CESAM:

CESAMES Systems Architecting Method

A Pocket Guide

July 2021

CESAM COMMUNITY

CESAM Community managed by CESAMES – 15, rue La Fayette – 75009 Paris – France

Publisher

C.E.S.A.M.E.S. – 15, rue La Fayette – 75009 Paris – France – SIRET: 518 815 741 00039

C.E.S.A.M.E.S. is a non-profit organization, created under the French law of July 1, 1901, that manages CESAM community, an international community of systems architects & engineers.

email: contact@cesam.community – Website: http://cesam.community/

Table of contents

Introduction

CESAMES Systems Architecting Method (CESAM) is a *systems architecting & modeling framework* which develops since 2003 in close interaction with many leading industrial companies in areas such as aeronautics, automotive, civil engineering, defense, energy, health, railway and space. It was initiated within the academic sphere in the context of the Dassault Aviation – DCNS – DGA – Thales – Ecole Polytechnique "Engineering of Complex Systems" industrial chair in France. Nowadays the members of CESAM Community act however as the *core developers & contributors* of the CESAM framework which is presented in a light version in this pocket guide.

CESAM is currently one of the *most used and disseminated* model-based systems architecting framework in the world, with more than 2,000 real system development projects on which it was operationally deployed in various industries and with around 10,000 trained engineers in more than 20 countries.

The success of CESAM is due to the fact that it is an *intuitive framework* which is very easy to use in practice. It is therefore especially dedicated to the working systems architects, engineers and modelers to help them better mastering the complex integrated systems they are dealing with.

We shall also point out that, thanks to its mathematical foundations, CESAM is, by design, *compatible with all the system modeling tools* of the market on which it can be easily implemented. It is thus possible to take benefit of the CESAM framework whatever your tooling choice and strategy is.

Let us now finish this short introduction by providing the structure of this pocket guide which is divided in two main sections. The first section intends to present the motivations of systems architecting, the discipline to which CESAM provides a rigorous language for expressing systems architectural ideas and writing model-based systems specifications. The second section discusses then the foundations of the CESAM framework with the presentation of the three core systems architectural visions – that is to say the operational, functional and constructional visions – which can be used to specify any system.

For further details, we shall finally refer to the extended and more detailed version of this guide, published by Springer Verlag.

1 Why Architecting Systems?

1.1 Product and Project Systems

Before starting, we need first to introduce a distinction that will be fundamental to understand better, both what is systems architecting and how to analyze many classical engineering issues. It indeed appears that engineered systems are always involving two kinds of systems:

- the first one is clearly the *product system*, i.e. the integrated hardware, software and "humanware" object which is under engineering in order to be finally constructed and put in service,
- the second one is the *project system*, i.e. the engineering organization – that is to say the engineers with their software & hardware resources – who is designing and developing the product system.

These two types of systems are of quite different nature: the product system is usually a technical-dominant system when the project system is clearly a human-dominant system. However, as shown in Figure 1, they are highly and permanently coupled during all the design & development phases of the product system: the project system typically monitors and transforms the implementation status of the product system through adapted implementation actions that are changing this implementation status.

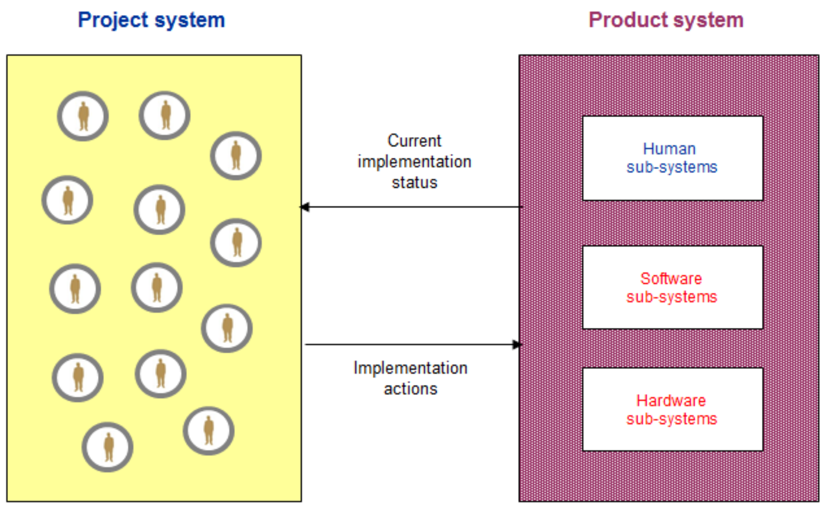

Figure 1 – Product versus project systems

The product / project distinction leads to two quite different ways of managing a system development:

- *Management mode 1 – project-oriented management:* this first management mode – clearly the most common – groups all classical project management activities where a system development project is followed by means of a project agenda and a task achievement monitoring.
- *Management mode 2 – product-oriented management:* this other management mode – probably also the less used – intends to monitor, supported by means of systems architectural views, the progression of the maturity-y and the progressive achievement of the desired product system.

These two management modes shall of course not be opposed since they are fully complimentary. A key good practice, on which systems architecting relies, claims that these two modes of management – respectively based on a project agenda and on systems architectural views – are both mandatory in the context of *complex systems* *development*. Note also that a product-oriented management is especially mandatory when developing a system with an important level of complexity (see next sub-section).

1.2 The Complexity Threshold

At this stage, it is now time to quickly explain what "system complexity" is and how it is related to systems architecting. In this matter, the key point to know is that the complexity of a system can be measured by the number of its external and internal interfaces. As a consequence of well-known complexity laws (see more details in the extended version of this guide), the engineering effort in a system development project will quickly be too important for being anymore handled by a single person, when the complexity of the system grows. In other terms, there will be always a *cognitive rupture moment* where the system complexity is too high to be any longer efficiently individually mastered by a systems architect or engineer. This rupture point is of course difficult to formally define, and it depends on the systems architecting & engineering maturity of a given industry, but as far as we could regularly see in practice, it can pragmatically be observed when complex systems designers began to express strong cognitive difficulties to master all dimensions of their system.

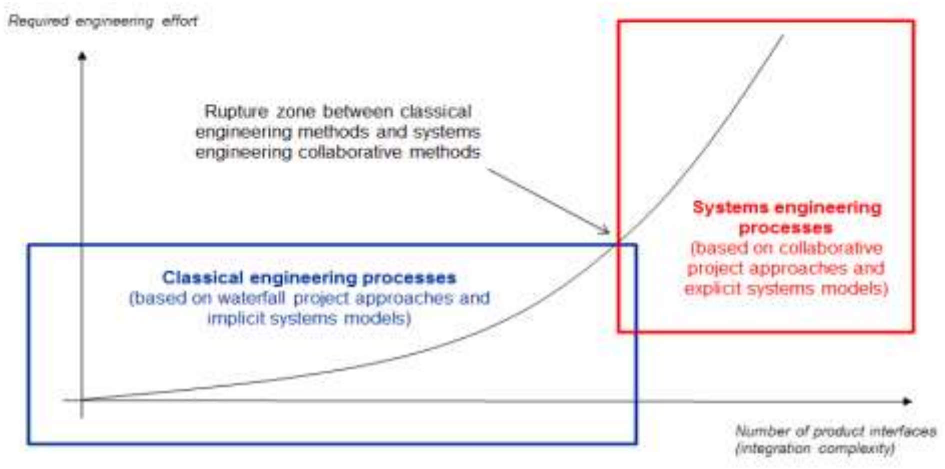
Required engineering effort
Rupture zone between classical engineering methods and systems engineering collaborative methods
Systems engineering processes (based on collaborative project approaches and explicit systems models)
Classical engineering processes (based on waterfall project approaches and implicit systems models)
Number of product interfaces (integration complexity)

Figure 2 – Project effort and integration complexity relationship

The consequence of this situation is that we can now distinguish two types of engineering, depending on whether the complexity of a given system lies before or after the complexity rupture zone that we just pointed out (see Figure 2). The first type of engineering, working perfectly well for low complexity systems, is what we will here call "*classical engineering*": it is usually based on waterfall development approaches, induced by a separation of engineering organizations by domain specialties, and on implicit systems models where key technical integration, knowledge only lies in the brains of a limited number of technical experts.

