g_2 / a_2} q_2'$
Formal models for DCS II

Path
A path in a LTS is a finite sequence of transitions
\[ q_1 \xrightarrow{g_1/a_1} q_2 \xrightarrow{g_2/a_2} q_3 \cdots q_n \xrightarrow{g_n/a_n} q_{n+1}. \]

Reachability
A state \( q \) of \( Q \) is reachable iff there exists a path from the initial state \( q^0 \) to \( q \).

Invariance
A set of states \( E \) is invariant iff any transition having as source a state of \( E \) has its destination state in \( E \).
# DCS functions and objectives

**$S' = \text{mk\_invar} \left(S, E\right)$ for $E \subseteq S$**

Returns a controlled system $S'$ such that all controllable transitions leading to states $q_{i+1} \notin E$ are inhibited.

**$S' = \text{kp\_reach} \left(S, E\right)$ for $E \subseteq S$**

Returns a controlled system $S'$ such that the controllable transitions entering subsets of states from where $E$ is not reachable are disabled.

**$E' = \text{reach\_under\_control} \left(S, E\right)$ for $E \subseteq S$**

Returns a subset $E'$ of the states of $S$ such that states in $E'$ are reachable by controllable transitions.

\[
\text{kp\_reach} \left(S, E\right) = \text{mk\_invar} \left(S, \text{reach\_under\_control} \left(S, E\right)\right)
\]
A short example of DCS
A short example of DCS
A short example of DCS
A short example of DCS

c = false

not c

E

S0

S1

S2

S3

S4
Outline

1. Introduction and context
2. Framework for automating the addition of FT with DCS
3. Specifying the hardware component failures
4. Handling different kinds of failures
5. Advanced DCS features
6. Case studies
7. Related work and conclusion
Framework for automating the addition of FT with DCS

- Only the hardware components can fail.
- The faults are uncontrollable events.
- The plant \( \mathcal{U} \) is specified modularly as the synchronous product of several LTSs.
- The plant \( \mathcal{U} \) represents all the possible behaviors, both the good ones and the bad ones.
- The fault model is specified as an LTS (composed synchronously with \( \mathcal{U} \)).
- The fault tolerance policy is specified as a reachability or invariance property (or more generally as a synchronous observer).
Defining the fault-intolerant system
Defining the fault hypothesis

- **What** HW components can fail
- If several components can fail, a list of failure configurations
- A fault model for each component that can fail
  - An LTS that specifies:
    - the behavior of the component when it fails
    - the duration of the failure
Defining a fault tolerance policy

Specified as a **DCS control objective**

- Invariance property, e.g.:
  - a subset of good behaviors should be invariant
  - a subset of bad behaviors should never be reached

- Reachability property, e.g.:
  - a subset of good behaviors should always be reachable

- More complex properties:
  - conditioned objectives
  - synchronous observers
Outline

1. Introduction and context
2. Framework for automating the addition of FT with DCS
3. Specifying the hardware component failures
4. Handling different kinds of failures
5. Advanced DCS features
6. Case studies
7. Related work and conclusion
Processor failures I

Permanent failures / Transient failures / Degraded transient mode

The events $f$ and $d$ are uncontrollable / For $r$ it depends
Processor failures II

One failure at a time

The events $f_i$ are $d_i$ are local / The events $f'_i$ and $d'_i$ are uncontrollable
At most two failures at a time (possibly simultaneous)

The events $f_i$ are $d_i$ are local / The events $f'_i$ and $d'_i$ are uncontrollable
Processor failures II

The events $f_i$ are $d_i$ are local / The events $f'_i$ and $d'_i$ are uncontrollable
Actuator failures I

Valve with permanent failures / With transient failures
Actuator failures II

Three-state braking system with permanent failures / With degraded modes
Sensor failures

Liquid level sensor with permanent failures / With transient failures
Outline

1. Introduction and context
2. Framework for automating the addition of FT with DCS
3. Specifying the hardware component failures
4. Handling different kinds of failures
5. Advanced DCS features
6. Case studies
7. Related work and conclusion
Classification of failures

[Avizienis, Laprie & Randell, IFIP WCC’2004]

- **Their domain**: in value or temporal (in the latter case, their duration must also be specified)

