MODPROD workshop

Dependability studies, focus on fault trees and Figaro language

Marc Bouissou
ECP, Laboratoire Génie Industriel & EDF R&D, dépt Management des Risques Industriels

Lena Buffoni
PELAB, Linköping University

Installation for the tutorial

- For Windows only
- USB keys with a special version of OpenModelica
- IMPORTANT: copy the contents of the USB before installing
- Copy the exercise folder
Outline

- Introduction : socio-technical systems
- Dependability studies and lifecycle
- General approach of a dependability study
- Static and dynamic systems & models
- Introduction to fault trees
- Principles to build fault trees
- Processing methods for fault trees
  - Minimum cut sets generation
  - Probability calculation
- Automatic generation of fault trees
  - The Figaro modeling language
  - Demonstration of the KB3 tool
- Figaro and OpenModelica

Socio-technical system

- System =
  - set of components in interaction
  - for a given purpose
- Socio-technical ==>
  - hardware
  - software (in most cases)
  - operator(s)
- Organic hierarchical decomposition possible for large systems
  - Ex: a nuclear power plant contains about 100 "elementary systems"
  - This decomposition is useful when it corresponds to a functional decomposition
  - It is often true for thermo-hydraulic systems, but false for distributed programmable systems
Decomposition of a NPP

NPP

Function: produce electricity

System

Functions of the system
Needs several components
- ex: cool the condenser

Important component or set of components
- ex: pump, regulation valve, heat exchanger

Sub-component
- ex: pump, motor,…

Subset
- ex: contactor

Item
- ex: bearing

Elementary system: Emergency Feedwater System
**Elementary system: electrical supplies**

**Operators and components in their environments**

- institution
- team
- installation
- system
- individual
- component
- Socio-technical system
Dependability (RAMS) studies & lifecycle

Life cycle of a product or installation & related dependability activities

- Development
- Feasibility, prototype
- Design optimization
- Mass production / building
- Deployment / use
- Feedback of experience
- Reliability centered maintenance
- Dismantling / recycling
The importance of early phases

- Need definition and planning
- Concept/Preliminary design
- Final Design
- Construction
- Operate and maintain

Source: Blanchard, B.S., Design and Manage to Life Cycle Cost, Forest Grove, OR, MA Press, 1978

Design optimization

- Find the best trade-off: costs / dependability
- Application of the ALARP principle
- Example:

The choice is not so simple: 2 pumps require check-valves in addition to avoid inverse flows => additional failures are possible
Pumps with the same reliability & maintainability, or not? The first solution is more "survivable", the second one requires less maintenance (but forbids online preventive maintenance)
Design optimization

- Choices to be made:
  - architecture (redundancies)
  - spare stocks
  - technologies (e.g. hydraulic or electric or pneumatic motors for valves ?)
  - materials (resistance to stress, corrosion, radioactivity…)

- Architecture alternatives are not easy to compare automatically
- Usually the architecture is decided by the designers according to functional considerations rather than dependability considerations
- However, they may wish to compare a few possibilities => tools are required for quick evaluations

---

General approach for a system dependability study

From the book by Rausand & Hoyland
First modeling of the system

Steps

1. Determine the system bounds with regard to other systems, in particular to fix the support systems
2. Acquire a correct knowledge of the functioning of the system
   - Functional analysis
   - Preliminary risk analysis, HAZOP, FMECA
3. Choose the level of detail

=> first modeling of the system, for example in the form of a diagram, accompanied with the main functional hypotheses

Mission: heat the water at temperature \( T = 65^\circ C \), and maintain level \( H \) between two bounds...

Choice of the detail level of the model

A crucial issue

- too coarse => we cannot answer all the questions
- too detailed =>
  - Model is intractable
  - Reliability data may not be available
Static and dynamic systems & models

(for dependability analysis)

Combining two components: series system

\[ A(t) = A_1(t)A_2(t) \]

\[ R(t) = R_1(t)R_2(t) \]
Combining two components: parallel system (active redundancy)

The reliability cannot be computed as a function of reliabilities or availabilities.