Such an engineering approach unfortunately does not work anymore when the system complexity threshold is crossed. One arrives then in a domain where transversal collaboration becomes crucial to complete individual expertise and where explicit systems models are now mandatory since it will be not anymore possible to handle implicitly the complexity that one is facing. In other words, one enters in "*systems engineering*" which is the engineering approach especially dedicated to integrative complexity mastering.

1.3 Addressing Systems Architecting becomes Key

Due to that complexity threshold which is – in our understanding – the root cause of most engineering issues observed in complex systems development, addressing systems architecting – the core of systems engineering – becomes therefore key for engineering organizations that are dealing with complex systems.

High complexity is indeed exactly the context where systems architecting will bring all its value. This discipline especially allows reasoning about hardware & software components (technical dimension) and human factors (human dimension) related to a given system, both in a unified framework and in subsidiary with all existing disciplines & engineering fields (see Figure 3). To achieve this goal, the fundamental principle on which systems architecting relies, is a *logical approach to engineering* brought by systemics. As one can easily understand, it is merely impossible to strongly consolidate all formalisms used to model and work with the various components of a heterogeneous system[1]. Systems architecting thus proposes a purely logical approach to federate all these "local" formalisms, rather than trying to consolidate them in a unique global formalism.

The main idea here is to simply use logic as a pivot language in order to be able to work in the same way with all hardware, software and "humanware"-oriented disciplines involved in a complex system design & development context. Each discipline can indeed easily formulate its requirements, constraints, findings or models using the universal language provided by *logical predicates,* which are formally Boolean functions telling us whether a given property is satisfied or not depending on the values of its entries (see [25]). In a system context, a logical predicate is thus nothing more than a statement that may true or false for a given system. A typical reasoning in systems architecting will thus be based on logical predicates associated, on one hand with the whole considered system and on the other hand with the different involved components and disciplines.

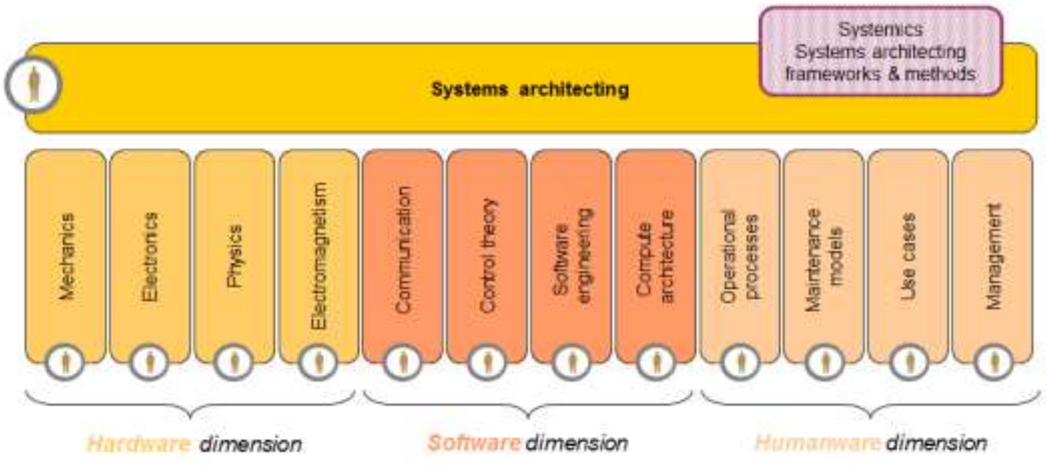

Figure 3 – The integrative & collaborative dimension of systems architecting

As we can see, there is therefore absolutely no magic in the promise of systems architecture to provide a unified framework for working with all dimensions of any system, as it is a simple consequence of the universality of mathematical logic! One may also notice that systems architecting can be seen as an observational modeling discipline: the construction of a system model – in the meaning of systems architecting – is indeed the result of the observation of all system components, guided by the engineering domains involved in the considered system, and of the federation of all these observations in a unique logical model of system level.

Last, but not least, it is important to also point out that systems architecting always integrates a strong collaborative dimension: it is indeed simply not possible to construct a system model without implying all people who are representing the different involved system components and engineering domains as soon as one wants

[1] Electromagnetism or fluid mechanics are for instance based on partial differential formalisms such as Maxwell or Navier-Stokes's equations, when signal processing relies on a distinction between time and frequency domains, leading to Fourier or Z transforms formalisms, which have all typically nothing to do with the logical and discrete formalisms used in information technology. Human factors are moreover of a totally different nature, without any strong mathematical background, but they shall also be included in the global picture. As a consequence, all these frameworks can just not be unified.

to get a realistic model. Moreover, sharing a system model is also a key good practice for ensuring its robustness. Collaboration is thus at the center of any systems architecting approach.

All these elements allow easily understanding that systems architecting is the discipline by excellence which is required to efficiently address systems integration issues. At this point, it may thus be useful to position precisely systems architecting within systems engineering (see Figure 4).

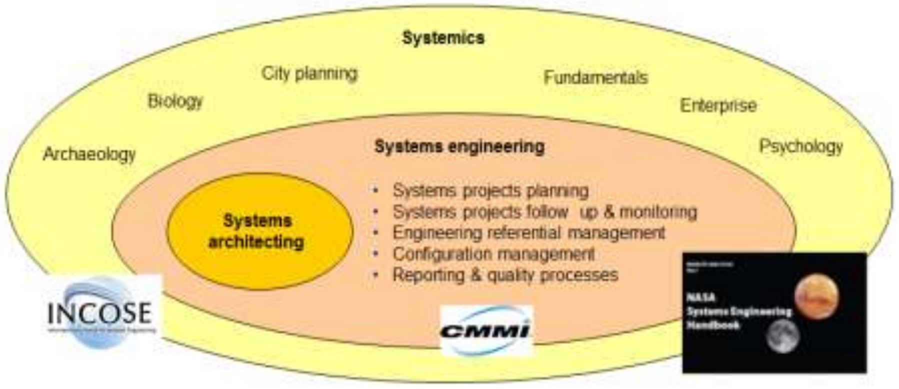
Systemics
City planning
Fundamentals
Biology
Enterprise
Archaeology
Systems engineering
Psychology
Systems architecting
Systems projects planning
Systems projects follow up & monitoring
Engineering referential management
Configuration management
Reporting & quality processes
INCOSE
CMMI
NASA Systems Engineering Handbook

Figure 4 – Relative position of systems engineering and systems architecture within systemics

To this purpose, let us first recall that systems engineering is nothing else than systemics applied to engineering. Systemics[2] here refers to the discipline that deals with systems. It provides holistic vision and analysis methods (see [27]), integrating crosswise all dimensions of a given topic. Systemics applies to many application domains such as archaeology, biology, city planning, enterprise[3] or psychology to provide few non-exhaustive examples. From a systemics perspective, systems engineering is thus just another application domain.

This fact must of course not make us forget that *systems engineering* (cf. [26] for more details) also has its own tradition that goes back to the 1950's, with first textbooks in the 1960's (see **Erreur ! Source du renvoi i ntrouvable.**) and first industrial processes formalization with the seminal NASA systems engineering handbook in the 1970's (see [16] for current edition). Many other textbooks (such as [2], [3], [4], [6], [12], [13], [14], [15], [17], [18], [21]) and industrial standards (such as [1], [9] or [11]) were then constructed in the line of that initial works. More recently, the International Council on Systems Engineering (INCOSE) emerged in the 1990's. It is currently federating and developing the domain at international level (see for instance [9] for the INCOSE systems engineering handbook).

Following INCOSE (see again [9]), systems engineering defines as "an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem (operations, performance, test, manufacturing, cost & schedule, training & support, disposal)". When one analyses more precisely the systems engineering processes, one can however split them naturally according to a product/project distinction, as discussed in section 1.1, in the following two types of activities of quite different nature:

[2] Systemics can be traced back to the 50's with the seminal work of H. Simon published in 1962 (cf. [19]). One usually also cite von Bertalanffy, who tried – but without being terribly convincing – to construct a "general systems theory" (cf. [22]).