- **Their coherence** w.r.t. all the users

- **Their detectability** by the user

  - E.g., crash failures are permanent and coherent temporal failures

  - E.g., Byzantine failures are incoherent value failures
Crash failures

Processor model

\[ \bar{f} \]
\[ \Rightarrow \]
\[ \text{OK} \]
\[ \Rightarrow \]
\[ f \]
\[ \Rightarrow \]
\[ \text{ERR} \]

Task model

\[ R^i \]
\[ j \]
\[ r_j \]
\[ a_1^j \]
\[ a_2^j \]
\[ a_3^j \]

\[ A_1^j \]
\[ A_2^j \]
\[ A_3^j \]
\[ T^j \]
\[ t_j \]
\[ t_i \]
\[ a_1^i \]
\[ a_2^i \]
\[ a_3^i \]

DCS objective

No task is active on a failed processor:

\[ S' = mk\_invar\left( S, \neg \bigvee_{j=1}^{n} \bigvee_{i=1}^{p} (A_i^j \land ERR_i) \right) \]
Value failures

For a Boolean data: replace it by one uncontrollable input

For a numerical data: discretize its domain and replace it by several uncontrollable inputs

abstract interpretation
Byzantine failures

A basic non-faulty component / Same functionality with Byzantine failures

Byzantine failure behavior

When the component $C$ is faulty, each one of its outputs is replaced by an uncontrollable input
Outline

1. Introduction and context
2. Framework for automating the addition of FT with DCS
3. Specifying the hardware component failures
4. Handling different kinds of failures
5. Advanced DCS features
6. Case studies
7. Related work and conclusion
Shared resources

- Shared access: \textit{true}

- Mutual exclusion: \[ S' = \text{mk$_\text{invar}$} \left(S, \neg(A_1 \land A_2)\right) \]

- Continuous access: \[ S' = \text{mk$_\text{invar}$} \left(S, A_1 \lor A_2\right) \]
Critical section

State $S_i$ of tasks $\tau_1$ and $\tau_2$ is a critical section

$$S' = \text{mk_invar} \left( S, \neg((A_1, S_1, i) \land (A_2, S_2, i)) \right)$$
Optimal discrete controller synthesis I

[Kumar & Garg, SIAM J. Cont. Opt., 1995]  [Tronci, CDC’96]

<table>
<thead>
<tr>
<th>task</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>bound $b_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$P_2$</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>power consumption $C$ per processor</th>
<th>quality $Q$ per processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2</td>
</tr>
</tbody>
</table>
Optimal discrete controller synthesis II

Basic fault tolerance DCS objective:

\[ S' = \text{mk\_invar} \left( S, \neg \bigvee_{j=1}^{3} \bigvee_{i=1}^{3} (A_i^j \land \text{ERR}_i) \right) \]

Optimal DCS objectives:

\[ S'' = \text{mk\_invar} \left( S', \forall 1 \leq i \leq 3, \sum_{j=1}^{3} C_i^j \leq b_i \right) \]

\[ S''' = \text{maximize\_step} \left( S'', Q = \sum_{i=1}^{3} \sum_{j=1}^{3} Q_i^j \right) \]
Optimal discrete controller synthesis III

![Diagram showing the synthesis process with conditions and outcomes.]

- **Quality Condition**: $3 + 2 + 5 = 10$

- **Power Conditions**:
  - $4 \leq 5$
  - $2 \leq 3$
  - $4 \leq 6$

- **Inconsistent Conditions**:
  - **$b_1$**: $2 + 2 + 3 = 7$
  - $2 \leq 5$
  - $2 \leq 6$
  - $3 + 2 + 5 = 10$

- **Inconsistent Condition** (ERR$_2$):
  - $2 + 2 + 3 = 7$

- **Inconsistent Conditions**:
  - **$b_3$**: $3 + 2 + 5 = 10$
  - $3 + 5 + 5 = 13$
  - $4 \leq 5$
  - $3 + 4 > 6$
Optimal DCS on paths
Conditioned discrete controller objectives

In order to model degraded modes:

- Mode $i$: Condition $cond_i$ / DCS objective $dcs\_obj_i$

**Conditioned objective**

\[
\bigwedge_{i=1}^{n} (cond_i \Rightarrow dcs\_obj_i)
\]

**Example**

\[
(no \ failure \Rightarrow nominal\_dcs\_obj) \ \land \ (one \ failure \Rightarrow dcs\_obj_1) \ \land \ (two \ failures \Rightarrow dcs\_obj_2)
\]
Synchronous observers

