Elements of Systems Architecture

Daniel KROB – CESAMES & INCOSE Fellow

AEIS – September 13, 2021

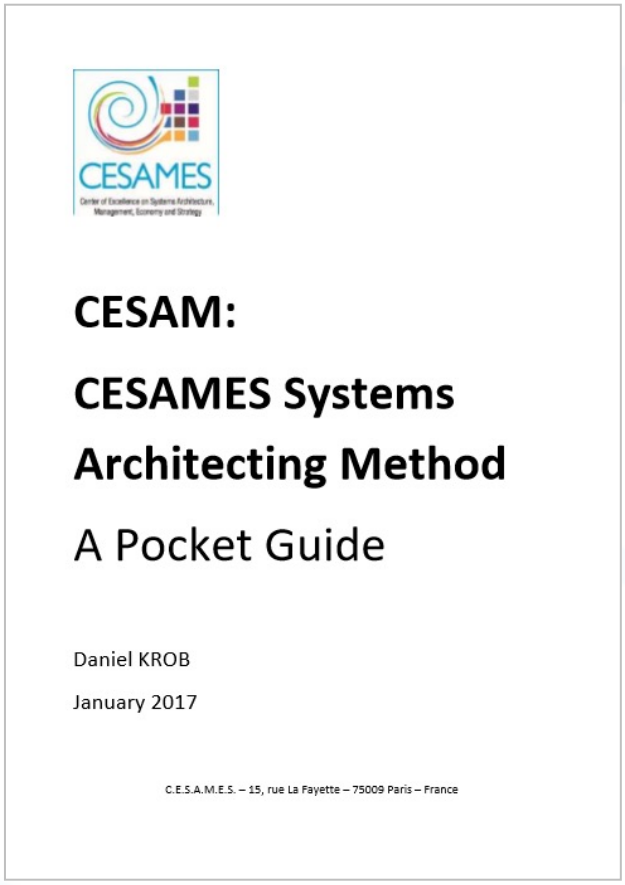
Table of contents

CESAMES
Center of Excellence on Systems Architecture, Management, Economy and Strategy

0 Brief overview of Daniel Krob's scientific career

- 2012-: president of CESAMES group
- 1989-: researcher, then senior researcher in computer science, National Center for Scientific Research (CNRS) – currently laid-off
- 1997-2002: founding director of the Laboratoire d'Informatique Algorithmique: Fondements & Applications (CNRS & Université Paris 7)
- 1995-2019: professor, then institute professor, Ecole Polytechnique
- 2003-2015: founder & director of the Dassault Aviation – DCNS – DGA – Thales industrial chair "Engineering of complex systems" of Ecole Polytechnique
- 2015-2017: scientific director of SystemX technological institute
- 2014: fellow of the International Council on Systems Engineering (INCOSE)

0 CESAMES public systems architecting methodology



CESAMES

Center of Excellence on Systems Architecture, Management, Economy and Strategy

CESAM:

CESAMES Systems Architecting Method

A Pocket Guide

Daniel KROB

January 2017

C.E.S.A.M.E.S. – 15, rue La Fayette – 75009 Paris – France

CESAM Pocket Guide Organization

The CESAM pocket guide is organized in eight chapters, plus some appendices specifically dedicated to more specialized material, as described below.

- *Chapter 0 – Introduction to CESAM* that may be skipped for a first approach.
- *Chapter 1 – Why Architecting Systems?* which presents the main motivations of the systems architecting approach and thus of CESAM framework.
- *Chapter 2 – CESAM Framework* that provides an overview of all CESAM concepts.
- *Chapters 3 to 6*, that do present in details, one after the other, each architectural vision of the CESAM systems modelling framework:
  - *Chapter 3 – Identifying Stakeholders: Environment* Architecture,
  - *Chapter 4 – Understanding Interactions with Stakeholders: Operational Architecture,*
  - *Chapter 5 – Defining What shall Do the System: Functional Architecture,*
  - *Chapter 6 – Deciding How shall be Formed the System: Constructional Architecture.*
- *Chapter 7 – Choosing the best Architecture: Trade-off*, that introduces to systems architecture prioritization, a key tool for the systems architect.
- *Chapter 8 – Conclusion* which gives some hints to reader on how continuing the systems architecting journey initiated by this pocket guide.

https://hal.archives-ouvertes.fr/hal-02561111v2/document

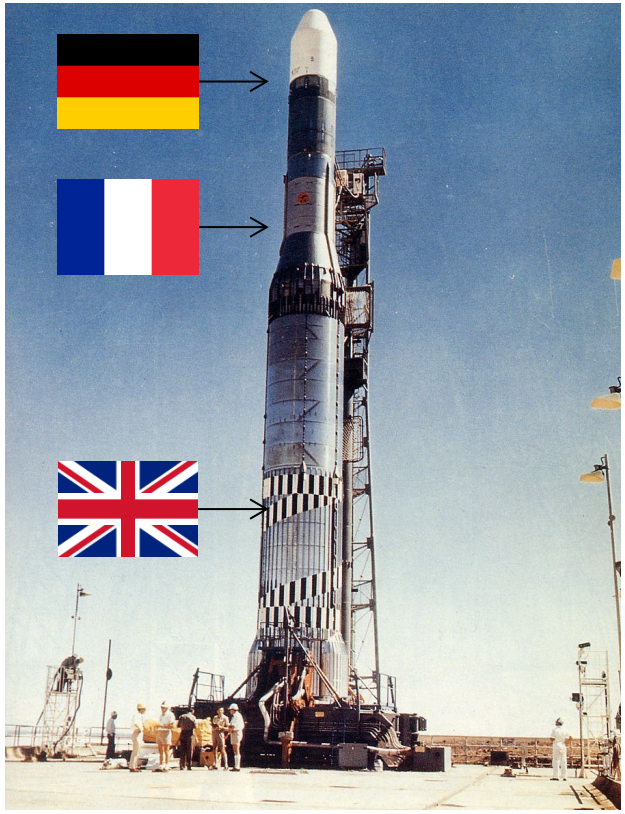
- **CESAM Level 1:** 9,000 trainees
- **CESAM Level 2:** 1,000 senior architects
- Deployed in **30 countries**

The public **CESAMES systems architecting method (CESAM)**, promoted by CESAMES, can be traced back to **NASA and INCOSE** systems engineering frameworks.

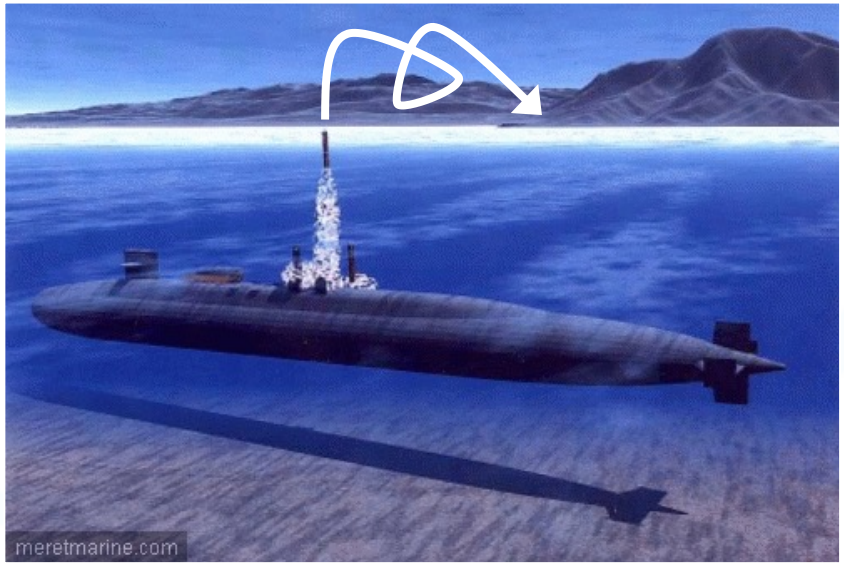
Table of contents

CESAMES
Center of Excellence on Systems Architecture, Management, Economy and Strategy

1 System issue 1 – Addressing critical integration issues



Europa launcher: 1964-1970
No success on 6 attempts due to lack of cooperation between main contributors



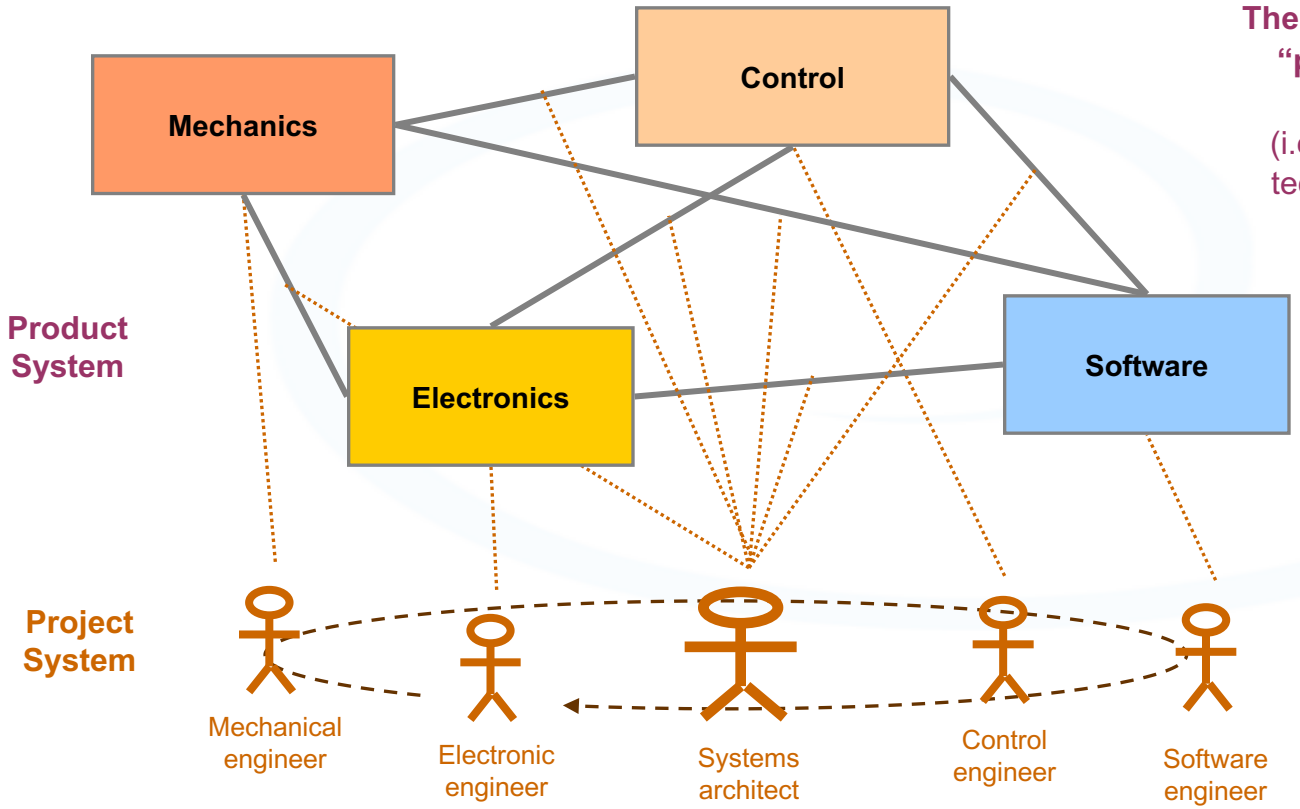
First submarine missiles: 1970-1980
Crash due to lack of cooperation between subsea & air engineering departments

Systems engineering is born in 1950-1980 within **defense & space**
as a key tool to solve the critical integration issues of this industry

1 Systems engineering technical answer 1 – Providing explicit system integration models

Examples of collaboratively constructed systems architecture views (for an engineering project in Japan)

1 Systems engineering human answer 1 – Introducing systems architect roles

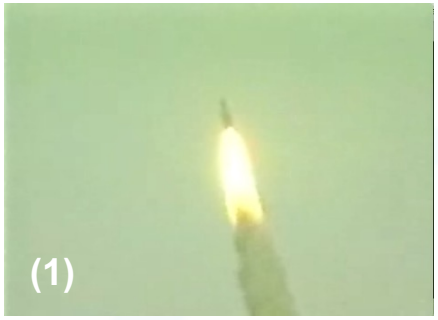


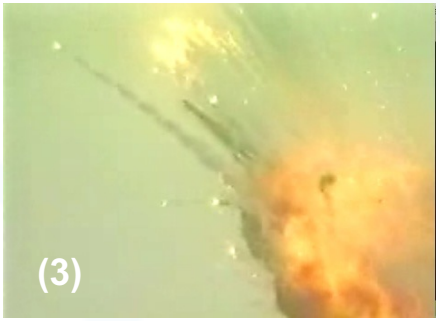
The key problem of the "product system": INTEGRATION (i.e. robustness of its technical interfaces)

The key problem of the "project system": CONVERGENCE (i.e. robustness of its human interfaces)

The **systems architect** is responsible for the **integration** of the **technical systems** (i.e. of the technical interfaces) and of the **human systems** (i.e. he/she has to make the stakeholders of a technical perimeter **converge** towards a unique vision

