AltaRica models and tools for system safety assessment
Best practices and lessons learnt from the aerospace domain

29/10/2014
Christel.Seguin@onera.fr
Tutorial outline

• System Safety Assessment

• AltaRica Basics
  • AltaRica Data Flow Language
  • Tools

• Modelling process

• FAQ about models
System safety analysis
and limits of current approaches
**System from safety perspective**

**System**: any material or immaterial “object”, or set of material or immaterial objects or operators, that is elaborated, adapted, set-up or used by humans with the specific purpose to *provide some intended service to its clients*. Eg: air traffic control, aircraft + pilot, flight-control system, computers, sensors, actuators …

![Diagram of an aircraft with labels for different systems]

- A380, Rafale, B787
- Flight control, hydraulic, electrical, flight warning, …
- Flight control computers, …
- Aircraft systems
- Equipment
- Aircraft
System from safety perspective

**Safety**: “system state where an acceptable level of risk with respect to fatality or injury of human is not exceeded”.

**Failure**: deviation of the service provided by the considered system, with respect to the expectations.

**Failure mode**: way by which a failure appears (e.g. fail-silent, erroneous value, …)

**Fault**: cause of a potential failure

**Undesirable event**: Any adverse event or situation that could be due to the considered system and its potential failures. Also called **Failure Condition**
Example of Safety Critical System

ATA29: Hydraulic system

SSA ATA29: quantitative & qualitative requirements

- "Total loss of hydraulic power is classified Catastrophic,
- the probability rate of this failure condition shall be less than $10^{-9} /FH$.
- No single event shall lead to this failure condition"

Design solution to meet the requirements

- redundancies & component reliability
Processes: certification / safety assessment / V&V

Certification preparation process
- Aircraft Airworthiness requirement assignment
- CRI for novelties and associated means of compliance
- Certification plan

Safety assessment/Safety analysis process

Safety Plan

Function and implementation oriented safety assessment:

Functional Hazard Assessment (FHA):
Failure Condition Identification and allocation of safety objectives (qualitative and quantitative). Aircraft level FHA and System level FHA

Multi system and system Safety Assessment
- Multi systems (Aircraft level): PASA/ASA (Preliminary and final Aircraft Safety Assessment)
- System level: PSSA/SSA (Preliminary and final System safety Assessment)

Common Cause Analysis (CCA)
- Particular Risk Analysis (PRA): e.g. engine burst, bird impact.
- Human Error Analysis (HEA): Crew error, Maintenance error.
- Common Mode Analysis (CMA)
- Zonal Safety Analysis (ZSA): Installation review

Safety Validation/Verification and safety assurance process

Certification readiness
Final certification
Acceptable means of compliance deliverables:
Safety dossier (safety synthesis / Safety case, FHA, SSA, ASA, PRA, etc.)

Certification activity
Safety assessment/analysis activity
V/V and Assurance Process activities
Classical failure propagation models and safety assessment techniques (cf ARP 4761)

• Failure mode and effect analysis (FMEA)
  – Model: from a local failure to its system effects / natural languages

<table>
<thead>
<tr>
<th>FUNCTION NAMES</th>
<th>FUNCTION CODE</th>
<th>FAILURE MODE</th>
<th>RATE</th>
<th>FLIGHT PHASE</th>
<th>FAILURE EFFECT</th>
<th>DETECTION METHOD</th>
<th>COMMENTS</th>
</tr>
</thead>
</table>

Functional FMEA template

• Fault tree analysis (FTA)
  – Model: from a system failure to its root causes / boolean formulae
  – Computation: minimal cut sets / probability of occurrence of top event

• And also Markov chain ....