[3] Systems approach applied to enterprises typically gives rise to enterprise architecture frameworks (see [23]) such as for instance TOGAF® (cf. [20]).

- Project-oriented activities, such as project planning, follow up & monitoring, engineering referential & configuration management, reporting & quality, which are all related to systems project management,
- Product-oriented activities, such as requirements engineering, operational, functional & constructional architecting, trade-off & safety analyses, verification & validation, which do cover exactly the scope of systems architecting.

In that perspective, systems engineering can be seen as the union of systems project management and systems architecting, which can thus also be analyzed as the core part of systems engineering, dedicated to the design & construction of robust systems models. Systems architecting shall especially be seen as the discipline synthesizing the methods and the tools which allow an exhaustive & coherent modeling of a system (in its triple operational, functional & constructional dimensions) in order to manage it efficiently all along its lifecycle (design, test, deployment, maintenance, …).

1.4 The Value of Systems Architecting

To complete our discussion on systems architecting, let us now focus on the value brought by this discipline, which was quite well analyzed by E. Honour in [8]. The key point to understand is that the systems engineering approach in which systems architecting takes place, mainly consists in redistributing the engineering effort towards upstream phases of the system development project, in a "definition" phase, in order to anticipate design risks as early as possible within such a project.

Figure 5 below, extracted from [8], perfectly illustrates this paradigm, where more time is initially spent better understanding the system to develop and thus reducing the project risks in the future, compared to a traditional design. Such an approach strongly relies on systems architecting since the "definition" phase will typically contain a strong part dedicated to the construction of a systems architecture file that will allow to analyze the system under design in all its dimensions[4] , the other part being devoted to the project planning. In that perspective, systems architecting can be seen as a risk management good practice in complex systems development contexts.

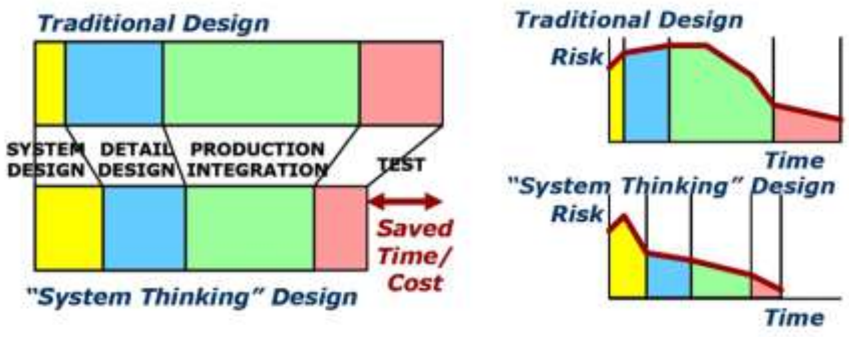

Figure 5 – Systems Architecting as a risk management practice[5]

One can thus understand that systems architecting is a key tool for mastering systems design and development projects in the respect of their quality, cost, time and performance constraints (QCDP), as soon as one deals with complex systems produced by the integration of many technical systems (hardware & software) and human systems (people & organizations). Many evidences support this point of view (see again [8] for examples).

[4] Typical dimensions to consider in a system analysis are stakeholders, lifecycle, use cases, operational scenarios needs, functional modes, functions, functional dynamics, functional requirements, configurations, components, constructional requirements, constructional dynamics, critical events, dysfonctional modes & behaviors and verification & validation.

[5] This figure was reproduced from [8].

We shall finally recall some key principles that are at the heart of the value provided by systems architecting and that shall be followed as soon as one wants to master a complex system design & development:

- **Provide simple, global, integrated and shared visions of a system:** "to be simple" and to capture completely all dimensions of a given problem in complex environments, which does not mean "to be simplistic", is always very difficult …
- **Think first about needs and not about solutions:** thinking first about technical solutions is unfortunately the common rule in most of systems development projects. One shall thus always remember that the "customers" of a system project shall fundamentally feed the systems architecting process.
- **Sort out all "spaghettis" in a system design:** this is indeed key if one wants to avoid mixing everything (objectives, functions, technical constraints, etc.) and confusing ourselves while confusing others, which are also again very common bad practices in too many complex systems contexts.

1.5 The key Role of Systems Architects

Systems architecting would of course not exist without the right key people to support the systems architecting process, that is to say *systems architects*! A systems architect shall indeed fundamentally be the main responsible of the systems integration issues. He/she shall thus ensure that the interfaces of the subsystems of the target system under design are reasonably robust for the purpose of the system development project, or in other terms that they will not be questioned (or as little as possible) by the technical managers in charge of the design of the different subsystems that form altogether a given system.

To achieve such objectives, it is necessary to play on a different dimension than the purely technical one. A key problem within system design is indeed that it is usually not enough to construct the "best" possible system architecture for it to be automatically retrieved and used by everyone. The global optimum, typically with respect to quality, cost, time and performance criteria, for an integrated system is indeed never[6] the union of the local optima of each of the subsystems that compose it. The consequence is that each subsystem shall necessary individually be sub-optimal with respect to these criteria if one wants to reach global optimality at system level, which is a rather counter-intuitive result!

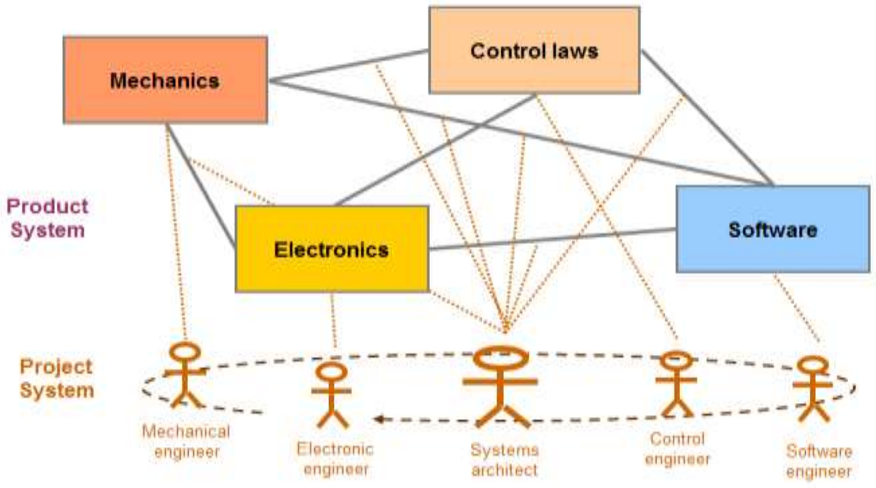

Figure 6 – The key role of the systems architect

[6] A classical result of control theory establishes that this assumption is true as soon as the system has not a linear behavior with respect to its entries, which is never the case for complex systems.

This fact is unfortunately not easy to accept for those responsible for the subsystems who will not design their subsystems with a global vision, as the architect of the whole system may have[7]. To solve this "human" problem, which is intrinsic to the design of integrated systems, one must put stakeholder alignment mechanisms at the heart of any system architecting process. These alignment activities shall ensure that the structuring systems architecting choices will always be shared by their stakeholders. Systemically speaking, such mechanisms will indeed guarantee that the technical interfaces of the product system will always be discussed and accepted at the "human" interfaces to which they are allocated within the project system (see Figure 6).

This rapid analysis therefore shows that a system architect must always be fundamentally capable of bringing the players together on the architectural choices for which he is responsible. This convergence work – which is a substantial and essential part of the systems architect job – is however not simple at all. It indeed requires mastering, independently of purely technical competency, a set of completely different soft skills that could be qualified of "human" architecting and engineering since they are highly involving the "human" dimension of the project system. At that level, the systems architect shall typically have the two following key competencies:

- *Competency 1 – Identifying all stakeholders of a given system architecture*: this first competency looks as an apparently simple activity, but it is often terribly difficult it to achieve in practice since it requires confronting the complex and changing reality of systems environments & engineering organizations. Ensuring the completeness and validity of a particular organizational analysis is indeed never easy. It is notably classical to forget key players within a given system scope and to identify erroneously others. Moreover, environments & organizations are often changing rapidly. Making a stakeholders mapping should thus always be constantly updated if one wants to keep pace with reality.
- *Competency 2 – Aligning the stakeholders on a same architectural solution*: this second competency is a real "facilitation" skill in the best sense of the term since such a work requires a real difficult technical and human know-how. A systems architect will only succeed in bringing the stakeholders together on given architectural choices if he/she plays in a coherent manner both on the technical level, which he/she must of course fully master to be credible, and on the human level, in order to obtain a strong consensus that will stand the test of time. This facilitation skill is therefore probably the most critical and important skill that any systems architect must have. It clearly differentiates systems architects from the other engineers.