Combining two components: parallel system (standby redundancy)

- The history previously given as an example is no longer possible
  - When Cpt 1 works, Cpt 2 is in standby and cannot fail
  - Example of dependency between the components

- A global approach of the system is needed
  - Impossible to reason component by component
  - Calculations are more complex…

- Other examples of dependencies
  - Reconfigurations
  - Shared maintenance resources
  - Common cause failures
  - …
Characteristics of reliability models

- Combinatorial explosion of the possible failure combinations
- Discrete state models – often Boolean
- Two main categories of models

Fault-tree

Explains an undesirable event as a function of failures of components in the form of a Boolean function
### State graph

- FU blows out, grid fails
- L1 fails
- L2 fails
- L2 fails, FU blows out, I2 opens, grid fails
- I1 opens
- I2 opens
- L1 fails, FU blows out, I1 opens, grid fails
- L2 opens, grid fails

Simple because no repair

### The limits

- Static models assume the independence of components -> (more and more) precise calculations on approximate models
- Dynamic models: more accurate, but combinatorial explosion -> imprecise (or incomplete) calculations
Fault trees

A powerful method, facilitating the \textit{construction} of the model and its \textit{processing}

Scope of fault trees

- One (or more) Boolean undesirable event(s)
- Two physical (or intrinsic) states for components: working or failed
- Two states for the functions fulfilled by the components
- Independence of \textit{components} (w.r.t. to their failure and repair processes)
- Deterministic interactions between \textit{functions} (e.g. loss of power supply $\Rightarrow$ loss of the function of all components using this power supply)
Local model and global model

- Repetitive process of determination of the causes:
  - The fault tree is formed by successive levels such that every event is generated by other events, through logical operators.
  - The complete tree represents the global model of the system (for a particular undesirable event).
  - A part of the tree represents a local model.

Definition of a fault tree

- Mathematical model connecting the occurrence of a top event to those of a set of basic events.
- The events are associated to Boolean indicator variables (X=1 means (by abuse of notation): the event X is realized); we identify the union, the intersection of the events with the Boolean operators OR, AND.
- A fault tree is thus a Boolean function.
- Normalized graphic representation.
- The basic events must be independent, if we want to use the fault tree for probability calculations.
Graphic representation

Most common gates

- **OR gate**:
  - Event: \(A \lor B \lor C\)
  - Diagram:

- **AND gate**: 
  - Event: \(A \land B \land C\)
  - Diagram:

- **K/N gate**: (equivalent to one OR on a set of AND)
  - Event: \(\frac{2}{3} \text{ Among } A, B, C\)
  - Diagram:

Less often used gates: NOT, NOR, NAND...
They create "non coherent" trees, more difficult to process.

Use of the transfer gates

- The sub-tree \(\{1\}\) is defined once and can be referenced in multiple places in the fault tree.

The possibility of having repeated events or sub-trees makes the fault "tree" in fact a graph without circuit.
Textual representation (example)

\[ Y = Y_1 \& Y_2; \]
\[ Y_1 = X_1 \| Y_3; \]
\[ Y_3 = X_2 \| X_3 \| X_4; \]
\[ Y_2 = X_5 \| Y_4; \]
\[ Y_4 = X_6 \| X_3 \| X_4; \]
\[ Y = F (X_1, X_2... X_n) \]

Method of construction (1/2)

- Start from the top event (usually a loss of function of the system)
- Look for the immediate, necessary and sufficient causes of the under development event; when this inventory is ended,
- Choose a new event to develop, and repeat the previous point

Advice

- Continue the decomposition upon reaching independent basic events of known probability
- Develop in width (level by level) rather than in depth first
- Give labels to each gate; this is required for the readability of the tree
Method of construction (2/2)

- 2 types of approaches are possible:
  a: Ascent of the interaction flows
  b: Progressive dive in the detail of the sub-systems