FT unannounced loss of wheel braking
Drawbacks of the classical Safety Assessment Approaches

• Fault Tree, FMEA
  - Give failure propagation paths without referring explicitly to a commonly agreed system architecture / nominal behavior =>
    - Misunderstanding between safety analysts and designers
    - Potential discrepancies between working hypothesis

• Manual exhaustive consideration of all failure propagations become more and more difficult, due to:
  - increased interconnection between systems,
  - integration of multiple functions in a same equipment
  - dynamic system reconfiguration
Model based safety assessment rationales

• Goals
  – Propose formal failure propagation models closer to design models
  – Develop tools to
    • Assist model construction
    • Analyze automatically complex models
  – For various purposes
    • FTA, FMEA, Common Cause Analysis, Human Error Analysis, …
    • since the earlier phases of the system development

• Approaches
  Extend design models (Simulink, SysML, AADL…) with failure modes
  Build dedicated failure propagation models (Figaro, AltaRica, Slim…)
  Transform into analyzable formalisms (boolean formulae, automata, …)
  Develop specialized analysis tools
Basics of AltaRica dataflow language
AltaRica language at a glance

- Language designed in late 90's at University of Bordeaux
  - for modelling both *combinatorial* and *dynamic* aspects of *failure propagation*
  - in a *hierarchical* and *modular* way
  - *formally*.

- Typical content of a basic AltaRica *node*

  ![Diagram](image)

  - **Input flows**
  - **Assertion**
    - `output = f (inputs, states)`
  - **Output flows**
  - **Transitions**
    - normal state \(\xrightarrow{}\) error state \(\xrightarrow{}\) normal state
  - **fault occurrence event**
A leading example: the basic reliability block

- Let be a basic system component *Block* that
  - receives
    - one Boolean input I,
    - an activation signal A and
    - a resource signal R.
  - produces
    - a Boolean output O

- **Block** performs *nominally* the following transfer law
  - O is true iff I, A and R are true.

- **Block** may *fail*.
  - In this case, the output O is false.

- **Initially**, the block performs the nominal function
From concepts to a concrete syntax:

node Block
  flow
    O:Bool:out; 
    I, A, R :Bool: in;
  state
    ok: Bool;
  event
    fail;
  init
    ok := true;
  trans
    ok |- fail -> ok := false;
  assert
    O = (I and A and R and ok);
  edon
AltaRica semantics

From AltaRica code to mode automata

```
ok=true
O=(I and A and R)

ok=false
O=false
```

```
ok=true
O=(I and A and R)

fail

ok=false
O=false
```
AltaRica semantics

From mode automata to **Kripke structure** =

\(<\text{Configurations, Data Assignation in configurations, Relations between configurations}>\)

**Runs** of a mode automata =

paths of the derived Kripke structure that start from one possible initial configuration

*8 possible initial states for the basic block*

- ok=true, I=A=R=O=true
- ok=true, l=A=true, R=O=false
- ok=true, I=A=R=O=false
- ok=false, I=A=R=true, O=false
- ok=false, I=A=true, R=O=false
- ok=false, I=A=R=O=false

......
Operational semantics of a mode automata (or "how to build the equivalent Kripke structure?")

- **Configuration=**
  - Assignment of a value to all flows and state variables

- **Computation of initial model configurations**
  - Assign the values set in the `init` clauses to all state variables
  - If all input flows are connected to the output of one component
    - the initial configuration is unique, compute the values of output flows according to the laws in the `assert` clauses
  - Else, there are potentially several initial configurations
    - consider all combinations of domain values for the free input variables
    - Complete all potential configurations by computing the values of output flows according to the laws in the `assert` clauses
Operational semantics of a mode automata (or "how to build the equivalent Kripke structure?")

- **Enabled transition** =
  - transition whose guard is true in the current model configuration

- **Computation of the next model configurations**
  - For each enabled transition, build a next configuration

  - In each next configuration:
    - Assign state values according to the selected transition list of assignments
    - Compute the values of input / output flows as in the initial configuration according to the laws in the `assert` clause + duplication of configurations if there are free inputs

- Iterate the computation until no new configuration is reached
Internal operations on mode automata

- Parallel composition: free product of mode automata
  - preserves all states, variables, transitions, assertions
  - interleaving parallelism (only one transition at a time)
- Ex: two parallel Boolean blocks

```
block1.ok = block2.ok = true
block1.O = (block1.I and block1.A and block1.R)
block2.O = (block2.I and block2.A and block2.R)
```

```
block1.ok = false, block2.ok = true
block1.O = false
block2.O = (block2.I and block2.A and block2.R)
```

```
block1.ok = true, block2.ok = false
block1.O = (block1.I and block1.A and block1.R)
block2.O = false
```

```
block1.ok = block2.ok = false
block1.O = false
block2.O = false
```

```
Internal operations on mode automata

- Interconnection: mapping an input of an automaton with an output of another automaton
  - preserves all states, variables, transitions, assertions
  - Introduces new assertions: Block2.I = Bloc1.O for all pairs of connected interfaces
  - interleaving parallelism (only one transition at a time)
  - ! allowed only if variables are not circularly defined
- Ex: two series blocks