[7] The difficulty of understanding the importance of system architecting comes from this situation. Engineers usually deal with technically homogeneous subsystems of a given system (i.e. the "boxes" of Figure 6) which are clearly visible to all. On the other side, the systems architect takes care of the system interfaces (that is to say the "arrows" in Figure 6) that nobody sees because they are, strictly speaking, not material. The architecture work is therefore done somewhere in the invisible! It is thus difficult to understand for the uninitiated, while fundamental since it fixes the framework in which to do engineering (a bad architectural framework can typically only lead to a "bad" system).

2 CESAM Framework 101

2.1 Introduction to CESAMES Systems Architecting Method (CESAM)

CESAM is a systems architecting & modeling framework dedicated to the working systems architects, engineers or modelers to help them better master the *complex integrated systems* they are dealing with.

The CESAM framework has several unique features:

1. First, CESAM has sound mathematical fundamentals which are providing a *rigorous and unambiguous semantics to all introduced architectural concepts*. This first property is key for ensuring an efficient and real understanding between the stakeholders of a system design & development project.
2. These bases do ensure that CESAM is a *logically complete & lean systems modeling framework*: in other terms, the architectural views proposed by CESAM are necessary and sufficient to model any integrated system. This second key property guarantees both the completeness of a system model, when based on CESAM framework, and that no useless modeling work will be done when using CESAM.
3. Finally, CESAM is *practically robust and easy-to-use* both by systems architects and systems modelers. This was indeed pragmatically observed among the large amount – more than 2,000 in 2021 – of various concrete systems within many different industries (aerospace, automotive, civil engineering, defense, energy, health, railway, etc.) that were modeled and architected using CESAM.

As a matter of fact, CESAM is currently probably one of the *most used and disseminated* model-based systems architecting framework in the world, with more than 2,000 real system development projects[8] on which it was operationally deployed in various industries and with around 10,000 trained engineers in more than 20 countries.

Note also that CESAM framework – due to the right level of abstraction that it promotes – can be implemented and used with both quite all existing system modeling frameworks and software tools of the market.

2.1.1 A Mathematically Sound System Modeling Framework

CESAMES Systems Architecting Method (CESAM) is the result of 12 years of research & development, that were initiated within the Dassault Aviation – DCNS – DGA – Thales – Ecole Polytechnique "Engineering of Complex Systems" industrial chair in France and continued by CESAM Community, including permanent interactions and operational experimentations with industry. The CESAM framework was indeed used in practice within several leading international industries in many independent areas with a success that has never wavered. This huge balanced theoretical and experimental effort resulted in *a both mathematically sound and practical systems modeling framework*, easy to use by working systems engineers and architects.

We will however not develop here the mathematical foundations – based on mathematical logics & semantics – of the CESAM framework. To learn more about them, we shall refer to the extended and more detailed version of this guide, published by Springer Verlag.

2.1.2 A Framework Focused on Complex Integrated Systems

Another key point to stress is that CESAMES Systems Architecting Method (CESAM) provides a model-based systems architecting & engineering framework which is fundamentally oriented to support the development of *complex integrated systems*. In other words, CESAM is especially dedicated and adapted to the architecting and

[8] By "real" system development projects, we mean operational / industrial system development projects with real business objectives & challenges, not just proofs of concepts.

modeling of complex systems, that is to say non-homogeneous systems, mixing typically hardware, software and maybe “humanware” components, which result from an integration process.

In this matter, let us recall to be more specific that *system integration* is the fundamental mechanism[9] that allows in practice to build a new system from other smaller systems, by organizing them so that the resulting integrated system can accomplish – within a given environment – its missions. Any system is therefore the result of multiple integration mechanisms, starting from its elementary parts up to arriving to the global system.

2.1.3 A Collaboration-Oriented Architecting Framework

CESAM is not only just a mathematically sound system modeling framework, specifically focused on complex integrated systems. It is also an *architecting framework* which means that it is intended to efficiently support all design decisions that any systems architect needs to regularly take during a complex system prototyping or development project. It should indeed be remembered that systems modeling is not an end in itself, but just a tool for systems architecting which is the key design process addressed by the CESAM framework. Architecting here means finding an optimal solution that fulfills a series of external needs and constraints. It can thus be seen as an optimization process which has to construct and select the “best” system among a series of possibilities. Choice is thus intrinsic to that activity. Being able to make the “good” choices in a rational way is thus always key in any system design project. This is exactly the purpose of the CESAM framework which provides to the working systems architect several systemic views as a support to *collaborative architectural decisions*.

It is indeed important to remember that a technical system does never exist alone and that it cannot be designed independently of the people who are engineering it. A “good“ systems architecture is in particular always an architecture that all stakeholders do share, thus a balance between technical and socio-technical considerations. The first job of a systems architect shall thus always be to understand and identify the organizational architecture in which a given system development takes place: the technical architecture of the system under design is indeed usually highly correlated to that organizational architecture. This explains why a systems architect must always manage the two following types of activities which are of very different nature:

- On one hand, *technical activities,* fundamentally centered on the definition of integrated global system models, making explicit interfaces between all components of a system,
- On the second hand, *facilitation activities* centered on the construction of convergence on these system models, creating a common vision in this matter between all stakeholders of the system.

System models are thus key for ensuring a collaboration of quality which is mandatory in the context of complex integrated systems development. In these matters, the basic tool for managing collaboration and creating architectural convergence is *the collaborative systems architecture workshop*.

The principle of such a workshop is quite simple since it consists in putting in the same place at the same time[10] all stakeholders that must converge on a given architectural solution and submitting then them a first version of the intended system architecture. Note that the effective implementation of a collaborative systems architecture workshop presupposes that one previously achieved an organizational architecture analysis, which abstracted efficiently and sufficiently the "field" of a given system architecture by only identifying a limited number (typically

[9] Integration is thus an operator between systems that maps a series of systems $S_1, \ldots, S_n$ into another new system S that results from the integration of $S_1, \ldots, S_n$. It is interesting to point out that most of the existing system “definitions” which can be found in the systems engineering literature (see for instance [6], [9], [11], [13], [14], [15], [17] or [18]) are defining systems as the result of an application of the integration operator. From a mathematical perspective, this gives unfortunately rise to logically inconsistent definitions of systems since the set on which acts this operator is never rigorously defined.

[10] A collaborative systems architecture workshop shall respect the core rules of classical theatre: unity of time, unity of location, unity of action. In other words, its participants shall work at the same moment in the same location towards a common and share objective.

no more than 15) of key players, who are truly representative of the entire organizational scope of the target system, with whom the required convergence work shall be managed.

2.1.4 A Business-Oriented Framework

Let us finally also stress that CESAM is a *business-oriented framework*. As already pointed out, the CESAM framework was indeed used in many operational industrial applications. In order to contribute to the increasing quality of systems development projects, we want therefore to share with the system community a part of that important practical experience through dedicated architecture frameworks.

