Tutoriel A3: A Reasoned Introduction to Model-Based Risk and Safety Assessments

Michel BATTEUX (IRT SystemX)
Tatiana PROSVIRNOVA (IRT Saint-Exupéry)
Antoine RAUZY (NTNU)
Note to the Reader

This tutorial about Model-Based Risk and Safety Assessment is strongly inspired by authors’ work on the modeling language AltaRica (and more precisely AltaRica 3.0).

Other authors may have a different vision of the subject.

We believe in a scientific approach of the questions debated here. For us, each and every assertion must be supported by strong mathematical arguments as well as sufficiently many practical experiments on sufficiently large case studies.

In our domain, reaching this high standard requires not only mathematical and algorithmic knowledge and rigorous experimental protocols, but also a huge effort of software development.

Michel Batteux
Tatiana Prosvirnova
Antoine Rauzy
Agenda

• What is Model-Based Risk & Safety Assessment?
• Behaviors + Structures = Models
• Behavior Modeling Frameworks
• Model Structuring Frameworks
• Model Synchronization
• Frequently Asked Questions
• Some References
WHAT IS MODEL-BASED SAFETY ASSESSMENT?
Preliminary Remarks

- Fault Trees, Block Diagrams, Event Trees and the like are models.
- Models are actually at the core of Risk and Safety Assessments since the very beginning of the discipline.

- **Model-Based Safety Assessment** (MBSA) differs thus from **Model-Based Systems Engineering** (MBSE) which is defined in contrast to text-based systems specifications.
What is a Model?

All these “things” are models in some way.
Models in (Safety and Reliability) Engineering

Cognitive Model → Mathematical Model → mind & paper models

Cognitive Model → Computerized Models

Cognitive Model → Code

Cognitive Model → Graphical Representation
Computerized Models

Computerized models (including graphical ones):

• are sequences of symbols that obey a given syntax (grammar);

• have a formal semantics (they are interpreted in a given mathematical framework);

• are designed primarily to perform calculations of risk related performance indicators.
“Classical” Modeling Formalisms

**Boolean formalisms**

- Fault Trees
- Blocks Diagrams
- Event Trees

**Transitions Systems**

- Markov Chains
- Stochastic Petri Nets

Note: for some applications, Bayesian networks are worth to consider
Issues with “Classical” Models

Classical modeling formalisms lack of expressive power and/or are very close to mathematical equations (lack of structure).
→ Distance between systems specifications and models;
→ Models are hard to design and even harder to share with stakeholders and to maintain throughout the life-cycle of systems.
Power Supply System(*)

Assess the probability that the Busbar cannot be powered and find the sequences of events that lead to this situation

Fault Tree

Loss of Busbar Power

Loss of Primary Power Supply

Loss of Offsite Power

Loss of Line 1

CBU1 Failure

TR1 Failure

Loss of Power Lines

CBDU1 Failure

TR2 Failure

Loss of Line 2

CBU2 Failure

CBDU2 Failure

Loss of Backup Power Supply

DG Failure

CBDU3 Failure
Mathematical issues (well known and accepted):
• Warm/Cold redundancies cannot be represented with Fault Trees
• Orders of events cannot be taken into account
• Common cause failures must be represented separately
• ...but the Markov chain for such system cannot be designed by hand (at least $2^9 = 512$ states)
Modeling issues:
- Model does not reflect the architecture of the system (no way back)
- Model hard to check for correctness and completeness
- No possible “visual” simulation
- One model per safety goal

Issues

difficult

nearly impossible
The Promise of MBSA

Modeling systems at **higher level** so to reduce the distance between systems specifications and models (without increasing the complexity of calculations).

```plaintext
class HydraulicPump
    Boolean working (init = false);
    event failure (delay = exponential(lambda));
    transition
        failure: working -> working := false;
end
```

Systems Specifications

Models
Complexity of Calculations

- **Calculations** of risk and safety related indicators are **extremely resource consuming**.
- This is not a problem of technology, it has been **mathematically proven** that they are **computationally intractable**.