```
block1.ok=true, block2.ok=false
block1.O=block2.I=
(block1.I and block1.A and block1.R)
block2.O=false
```

```
block1.ok=false, block2.O=true
block1.O=block2.I=false
block2.O=false
```

```
block1.ok=false, block2.ok=false
block1.O=block2.I=false
block2.O=false
```

```
block1.ok=true, block2.ok=false
block1.O=block2.I=
(block1.I and block1.A and block1.R)
block2.O=false
```

```
block1.ok=false, block2.O=true
block1.O=block2.I=false
block2.O=false
```

```
block1.ok=false, block2.ok=false
block1.O=block2.I=false
block2.O=false
```

```
block1.ok=true, block2.ok=false
block1.O=block2.I=
(block1.I and block1.A and block1.R)
block2.O=false
```

```
block1.ok=false, block2.O=true
block1.O=block2.I=false
block2.O=false
```

```
block1.ok=false, block2.ok=false
block1.O=block2.I=false
block2.O=false
```
Internal operations on mode automata

- Parallel composition with event grouping: synchronized product of mode automata
  - preserves all states, variables, transitions of ungrouped event, assertions
  - Introduces new grouped transitions $E: <e_1, ... e_n>$
    - Initially $G_1 |-> e_1 -> Ass_1 .... G_n |-> e_n -> Ass_n$
    - Replaced by
      strong synchronisation: $G_1$ and... and $G_n |-> E -> Ass_1; ...Ass_n$
      broadcast: $G_1$ or... or $G_n |-> E$ if $G_1$ then $Ass_1; ...$if $G_2$ then $Ass_n$
  - interleaving parallelism (only one atomic or a grouped transition at a time)

- Ex: modeling of common cause of failures not propagated by interfaces
  - Explosion, fire, loss of power, ... of a zone
Event weight

- Weights may be allocated to events in order to constrain the model simulation

- They are introduced in two ways
  - By definition of a partial priority order between the model events
    - Ex: A<B, A<C, B<C
    - at each simulation steps, the triggered event is one of the enable events with the highest priority
  - By allocation of probability laws to event occurrences at time t
    - Ex: A: Dirac(0)
      When A is enabled at time t, it SHALL be triggered at t
    - Ex: B: Exp(-λt)
      When B is enabled at time t, its occurrence probability is Exp(-λt)
Comments about mode automata

• They encompass
  • Boolean formulae thanks to assertion part
  • labeled transition system (e.g. Petri Nets) thanks to the transition part

• They enable compositional system models thanks to well founded operations on automata
Demo: a simplified version of the hydraulic system of the A320 edited with Cecilia OCAS Dassault Aviation
Safety assessment tools
Several tools for analysing AltaRica dataflow models

- Cecilia OCAS from Dassault Aviation
  - Used for the first time for certification of flight control system of Falcon 7X in 2004
  - Tested by contributors of ARP 4761 (cf MBSA appendix)
- AltaRica free suite from Labri
  - compatible with data flow restriction
  - http://altarica.labri.fr/wp/
- Safety Designer from Dassault System
- Simfia from APSYS Airbus group
- RAMSES from Airbus
- And plugins to independant tools
  - NU-SMV (FBK Trento), MOCA-RP (Satodev Bordeaux), Arc (LaBri Bordeaux) EPOCH (ONERA Toulouse)….

+ potential compatibility with tools for AltaRica 3.0 ?
Several functions provided by the various tools

- Simulation
- Fault tree generation
- Sequence generation
- FMEA generation
- Stochastic simulation (MOCA-RP)
- Model-checking (ARC, Nu-SMV)
- Probabilistic model-checking (EPOCH)

Minimal set supported by all integrated workbenches
Demo of the minimal set of functions with Cecilia OCAS

Interactive simulation = user driven exploration of the Kripke structure

→ play simple combination of failures (in the style of FMEA)
OCAS Fault-Tree generation

- The fault tree can be exported to other tools (e.g. Arbor,...) to compute of minimal cut sets and probabilities
Demo of the minimal set of functions with Cecilia OCAS

- OCAS Sequence Generator
  - Automatic generation of failure sequences that lead to the observation of the failure conditions
  - Limit on the number of failures to be considered