1 System issue 2 – Undesired emergence (1/2)

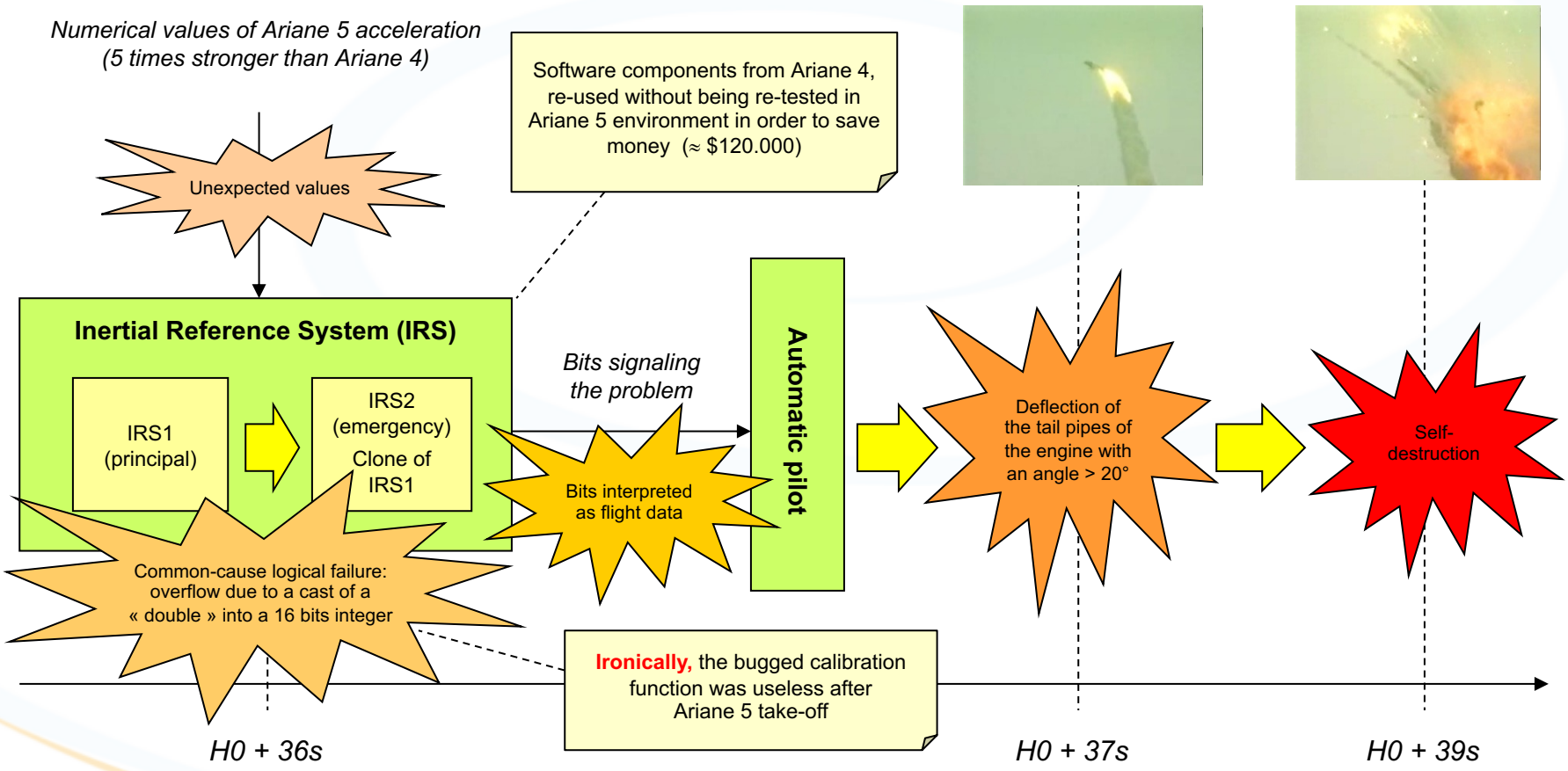




- **The crash of June 4, 1996:**
  - H0-H0+36 s.: perfect flight until second 36 after take-off (1)
  - H0+36,7 s.: simultaneous failure of the two inertial systems
  - H0+37 s.: activation of the automatic pilot which misunderstood the inertial systems data and corrected brutally the trajectory of Ariane 5 (2)
  - H0+39 s.: boosters brake, leading to the launcher self-destruction (3)
- **Immediate costs of the crash:**
  - Direct cost: $ 370 m (load lost)
  - Induced cost: 1 month of work to recover the most dangerous fragments (e.g. fuel stock) in Guyana swamps (“La Mangrove”)
- **Huge indirect costs due to Ariane 5 program delaying:**
  - Ariane 5 second flight was performed one year later
  - Ariane 5 first commercial flight was performed 3 years later (December 10,1999)

1 System issue 2 – Undesired emergence (2/2)

Source: Lions commission report, 1996



The Ariane 5 system **was incorrect by design**

1 Systems engineering answer 2 – Integrating model-oriented verification & validation activities along the design

Verification & validation practices can be split into **model–oriented** and **integration-oriented** practices.

Integration-oriented V&V

Model-oriented V&V

Model validation

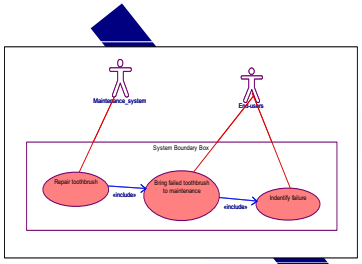
Model verification

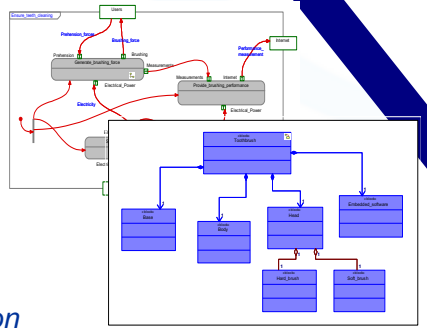
Operational tests

Integrated system validation

Functional & constructional tests

Integrated system verification





Brosse-@-dents électronique

Model & integration-oriented verification & validation practices (V&V) are **key** and shall be used **transversally along the design lifecycle** of a system.

1 System issue 3 – Missing key stakeholders

The Calcutta subway case (1/2)

- **What happened**
  - A strong heat wave (45°C = 113°F in the shadow) stroke India during summer
  - The cockpit touch screens of Calcutta subway trains became completely white
  - The subway stopped operating during a few days (leading to a big chaos)
  - The touch screens was immediately retested,
  - But they operated fine under high temperature conditions …

The Calcutta subway

Touch screen in the cockpit

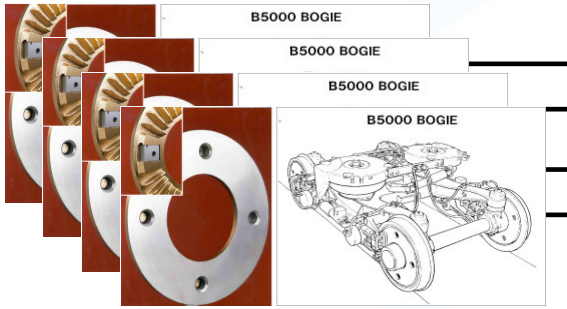
1 System issue 3 – Missing key stakeholders

The Calcutta subway case (2/2)

1 Heat impact: metal expansion → each brake system operates continuously out of its safety borders

2 Computer tries – unsuccessfully – to bring back all brake systems inside their safety borders

4 Observed side effect



Measures

Bus

Computer

Bus

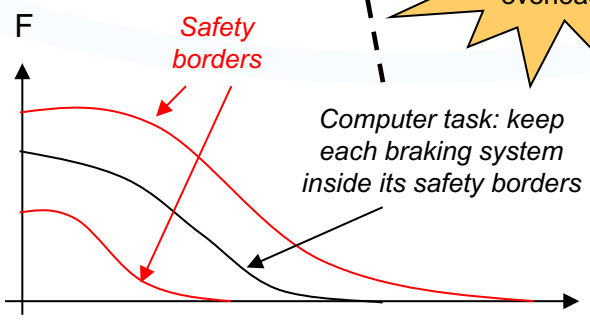
Instructions

3

Touch screen in the cockpit

The network is overloaded

Heat: temperature > 45° C



Computer task: keep each braking system inside its safety borders

Heat impact: the control law is false

The **India stakeholder**, characterized by summer « high temperature » contexts, was ignored during design: **reality is always stronger than documentation**

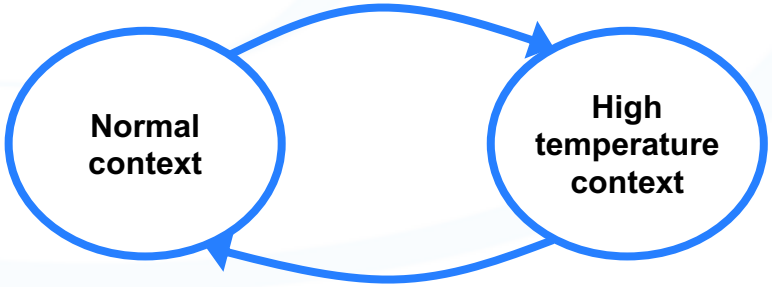
1 Systems Engineering Answer 3 – Operational architecture

1. India is a stakeholder of the Calcutta subway
2. India has summers where temperature can reach 50°C in the shadow

Stakeholder & constraint analysis

The system of interest: the Calcutta subway

T > T0 | High temperature special control law

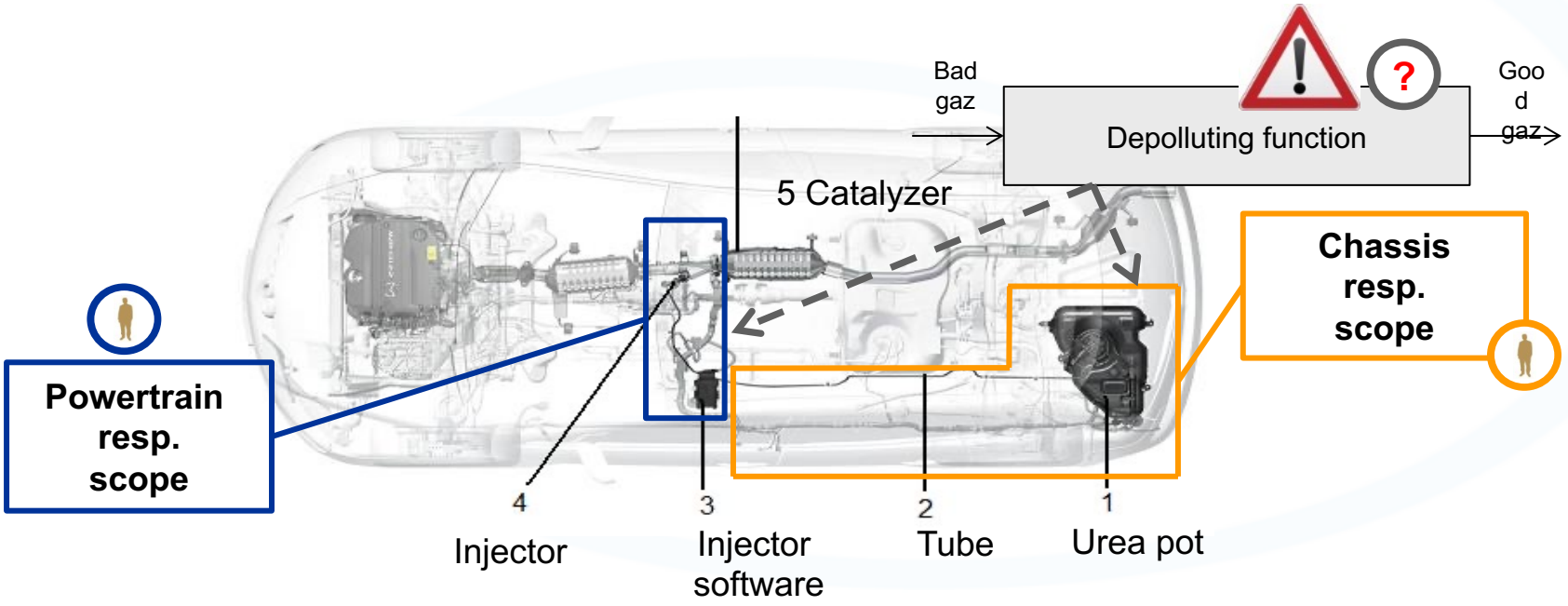


T < T1 | Normal control law

Lifecycle analysis leads to identify an Indian specific operational context: the « high temperature context »

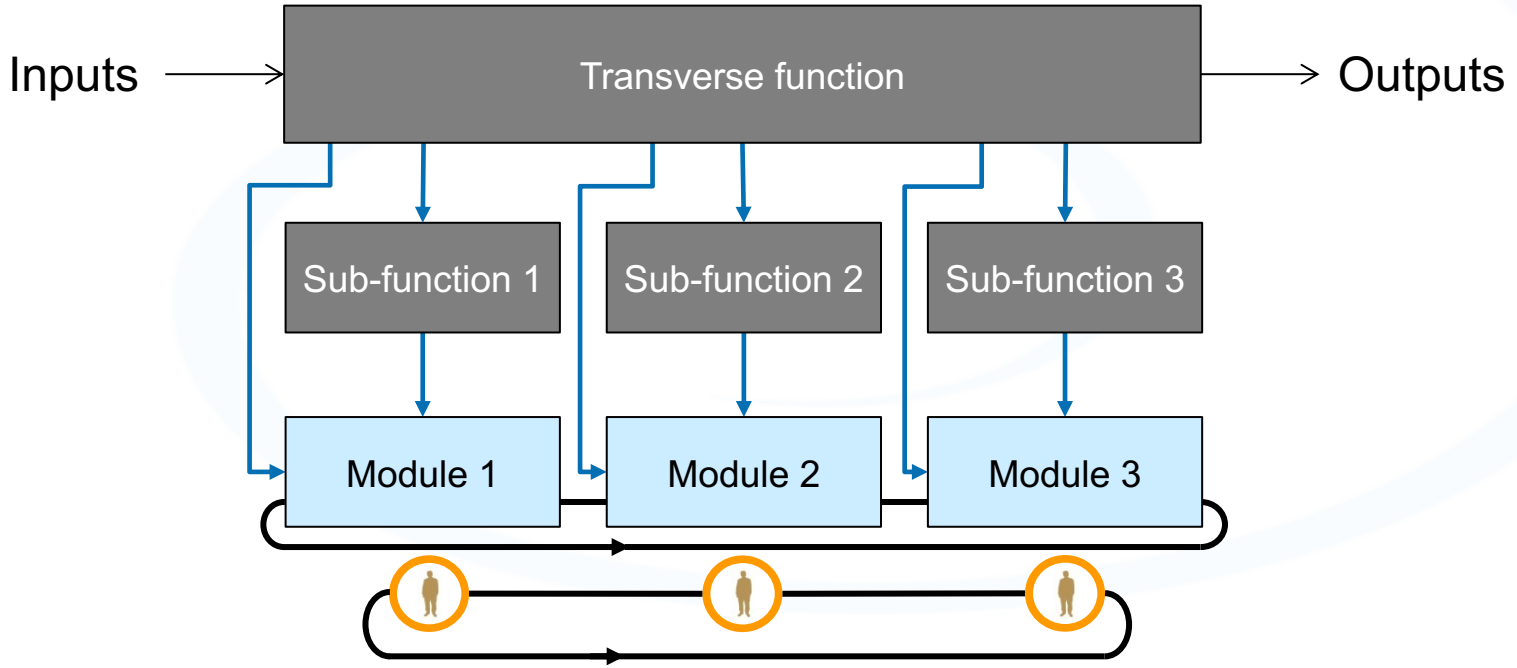
Operational architecture analyzes the **environment** of a **system** that contains all the **external systems** that may have an influence or an interaction with the system of interest

1 System Issue 4 – Forgetting to manage transversal behaviors



A car constructor wanted to implement a **depolluting technology** working by urea injection in the engine exhaustion gaz. The depolluting function is cut into two constructional subsets under the powertrain & chassis project responsibilities with **nobody in charge of the global coherence** of the transverse function. After 5 years of design: a non robust system at the very end plus **5 M€ budget over costs**.