→ **Models** result always of a **tradeoff** between the accuracy of the description and the ability to perform calculations.
BEHAVIORS + STRUCTURES = MODELS
Behaviors + Structures = Models

**Mathematic framework**
- Ordinary Differential Equations
- Mealy Machines
- Probabilistic Boolean Algebras
- Petri Nets
- Bayesian Networks
- Guarded Transitions Systems
- ...

**Structuring paradigm**
- Block Diagrams
- Object-Oriented
- Prototype-Oriented

- Modelica
- Lustre
- Fault Trees

- Reliability Block Diagrams
Special Case: Architecture Languages

**Mathematic framework**
- ... 
- Empty 
- ...

**Structuring paradigm**
- (extended) Block Diagrams 
- ...

**SysML structural diagrams**
(BDD, IBD)

![Diagram showing SysML structural diagrams](image)
Questions

• What are the good mathematical frameworks for risk and safety assessment?

• What are the good structuring paradigms for these mathematical frameworks?

Recall: no universal panacea…
BEHAVIOR MODELING
FRAMEWORKS
“Classical” Modeling Formalisms

**Boolean formalisms**
- Fault Trees
- Blocks Diagrams
- Event Trees

**Transitions Systems**
- Markov Chains
- Stochastic Petri Nets

Common Characteristics:
- Event-Based
- Probabilistic
Boolean models are automatically transformed into equivalent Fault Trees before assessment.
Assessment Algorithms

Model (Fault Tree) → Minimal Cutsets, Prime Implicants → Binary Decision Diagrams

Indicators
- Unavailability
- Importance Factors
- Safety Integrity Level
- …
Pros & Cons

• Pros
  ▪ Well mastered
  ▪ “Easy” to understand
  ▪ Efficient assessment algorithms (see articles by A. Rauzy)
  ▪ Many available software
  ▪ …

• Cons
  ▪ Lack of expressive power
  ▪ Very distant from systems specifications
  ▪ One model per safety goal
  ▪ …

• Possible extension
  ▪ Finite domain algebra, e.g. \{low, medium, high\}
Transitions Systems

Modeling
- Much more expressive power than Boolean formalisms
- Lack of structure (Markov chains, Petri nets)

Assessment
- Compilation into fault trees (not always possible)
- Compilation into Markov chains (not always possible)
- Sequence generation
- Monte-Carlo Simulation
- Model-checking
- ...

Generic mathematical framework
- Guarded Transitions Systems
Guarded Transitions Systems

• The state of the system is represented by means of (state) variables.

• Variables take their value into domains (Boolean, sets of symbolic constants, integers…)

• Variables change of value when and only when an event occur, i.e. when the transition it labels is fired.

• A transition is fireable only when its guard (pre-condition) is satisfied.

• Events are associated with (stochastic) delays and/or with probabilities
The synchronized composition of two (or more) GTS is a GTS.

Synchronizations
- Main.failure & Spare.start
- Main.failure & Spare.failureOnDemand
- Main.Repair & Spare.stop
Composition

The **synchronized composition** of two (or more) GTS is a GTS.
Flow Variables

- Flows of information/matters/energy circulating in the system are represented by means of (flow) variables.
- Flow variables take their value into domains (Boolean, sets of symbolic constants, integers...)
- Flow variables depend functionally on state variables: their value is entirely determined by the values of state variable
The engine E1 is fueled through T1, and V1:

- not T1.isEmpty \implies T1.outFlow
- T1.outFlow \implies V1.leftFlow
- V1.leftFlow and not V1.closed \implies V1.rightFlow
- V1.rightFlow \implies E1.inFlow
Now, the engine E1 is fueled through T2, V2 and V3:
• not T2.isEmpty ⇒ T2.outFlow
• T2.outFlow ⇒ V2.rightFlow
• V2.rightFlow and not V2.closed ⇒ V2.leftFlow
• V2.leftFlow ⇒ V3.rightFlow
• V3.rightFlow and not V3.closed ⇒ V3.leftFlow
• V3.leftFlow ⇒ E1.inFlow
Now the engine E1 is not fueled
• not T2.isEmpty ⇒ T1.outFlow
• T1.outFlow ⇒ V1.leftFlow