```c
/*
orders(MCS('hydrau_total_loss.O.true')) =
orders product-number
3  35
4  56
total  91
end */
products(MCS('hydrau_total_loss.O.true')) =
{ 'EDPg.fail_loss', 'EMPb.fail_loss', 'disty.fail_loss'}
{ 'EDPg.fail_loss', 'EMPb.fail_loss', 'rsvy.fail_loss'}
{ 'EDPg.fail_loss', 'Elec2.fail_loss', 'disty.fail_loss'}
{ 'EDPg.fail_loss', 'Elec2.fail_loss', 'rsvy.fail_loss'}
...```
! Classes of model

- Deterministic model
  - there is no configuration such that at least two transitions with the same event name are enabled and the next configurations are different,
  - In a deterministic model, the current configuration is a function of the sequence of selected events since the initialisation

- Static/Dynamic Model (for deterministic models)
  - **Static** Model: the order of the events in sequence as no influence on the current configuration
  - **Dynamic** Model: the last property is not verified => use sequence generation rather than fault tree generation
Tool Performances 1

- MISSA (2008-2011) integrated case study:
  - Collaborative model of the systems used by the deceleration function
**Tool Performances 2**

- Models for Preliminary System Safety Assessments: tractable

<table>
<thead>
<tr>
<th>System</th>
<th>Organisation Responsible</th>
<th>Modelling Approach</th>
<th>Number of Basic Components</th>
<th>Overall number of events</th>
<th>Number of FCs from FHA</th>
<th>MCS computation time (to max of 3 failures)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wheel Braking System</td>
<td>Airbus (UK)</td>
<td>FEM</td>
<td>517</td>
<td>974</td>
<td>9</td>
<td>Minutes to hours depending on the observer</td>
</tr>
<tr>
<td>Ground Spoilers &amp; Thrust Reversers</td>
<td>EADS APSYS</td>
<td>FEM</td>
<td>111</td>
<td>100</td>
<td>6</td>
<td>~40 minutes</td>
</tr>
<tr>
<td>Weight on Wheels</td>
<td>Dassault Aviation</td>
<td>FEM</td>
<td>35</td>
<td>48</td>
<td>1</td>
<td>&lt;1 minutes</td>
</tr>
<tr>
<td>Integrated Modular Avionics</td>
<td>THALES</td>
<td>FEM/FLM Hybrid</td>
<td>80</td>
<td>200</td>
<td>25</td>
<td>~5 minutes</td>
</tr>
<tr>
<td>Electrical Power Distribution</td>
<td>Alenia Aeronautica</td>
<td>FEM/FLM Hybrid &amp; ‘pure’ FEM</td>
<td>~30</td>
<td>~50</td>
<td>5</td>
<td>~5 minutes</td>
</tr>
<tr>
<td>Hydraulic Power Distribution</td>
<td>Airbus (Germany)</td>
<td>FEM/FLM Hybrid</td>
<td>78</td>
<td>70</td>
<td>8</td>
<td>&lt;1 minutes</td>
</tr>
</tbody>
</table>

- Detailed models are not easily "provable"
  - Hybrid models mixing discrete and continuous time evolution laws
  - Timed models, to reason about system delays
Needs for new tools

• How to exploit easily thousand of cut sets?
  – Tool to fold / unfold cutset details (eg APSYS Sirocco)
  – Optimization of DAL allocation (eg ONERA DAL calculator)

• ... 