1 Systems Engineering Answer 4 – Functional architecture (1/3)



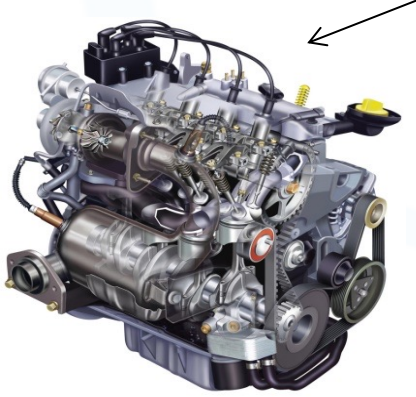
Transverse functions are modeling **functional cooperation** between **hardware and software modules** and hence indicate where **project actors** shall collaborate.

Mastering these transverse functions is therefore key!

1 Systems Engineering Answer 4 – Functional architecture (2/3)

Provide Energies

Provide Torque | **Provide Electricity** | **Provide Vacuum**

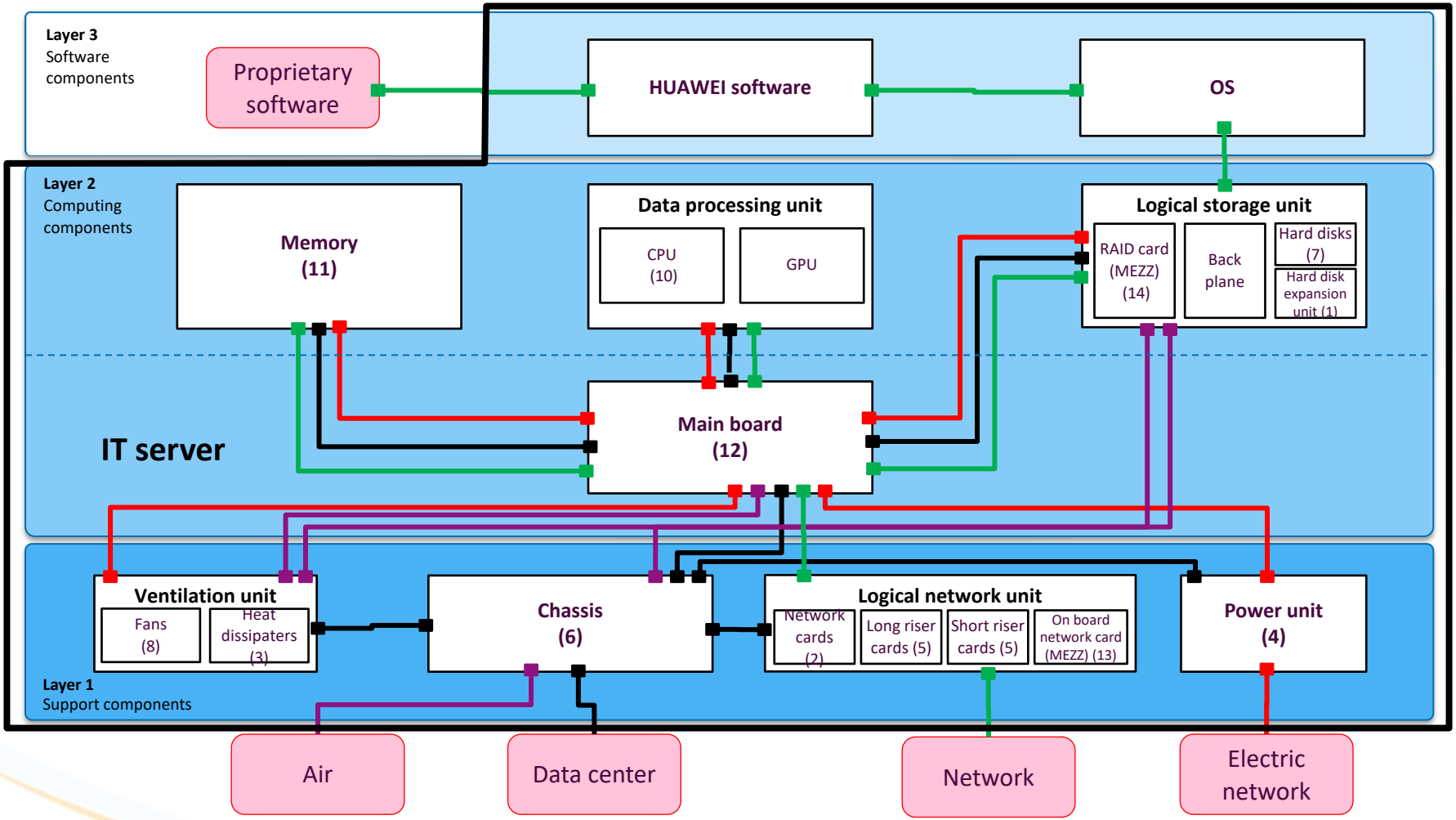


Thermic Vehicle

Electric Vehicle

Moreover functions shall be **technology independent**: they thus provide a **stable basis** for product design that allows to **optimize concrete engineering choices**

1 Systems Engineering Answer 4 – Functional architecture (3/3)



Note that the stability of a functional architecture also allows to **standardize** it which may be interesting to design standard structuring interfaces.

Table of contents

CESAMES
Center of Excellence on Systems Architecture, Management, Economy and Strategy

2 Key system concept – Time scale and states

A **time scale T** is a totally ordered set with the two following properties:

- T has a unique minimal element t_0
- each element $t \in T$ has a (unique) greatest upper bound within the time scale, called its successor and denoted t+ within T.

Time scales are used to model the **moment of times where one can observe a system**. Up to rescaling, two types of time scales are key in practice:

- **discrete** time scales where t+ = t + 1,
- **continuous** time scales where t+ = t + dt where dt stands for an infinitesimal quantity.

Discrete time scales model event-oriented systems (such as software systems regulated by a discrete clock) when continuous time scales model physical continuous phenomena.

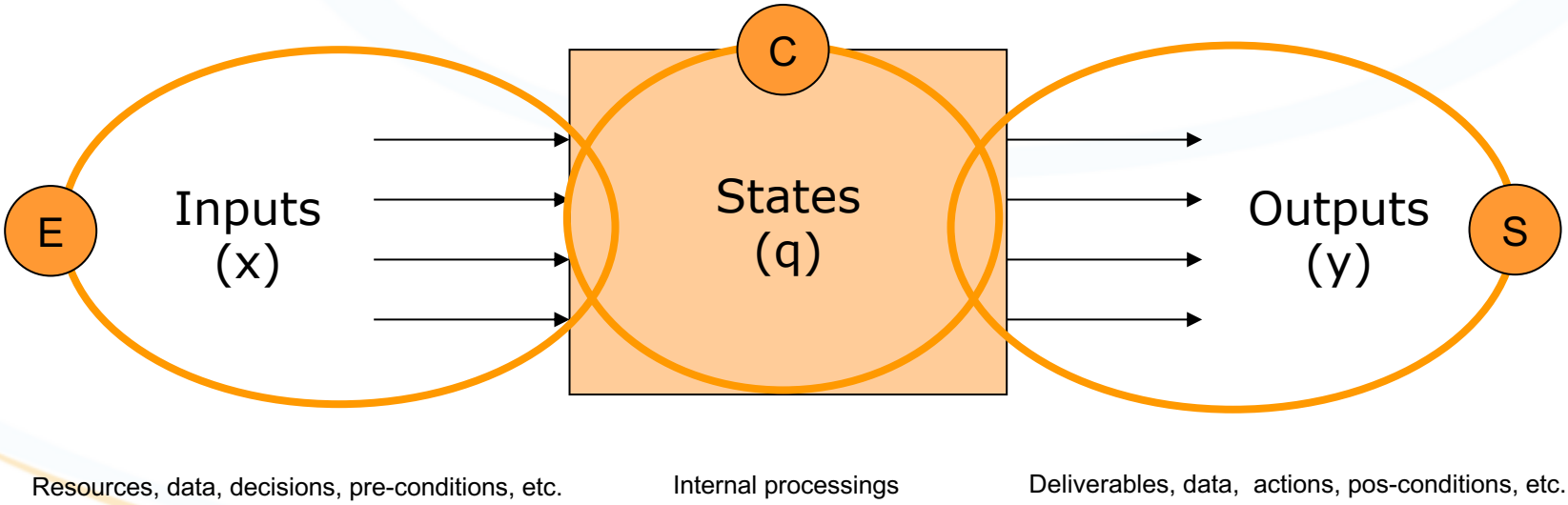
A **state** q associated with a time scale T is then just a subset of T, when a **state set** Q associated with T refers to a partition of T, that is to say a set of states such that:

- each moment of time of T is in some state $q \in Q$,
- no moment of time of T can be in two different states $q \in Q$.

2 Key system concept – System (1/2)

Definition 1 – Formal system – Let T be a time scale. A **formal system** S is characterized on one hand by an input set X, an output set Y and a state set Q associated with T and on the other hand by two kinds of behaviors that link these systemic variables among T :

- a **functional behavior** that produces an output $y(t+) \in Y$ at each moment of time $t \in T$, depending on the current input $x(t) \in X$ and internal state $q(t) \in Q$ of the system,
- an **internal behavior** that results in the evolution $q(t+) \in Q$ of the internal state at each moment of time $t \in T$, under the action of a system input $x(t) \in X$.



Standard representation of the signature of a system

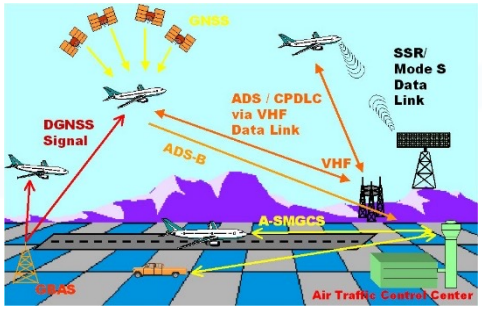
2 Key system concept – System (2/2)

hardware systems
(physical laws)

software systems
(logical laws)

Everything is a system

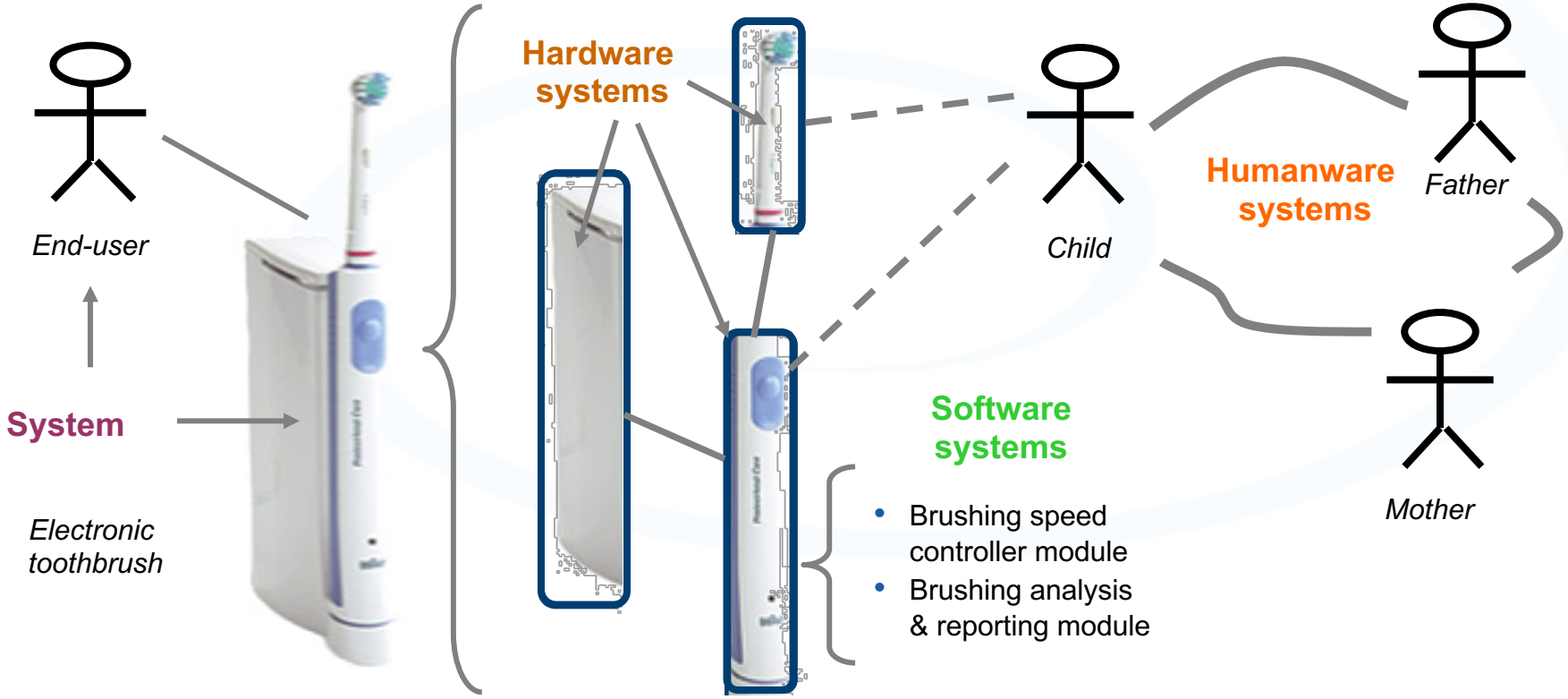
”humanware” systems
(laws of human behavior)



systems of systems
(interoperability standards)

Definition 2 – Real system – An object of the real world will be called a **real system** as soon as its structure and its behavior can be described by a formal system that will be then called a **model** of the considered real system.