The other flow variables are reset to their default values (false).
Hierarchical

GTS make it possible:

- To design models of systems by composing models of subsystems into hierarchies.
Implicit Representation of the State Space

GTS make it possible:

- To represent in an **implicit way** actual states and transitions of the system (**reachability graph**).
- To avoid (to some extent) the combinatorial explosion of the size of the model and to allow approximate calculations based on most probable scenarios/states.
GTS versus (Dynamic) Fault Trees

**Basic Event**
- Status: working or not working
- Activity: active or not active
- Failure: not working or failed

**Gates**
- Status: children status
- Activity: children activities

Idea: Basic Events and Gates
- Calculate their status (working or failed) bottom-up;
- Are activated top-down (in regular Fault Trees, basic events and gates are always active).

GTS generalize (at no cost) Dynamic Fault Trees
GTS versus Petri Nets

GTS generalize (at no cost) Stochastic Petri Nets (and various extensions of).
Wrap-Up

• Two main mathematical frameworks for risk & safety assessments:
  - Probabilistic Boolean algebra (fault trees)
  - Transitions systems
• Both have advantages and drawbacks
• Guarded Transitions Systems are the most generic framework of the second category
MODEL STRUCTURING FRAMEWORKS
Composition

One cannot expect models of complex systems to be simple. To capture interesting aspects they have to be complex too, and therefore they must be structured.

The simplest structuring relation is the composition: a system composes a component means that the component “is part of” the system. Many modeling formalisms implement composition.

Note: S.A.V is different from S.B.V. although both components are “named” V.
Prototypes

In a hierarchical decomposition, each block (S, S.A, S.A.V…) is supposed to be unique. A block with a unique occurrence is called a prototype.

In general, at system level, many blocks are unique.
However, it is often the case that components (or even subsystems) are similar (e.g. S.A.V and S.B.V, S.A and S.B). Having only prototypes is not very suitable for knowledge capitalization and reuse.

Classes are on-the-shelf, reusable modeling components. Classes can be instanced in a model, e.g. V is an instance of the class Valve in the class Train. An instance of a class is called an object.

Several modeling formalisms implement classes, but extremely few both prototypes and classes.
The Box-in-Box-in-Box-in-Box Issue

It is not possible to modify a class through its instance, because it would impact not only that particular instance, but all (possibly unknown and even not yet created) instances of the class.
In some cases, we want to modify or extend the characteristics of a modeling component/class without changing its nature. In these cases, composition is not really suitable because we would like to be able to substitute the modified/extended component for any occurrence of the original one.

_Inheritance_ makes it possible. Inheritance is a “is-a” relation between modeling components, e.g. an AutonomousTrain is a Train.

Very few modeling formalisms implement inheritance.
In some cases, we want to capture that a subsystem needs some component, but that this component is not part of the subsystem and may be shared by several subsystems.

**Aggregation** makes it possible. Aggregation is a “uses” relation between modeling components, e.g. a PoweredTrain aggregates/uses a PowerSource.

Very few modeling formalisms implement aggregation.
Wrap-Up

- Model structuring mechanisms are (almost) independent of behavioral constructs. They originate from mechanisms to structure programs.

- **Prototypes, classes, composition** *(is-part-of relation)*, **inheritance** *(is-a relation)* and **aggregation** *(uses relation)* are the fundamental concepts of model structuring.
  - Prototypes + composition: **hierarchical modeling** paradigm.
  - Classes + composition: **structured modeling** paradigm.
  - Classes + composition + inheritance: **object-oriented** paradigm
  - Prototypes + Classes + composition + inheritance + aggregation: **prototype-oriented** paradigm
MODEL SYNCHRONIZATION
A Double Challenge

- Systems designed by industry are more and more complex.
- To face this complexity, the different engineering disciplines (mechanics, thermic, electric and electronic, software, safety…) virtualized their contents to a large extent, i.e. they are designing models. Each system comes with dozens of models.