• Design optimization
  – Safety budget allocation
  – Mapping of function over resources
  – Synthesis of controller under safety constraints

• One model for all analysis: not realistic
  => management of model consistency all over the design
Modelling guidelines
Reminder: example of AltaRica component

How to build such a model starting from technical documents?

```plaintext
node Block
  flow
    O: Bool: out;
    I, A, R : Bool: in;
  state
    ok: Bool;
  event
    fail;
  trans
    ok | fail -> ok := false;
    assert
      O = (I and A and R and);
  init
    ok := true;
  Edon
```

Abstract domain of value

```
ok = true
O = (I and A and R)
```

```
fail
ok = false
O = false
```
Presentation of a leading example: COM/MON pattern

Case study: a command/monitoring pattern of safety architecture to compute correct orders even if one fault occurs.

Structure (cf figure below):
- Two numerical functions Fc and Fm
- A comparator cmp that raises an alarm when the function outputs are different
- A contactor ct that selects a function output. Initially, the command output is selected. One switches to no output as soon as an alarm is raised.

The functions have two failure modes:
- they may produce an erroneous output
- they may produce no output at all.

The safety requirements of interest for this pattern are:
- FC_B1: an erroneous output is major.
- FC_B2: the output loss is minor
Information needed before starting the failure propagation model of a system

Model purpose:
- fully cover the system under study
- enable the observation and the analysis of a set of failure conditions on this perimeter

=> Before starting the model, identify:
- Components in the study perimeter and their granularity
  * Ex: 4 internal components in the Com-Mon + 4 input components + 1 output component
  The internal components will be decomposed in later assessment phases, according to the function implementation
- Expression of the failure conditions in relationship with the model perimeter
  * Ex: loss of aircraft control ~> no output provided by component "ct"
- Propagation laws inside each atomic components that results both:
  * from functions performed in the nominal case
    * Ex: safety barriers = "cmp" + "ct"
  * from potential faults and observable failure modes
    * Ex: from FMEA, failure mode = erroneous and loss
Meaningful abstract domains for the component interfaces depend on:

- Failure mode of the component inputs
- Failure modes of the component output that shall be propagated and observed
  - Ex: a domain for numerical data =\{ok, lost, erroneous\}
- Class of input or output nominal values that impact the failure propagation results
  - Ex: alarm_raised: bool

To get homogeneous modeling hypothesis, use as much as possible the same abstract domain for all the interfaces of the same kind (same technology, same role, ...
Meaningful component internal states and related events

- States: internal parameter that impact the functions provided by the component
  - operational modes: \textit{ex manual /autopilot mode}
  - failure modes
  - "artificial" states: used to store a value
- Events: used to observe the change of the internal states
  - environment order,
  - failure occurrence ... :
Component behavior

Transitions = dynamic of the internal states (nominal or not)
  • Identify guards and effects of each event;
  • Set priority between events when needed
    ! Infinite loop of events with highest priority

Assertions = computation laws over the abstract domains:
  • For each output, build a decision table:

<table>
<thead>
<tr>
<th>Output Name</th>
<th>Condition over inputs &amp; modes</th>
<th>Output Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>not ok</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td>I&amp;A&amp;R</td>
</tr>
</tbody>
</table>

  $O = \text{case} \{ 
    \text{not ok : false,} 
    \text{ else (I and A and R) } 
  \};$

  • ! Circular definitions shall be broken
  • introduction of artificial states to store previous value and serialize the definition
1. Define a library containing the following components
   - inputs,
   - function,
   - comparator,
   - contactor and
   - FC observers

2. Build a system COM/MON with this library of components

3. Assess the COM/MON system
   - Generate sequences for each FC
   - Are the safety requirements met? Justify your answer.
   - Can the COM and MON units share the same inputs or the same power source? Justify your answer.
Main abstract domain for components

