CSDM Asia 2021 – Beijing 11/04/2021

The Model-Based Systems Engineering Trinity: Syntax, Semantics, Pragmatics

Prof. Antoine B. Rauzy

Department of Mechanical and Industrial Engineering
Norwegian University of Science and Technology
Trondheim, Norway

&

Chair Blériot-Fabre
CentraleSupélec/SAFRAN
Paris, France

Model-Based Systems Engineering

We entered the era of Model-Based Systems Engineering (MBSE) but:

- How to make the MBSE process efficient?
- Why do we design models?
- What do we do with models?
- What is a (good) model?
- What is a (good) modeling language?
- What is a (good) modeling environment?

These questions are serious and need serious answers.

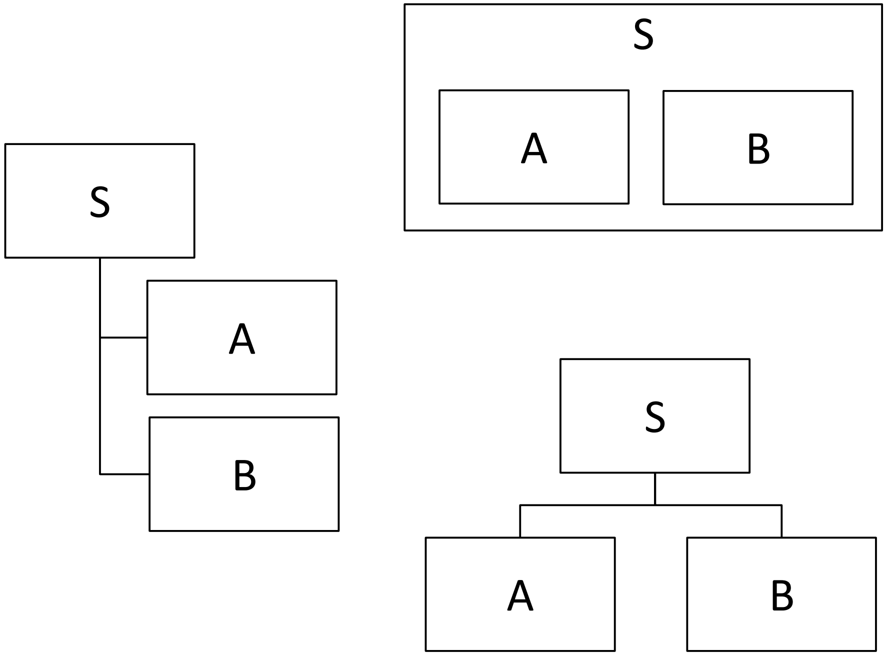
We need to establish the foundations of Model-Based Systems Engineering.

Rule 1. Diagrams are not models

Models are **mathematical objects**

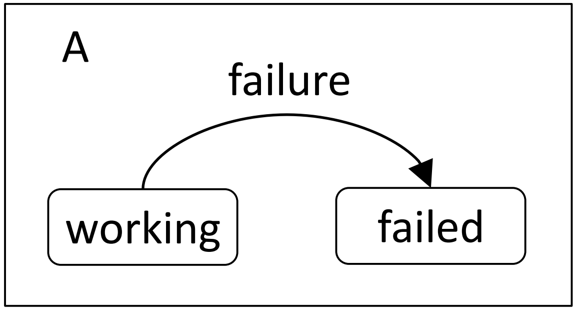
Diagrams are (or more exactly should be) **graphical representation** of models

```
block S
    block A
        ...
    end
    block B
        ...
    end
end
```