[Cieslak, Desclaux, Fawaz & Varaiya, IEEE TAC, 1988]
Outline

1. Introduction and context
2. Framework for automating the addition of FT with DCS
3. Specifying the hardware component failures
4. Handling different kinds of failures
5. Advanced DCS features
6. Case studies
7. Related work and conclusion
Case study I: two tanks

Two tanks, two pipes, and four valves (four actuators: \( V_0 \), \( V_1 \), \( V_2 \), and \( V_3 \) :}
LTS of the system
Complete system specification
Conditioned DCS objectives

1. As long as the valve $V_0$ is not stuck in the faulty and open state, neither of the two tanks should ever over-flood:

$$S' = mk\_invar \left( \neg FO_0 \implies \neg \bigvee_{i=0}^{3} (N_3, N_i'), S \right)$$

2. If the four valves work fine, then the level in the left and right tanks should be regulated respectively at $N_2$ and $N_1'$:

$$S' = \bigwedge_{i=0}^{3} (O_i \lor C_i) \implies \text{reach\_under\_control} \left((N_2, N_1'), S \right)$$
Case study II: the Byzantine generals

[Lamport, Shostak & Pease, ACM TOPLAS, 1982]

- $n$ divisions of the Byzantine army, each commanded by its own general, are camped outside an enemy city.

- One of the generals is the commander of the army, while the $n - 1$ remaining ones are his lieutenants.

- The generals must decide on a common plan of action, either attack or retreat.

- Only by communicating with one another only by oral messages.

- The problem is that some generals are traitors.
The Byzantine generals

- The commander first sends an order (attack or retreat) to his $n - 1$ lieutenants:
  - If he is loyal, then he must send the same order to all his lieutenants
  - But if he is a traitor, then he can send different orders to his lieutenants, that is, incoherent orders

- Each lieutenant transmits the received order to all the other lieutenants (similar behavior as the commander)

- The loyal generals will reach a consensus for their plan of action
Conditions that must be satisfied

- **IC1**: All the loyal lieutenants obey the same order

- **IC2**: If the commander is loyal, then each loyal lieutenant obeys the order sent to him
Algorithm 1 Byzantine Commander \((m, v)\)
1 Send my order \(v\) to the \(n - 1\) lieutenants;

Algorithm 2 Byzantine Lieutenant \((m, i)\)
1 \(v_i := \) value received from the commander;
2 \textbf{if} \(m = 0\) \textbf{then}
3 \hspace{1em} Use as order the value \(v_i\);
4 \textbf{else}
5 \hspace{1em} Send \(v_i\) to the \(n - 2\) other lieutenants;
6 \hspace{1em} \textbf{forall} \(j \neq i\) \textbf{do}
7 \hspace{2em} \(v_j := \) value received from the lieutenant \(j\);
8 \hspace{1em} \textbf{end do}
9 \hspace{1em} Use as order the majority \(\text{maj}(v_1, v_2, \ldots, v_{n-1})\);
10 \textbf{end if}

**Lamport et al theorem**

If there are \(m\) traitors, then there must be at least \(3m + 1\) generals
Fault tolerant Byzantine generals

Environment

Commander

Lieutenant 1

Lieutenant 2

Lieutenant 3

\[ v \]

\[ u_1^c \]

\[ u_2^c \]

\[ u_3^c \]

\[ t_c \]

\[ t_1 \]

\[ t_2 \]

\[ t_3 \]

\[ v_1 \]

\[ v_2 \]

\[ v_3 \]

\[ u_1^1 \]

\[ u_2^1 \]

\[ u_3^1 \]

\[ u_1^2 \]

\[ u_2^2 \]

\[ u_3^2 \]

\[ u_1^3 \]

\[ u_2^3 \]

\[ u_3^3 \]

\[ v_1^1 \]

\[ v_2^1 \]

\[ v_3^1 \]

\[ v_1^2 \]

\[ v_2^2 \]

\[ v_3^2 \]

\[ v_1^3 \]

\[ v_2^3 \]

\[ v_3^3 \]
DCS objectives

- Property IC1 = unreachability of the states such that the predicate
  \[ \forall i \neq j, \text{Loy}_i \land \text{Loy}_j \land ((\text{Att}_i \land \text{Retr}_j) \lor (\text{Retr}_i \land \text{Att}_j)) \] is true:
  \[ S' = \text{mk}_\text{invar} (S, \forall i \neq j, \text{Loy}_i \land \text{Loy}_j \land ((\text{Att}_i \land \text{Retr}_j) \lor (\text{Retr}_i \land \text{Att}_j)) = \text{false}) \]