- There is a here double challenge:
  - Integrating the different engineering disciplines
  - Integrating the models they produce

- As a consequence, we need to design tools and methods to support this integration.

The emerging science (and engineering) of complex systems is a science (and engineering) of models
The diversity of models is irreducible

The level of abstraction of a model depends on what is to be observed, i.e. on the virtual experiments to be performed on that model.

There cannot be no such a thing as unique model or even a master model of a complex system.
Commonalities between models stand in their structuring

- Any **modeling language** is the **composition** of a **mathematical framework** and a set of **constructs to structure** models.

- The **structure of models** reflects the **structure of the system**, but only to a **limited extent**

<table>
<thead>
<tr>
<th>Structuring Constructs (Prototypes, Classes...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>differential equations</td>
</tr>
</tbody>
</table>

Simulink, Modelica, Lustre
Synchronization = Abstraction + Comparison

- The design/production/operation/decommissioning of a system involves the design of dozens if not hundreds of models. These models are designed by different teams in different languages at different levels of abstraction, for different purposes. They have different maturities.
- The question is how to synchronize these models, i.e. to ensure that they are speaking about the same system.
- Abstraction is a key tool for model synchronization.

- The suitable abstractors/comparators depend on the project, phase of the project…
FREQUENTLY ASKED QUESTIONS
What are the tools/languages supporting the MBSA approach?

- AltaRica
  - SimFia (EADS Apsys)
  - Safety Designer (Dassault Systemes)
  - Cecilia-OCAS (Dassault Aviation, not distributed)
  - OpenAltaRica tools (IRT SystemX & AltaRica Association)
  - ARC/AltaRica Studio (University of Bordeaux)
- Figaro (EdF)
- SAML (University of Magdeburg)
- HIP-HOPS (to some extent) (University of Hull)
- SOFIA (to some extent) (CEA-LIST)
- Petro (specific to Oil & Gas) (SATODEV)
## How mature is the MBSA approach?

### Helpful

- Theoretical framework
- High Level Models are much easier to design, to debug, to master, to maintain, to share, to reuse...
- Generalization of “classical” formalisms such as Block Diagrams, Markov chains, Generalized Stochastic Petri Nets
- Richness of assessment algorithms

### Harmful

- Trend to design too big and unique models
- Difficulty to handle systems whose architecture changes during the mission
- Initial cost to train analysts

### Internal origin

- Significant audience in France
- Certification process accepted by FAA and EASA (Dassault F7X), mentioned in last version of ARP4761
- Graphical simulation
- Used beyond safety analyses (performance analysis)

### External origin

- Development costs
- Redundant developments
References
Is the AltaRica project active?

• Yes! The OpenAltaRica project

www.openaltarica.fr  www.altarica-association.org
Is there a conference dedicated on MBSA?

- Yes!

Next International Conference on Model-Based Safety Assessment, IMBSA 2017, will be collocated with SAFECOMP 2017 in Trento (Italy)
SOME REFERENCES
Some References

Algorithms for Fault Tree assessment


Algorithms for Markov chains assessment


Stochastic Simulation for RAMS Studies

Some References

Dynamic Fault Trees


“Model-Based” extensions of Stochastic Petri Nets

Some References

Altarica Foundations & Specifications


Tutorial examples of Altarica

Some References

Compilation of AltaRica (GTS) into Fault Trees


Compilation of AltaRica (GTS) into Markov chains

Some References

Some selected PhD Theses related to AltaRica (mostly in French)

- Sophie Humbert. Déclinaison d'exigences de sécurité, du niveau système vers le niveau logiciel, assistée par des modèles formels. Université de Bordeaux I. April, 2008.
Some References

Additional References on Model-Based Safety Assessment

Some References

Paradigms to structure programs/models


Graphical Modeling & Architecture Modeling Formalisms