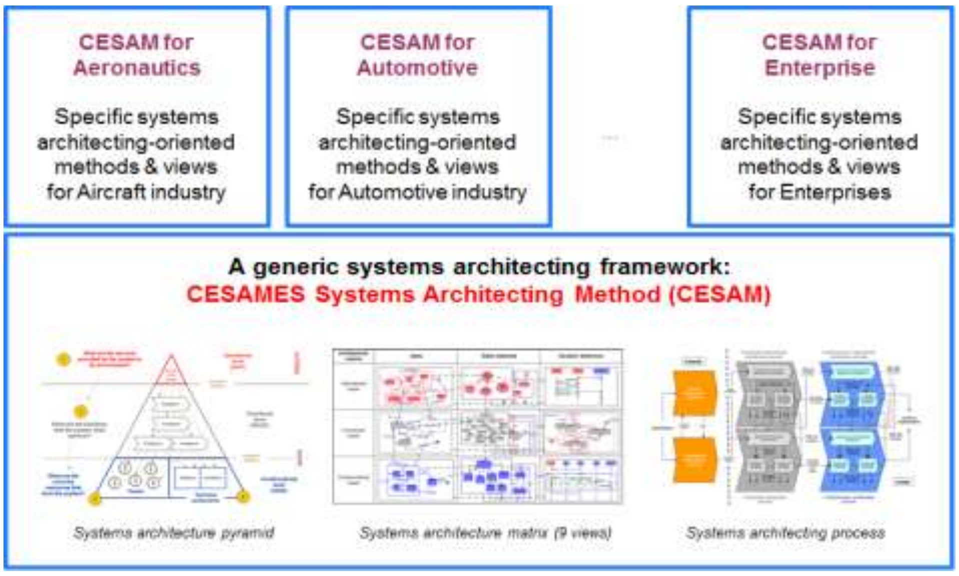
CESAM for Aeronautics
Specific systems architecting-oriented methods & views for Aircraft industry
CESAM for Automotive
Specific systems architecting-oriented methods & views for Automotive industry
CESAM for Enterprise
Specific systems architecting-oriented methods & views for Enterprises
A generic systems architecting framework:
CESAMES Systems Architecting Method (CESAM)
Systems architecture pyramid
Systems architecture matrix (9 views)
Systems architecting process

Figure 7 – Tentative structure of the CESAM framework

The CESAM systems architecting framework is therefore intended to be organized in the two following layers, as illustrated and described in the above Figure 7:

- **a generic layer**, consisting of a *generic system architecting framework* as introduced in this guide, which can be applied to any system in full generality,
- **a specific layer**, consisting of several *specific systems architecting methods & systems architecture views* per main application domain (aeronautics, automotive, enterprise, etc.).

The generic layer of the CESAM framework is fully presented in the extended and more detailed version of this guide, published by Springer Verlag. One can also find it on our community website https://cesam.community. In the same way, our community website shall regularly publish the specific domain-oriented instantiations of the CESAM framework.

2.2 Elements of Systemics

Before going further, we now need to introduce the notions of interface and environment. These first elements of systemics will indeed be mandatory for presenting further the CESAM framework.

2.2.1 Interface

The concept of interface is the first key systemic concept that we need to present. An *interface* models an interaction, an exchange, an influence or a mutual dependence between at least two systems. An interface may not necessarily have a concrete implementation: it is just a way of abstractly expressing the relative impacts that different systems have one on the others[11].

With respect to a given system, interfaces may then be either *external* (when they involve the considered system and some other external systems) or *internal* (when they are only relative to components of the system of interest). Figure 8 illustrates this notion on the electronic toothbrush example by showing two external interfaces between the toothbrush and the end-user and some internal toothbrush mechanical & electrical interfaces.

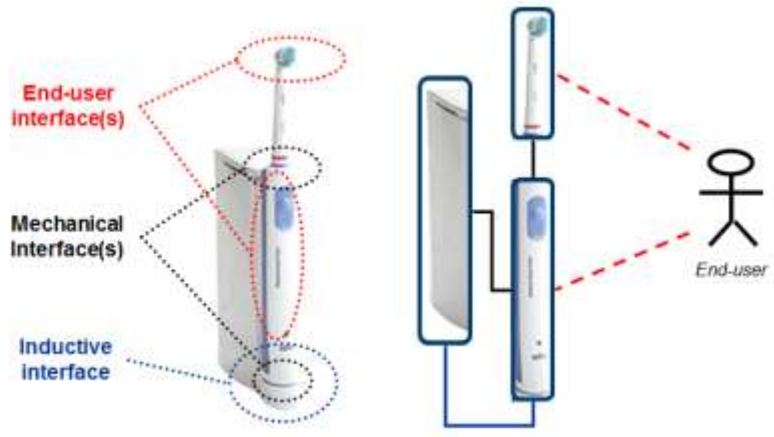

Figure 8 – Examples of interfaces for an electronic toothbrush

Note that the technical interfaces, corresponding to a concrete interaction or exchange between several systems, are usually the easiest to identify since they refer to a visible relationship between the involved systems. However, the invisible interfaces, relative to an influence or interdependence between different systems that may typically be of strategic, political, societal or regulatory nature[12], are also crucial.

2.2.2 Environment of a System

The recursive nature of systemic analysis naturally leads us to introduce the notion of *systemic environment* of a system, defined as any closed[13] super-system of a given system, which shall provide a natural basis to begin a recursive analysis of a given system.

A real system has of course many real systemic “environments”. However, the pragmatic constraints of any system design & development project led us to define *a reference environment* for any concrete system S, which will be called the environment of S by a slight abuse of language. This reference environment is the smallest “useful” systemic environment of the system of interest S. It is just the system that results from integration of S and all other real systems that have an influence on its design & development. We will thus neglect all other real external systems when one considers that they have no strong interdependence or no strong interaction with S.

Defining the environment of a system means defining its *border*, that is to say eliciting precisely what is inside and what is outside the system. Figure 9 typically illustrates this key concept on the electronic toothbrush

[11] Most people are making for instance the confusion between networks and interfaces. A network is indeed strictly speaking not an interface, but another system with which the system of interest has also a specific interface. In the first stages of a design, one may of course abstract it and only consider the logical interface between the systems it connects, but the abstract interface involved in such a mechanism shall not be mixed with the network system itself.

[12] Think also on the possible impacts of competitors, new technology, industrialization or maintenance on your system.

[13] A closed system is a system which is considered to have no external interfaces. No real system – apart of the physical universe in its whole – is of course a closed system. When we are considering the environment of a system as a closed system, we just mean that such an approximation is reasonably correct and useful in practice.

example. Note also that defining this border is typically the first key systems architecting decision that one must take in practice, since it allows specifying precisely the system of interest which is under design.

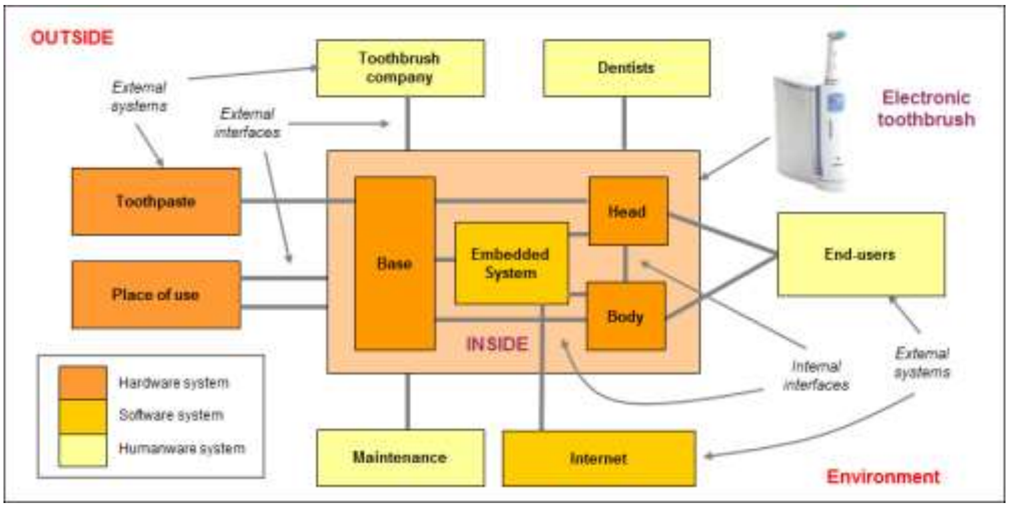

Figure 9 – Environment of an electronic toothbrush

2.3 The Three Architectural Visions

The three systems architectural visions will be our first key systems architecting tool for analyzing any system.