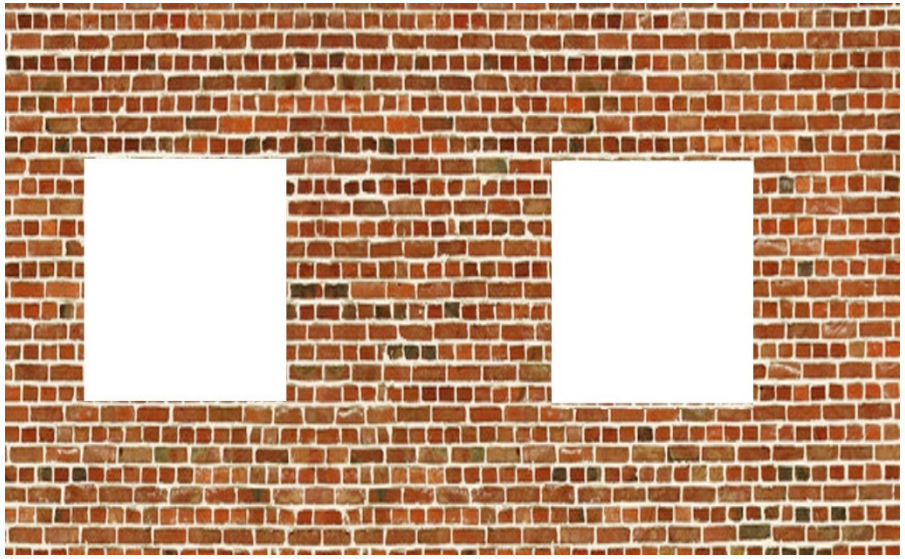
2 Key system concept – Integration – First unformal attempt



Integration is the process that allows to **build** a **system based on other systems** (**hardware**, **software** & **humanware**) that are organized in such a way that the resulting integrated system can perform – in a given environment – its mission

2 Key system concept – Emergence

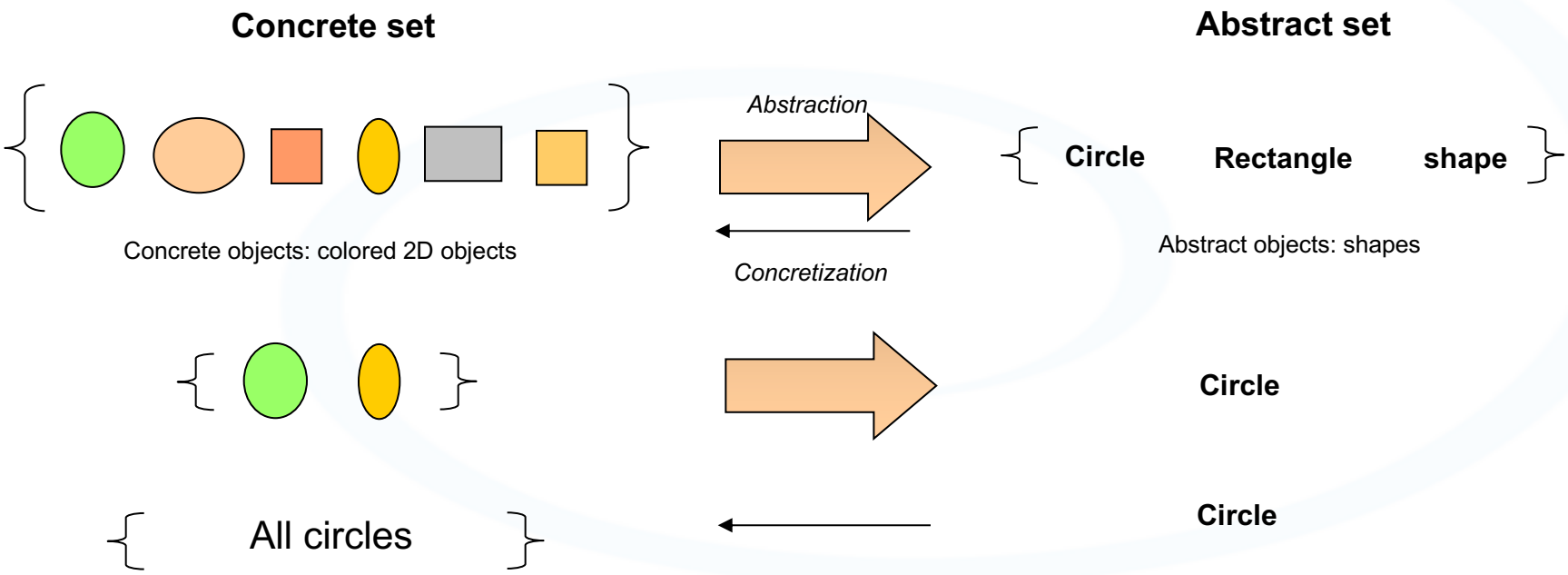
Bricks



An integration of bricks leads to a wall which may have a number of emergent properties

The point is that integration induces **emergence** which was not captured at all by the previous definition: an integrated system has **always emergent properties** that **cannot be deduced** from the **properties of the systems with which it is built.**

2 Key system concept – Abstraction



Formal definition:

- Abstraction(Concretization(A)) $\subseteq$ A
- C $\subseteq$ Concretization(Abstraction(C))

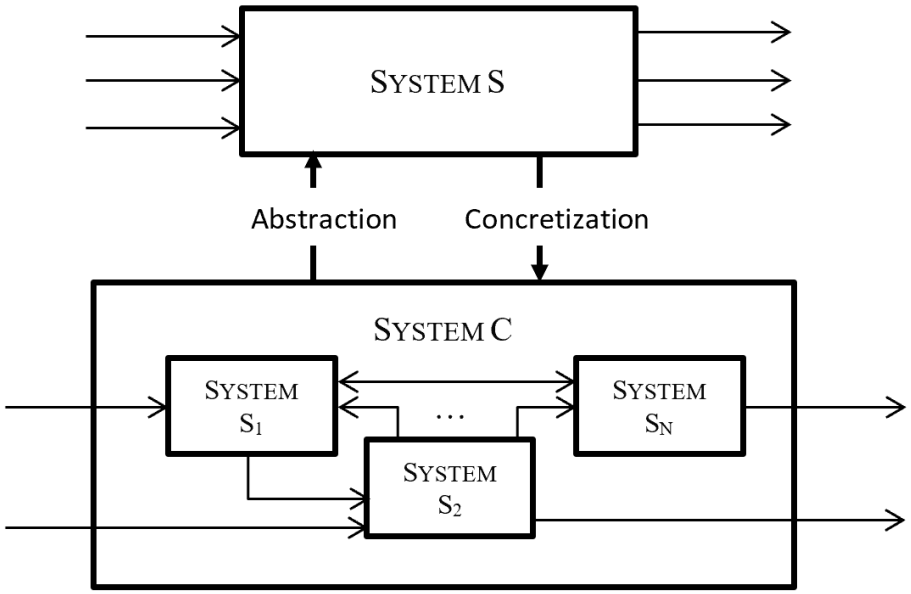
An **abstraction** is a **not** (too much) **destructive idealization** of a set of objects

Source: *Abstract interpretation*, Cousot & Cousot (1977)

2 Key system concept – Integration (1/2)

Definition 3 – Integration – Let S_1, … , S_N be a set of N (formal) systems. One says then that a (formal) system S is the result of the **integration** of these systems if there exists on one side a (formal) system C obtained by composition of S_1, …, S_N and on the other side dual abstraction and concretization operators that allow to express:

- the system S as an abstraction of the system C,
- the system C as a concretization of the system S.



Formal integration of systems

2 Key system concept – Integration (2/2)

The definition of integration has several immediate important consequences:

1. One **never obtains a complete model** of an integrated system by **composing the models of its sub-systems**.

   This first property motivates the existence of a specific discipline dedicated to the construction of models of integrated systems which is **systems architecting**.

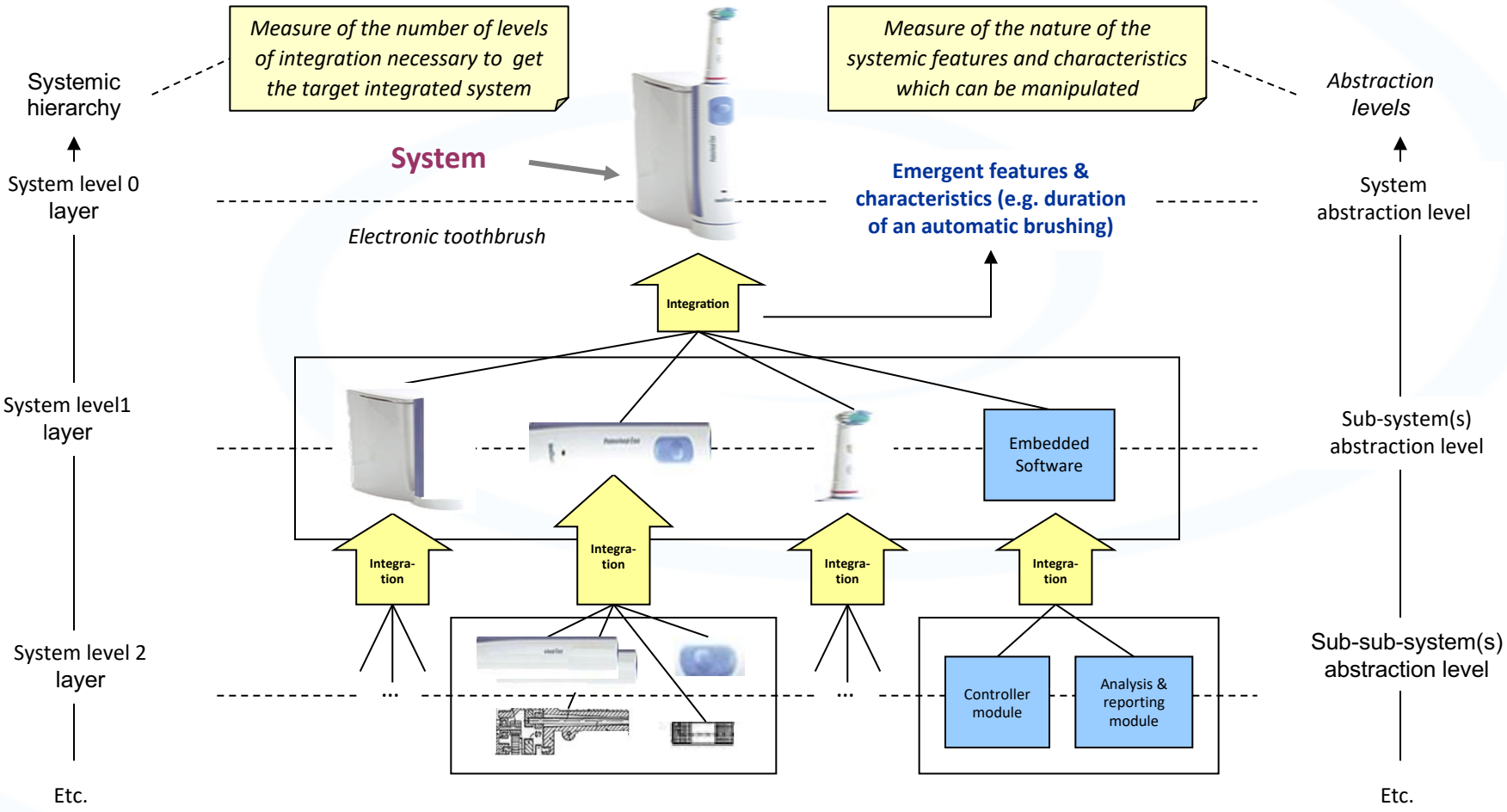
2. The **knowledge of the properties of the sub-systems** of a system **does not give us the full knowledge of the properties of the system**. As a matter of fact, it is much easier to deduce the properties of the sub-systems of a given system from the properties of this system, as far as one knows the integration law.

   This second property motivates to favor a **top-down approach** for modelling an integrated system.

   Note that in practice, one must take into account bottom-up feedbacks which results in a mixed **top-down major and bottom-up minor** approach for systems architecting.

3. In the integration definition, abstraction applies both on the input / output sets and on the time scales. As a consequence, the **systemic hierarchy** which is induced by integration is both a **time & space abstraction hierarchy**.