Rule 2. Models Have a Syntax

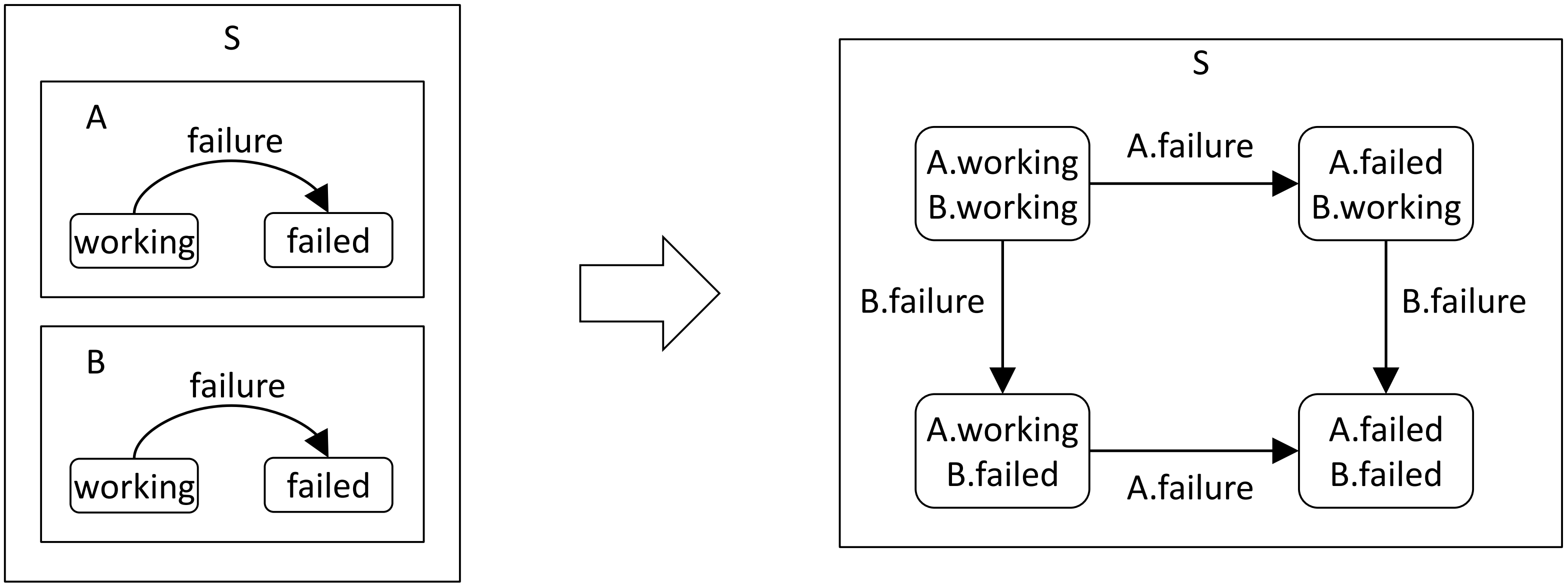
```
block A
    state working;
    state failed;
    transition
        failure: working -> failed;
end
```



Models are written in **modeling languages**.
There should be a unambiguous means to determine whether a given text (or diagram) is a correct a model or not. This means is called the **syntax** of the models, often described by means of a **grammar**.

Block ::= **block** Identifier StateDeclaration* Transition* **end**
StateDeclaration ::= **state** State ;
Transition ::= **transition** Event : State -> State ;
Event ::= Identifier
State ::= Identifier

Rule 3. Models Have a Semantics

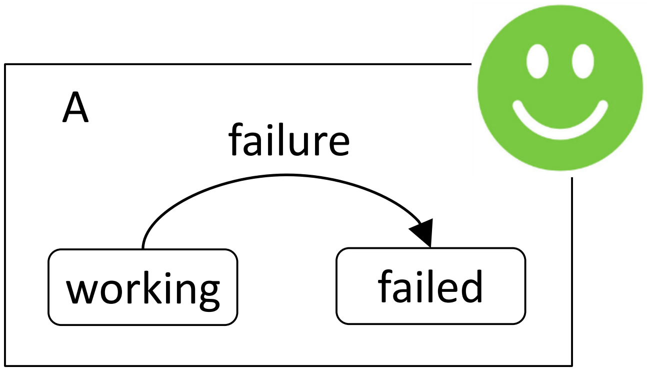


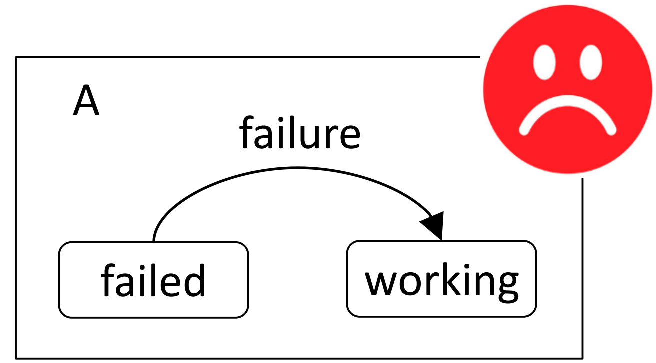
There should be an unambiguous way to interpret models into mathematical objects. This interpretation is the **semantics** of the model.