/* Code of the "function" component

Profile of the "function" component

Initialization
and transition of states
for the "function" component

Table to decide the value of
the component output in any case

Probability laws of
the component failure modes

```plaintext
domain FailType = {err, lost, ok};

node Function
  flow
    O:FailType:out;
    A:bool:in;
    I:FailType:in;
  state
    S:FailType;
  event
    fail_loss,
    fail_err;
  init
    S := ok;
  trans
    (S = ok) | fail_loss -> S := lost;
    (S = ok) | fail_err -> S := err;
  assert
    O = case {
      ((S = ok) and A) : I,
      ((S = lost) or (not A)) : lost,
      else err
    };
  extern
    law <event fail_loss> = exp(1.0E-4);
    law <event fail_err> = exp(1.0E-5);
edon
```
/* Code of the "data input " component

node Source
  flow
    O:FailType:out;
  state
    S:FailType;
  event
    fail_loss,
    fail_err;
  init
    S := ok;
  trans
    (S = ok) | fail_loss -> S := lost;
    (S = ok) | fail_err -> S := err;
  assert
    O = S;
  extern
    law <event fail_loss> = exp(1.0E-4);
    law <event fail_err> = exp(1.0E-5);
edon
/* Code of the "power input " component 

node Power
  flow
    O:bool:out;
  state
    S:FailType;
  event
    fail_loss;
  init
    S := ok;
  trans
    (S = ok) |- fail_loss -> S := lost;
  assert
    O = case {
      (S = ok) : true,
      else false
    };
  extern
    law <event fail_loss> = exp(1.0E-4);
edon
COM/MON correction

/* Code of the "comparator" component

node Comparator
  flow
    In1:FailType:in;
    In2:FailType:in;
    Out:bool:out;
  assert
    Out = case {
      (In1 = In2) : false,
      else true
    };
  edon

! The exercise does not specify failure mode of the comparator =>
there is neither declared failure event nor a transition part in the code
### COM/MON correction

<table>
<thead>
<tr>
<th>Contactor without memory</th>
<th>Contactor with memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>node Contactor_be1</td>
<td>node Contactor_be2</td>
</tr>
<tr>
<td>flow</td>
<td>flow</td>
</tr>
<tr>
<td>In:FailType:in;</td>
<td>In:FailType:in;</td>
</tr>
<tr>
<td>Alarm:bool:in;</td>
<td>Alarm:bool:in;</td>
</tr>
<tr>
<td>Out:FailType:out;</td>
<td>Out:FailType:out;</td>
</tr>
<tr>
<td>assert</td>
<td>state</td>
</tr>
<tr>
<td>Out = case {</td>
<td>Open:bool;</td>
</tr>
<tr>
<td>Alarm : lost,</td>
<td>event</td>
</tr>
<tr>
<td>else In</td>
<td>open_ct;</td>
</tr>
<tr>
<td>}</td>
<td>Init open:= false;</td>
</tr>
<tr>
<td>edon</td>
<td>trans</td>
</tr>
<tr>
<td></td>
<td>not open and alarm</td>
</tr>
<tr>
<td></td>
<td>assert</td>
</tr>
<tr>
<td></td>
<td>Out = case {</td>
</tr>
<tr>
<td></td>
<td>Open : lost,</td>
</tr>
<tr>
<td></td>
<td>else In</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>extern</td>
</tr>
<tr>
<td></td>
<td>law &lt;open_ct&gt; =Dirac(0);</td>
</tr>
</tbody>
</table>

**Nominal event that open the contactor**

**Dirac law of the control event:**
*open_ct* has a highest priority than stochastic event like failure
Summary: template to assist component specification

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Service presentation (role, conditions of use, …)</th>
<th>Generic service that provides a correct output if and only if it receives an input, is activated, has the needed resource and is ok.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic_service</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Component interfaces**

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>Role</th>
<th>Orientation</th>
<th>Value type</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Service</td>
<td>In</td>
<td>Bool</td>
</tr>
<tr>
<td>A</td>
<td>Control</td>
<td>In</td>
<td>Bool</td>
</tr>
<tr>
<td>R</td>
<td>Resource</td>
<td>In</td>
<td>Bool</td>
</tr>
<tr>
<td>O</td>
<td>Service</td>
<td>Out</td>
<td>Bool</td>
</tr>
</tbody>
</table>

**Component control states**

<table>
<thead>
<tr>
<th>Control State Name</th>
<th>Role</th>
<th>Value type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ok</td>
<td>Reliability</td>
<td>(range of modes or attribute values)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bool (ok or not ok)</td>
</tr>
</tbody>
</table>

**Service measures**

<table>
<thead>
<tr>
<th>Measure Name</th>
<th>Measure parameters</th>
<th>Measure estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure rate</td>
<td>lambda</td>
<td>10.9 per hour</td>
</tr>
</tbody>
</table>