2 Key system concept – Systemic hierarchy



Every **system** can be decomposed according to a **systemic hierarchy**, which corresponds to its **integration levels**, but which is also a hierarchy of **abstraction levels** reflecting the **emergent features** induced by each integration process

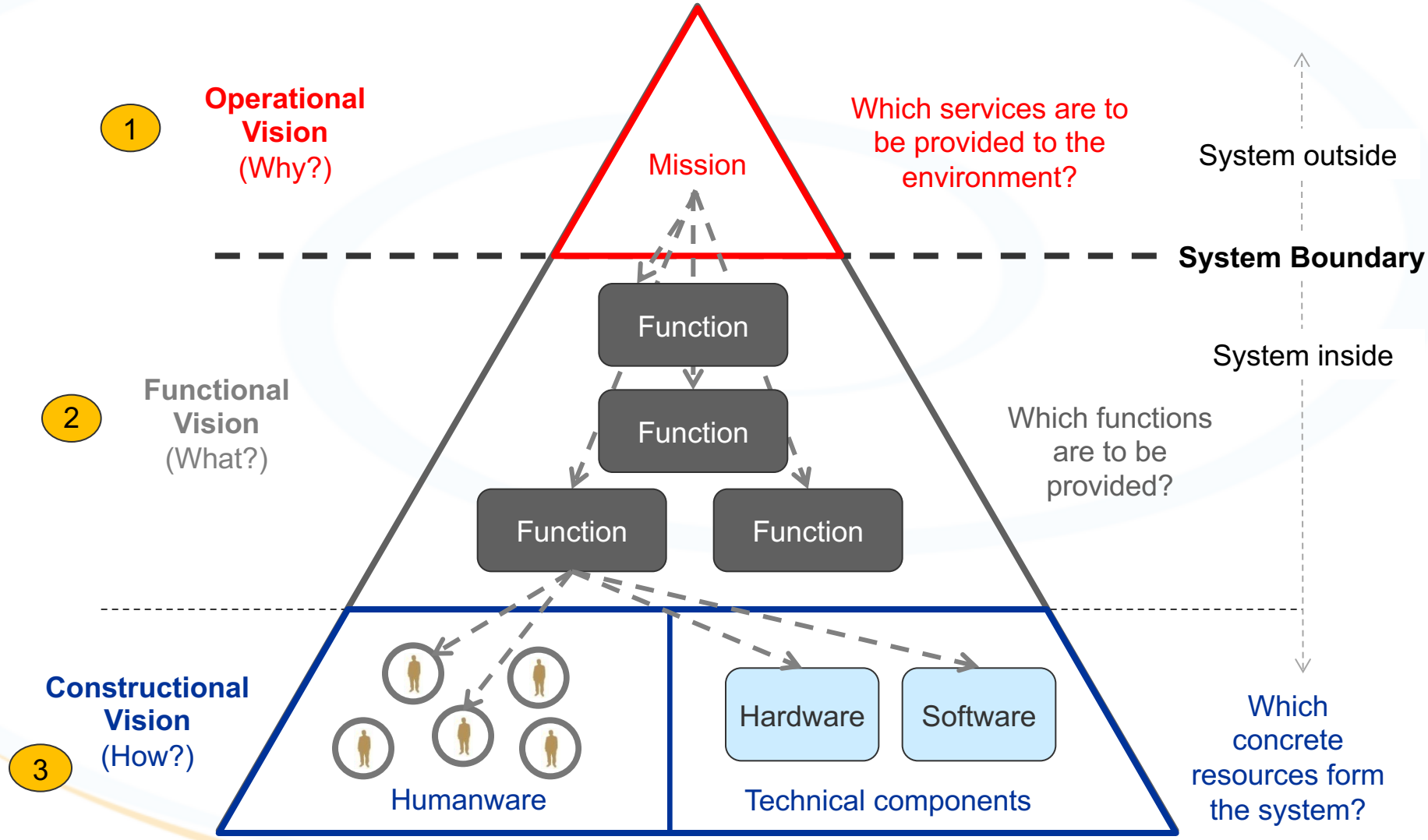
Table of contents

CESAMES
Center of Excellence on Systems Architecture, Management, Economy and Strategy

Table of contents

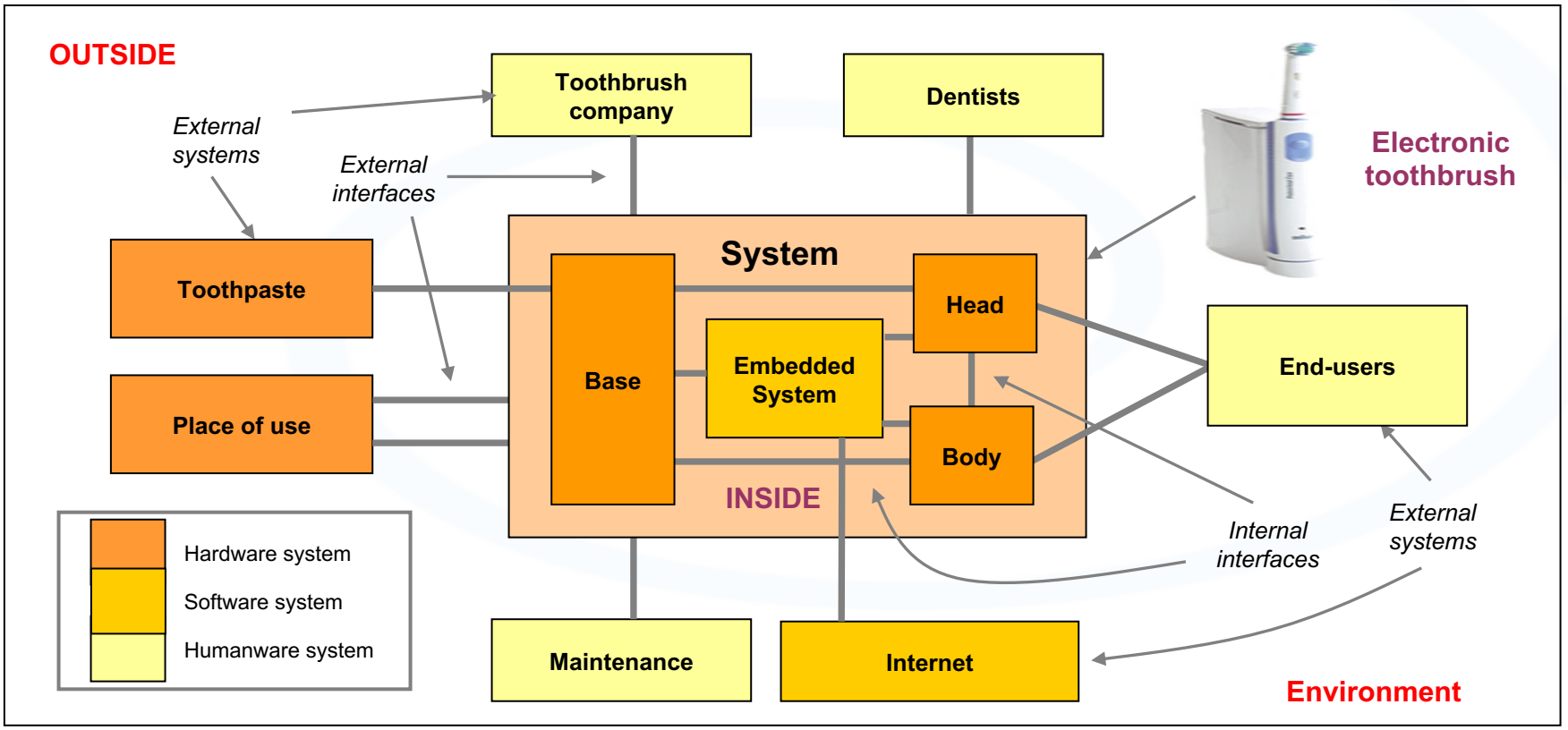
CESAMES
Center of Excellence on Systems Architecture, Management, Economy and Strategy

3 Key systems architecture concept – Architectural pyramid



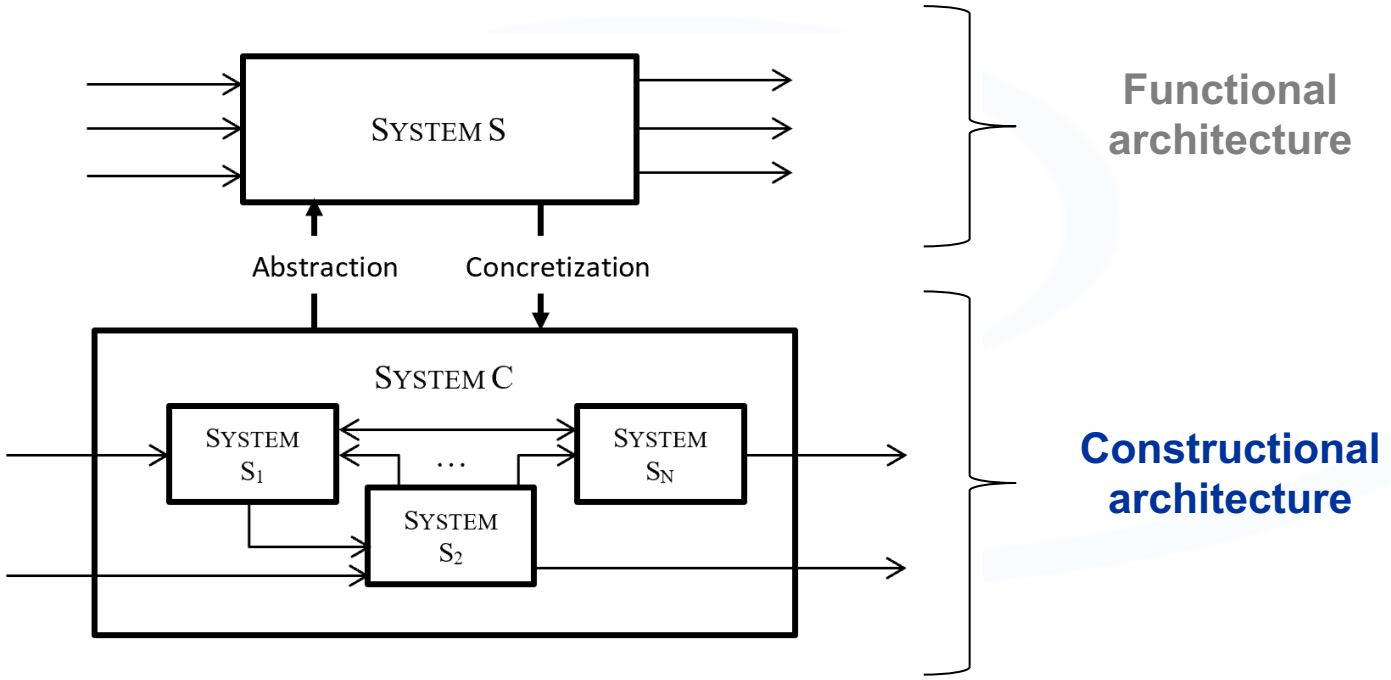
We can therefore study any system from **three** visions

3 Why three architectural visions? (1/2)



Any **system S** is a part of its **environment Env(S)** which is the "smaller" system that encapsulates S and which is supposed without any external interface. To understand how the system S interacts with the other systems of Env(S), one has to model this environment. This activity is classically called the **operational architecture** of the system S, even if strictly speaking it is the architecture of the environment of S.

3 Why three architectural visions? (2/2)



To describe an integrated system, one must – by definition – describe how its different sub-systems are composed, which correspond to **constructional architecture**, but also the behavior of the system in its whole, which correspond to **functional architecture**.

3 Relationships between the three architectural visions

Electronic toothbrush example

Electronic toothbrush

Operational vision

(analysis of the environment of the target system)

Functional vision

(abstract internal behavioral analysis of the target system)

Constructional vision

(concrete internal structural analysis of the target system)

OUTSIDE

INSIDE

Electricity

Power supply

End-users

Queries

Reports

Internet

Electricity

Forces

Queries

Measures

Interface

Interface

Interface

Provide electrical power

Generate brushing force

Measures

Provide brushing capability

Toothbrush

Electricity

Allocation

Allocation

Allocation

Electricity

Electricity

Holding-force

Body

Electricity

Brushing force

Measures

Embedded Software

Queries

Measures

Base

Mechanical Interfaces

Head

Toothbrush

Table of contents

CESAMES
Center of Excellence on Systems Architecture, Management, Economy and Strategy

3 A bit of archeology: Hoare equivalence (1969)

With each **computable function** f, that is say a function that can be computed by a software program, one can associate a **logical predicate**, called an Hoare assertion, which models the relationships between the pre-conditions and the post-conditions of the computable function f.

y = add(x): return(x+1)	x = x0 ⟶ y = x0+1
Example of a computable function	Associated Hoare assertion

Hoare theorem states then that one **can equivalently define** any computable function:

- either in extension by an algorithm,
- or in intension by an Hoare assertion.

This theorem is the cornerstone of **requirements engineering** in software engineering since it states that any function that can be computed by a software can be specified either through an algorithm or through a set of requirements (mathematically a set of requirements can be encoded into a single requirement).

3 Temporal system logics

Mimicking classical results of computability theory, one can associate with any system a **temporal system logics** defined as follows:

- S $\models$ TRUE for any system S,
- S $\models$ O(x,y,q) if and only if $x(t_0) = x$, $y(t_0) = y$ and $q(t_0) = q$ (when $x, y, q \neq \varnothing$),
- S $\models$ f AND g if and only S $\models$ f and S $\models$ g,
- S $\models$ NOT f if and only if one does not have S $\models$ f,
- S $\models$ X f if and only if $S[t_0+] \models f$,
- S $\models$ f U g if and only if $\exists\, t \in T$ such that $S[t] \models g$ and $S[u] \models f$ for all $u \in T$ with $u < t$.

