The COMON Case Study

Chaouki MAIZA, Simplice DJ OKO-DJ OKO, Erwan J AHIER, Pascal Raymond, Nicolas Halbwachs

SYNCHRON 2011
Outline

1. The Lurette Test Tool
2. COMON - Continuous Test Chain
3. Presentation of some oracles
4. Presentation of some scenarii
5. Demo
Lurette - Automatic Functionnal Tests of Reactive Systems
Lurette - Automatic Functionnal Tests of Reactive Systems

Black box functionnal test
- Comparing an implementation and a specification

Automatical test
- Generation of stimuli (SUT inputs)
- Tests results checking (oracle)

Based on a formal description of
- System Under Test properties
- Different hypothesis done on the environment

Reactive Systems
- The SUT reacts to the environment that it tries to control (feedback)
- A realistical environment must acts the same way
Data Flow - Global View

The environment can be seen as a Non-deterministic reactive system

Constrainted test vector generator

Chaouki MAIZA, Simplice DJOKO-DJOKO, Erwan JAHIER, Pascal Raymond, Nicolas Halbwachs

The Lurette Test Tool

SYNCHRON 2011
The environment can be seen as a non-deterministic reactive system. A constrained test vector generator is used to interact with the SUT (System Under Test) within this environment.
The environment can be seen as a
Non-deterministic reactive system
Constrainted test vector generator
The environment can be seen as a
Non-deterministic reactive system
Constrainted test vector generator
Lurette Test Process
Lurette Test Process
Lurette Test Process
Lurette Test Process

mauvais codage

problème

Lurette

Chaouki MAIZA, Simplice DJOKO-DJOKO, Erwan JAHIER, Pascal Raymond, Nicolas Halbwachs
Lurette Test Process

T < 100

Spéc imprécise

mauvais codage

problème

Lurette
Lurette Test Process

\[ T < 100 \]
valid => T<100

Spéc imprécise

mauvais codage

problème

Lurette
Lurette Test Process

\[ T < 100 \]
valid \( \Rightarrow \) \( T < 100 \)
valid and nominal \( \Rightarrow \) \( T < 100 \)

Spéc imprécise

mauvais codage

problème

Lurette
Lurette Test Process

- $T < 100$
- valid $\Rightarrow T < 100$
- valid and nominal $\Rightarrow T < 100$

Spéc imprécise

mauvais codage

problème

Lurette $\Rightarrow 100\%$ couv ?

ok

Chaouki MAIZA, Simplice DJOKO-DJOKO, Erwan JAHIER, Pascal Raymond, Nicolas Halbwachs

The COMON Case Study

SYNCHRON 2011
Lurette Test Process

- Lurette Test Tool
- Lurette Test Process
- On raffine (scénario)
- Non
- ok couv ?
- 100%
- valid and nominal => T<100
- valid => T<100
- T < 100
- Spéc imprécise
- mauvais codage
- problème
- Lurette
- ok
- 100% couv ?
- Non
- On raffine (scénario)
Lurette Test Process

La spécification est imprécise

Problème

Mauvais codage

On raffine (scénario)

Lurette

Oui

100% couv ?

Non

valid => T<100
valid and nominal => T<100

T < 100
Outline

1. The Lurette Test Tool
2. COMON - Continuous Test Chain
3. Presentation of some oracles
4. Presentation of some scenarii
5. Demo
COMON - Continuous Test Chain

Control-Command for civil nuclear systems

- ATOS WorldGrid
- Rolls-Royce Civil Nuclear
- Corys
- VERIMAG
## COMON - Continuous Test Chain

Control-Command for civil nuclear systems

- ATOS WorldGrid
- Rolls-Royce Civil Nuclear
- Corys
- VERIMAG

<table>
<thead>
<tr>
<th>N0</th>
<th>N1</th>
<th>N2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>Interface</td>
<td>Interface</td>
</tr>
</tbody>
</table>

Chaouki MAIZA, Simplice DJOKO-DJOKO, The COMON Case Study, SYNCHRON 2011
## COMON - Continuous Test Chain

Control-Command for civil nuclear systems

- ATOS WorldGrid **ADACS** : visualization
- Rolls-Royce Civil Nuclear **SCADE** : simulation
- Corys **ALICES** : simulation
- VERIMAG **Lurettte** : test manager

<table>
<thead>
<tr>
<th>RRCN -&gt; N1CL</th>
<th>Interface</th>
<th>N2 &lt;= ATOS</th>
<th>Interface</th>
<th>N1 &lt;= ALICES</th>
<th>Interface</th>
<th>N0 &lt;= ALICES</th>
</tr>
</thead>
</table>
1. Develop an executable model (in ALICES) of all the sub-systems.
1. Develop an **executable model** (in ALICES) of all the sub-systems.
2. Test those models by confrontation to a **formal** description of the system properties (written in natural language).
1. Develop an **executable model** (in ALICES) of all the sub-systems
2. Test those models by confrontation to a **formal** description of the system properties (written in natural language)
3. Integration of real systems (emulation/stimulation)
   - the oracles of point 2 can be reused
   - the models of point 1 can serve as references
1. Develop an **executable model** (in ALICES) of all the sub-systems.
2. Test those models by confrontation to a **formal** description of the system properties (written in natural language).
3. Integration of real systems (emulation/stimulation)
   - the oracles of point 2 can be reused
   - the models of point 1 can serve as references

Testing an open reactive system is made by **simulating its environment** and confronting it to an executable model.
1. Develop an **executable model** (in ALICES) of all the sub-systems.