2.3.1 Architectural Visions Definition

The heterogeneity of the environment of a system requires to address it by different axes of architectural analysis in order to integrate the whole set of various perceptions provided by the different system stakeholders. Such a consideration naturally leads to organize these points of views according to different architectural visions that are both necessary due to the variety of any systemic environment, but also useful since they allow decoupling the representations of a given system in different "properly" interrelated separated views[14] which always leads to better clearness and flexibility in terms of system design & development management.

As a matter of fact, each integrated system S can always be completely analyzed from three different and complementary perspectives that generate three generic *architectural visions* (operational, functional and constructional), each of them grouping different types of systemic models, as defined below:

- **Architectural vision 1 – Operational vision:** strictly speaking, the operational vision groups only the models of the environment of S – and not of S itself – which are involving S. Such operational models are thus describing the interactions of S with its environment.
- **Architectural vision 2 – Functional vision:** the functional vision groups all system level models that are describing the input/output dynamics of S, without referring to its concrete implementation through components. Such functional models are thus abstractly modeling the behaviors of S.
- **Architectural vision 3 – Constructional vision:** the constructional vision groups the system level models that are explaining how S is obtained from its lower-level components. Such constructional models are thus describing the structure of S.

[14] This way of managing different views on the same system is in fact quite common in usual life. Think for instance of a tourist visiting a city. He/she will probably use many different views, typically provided by a touristic guide, a metro map and a city map. To find his/her way, he/she may for instance first chose the monument to visit in the touristic guide, then move there using the metro map and finally manage the local approach using a city map. In architectural terms, the tourist is thus taking information in different coupled views and integrating them in order to take the "good" decision!

Note that other names do classically exist for addressing the last constructional vision. One may for instance speak of *structural vision*. Many frameworks are also speaking of *logical vision* to denote the constructional vision in the CESAM meaning. The physical vision which is introduced by some other architectural frameworks is then typically a refinement of that constructional vision.

An illustration of these three different architectural visions is provided by Figure 10 on the electronic toothbrush example. We sketched there three different types of models, with their connections, each of them illustrating a different systems architectural vision. One sees that the operational vision is not interested by the toothbrush behavior or structure, but just by describing its interactions with some external systems, which are here power supply, end-users and internet. On the other hand, the functional vision gives the main toothbrush behaviors – i.e. provide electrical power, generate brushing power and provide brushing capability – that allow producing the external interactions which were captured by the operational vision. At last, the constructional vision shows how to implement concretely these internal behaviors through suitable components, here a base, a body, an head and an embedded software.

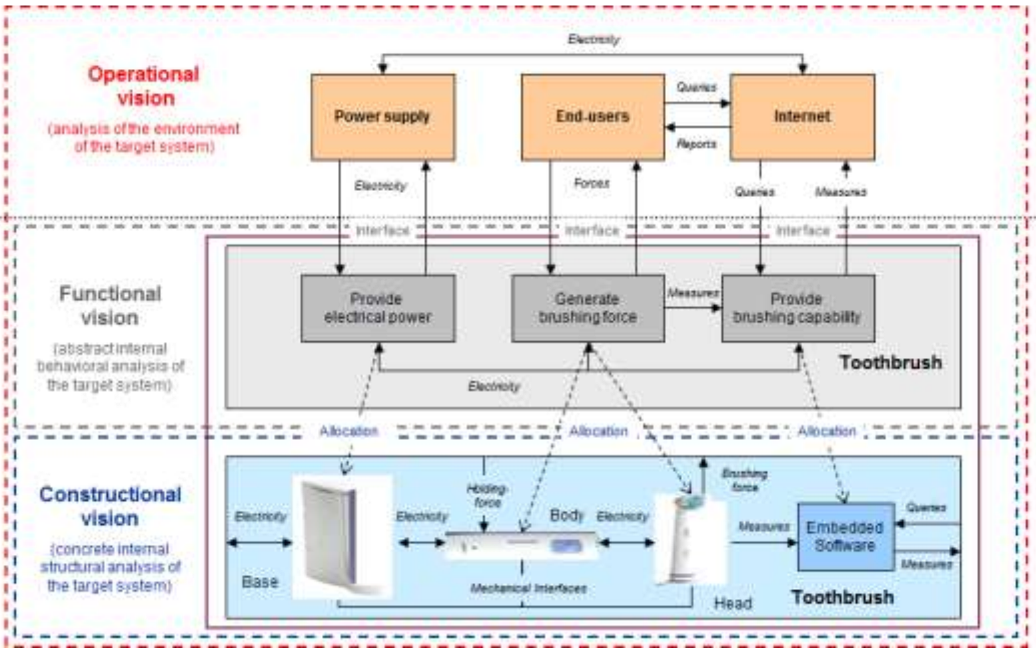

Figure 10 – Illustration of the three architectural visions on an electronic toothbrush

It is also important to point out that the previous architectural vision definitions are consistent. The key point is here only to be sure of the existence of functional models as defined above. This is however directly connected to the emergence postulate that claims that the mere knowledge of the models of the components of a system and of their interaction laws is never sufficient to model the system that results from their integration. This explains why any system always has functional models, whose core fundamental role is to express the emerging behaviors[15] that one will never be able to capture and read within constructional models[16].

[15] Unfortunately, this is not a common understanding of the functional vision. When doing "functional analysis", most people are indeed just modeling the functions of the components of a given system, which is not at all the core of functional analysis in our meaning since this activity shall focus on describing functions at system level, and not at component level.

[16] To understand this phenomenon, consider the example of a car whose constituent high-level systemic components are the car body, powertrain, cockpit, chassis and embedded electronics. The interaction of these components typically allows for features like "obstacle detection" which requires the cooperation of a radar (placed in the car body), an embedded software (within embedded electronics), a LED (positioned in the passenger cockpit), and possibly chassis or powertrain if one wants to act on the brakes and or to reduce engine torque when a given obstacle is too close to a car. Such a "transverse" feature

We can thus now understand why it is necessary to have three different types of models to model in practice a real system: the operational vision captures the external viewpoint while functional and constructional views do capture the internal perspective, by modeling respectively, on one hand, the emergent behaviors and, on the other hand, the concrete constitution of the considered system.

2.3.2 Architectural Visions Overview

Let us thus now present more in details the three different – operational, functional and constructional – systems architectural visions that we introduced in the last section.

2.3.2.1 Operational Vision

The *operational vision* provides "black box" models of a given system where one does not describe the system of interest, but rather its interactions and its interfaces with its environment. Its core motivation is to understand in that way the motivation – that is to say the "Why" – of the system.

In this matter, the key point to understand is that an operational analysis manipulates concepts at environment level, which are mixing both the system of interest and its external systems. The operational concept of "mission" of a system is a typical example of such a situation. Formally speaking, a *mission* for a given system S can indeed be defined as a function of the environment of reference Env(S) of that system[17]. When one analyzes for instance the "guarantee dental hygiene" function[18], whose functional behavior consists in transforming dirty teeth into healthy teeth and/or maintaining teeth in a healthy state, one can see that a toothbrush can clearly not achieve alone this feature, which also requires at least an end-user, toothpaste and water, plus may be a dentist. Such function can thus only be allocated to the environment of the toothbrush and not to the toothbrush alone. In other words, "guarantee dental hygiene" is hence not a function of the electronic toothbrush, but a function of its environment, which means that it shall be interpreted as a mission – and again not an (internal) function[19] – of the toothbrush, according to our above definition.