It is also interesting to provide explicitly the semantics of the two additional temporal operators that we introduced above, as it can be deduced from the previous definitions:

- S $\models \Box$ f if and only if for all $t \in T$, one has $S[t] \models f$,
- S $\models \Diamond$ f if and only if there exists $t \in T$ such that $S[t] \models f$.

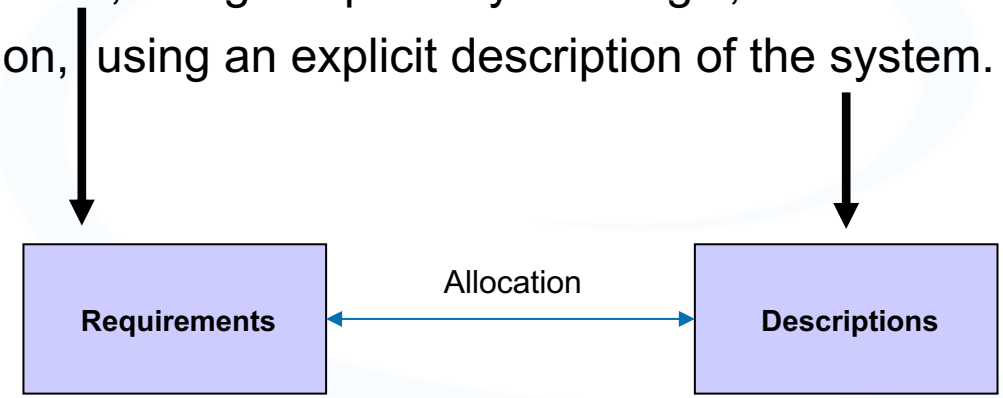
A **system requirement** is then just any **logical property of a system** described using such a temporal system logical. A maintainability property can for instance be stated as:

$$\text{Maintainability} = \Box\,(\,\text{NOT O}(\varnothing, \varnothing, \text{“normal“}) \rightarrow \Diamond\, \text{O}(\varnothing, \varnothing, \text{“normal“})\,)\,.$$

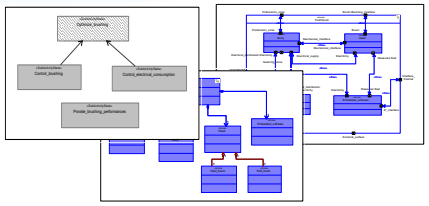
3 Systems can be described either in intention or in extension

Exactly as in classical computability theory, one can prove that one **can describe equivalently** a system:

- either in intention, using temporal system logic,
- or in extension, using an explicit description of the system.



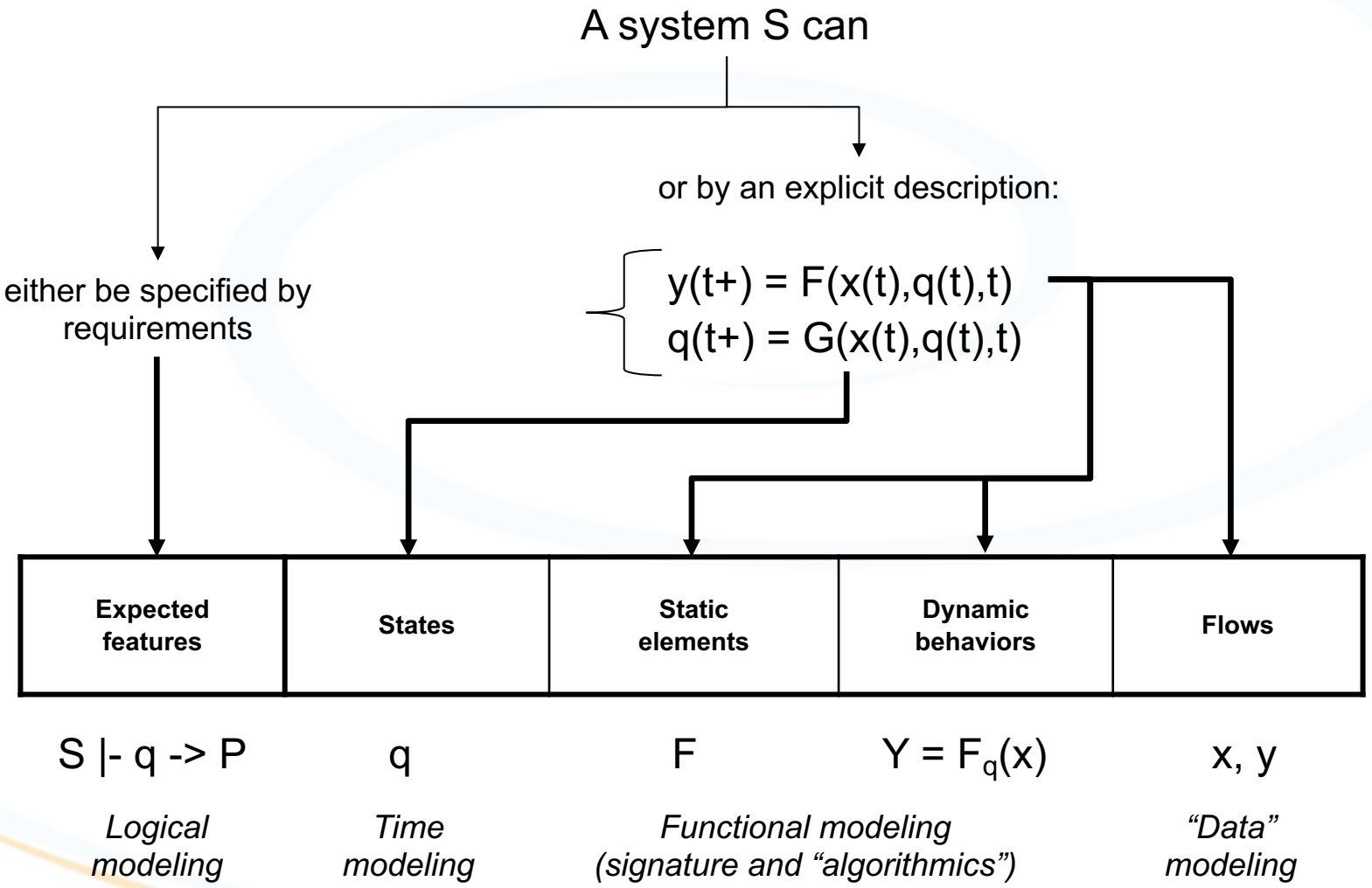
Requirement type: the system> shall <do or be something> …



Mathematically these two modes of specifications are equivalent. However **their practical complexity is not equivalent at all**. As a consequence, any concrete system specification is always a mixture of these two types of specifications.

3 CESAM systems architecture framework

Generic structure of a description view



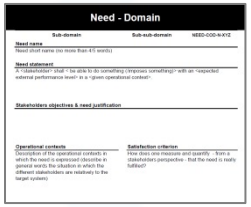
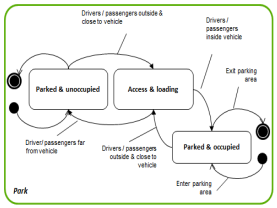
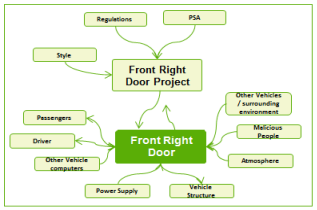
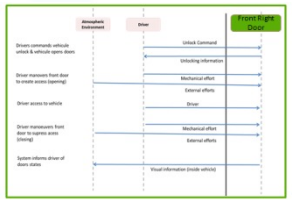
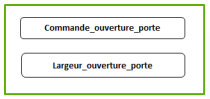
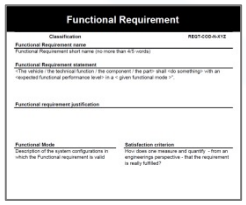
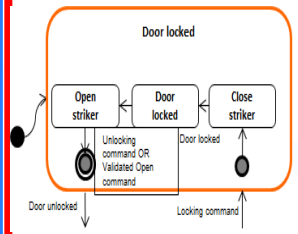
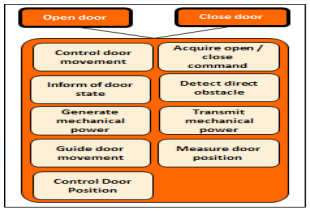
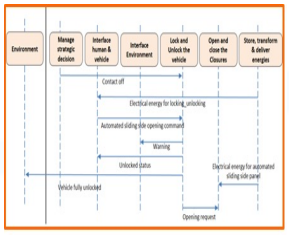
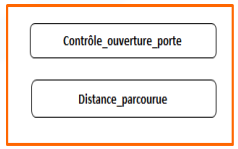
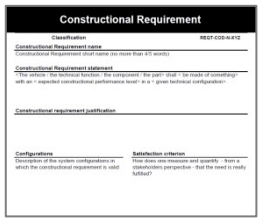
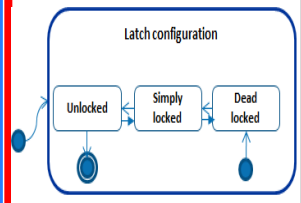
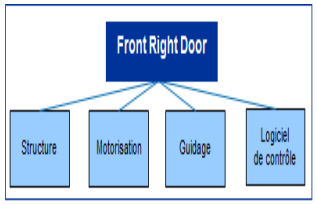
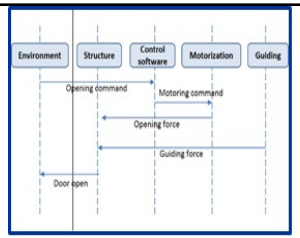
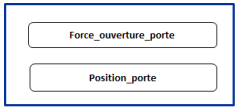
Generic structure of **CESAM framework**

3 CESAM systems architecture framework

Requirements and descriptions

CESAM

Sliding side door

Systems architecture visions	Requirements: Expected features	Description: States	Description: Static elements	Description: Dynamic behaviors	Description: Flows
Operational vision					
Functional vision					
Constructional vision					