- Advantages:
  a: Systematic character, easy to automate
  b: Yields trees more compact and easier to process

- Drawbacks:
  a: Raises big problems in case of existence of loops
  b: Difficult to automate

Examples of the method a)

Example 1
U.E. = no output signal for E

Example 2
U.E. = T2 too high
Example 1

Unwanted event: no output signal for E

Loss of E

No signal at the input of E

Loss of D

No signal at the input of D

No signal at the output of B

No signal at the output of C

Loss of B

Loss of A

Loss of C

Loss of A

Example 2

Unwanted event: T2 too high

Q too high

T1 too high

q too low

t1 too high
Exercise

Unwanted event: no water at the output of the system

Modeling common cause failures

A belongs to a group of 3 identical components A, B, C, susceptible to CCFs

The probabilities of the basic events are computed according to one of the models presented in the module “BasicSysDep”
Qualitative processing of (coherent) fault trees

Definition of the minimal cut sets

- **CUT SET**: set of basic events whose (simultaneous) occurrence ensures the occurrence of the top event
- **MINIMAL CUT SET**: cut set containing no other cut set
- **ORDER** of a cut set: number of events that it contains
- The set of all the minimal cut sets constitutes a canonical representation of the fault tree

Example: two minimal and one non minimal cut sets
Calculation of the minimal cut sets

Consists in rewriting the Boolean function, using Boole’s algebra laws

\[ X + X = X \cdot X = X \]

\[ X + 1 = 1, \quad X + 0 = X \]

\[ X \cdot 0 = 0, \quad X \cdot 1 = X \]

\[ X \cdot (Y + Z) = X \cdot Y + X \cdot Z \]

\[ X + Y \cdot Z = (X + Y) \cdot (X + Z) \]

Boolean product \( \Leftrightarrow \) intersection of events

Boolean sum \( \Leftrightarrow \) union of events

Example

\[ Y = Y_1 \cdot Y_2 \]
\[ Y_1 = X_1 + Y_3 \]
\[ Y_3 = X_2 + X_3 + X_4 \]
\[ Y_2 = X_5 + Y_4 \]
\[ Y_4 = X_6 + X_3 + X_4 \]

\[ Y = (X_1 + X_2 + X_3 + X_4) \cdot (X_5 + X_6 + X_3 + X_4) \]

\[ Y = [(X_1 + X_2) \cdot (X_3 + X_4) \cdot (X_5 + X_6) + (X_3 + X_4)] \]

\[ Y = (X_1 + X_2) \cdot (X_5 + X_6) + X_3 + X_4 \]

\[ Y = X_1 \cdot X_5 + X_1 \cdot X_6 + X_2 \cdot X_5 + X_2 \cdot X_6 + X_3 + X_4 \]
Algorithms

- They proceed from the leaves towards the root (bottom-up) or from the root towards the leaves (top-down).
- They can also combine the two approaches.
- Reduction of the combinatorial explosion:
  - without loss of information thanks to the location of independent sub-trees (modules) \((X_1 + X_2, X_5 + X_6)\) in the previous example which are not developed (they are considered as "macro basic events")
  - with loss of information thanks to the elimination of the cut sets of too high order or too low probability
- The techniques of direct rewriting tend to be replaced by the use of BDD (Binary Decision Diagrams).

Use of the minimal cut sets

- Validation of the model.
- They allow to detect the weak points of the system (but be careful: according to probabilities, the cut sets of smaller order are not necessarily the most likely).
- They allow the identification of false redundancies and potential common cause failures (this requires that we can associate characteristics to the basic events, such as: place, conditions of environment, manufacturer…)
- Support of the probability calculation (see next chapter).
Quantitative processing of fault trees (coherent or not)