2. Test those models by confrontation to a **formal** description of the system properties (written in natural language).

3. Integration of real systems (emulation/stimulation)
   - the oracles of point 2 can be reused
   - the models of point 1 can serve as references

Testing an open reactive system is made by **simulating its environment** and confronting it to an executable model.

The central point in this approach is obtaining, testing and validating an **executable model** earlier.
Experiments done during this case study
Experiments done during this case study

Formalisation from natural language into
  - Lustre for the oracles
  - Lutin for the environments/stimulators
Experiments done during this case study

Formalisation from natural language into
- Lustre for the oracles
- Lutin for the environments/stimulators

Macrolibrary of generic nodes to help writing oracles and stimulators
- Checking that a numerical value remains in a specific range
- Calculating different system states (situation):
  - nominal, low power, 2/3-1/3, loop 1, emergency
- Detecting the stability of a variable
- Doing an action and waiting for the stability
- etc.
Outline

1. The Lurette Test Tool
2. COMON - Continuous Test Chain
3. Presentation of some oracles
4. Presentation of some scenarii
5. Demo
An oracle supervising threshold crossings

When a high threshold is crossed, the N2 must show an alarm after 5 seconds and reciprocally.
An oracle supervising threshold crossings

When a high threshold is crossed, the N2 must show an alarm after 5 seconds and reciprocally.

\[
\text{alarm\_raised} = \text{timer}(\text{alarm, 5.0});
\]
\[
\text{high\_threshold\_crossed} = \text{timer}((v > \text{high\_threshold}), 5.0);
\]

\[
\text{ok1} = (\text{falling\_edge(hi} \text{gh\_threshold\_crossed)} \Rightarrow \text{alarm\_raised});
\]
\[
\text{ok2} = (\text{alarm} \Rightarrow \text{high\_threshold\_crossed});
\]
An oracle supervising threshold crossings

When a high threshold is crossed, the N2 must show an alarm after 5 seconds and reciprocally.

\[
\begin{align*}
\text{alarm\_raised} & = \text{timer}(\text{alarm}, 5.0); \\
\text{high\_threshold\_crossed} & = \text{timer}(v > \text{high\_threshold}), 5.0; \\
\text{ok1} & = (\text{falling\_edge}(\text{high\_threshold\_crossed}) \Rightarrow \text{alarm\_raised}); \\
\text{ok2} & = (\text{alarm} \Rightarrow \text{high\_threshold\_crossed}); \\
\end{align*}
\]

NB : it is the same for low, very high and very low thresholds
An oracle supervising the system situations

We extracted different system situations from the natural language description: nominal, low power, emergency.
Then we expressed relations between these properties:
nominal $\iff$ low power $\iff$ emergency
An oracle supervising the system situations

We extracted different system situations from the natural language description: nominal, low power, emergency

Then we expressed relations between these properties:

 nominal <-> low power <-> emergency

```
node oracle_situation(nominal, low_power, emergency : bool)
returns(ok:bool);
  ok = true ->
    (pre nominal => nominal or low_power) and
    (pre low_power => low_power or nominal or emergency) and
    (pre emergency => emergency or low_power);
```
In nominal mode, after each operator order, all the sensors values must be stables after 5 minutes
Presentation of some oracles

An oracle supervising the system stability

In nominal mode, after each operator order, all the sensors values must be stable after 5 minutes

\[
\begin{align*}
C &= \text{true\_since(no\_new\_order, 300.0) and nominal;} \\
ok &= (C \Rightarrow \text{is\_stable})
\end{align*}
\]
Presentation of some oracles

An oracle supervising the system stability

In nominal mode, after each operator order, all the sensors values must be stabiles after 5 minutes

\[
C = \text{true\_since(no\_new\_order, 300.0)} \quad \text{and nominal;}
ok = (C \Rightarrow \text{is\_stable})
\]

Covering this test means generating a sequence where \( C \) is true
An oracle supervising the system stability

In nominal mode, after each operator order, all the sensors values must be stables after 5 minutes

$$C = \text{true\_since(no\_new\_order, 300.0) and nominal;}$$
$$ok = (C \Rightarrow \text{is\_stable})$$

*Covering* this test means generating a sequence where $C$ is true

Yet, the need of a scenario where the operator orders dont change too quickly
Outline

1. The Lurette Test Tool
2. COMON - Continuous Test Chain
3. Presentation of some oracles
4. Presentation of some scenarii
5. Demo
A (pseudo-)random failure generator

Goal: generating \( n \) failures randomly without triggering classified actions
A (pseudo-)random failure generator