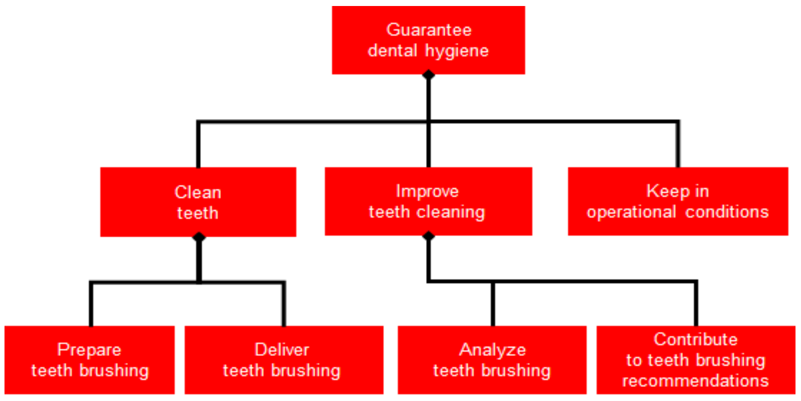

Figure 11 – Operational vision – Mission Breakdown Structure (MBS) of an electronic toothbrush

The operational vision relies on other operational concepts such as life cycle, operational contexts, operational scenarios or operational objects. All these concepts may of course be managed at different levels of abstraction and grain. Figure 11 shows for instance the *Mission Breakdown Structure* (MBS) of an electronic toothbrush

---

is clearly difficult to catch in a purely constructional car model, when one will see the flows exchanged between the various involved components of the vehicle without being able to understand their overall logic. Only a functional model at car level will allow capturing the semantics of such a transverse function.

[17] In other terms, Mission(S) ≡ Function(Env(S)) for every system S.

[18] Due to the functional nature of a mission, we do recommend naming a mission as a verb in infinitive form.

[19] The role of functional analysis is to extract, strictly speaking, the functions of a system of interest which are hidden within its missions, that is to say the internal behaviors of the considered system that are only involving the system.

where its core missions are put in a hierarchy, according to the fact that a high-level mission needs the lower levels missions to be achieved.

The operational vision can also be seen as a natural interface between engineering actors and their stakeholders. Typical examples of operational models are indeed for instance marketing or usage models (that describe how a product system will be seen by the market or used the end-users), business models (which explain how the industrial manufacturer will earn money with a product system) and development, construction or maintenance models (that specify how a product system will be managed by the associated design, manufacturing or support systems). In this matter, the role of the operational vision is to capture and express rigorously the information contained in these different operational models within a language that can be easily understood by the system designers and used in the development process.

2.3.2.2 Functional Vision

The *functional vision* provides "grey box" models of a given system of interest where one begins to apprehend the inside of the considered system, but only in terms of input/output abstract behaviors and not of concrete implementation choices, in order to begin understanding more deeply what the system does, without however knowing at this point how it is concretely structured. Its core motivation is to elicit in that way the behavior – that is to say the "What" – of the system.

The core notion of the functional vision is of course the notion of "function" of a system, which refers to an input/output behavior of the considered system. In other terms, a *function* associated with a given system models a transformation process – which can be achieved by physical, software or even "humanware" resources – that transforms a given series of inputs into a given series of outputs. This explains why a common pattern to name a function is a verb followed by a complement, the generic patterns being typically "Do something" or "Transform inputs into outputs". In any case, one shall always check when defining a function of a given system whether it expresses such a transformational behavior.

Contrarily to the operational vision, all functional concepts – such as functional modes, functional scenarios or functional objects – are now uniquely referring to the system of interest, without involving any external system. All these concepts can again be managed at different levels of abstraction / grain. Figure 12 shows for instance the *Functional Breakdown Structure* (FBS) of an electronic toothbrush, where its main functions are put in a hierarchy according to the fact that a given high-level function needs the lower levels functions to be achieved, meaning that the "algorithm" of the high-level function involves the lower functions as sub-routines.

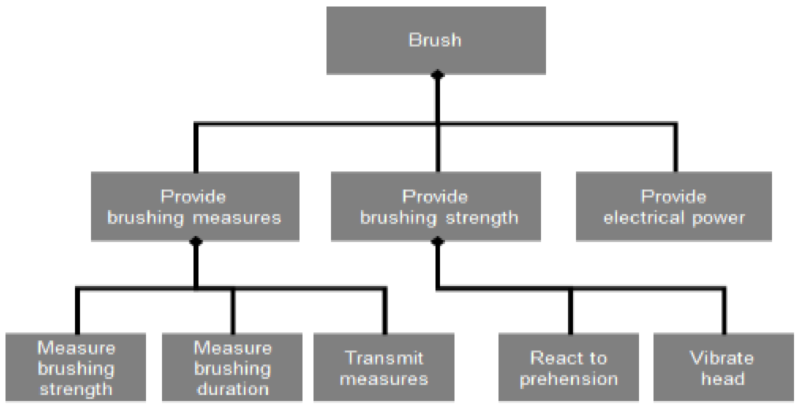
Brush
Provide brushing measures
Provide brushing strength
Provide electrical power
Measure brushing strength
Measure brushing duration
Transmit measures
React to prehension
Vibrate head

Figure 12 – Functional vision – Functional Breakdown Structure (FBS) of an electronic toothbrush

We may now point out that a key difficulty of functional analysis is the identification of *transverse functions* that is to say of functions that cannot be directly allocated to a single component of a given system. Such functions are indeed capturing the emergent behaviors resulting from the cooperation between the different components of a system, which by definition cannot be easily observed at constructional level. It is therefore always important to identify these functions in order to master the integration process since these functions are also telling us where different teams in charge of different components shall work collaboratively[20]. Within a functional breakdown structure, one may thus normally always find both component functions and transverse functions. Unfortunately, most of engineers are often forgetting transverse functions in their analyses, which leads them to lose the most important value of functional analysis from a systems architecting perspective.

Another key point is that the functional vision is fundamental in systems architecting since it provides the deep invariants of any system. Any communication network will achieve for instance always the same basic functions such as “receive messages”, “route messages” or “deliver messages”, independently of its implementation technology that may be, either purely manual (think to your snail mail operator) or based on many different technical implementations (Hertzian waves, twisted cables, copper wires, optical fiber, etc.). In other words, *technology changes, but functional architecture remains*. As a natural consequence, functional architecture always provides a robust basis for architecting a system. It indeed allows the systems architect to reason on a system independently of technology and thus to define, analyze and evaluate different implementation options for a given functional architecture. Such an approach is key to choose the best solution.

Good systems architectures are also based on *functional segregation principles*. This simply means that some key functional interfaces must be strictly respected at constructional level[21]. This gives rise to layered architectures where components are clustered in different independent functional layers connected by functional interfaces. Typical classical examples of such layered architectures are computer, mobile phone or communication network architectures that are organized in different independent layers, starting from the physical layer up to arriving to service layers (see for instance [24] for more details). One must thus for instance be able, on one hand, to change a signal processing protocol within the physical layer without any impact on the service layer and, on the other hand, to implement a new service or a service evolution in the service layer without any impact on the physical layer. Such achievements are typically obtained by means of robust functional interfaces, stable among time, in order to absorb the technological evolutions that will naturally arrive in the life of any system[22].

It is finally interesting to observe that the standard vocabulary used to discuss about the functional vision is traditionally different, depending on whether the considered system is a technical or an organizational system. The term “function” is for instance usually reserved for technical systems, when one rather uses terms such as “process”, “activity” or “task” to express the same behavioral concept when dealing with organizational systems. It shall however be clearly understood that processes, activities or tasks are in fact nothing else than functions of a given organizational system, considered at different levels of abstraction.

2.3.2.3 Constructional Vision

The *constructional vision* provides white box models of a system where one describes all concrete hardware, software and “humanware” components of a given system with their interactions. Its core motivation is to elicit

[20] An example of transverse function is the thrust reversing function on an aircraft: this function, which reverts the air flow passing in an aircraft engine to decrease the speed of the aircraft when on ground, is provided by the cooperation of a cylinder that pushes a trap both located in the nacelle, the engine itself and a critical embedded software that coordinates the involved nacelle and engine components when thrust reversing operates. Such a function is typically transverse since distributed on several hardware & software components which are located moreover provided by different suppliers (typically one for the nacelle, one for the engine, one for the embedded system): identifying the transverse nature of such a function and putting it under control in the aircraft development project is thus totally key to ensure the success of its integration.

[21] In this matter, the role of the systems architect is to guarantee that such interfaces will never be violated in the design.