Calculations of probability, importance factors

The Poincaré formula

Also called the inclusion-exclusion principle

\[
\Pr(UE) = \Pr(C_1 \cup C_2 \cup \ldots \cup C_i \cup C_n)
\]

\[
\Pr(UE) = \sum_{i=1}^{n} \Pr(C_i) - \sum_{i<j} \Pr(C_i \cap C_j) + \sum_{i<j<k} \Pr(C_i \cap C_j \cap C_k) - \ldots + (-1)^n \Pr(C_1 \cap C_2 \cap \ldots \cap C_n)
\]

Intractable for large numbers of minimal cut sets

In practice, the approximation given on next slide is used
Calculation from the minimal cut sets (coherent trees)

Theorem of Poincaré:

\[
\sum_{i \neq j} \Pr(C_i) - \sum_{i \neq j} \Pr(C_i \cap C_j) \leq \Pr(\bigcup C_i) \leq \sum \Pr(C_i)
\]

The probability of every cut set is given by the product of probabilities of the basic events which compose it (because they are independent). And for an intersection of cut sets?

This approximation is good only if all basic events are of low probability. The fact that minimal cut sets have a low probability is not sufficient!

Summary...

- Fault trees are a very practical type of models by their readability, their capacity to model large systems, with complex structures.
- There are numerous software* to process them; the most recent are based on BDD techniques, that allow exact calculations (thus valid whatever the probabilities, and for non coherent as well as coherent fault-trees).
- Their essential limitation is their incapability to take into account dependences between components (e.g: functioning in normal/standby, limitation of the number of repairmen...)

* But very few of them are free:
Automatic construction of fault trees

Principles of the KB3 tool developed by EDF

Demonstration

Why automation tools?

- Needs for the PSA of nuclear power plants
  - Consistency
  - Traceability
- Needs for the studies to be performed during the design of new systems
  - Speed
  - Accessibility to non specialists
**Principles of the KB3 workbench**

- **Knowledge Base (Figaro language)**
  - Description of components
  - Graphic input of system models
  - Figaro 0 (textual model)

- **Fault tree generator**
  - Towards commercial tools (Aralia, Risk-Spectrum)
  - Minimal cut sets
  - Reliability
  - Availability

- **Automatic generator of Sequences: FIGSEQ**
  - The most likely sequences
  - Reliability, MTTF
  - Availability

- **Monte Carlo simulation software: YAMS**
  - The most likely sequences
  - Reliability, Availability
  - Mean values of random variables

---

**A knowledge base = a graphic language to build models**

- **A KB is defined by two main files (+ icons)**
  - Generic behavior models of the components of the KB in FIGARO language: defines the semantics of the graphic language
  - Parameter settings of the graphic interface
    - Define the "grammar": what is allowed / forbidden in terms of connections
    - Improve the user-friendliness by the definition of changes of colors, texts and icons in interactive simulation
A simple telecom network

KB3: benefits

- Consistency of studies
- Traceability of studies
- Quality of studies
- Speed (between 40% and 80% of time saving for a study)
- Accessibility to non-specialists
- A unique tool allowing to use numerous graphic formalisms (→ learning reduced to the notions specific to the used knowledge base)
The FIGARO tools

- Visual Figaro: Library editor
- TradBDC: Translator Fr-Eng
- FigArbre: FT generator
- Fault tree (XML)
- FigP: Figaro Processor
- Figaro 1 (library)
- Figaro 1 (Objects list)
- Figaro 0 (instantiation of library on objects)
- Fig0Debug: Interactive simulation
- YAMS: MC Simulation
- OpenModelica: Model input
- Figseq: Sequence exploration

Text only tools
What is on the USB key

Visual Figaro: Library editor

TrabDCC: Translator Fr-Eng

FigArbre: FT generator

Fault tree (XML)

Figaro 1 (library)

OpenModelica: Model input

FigP: Figaro Processor

Figaro 1 (Objects list)

Figaro 0 (instantiation of library on objects)

GRIF.Tree: Graphical FT editor
Minimal cut sets, Pr(top event)
Importance factors

Fig0Debug: Interactive simulation

Examples

INTERFACING MODELICA AND FIGARO
### Why interface Modelica with Figaro?