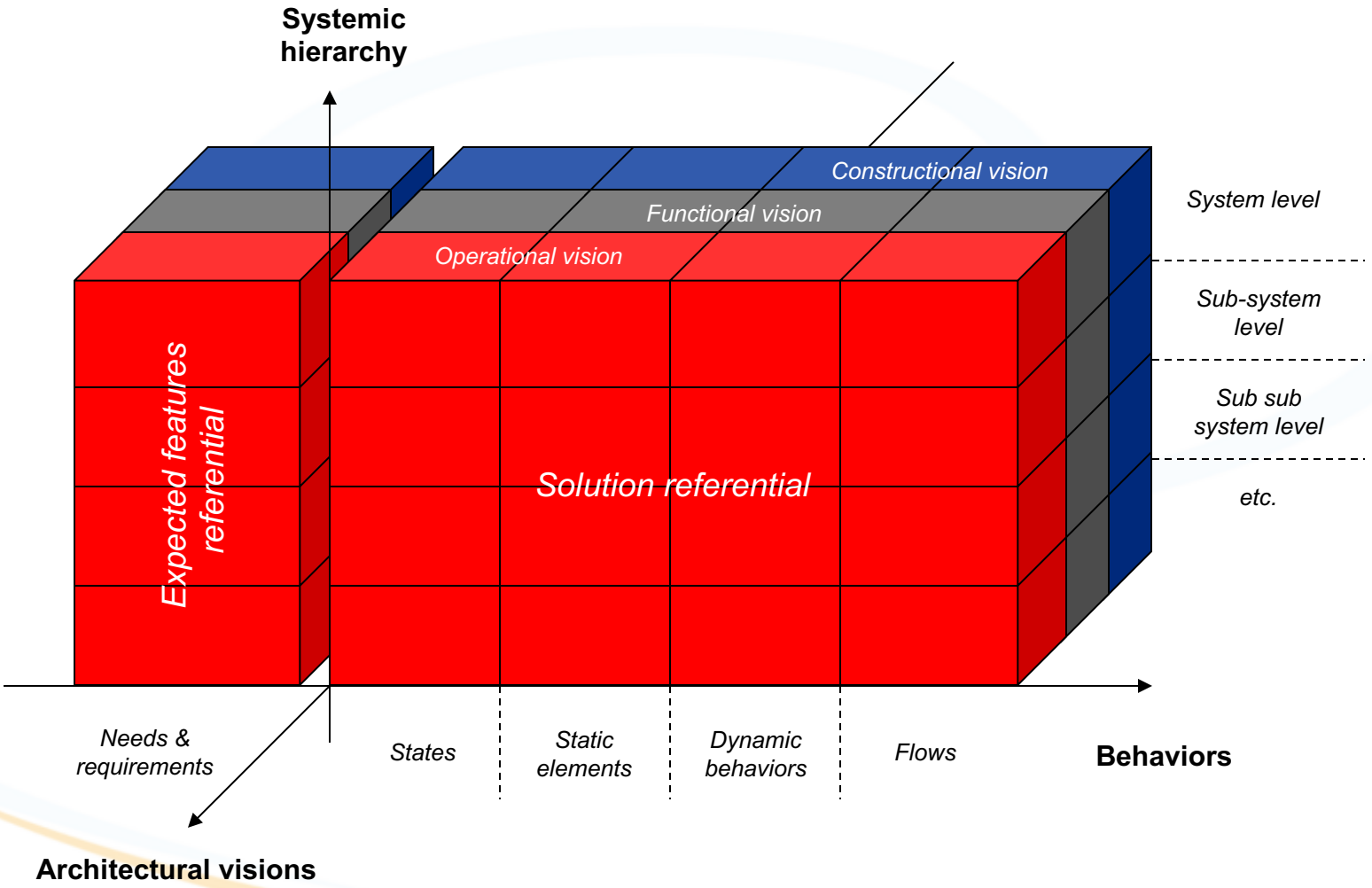
Towards specification files

Generic structure of a **systems architecture referential**

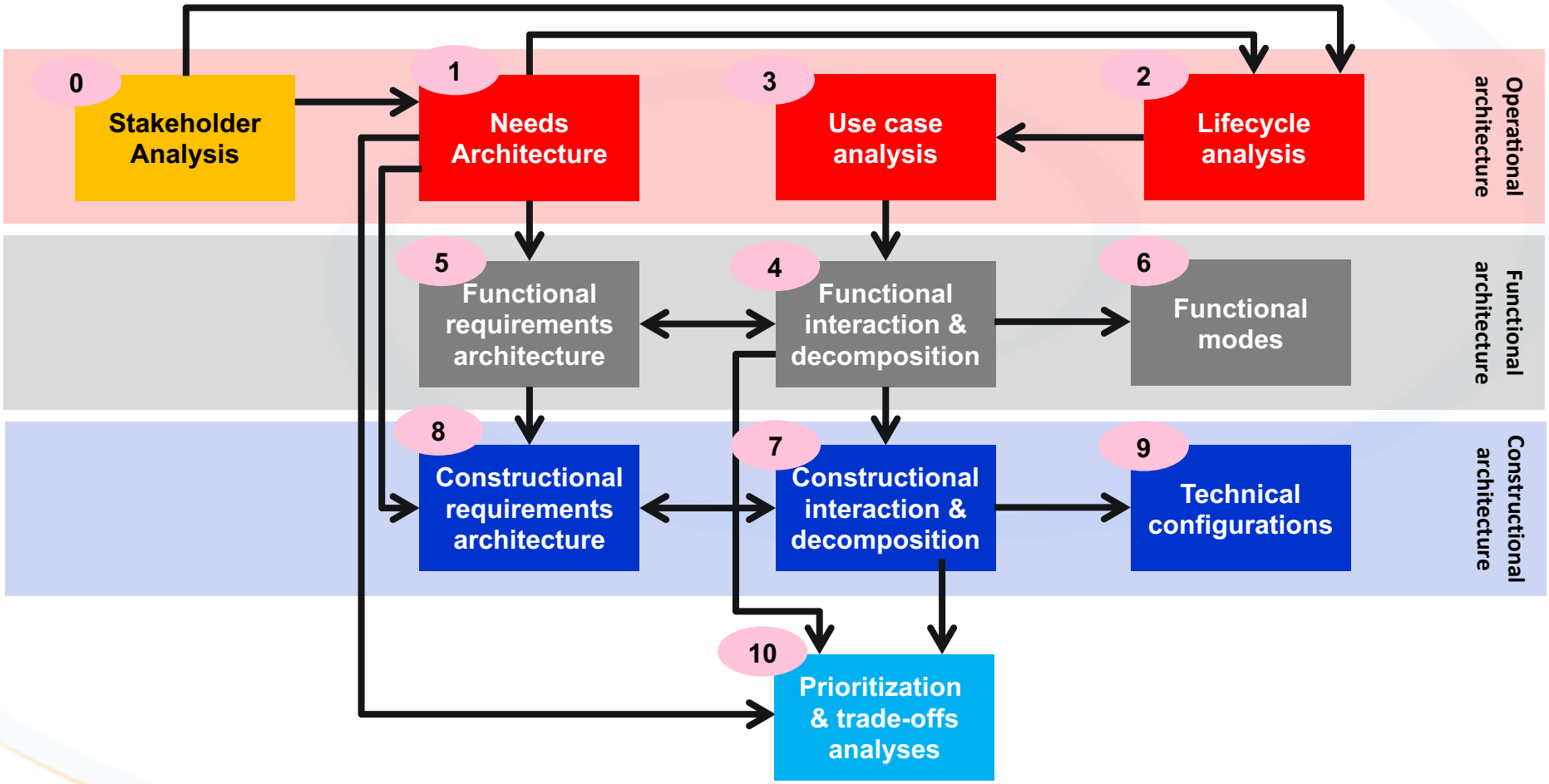
3 CESAM systems architecture framework

Synthesis: the CESAM systems architecture cube

An analysis method for any given system



3 Main steps of the systems architecting process



The **systems architecting process** – which is **not sequential** at all – consists in moving in the systems architecture cube following a standard way.

Table of contents

CESAMES
Center of Excellence on Systems Architecture, Management, Economy and Strategy

Table of contents

CESAMES
Center of Excellence on Systems Architecture, Management, Economy and Strategy

4 System complexity hierarchy

Type of system	Characteristics	Typical example	Design strategy
Ecosystem	Weak coupling of systems of systems		Actors influence
System of systems	Weak coupling of distributed systems		Interfaces standardization
Product	Strong coupling of fixed components	Aircraft 	W-cycle
Product component	Does not exist independently of a product	 Aircraft engine	V-cycle

Components, integrated products, systems of systems and ecosystems are **all systems**.
From a modelling perspective, they **can all be modelled using the same formalism**.

4 The difference between systems of systems and usual systems engineering rather relies on specific design issues

Distribution

Inter-operability

Competition

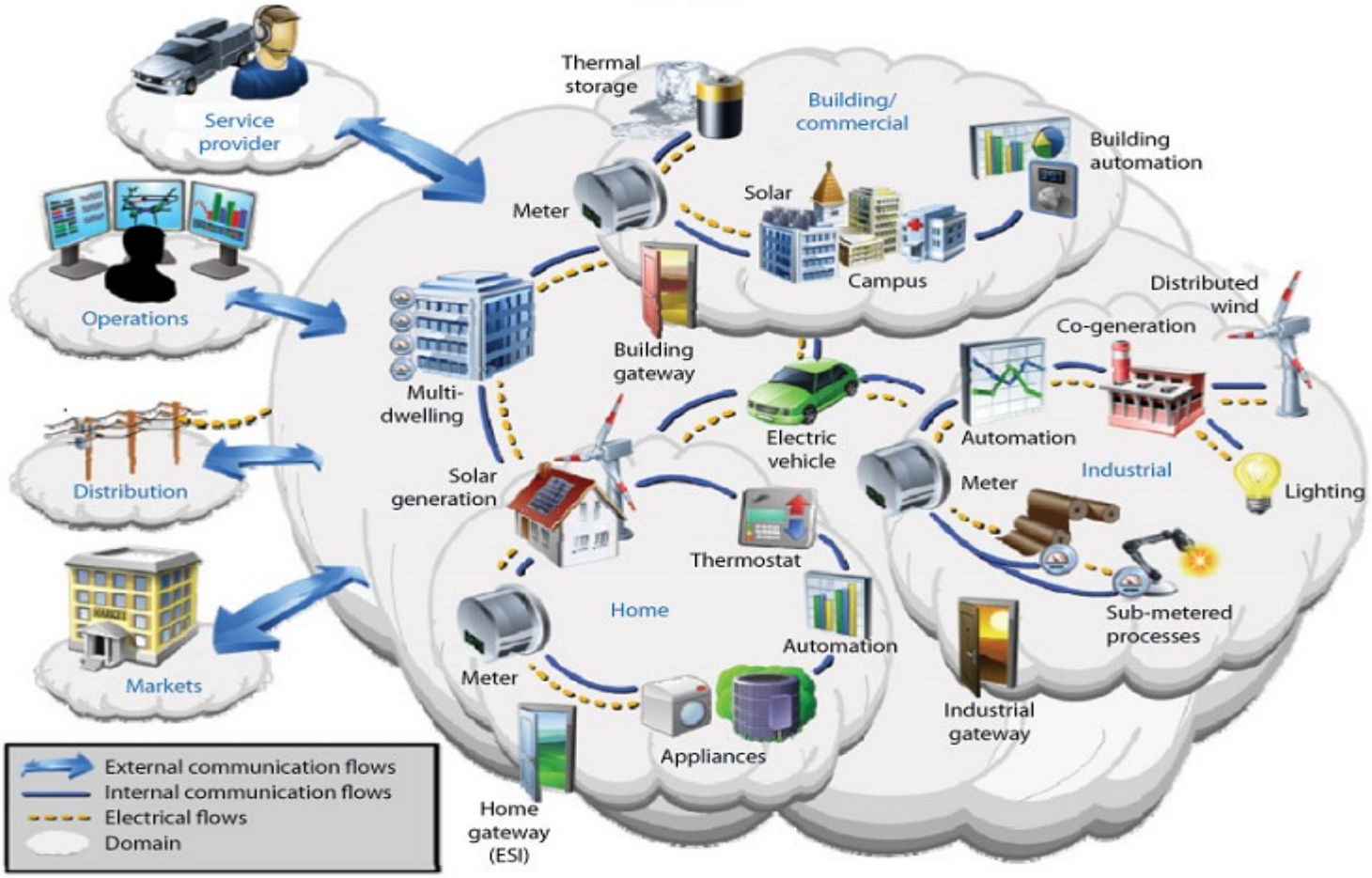
Unpredictability

Specific systems of systems features that are difficult to take into account in their design

The main difference between systems of systems and usual systems engineering is thus **not a matter of modelling**, but rather of **difficult design issues, specific to systems of systems**, that must be taken into account by the systems of systems architect

4 Example: electrical vehicle ecosystem (1/2)

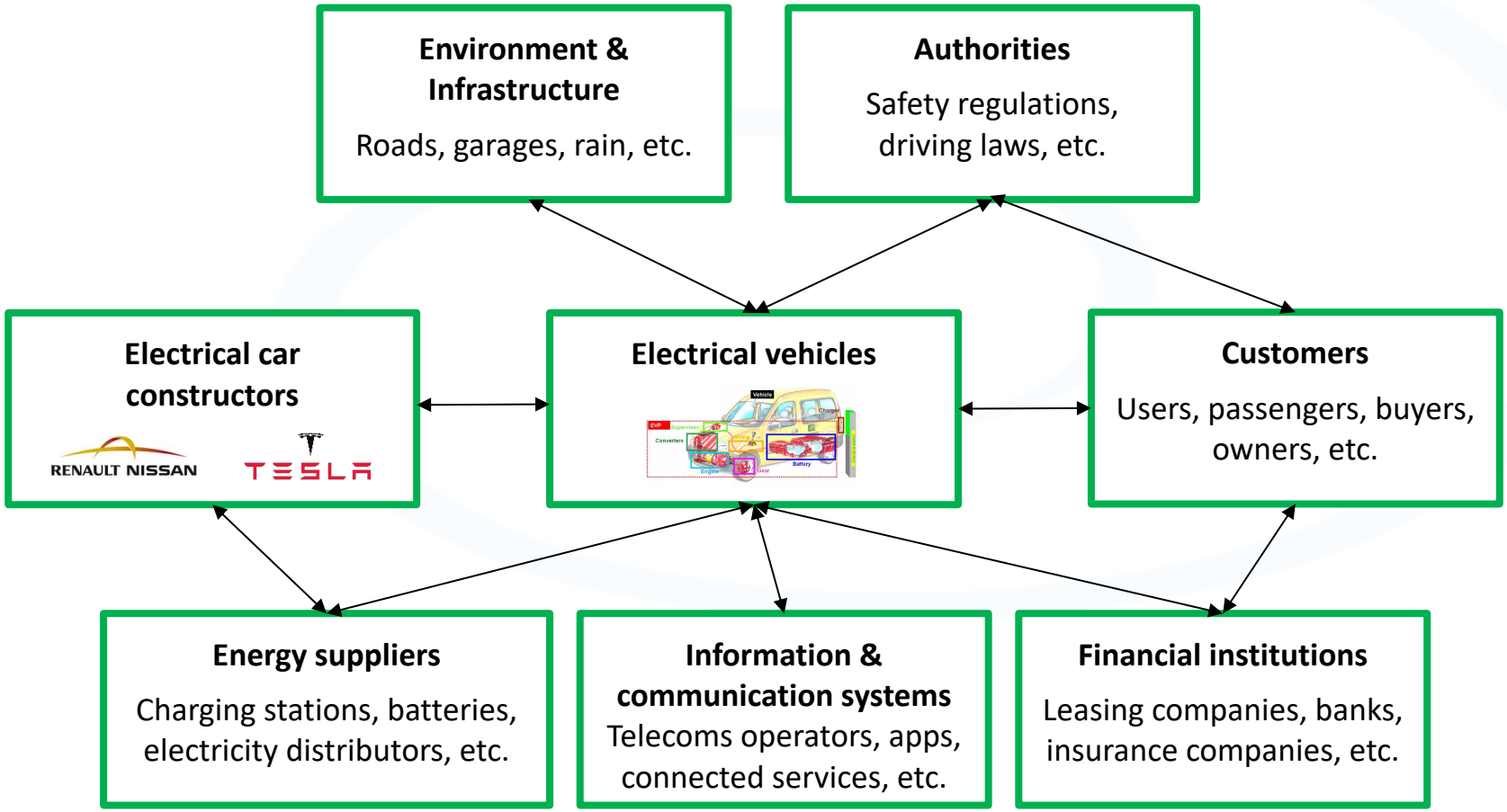
RENAULT NISSAN



Let us take the example of the **electrical vehicle ecosystem**
The problem to solve is to define its **dissemination strategy** within a territory

4 Example: electrical vehicle ecosystem (2/2)

RENAULT NISSAN

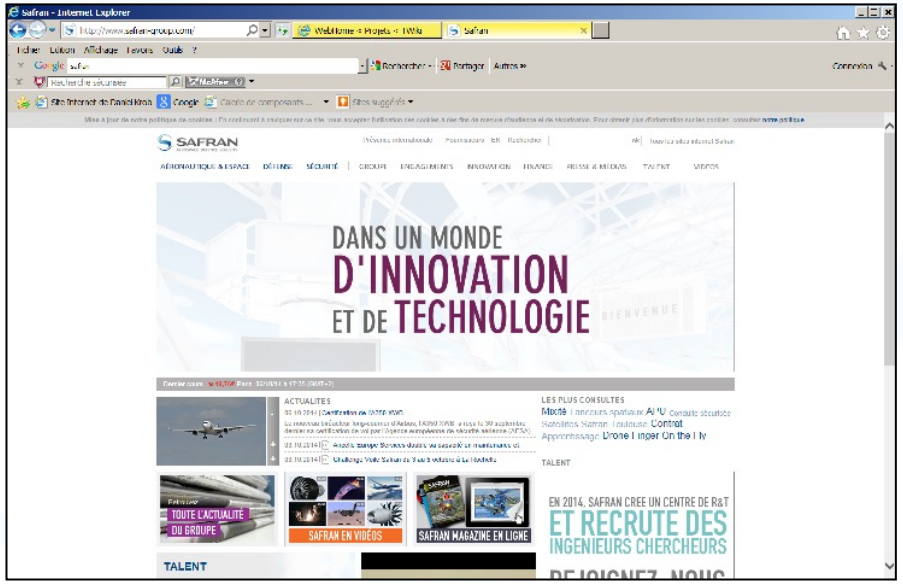


The **good win-win "to be" vision** can be seen as a **Nash equilibrium** that can be analyzed by means of game theory techniques appearing naturally in a system of systems context since the different systems composing the system of systems **are competing together** to capture the end-user value, but rarely used in usual systems engineering.