A formal semantics is the only way to justify computerized operations on models

Syntax and semantics are **domain independent**.

Rule 4. Models Have a Pragmatics



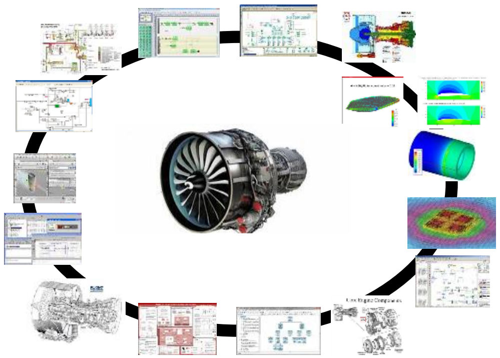


Properties of models are interpreted into **properties of real systems**.
This interpretation is called the **pragmatics** of models.

Facts about the pragmatics of models:

- It is at the very **core of the modeling process**.
- It is impossible to formalize as in requires a huge and **domain dependent** knowledge about systems.
- It is cultural and as such source of **ambiguities**.
- For these reasons, it should never be mixed up with the syntax and the semantics.

Rule 5. Pragmatic Modeling Objectives Determine the Choice of Mathematical Frameworks



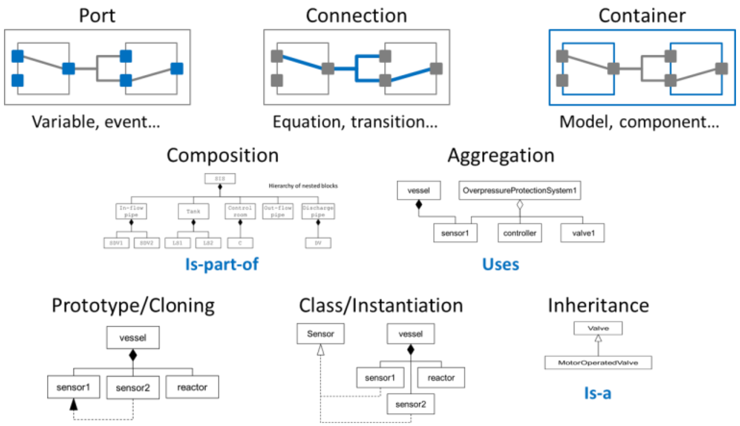
A model is always an **abstraction** of the system and is of interest because it is an abstraction.

The **properties of the system** to be studied determine the **mathematical framework** that should be used for the model.

Experiments performed on the model have a **cost**. This cost is a key driver for the choice of the mathematical framework and the level of abstraction of the model. The design of a model results always of a **tradeoff** between the **accuracy** of the description and the **cost** of experiments.

The **diversity** of models is **irreducible**.

Rule 6. Give me a Mathematical Framework, I will give you a Full-Fledged Modeling Language



In software engineering, the **object-oriented paradigm** is dominant, for good reasons.

Model-based systems engineering is ruled by the equation:

behavior + architecture = model

S2ML+X paradigm:

- X: suitable mathematical framework (Boolean equations, ODE, FSM, GTS…)
- S2ML (system structure modeling language): complete and versatile sets of **object-oriented** and **prototype-oriented constructs** to **structure models**

S2ML is domain independent.

Conclusion

Huge benefits can be expected from a full-scale deployment of model-based systems engineering. However, this requires:

- To set up solid **scientific foundations** for **models engineering**.
- To **bring to maturity** some **key technologies**.

The **biggest challenge** is to **train new generation of engineers**:

- With skills and competences in **discrete mathematics** and **computer science**,
- With skills and competences in **software engineering**,
- With skills and competences in **system thinking**,
- With skills and competences in **specific application domains**.