- Modelica models used in dynamic model verification of the functional behavior
- Figaro can be used for static or dynamic reliability analysis

These two ways to model systems are complementary!

---

### Advantages

- Modelica models allow a graphic representation of the system structure
- Automatic generation:
  - Reduces risk of introducing errors
  - Improves usability
  - Simplifies model maintenance
Mapping Figaro to Modelica

Same topology?

Figaro model

Modelica model  Simple telecom model
Mapping Figaro to Modelica – how?

The Figaro specific information in the Modelica model is expressed by inheriting from two abstract classes:

```model Figaro_Object
  parameter String fullClassName;
  parameter String codeInstanceFigaro;
  end Figaro_Object;

connector Figaro_Object_connector
  parameter String fullClassName;
  parameter String codeInstanceFigaro;
  end Figaro_Object_connector;
```

Links are full fledged objects in Figaro!

Mapping Figaro to Modelica

The Figaro specific information in the Modelica model is expressed by inheriting from two abstract classes:

```model Figaro_Object
  parameter String fullClassName;
  parameter String codeInstanceFigaro;
  end Figaro_Object;

connector Figaro_Object_connector
  parameter String fullClassName;
  parameter String codeInstanceFigaro;
  end Figaro_Object_connector;
```

Figaro-specific information used to generate the Figaro model
Mapping Figaro to Modelica

The Figaro specific information in the Modelica model is expressed by inheriting from two abstract classes:

```model Figaro_Object
parameter String fullClassName;
parameter String codeInstanceFigaro;
end Figaro_Object;
```

```connector Figaro_Object_connector
parameter String fullClassName;
parameter String codeInstanceFigaro;
end Figaro_Object_connector;
```

Through the use of inheritance the default values for these parameters can be specified once for a set of components.

Configuring Figaro generation in OMEdit

- Figaro database file
- Type of file to generate
- Fault-tree options
- Location of Figaro processor

Attention: Set the working directory to the working directory.
Generate the Figaro file from the model

Attention: Generate Figaro from the root package containing the model!

Exercise: Telecom network model

Modelica model

Figaro model
Exercise: Telecom network model

- In the exercises directory go to TelecomNetwork
- Open the description pdf, follow the instructions to configure the Figaro plugin.
- Generate a Figaro0 model.
- Simulate a failure in the telecom model by increasing the resistance in one of the links

Mapping Figaro to Modelica

- Not every Modelica object maps to a Figaro object
- Figaro properties can be specified for a type or for an instance
- In Figaro links are more complex objects than Modelica connectors, so they might need to be created in the Modelica model
Mapping Figaro Links to Modelica

Modelica model

Figaro model

R = 100000
**Generation of the Figaro model in OpenModelica**

To generate the Figaro model from the augmented Modelica model:
- the abstract syntax tree representation of the model is parsed
- the Figaro specific properties and Figaro object instances are exported
- the results are saved as a Figaro file

---

**Model Translation Process to Hybrid DAE to Code**

- **Modeling Environment**
  - **Frontend**
    - Modelica Graphical Editor
    - Modelica Textual Editor
  - **"Middle-end"**
  - **Backend**
    - Translator
    - Analyzer
    - Optimizer
    - Code generator
    - C Compiler
    - Simulation

- **Modelica Model**
- **Modelica Source code**
- **Flat model Hybrid DAE**
- **Sorted equations**
- **Optimized sorted equations**
- **C Code**
- **Executable**
OpenModelica scripting API:

```plaintext
function exportToFigaro
    input TypeName path "the class to be exported";
    input String database "the Figaro database";
    input String mode "the type of file to be generated";
    input String options "options for the generation of FTA";
    input String processor "the Figaro processor";
    output Boolean success;
end exportToFigaro;
```

Exercise: SRI model
### Exercise: SRI model

- Go to the SRI folder in the exercises
- Configure the Figaro plugin
- Add a third pump to the model
- Generate the fault tree
- Open the fault tree in the graphical editor GRIF