Goal: generating \( n \) failures randomly without triggering classified actions

Emergency: a constraint specifying the failures that will trigger a classified action
- 2 redundant components failing at the same time
- 3 ou 4 quadri-redundant sensors failing together
A (pseudo-)random failure generator

Goal: generating \( n \) failures randomly without triggering classified actions

Emergency: a constraint specifying the failures that will trigger a classified action
- 2 redundant components failing at the same time
- 3 ou 4 quadri-redundant sensors failing together

We randomly choose between the solutions of the constraints
A (pseudo-)random failure generator

Goal: generating $n$ failures randomly without triggering classified actions

**Emergency**: a constraint specifying the failures that will trigger a classified action
- 2 redundant components failing at the same time
- 3 ou 4 quadri-redundant sensors failing together

We randomly choose between the solutions of the constraints

A minimal scenario to
- avoid switching between failures at each cycle
- switch anyway after 200 cycles (or after a `reset`)

```
node (n : int ; reset : bool) returns (P1, P2, P3, P4, P5, P6) =
  let Emergency = two_false (P1, P2) or three_false (P3, P4, P5, P6) in
  let nb_failures = (if P1 then 1 else 0) + ... + (if P6 then 1 else 0) in
  loop {
    (not Emergency and nb_failures = n) | nb_failures = n
  } by loop 200 { maintain (P1) and ... maintain (P6) and not reset}
```
A (pseudo-)random failure generator

Goal : generating $n$ failures randomly without triggering classified actions

Emergency : a constraint specifying the failures that will trigger a classified action
- 2 redundant components failing at the same time
- 3 ou 4 quadri-redundant sensors failing together

We randomly choose between the solutions of the constraints

A minimal scenario to
- avoid switching between failures at each cycle
- switch anyway after 200 cycles (or after a reset)

```scheme
(node (n: int; reset: bool) returns (P1, P2, P3, P4, P5, P6) =
    let Emergency = two_false(P1, P2) or three_false(P3, P4, P5, P6) in
    let nb_failures = (if P1 then 1 else 0) + ... + (if P6 then 1 else 0) in
    loop {
        (not Emergency and nb_failures = n) |> nb_failures = n
    } fb
    by loop 200 {maintain(P1) and ... maintain(P6) and not reset}
)
```

Failures-i.lut; Failure Demo
A (pseudo-)random operator scenario

Change the opening order of a gate while supervising the values of sensors (level)
A (pseudo-)random operator scenario

Change the opening order of a gate while supervising the values of sensors (level)

1. Choose a target opening order ([0 ; 100])
A (pseudo-)random operator scenario

Change the opening order of a gate while supervising the values of sensors (level)

1. Choose a target opening order ([0 ; 100])
2. Use a node that change the opening order step by step until it reaches the target
A (pseudo-)random operator scenario

Change the opening order of a gate while supervising the values of sensors (level)

1. Choose a target opening order \([0 ; 100]\)
2. Use a node that change the opening order step by step until it reaches the target
3. When the target is reached, restarts in 1.
A (pseudo-)random operator scenario

Change the opening order of a gate while supervising the values of sensors (level)

1. Choose a target opening order \([0 ; 100]\) 
2. Use a node that change the opening order step by step until it reaches the target
3. When the target is reached, **restarts** in 1.

In order to bring the opening order step by step to the target:

1. Keep the same order **until** obtaining system stability
A (pseudo-)random operator scenario

Change the opening order of a gate while **supervising** the values of sensors (level)

1. Choose a target opening order ([0 ; 100])
2. Use a node that change the opening order **step by step** until it reaches the target
3. When the target is reached, **restarts** in 1.

In order to bring the opening order step by step to the target:

1. Keep the same order **until** obtaining system stability
2. Getting closer to the target order by at most **step** amount
3. If the target is not reached, **restarts** in 1.
A (pseudo-)random operator scenario

Change the opening order of a gate while supervising the values of sensors (level)

1. Choose a target opening order \([0; 100]\)
2. Use a node that change the opening order step by step until it reaches the target
3. When the target is reached, restarts in 1.

In order to bring the opening order step by step to the target:

1. Keep the same order until obtaining system stability
2. Getting closer to the target order by at most step amount
3. If the target is not reached, restarts in 1.

This illustrate Lutin ability to express complex scenarii.
A (pseudo-)random operator scenario

Change the opening order of a gate while supervising the values of sensors (level)

1. Choose a target opening order ([0 ; 100])
2. Use a node that changes the opening order step by step until it reaches the target
3. When the target is reached, restarts in 1.

In order to bring the opening order step by step to the target:

1. Keep the same order until obtaining system stability
2. Getting closer to the target order by at most step amount
3. If the target is not reached, restarts in 1.

This illustrates Lutin's ability to express complex scenarii.

-> operator.lut ; Operator demo
Outline

1. The Lurette Test Tool
2. COMON - Continuous Test Chain
3. Presentation of some oracles
4. Presentation of some scenarii
5. Demo
Demo!

The 4 systems communicate
Lurette pilots the tests
Use the 2 scenarios presented before
Use (most of) the oracles presented before