**Specification of the variations of “basic_service”**

<table>
<thead>
<tr>
<th>Variation Name</th>
<th>Guards</th>
<th>Triggers (condition/event)</th>
<th>Effects over states/interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service_ok</td>
<td>True</td>
<td>I and A and R and Ok/-</td>
<td>O=true</td>
</tr>
<tr>
<td>Service_ko</td>
<td>True</td>
<td>Not (I and A and R and Ok)/-</td>
<td>O=false</td>
</tr>
<tr>
<td>Component_faulty</td>
<td>True</td>
<td>True/fail (lambda)</td>
<td>Ok=false</td>
</tr>
</tbody>
</table>
FAQ
Modelling nominal dynamic reconfiguration

- Example: contactor of COM-MON

- Option 1: (cf previous slides)
  - The reconfiguration order is received via a flow
  - It triggers a deterministic nominal event with Dirac(0) law

- Option 2:
  - The reconfiguration order is synchronized with a specific fault occurrence
Modelling cascading failure

• Exemple: component destroyed when it receives over voltage

• Option 1:
  • The input failure mode is received via a flow
  • It triggers a deterministic failure event with Dirac(0) law

• Option 2:
  • The cascading failure is synchronized with the primary failure event

• The cut sets contain only the primary failure
Modelling Common Cause of Failure

• Exemple: component destroyed by fire in a same zone

• Option 1: the common point (eg the zone) is a component of the model
  • Possible to use a flow between the zone and its components

• Option 2: the common point is not a component of the model
  • Use synchronisation of the failure event of all the components in the zone
Modelling control loop

• Exemple

Device position

Control function ➔ Controled Device

order

• Option 1
  • Add a delay i.e. a component which stores the previous device position + use sequence generation

• Option 2
  • Break the propagation path when the failure condition occurred i.e. put two inputs to the control functions, the normal one + a status which inhibits the function computation
Modelling acausal system

- Example: a leakage in the hydraulic system is propagated backward and forward in the system.

- Option: each connecting interface is modelled by two flows, an input and output flow.

```plaintext
node REVBOOL_REVBOOL_blockR3
flow
  iconic : [1,2] : out ;
  O : HydBool_HydBool_bibool : out ;
  O^nom : bool : out ;
  O^rev : bool : in ;
  I : HydBool_HydBool_bibool : in ;
  I^nom : bool : in ;
  I^rev : bool : out ;
state
  S : bool ;
event
  fail_loss ;
init
  S := true ;
```

// I : input (nominal and reverse flow)
// A : activation
// O : outputs (nominal and reverse flow)
// S : status (true means ok, false means failed)
trans
// fail0 event sets the status of the component to false
S=true | fail_loss -> S=false ;
assert
// a block produces output if it receives input and has not failed
// and it is activated
O^nom= (I^nom and S) ;
// IF a block has not failed, is activated
// THEN it produces reverse input WHENEVER it receives reverse output
// ELSE it produces reverse input
I^rev=O^rev ;
```
Is MBSA limited to technical system?

- Possible to mix technical system and human procedures (cf Thalès model)
Thank you to partners of fp 5,6,7 projects for aeronautics and partner of space and helicopter
Thank you to past and current colleagues

ONERA colleagues {firstname}.{lastname}@onera.fr
- Charles Castel, Pierre Bieber, Claire Pagetti, Eric Bensana, Remi Delmas, Guillaume Infantes, Florent Teichteil, Julien Brunet, Xavier Pucel

Students
- PhD: Christophe Kehren, Laurent Sagaspe, Romain Bernard, Sophie Humbert, Romain Adeline, Jean-Charles Chaudemard, Kossi Tiassou, Christelle Jacob, Sebastien Maitrehenry, Jonathan Sprauel
- Master: Thomas Lucas, Beatrice Toussaint, Lucia Sanz, Andra Tonie

Other lab colleagues at LaBri, LIX, ISAE, LAAS, IRIT, …

For more information a lot of papers can be found at:
http://www.lix.polytechnique.fr/~rauzy/altarica/AltaRica.html