- Property IC2 = unreachability of the states such that the predicate
  \[ \forall i, \text{Loy}_c \land \text{Loy}_i \land ((\text{Att}_c \land \text{Retr}_i) \lor (\text{Retr}_c \land \text{Att}_i)) \] is true:
  \[ S' = \text{mk}_\text{invar} (S, \forall i, \text{Loy}_c \land \text{Loy}_i \land ((\text{Att}_c \land \text{Retr}_i) \lor (\text{Retr}_c \land \text{Att}_i)) = \text{false}) \]
Outcome of DCS

Most permissive environment model:

\[
\langle \text{Loy}_c \xrightarrow{e_c/t_c} \text{Tra}_c \rangle \parallel \langle \text{Loy}_1 \xrightarrow{e_1/t_1} \text{Tra}_1 \rangle \parallel \langle \text{Loy}_2 \xrightarrow{e_2/t_2} \text{Tra}_2 \rangle \parallel \langle \text{Loy}_3 \xrightarrow{e_3/t_3} \text{Tra}_3 \rangle
\]

DCS restricts the environment model by allowing only one general be a traitor.

It proves the theorem of Lamport et al. in the particular case of \( m = 1 \)

Proving the theorem in the general case would require parametric DCS
Outline

1. Introduction and context
2. Framework for automating the addition of FT with DCS
3. Specifying the hardware component failures
4. Handling different kinds of failures
5. Advanced DCS features
6. Case studies
7. Related work and conclusion
Related work

Fault-tolerant supervisors: [Cho & Lim, IEEE TRA, 1998]

Power transformer station controller: [Marchand & Samaan, IEEE TSE, 2000]

Automating the addition of fault-tolerance: [Kulkarni & Arora, FTRTFT’00]

Synthesis of fault-tolerant concurrent programs: [Attie, Arora & Emerson, ACM TOPLAS, 2004]

Supervisor for operating mode systems: [Kamach, Pietrac & Niel, IFAC WG’04]

Synthesis of quasi-static schedulings: [Cortadella et al, IEEE TCAD, 2005]

Fault-tolerant planning: [Jensen, Veloso & Bryant, WPUII’03]
Future work: distributed synthesis

Several complex issues

- Decentralized supervisory control [Lin & Wonham, IS’88]
  [Cieslak, Desclaux, Fawaz & Varaiya, IEEE TAC’88]
- How to make the controller tolerant to the failures of its processor?
- Undecidable without communication between the local controllers
  [Tsitsiklis, MCSS’89] [Tripakis, IEEE TAC’04]
Future work: distributed synthesis

Several complex issues

- Decentralized supervisory control [Lin & Wonham, IS’88]
  [Cieslak, Desclaux, Fawaz & Varaiya, IEEE TAC’88]
- How to make the controller tolerant to the failures of its processor?
- Undecidable without communication between the local controllers
  [Tsitsiklis, MCSS’89] [Tripakis, IEEE TAC’04]

Our proposal

- To distribute the controller afterwards
- Automatic distribution of LTSs [Caspi, Girault & Pilaud, IEEE TSE’99]
- Classical FT techniques to tolerate the failures of the processors and
  the communication links
- Lots of future work!
Conclusions: our contributions

- Framework for automating the addition of fault tolerance based on DCS
- All kinds of HW components: processors, communication links, actuators, and sensors
- All kinds of failures: crash, value, Byzantine
- Model of the plant: synchronous product of LTSs
- DCS objective: reachability and invariance properties
- Advanced DCS features: optimal DCS, conditioned objectives, synchronous observers
Conclusions: advantages of our framework

- Our method is fully **automatic**
- Separation of concerns
- It is **flexible**: it is easy to try several fault tolerant objectives
- The controlled system is fault tolerant **by construction**
- The failure recovery mechanism is **dynamic** (hence it does not induce too much redundancy overhead like static methods)
- With a **static guarantee** on the fault tolerance of the obtained system (unlike dynamic methods).