Table of contents

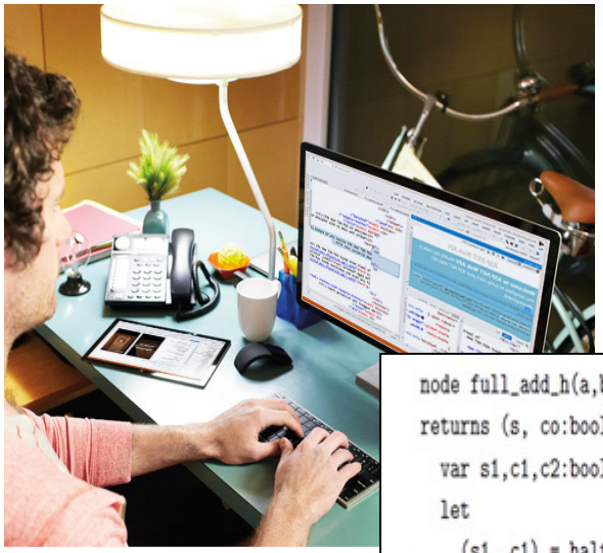
CESAMES
Center of Excellence on Systems Architecture, Management, Economy and Strategy

4 Since 50 years, one knows how to compile a software



Interpretation or compilation

4 And what if tomorrow, one could synthesize systems?



```
node full_add_h(a,b,c:bool)
returns (s, co:bool);
  var s1,c1,c2:bool;
  let
    (s1, c1) = half_add(a,b);
    (s, c2) = half_add(c, s1);
    co = c1 or c2;
  tel;
```

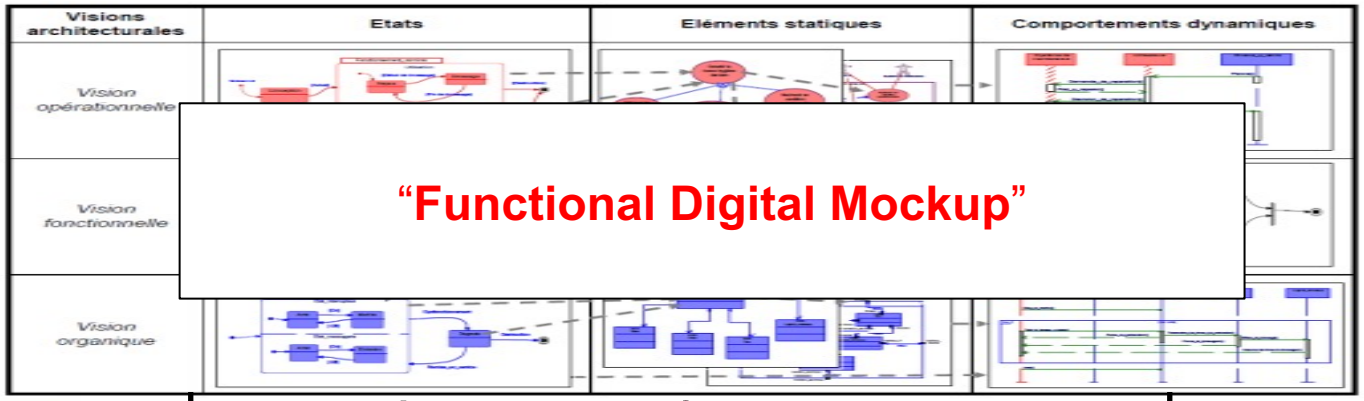
3D printer

Synthesis

4 What functional architecture for a system synthesis tool?

INTEGRATION LOGICS

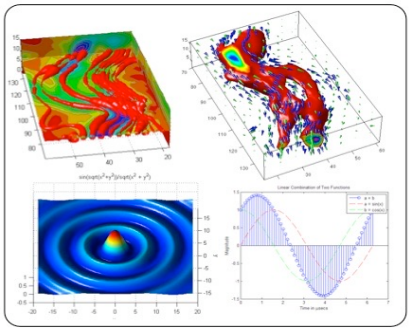
System Synthesis



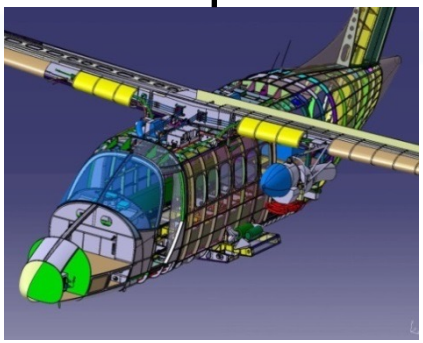
"Functional Digital Mockup"

"Functional" modelling

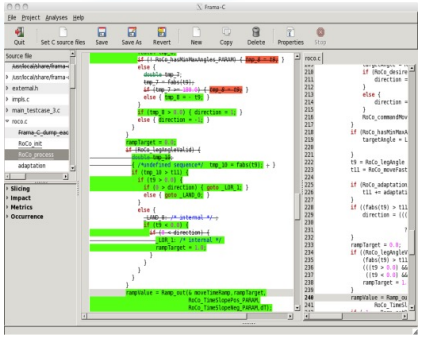
COMPONENTS



Multi-physics modelling



Geometric modelling



Embedded systems modelling

4 What fundamentals for a system synthesis tool?

- **The central target paradigm:** a **"system-oriented" programming language**, constructed on a **rigorous systemic semantics**, allowing to specify a system in a **completely non-ambiguous way**

- **The target modelling activity:** a **"programming" process for a system**, seen as an **abstract machine** managing and transforming **multi-scale physical, data and informational flows** along a **multi-scale discrete time**.

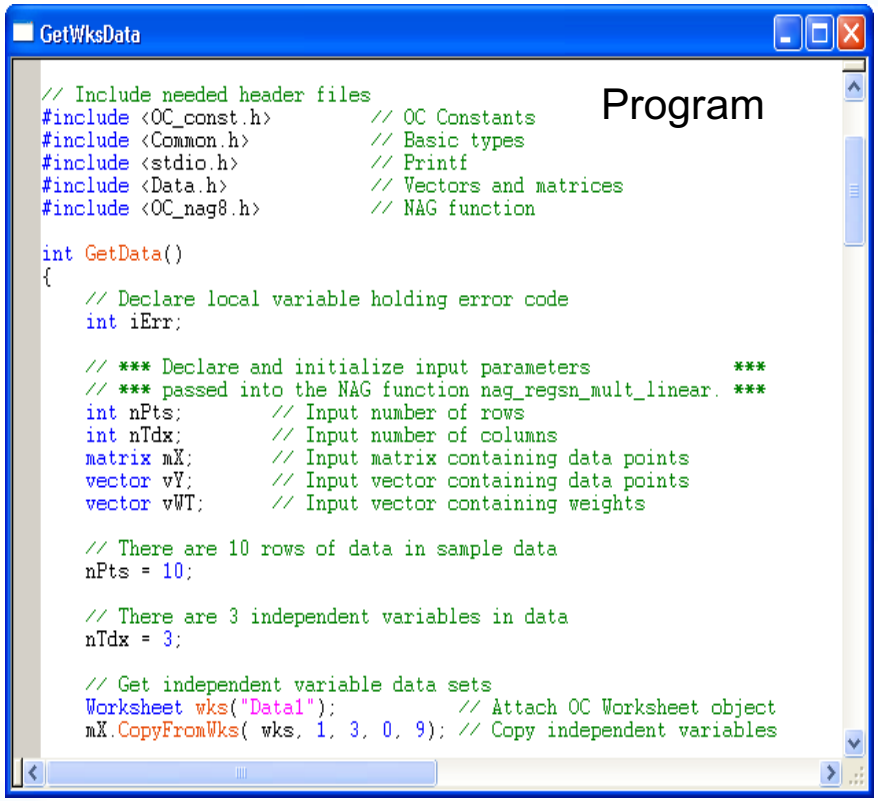
```
connector type Singleton(ePort p)
 define [p] // set of enabled interactions: {p}
end

connector type RendezVous(IntPort p1, IntPort p2)
 define [p1 p2] // set of enabled interactions: {p1 p2}
 on p1 p2 up {} down {p2.x = p1.x;}
end

connector type Broadcast(IntPort p1, IntPort p2, IntPort p3)
 define [p1' p2 p3]  // set of enabled interactions: {p1}, {p1 p2}, {p1 p3},
{p1 p2 p3}
 on p1       up {} down {}
 on p1 p2    up {} down {p2.x = p1.x;}
 on p1 p3    up {} down {p3.x = p1.x;}
 on p1 p2 p3 up {} down {p2.x = p1.x; p3.x = p1.x;}
end
```

Example of the first steps toward a system-oriented language (BIP – Joseph Sifakis)

4 The scientific issue to solve (1/2)



Program

Syntax

Denotational semantics:

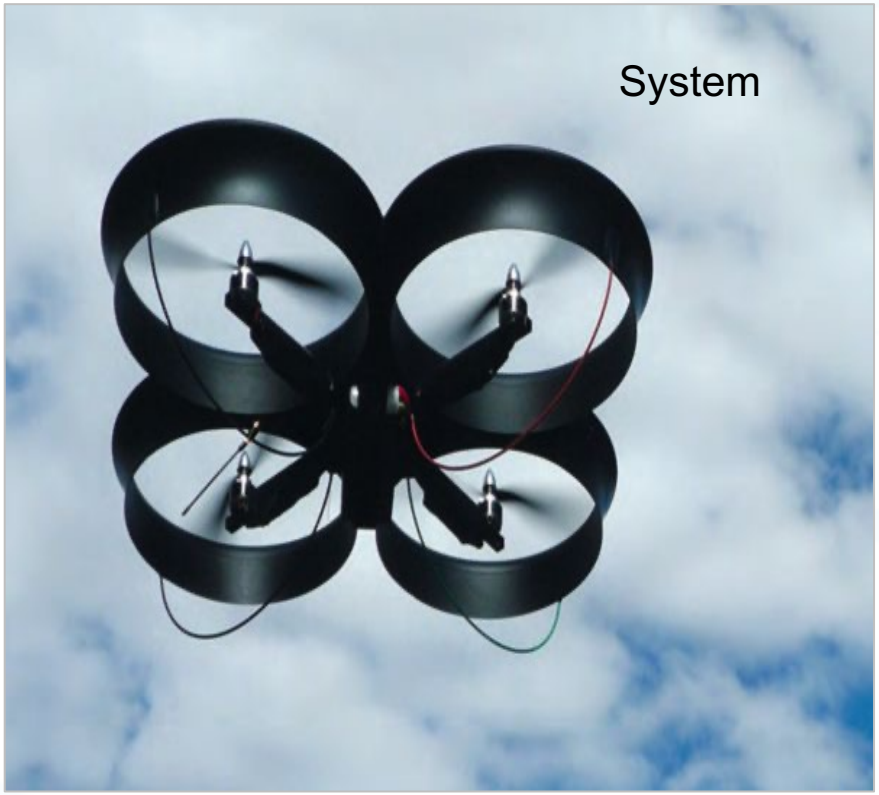
Logical syntax describing the mathematical object (function) associated with a program

Equivalence theorems (that guarantee that the program execution is the one which is described by its syntax)

Operational semantics:

Logical syntax describing the dynamics of the program (i.e. what is doing the program)

4 The scientific issue to solve (2/2)



Syntax?

Denotational semantics

Logical syntax describing the mathematical object (**formal system**) associated with a system

Equivalence theorems?
(that guarantee that the system execution is the one which is described by its syntax)

Operational semantics?

Logical syntax describing the dynamics of the system (i.e. what is doing the system)

Table of contents

CESAMES
Center of Excellence on Systems Architecture, Management, Economy and Strategy

5 A key role: the systems architect

1 – Architects definitely have a technical profile:

- They want to complete everything they start
- They have a rigorous & analytical position and can “drill down” when necessary
- They work in well-defined perimeters
- They are business & short-tem-oriented

2 – But good architects think different from many engineers:

- They look beyond the system boundary
  - They are receptive to commercial, political and cultural aspects
  - They think « solutions » and « stakeholders »
  - They always keep in mind the big picture
- They know modeling and innovation
  - They develop mental visualization capabilities
  - They can navigate through different design levels
  - They push innovation, while staying in logical and constrained frameworks
- They help organizations select balanced trade-offs
  - They tolerate ambiguity & combine their left and right-brain skills
  - They are communicators and facilitate decision-making
  - They manage system risks

CESAMES

Center of Excellence on Systems Architecture, Management, Economy and Strategy

C.E.S.A.M.E.S. INSTITUT
Limited Company registered in Paris under RCS 529 638 314
Legal Address: 15, rue La Fayette – 75009 PARIS – France
Contact: contact@cesames.net
Phone: +33 (0)1 40 15 54 20