[22] Another motivation for such functional segregation is abstraction. It would indeed be practically impossible to develop a software service by working directly at the level of the physical layer of a computer system since the physical world is here usually highly non deterministic with many probabilistic phenomena that must be hidden to a service developer.

in that way the concrete structure – that is to say the "How" – of the considered system. It is thus probably the most intuitive part of any system architecture.

The core concept of the constructional vision is of course the notion of "component" of a system, that refers to a concrete part of the considered system. In other terms, a *component* associated with a given system models a physical, software or even "humanware" [23] resource that belongs to the system. A common pattern to denote a component is thus just to use its usual technical or business name.

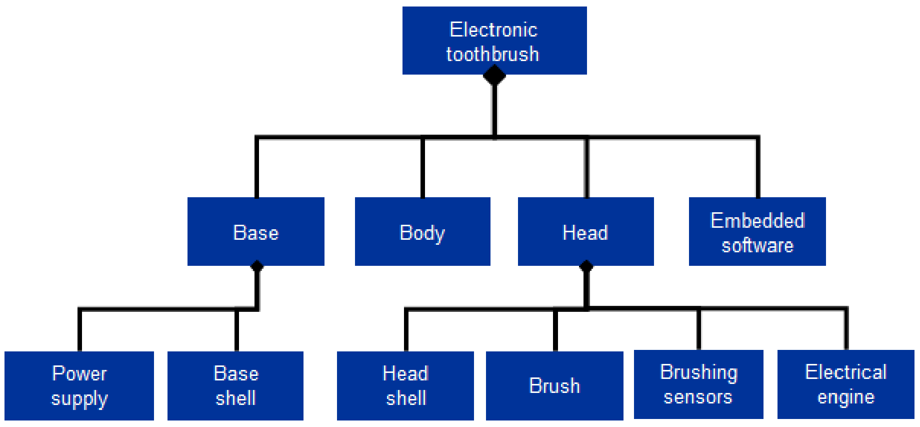

Figure 13 – Constructional vision – Product Breakdown Structure (PBS) of an electronic toothbrush

Exactly as in the functional vision, all constructional concepts – such as configurations, constructional scenarios or constructional objects – are again uniquely referring to the system of interest, without involving any external system. All these concepts can be managed at different levels of abstraction / grain. Figure 13 shows for instance the *Product Breakdown Structure* (PBS) of an electronic toothbrush, where its components are put in a hierarchy according to the fact that a high-level component results from the integration of the lower-level components.

Note that the term "architecture" usually only refers to the constructional architecture of a system. One shall be aware we will of course use this term in a much broader acceptation through our entire pocket guide, especially when speaking of systems architecture.

2.3.3 Relationships Between the Three Architectural Visions

Let us now point out the network of relationships existing between the three architectural visions, since they are at the heart of systems architecting. It is especially important to maintain these relationships during the different design phases, which is difficult due to the "highly iterative & recursive nature" of systems architecting [16].

Figure 14 shows the generic relationships existing between the architectural visions as explained below.

- The operational vision connects first with the two other visions since the missions of a system are naturally implemented by functions, but also components of the considered system. Another way to make this connection is to observe that all external flows, existing between the different external systems and the system of interest, must be internally captured or produced (depending whether the external flows are input or output flows) by functions of the system of interest[24].

[23] Men are part of systems when they have strong organizational dimensions (e.g. enterprise information systems) or when a human stakeholder (e.g. pilot, driver, operator, etc.) plays such a key role with respect to the system that it may be crucial to include him/her in the design, considering then an operated system rather that the underlying technical system alone.

[24] In other terms, it is sufficient to prolong each external flow, as identified in the operational vision, within the system to get the first functions of the system of interest. Functional analysis will then continue internally the same kind of analysis up to

- The functional vision connects back in the same way with the operational vision and forth with the constructional vision since abstract functions result from the implementation of a mission and must be concretely allocated to some set of constructional components.
- The constructional vision connects then back to the two other visions according to the implementation and allocation relationships that we highlighted.

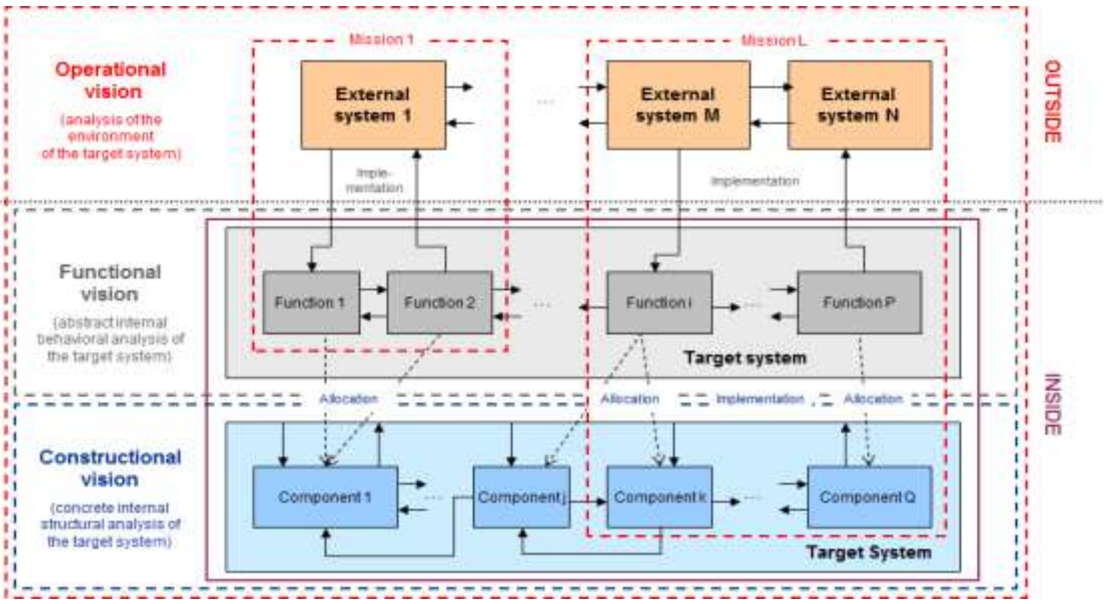

Figure 14 – Relationships between the three architectural visions

Note that **one should not think** that operational artefacts do only allocate to functional artefacts, which on their side do only allocate to constructional artefacts. Such a vision would indeed be dramatically false. Operational, functional and constructional visions shall indeed be analyzed as a circle of three interdependent visions. Figure 15 illustrates this non necessarily intuitive situation.

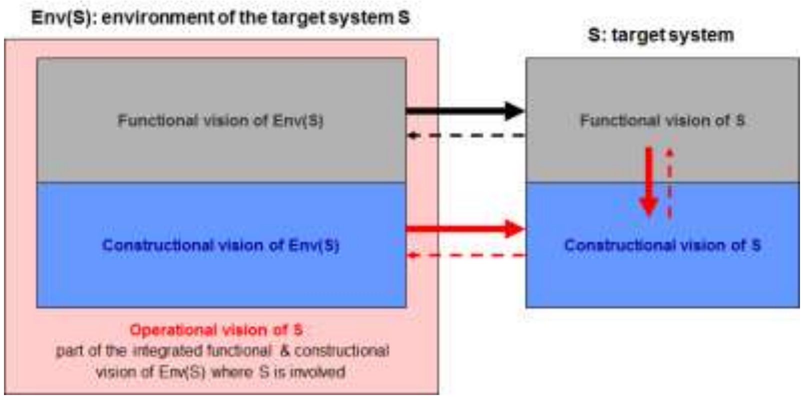
Env(S): environment of the target system S
S: target system
Functional vision of Env(S)
Functional vision of S
Constructional vision of Env(S)
Constructional vision of S
Operational vision of S
part of the integrated functional & constructional vision of Env(S) where S is involved

Figure 15 – Relationships existing between the three architectural visions

One must first understand that the operational vision is nothing else than a mixed functional and constructional description of the subset of the environment of the system of interest which involves this last system. As an

identifying exhaustively all functions of the considered system, which can be checked by a functional synthesis proving that all identified functions are forming a coherent functional network.

immediate consequence, the functional (resp. constructional) dimension of the environment Env(S) of a given system S naturally maps with the functional (resp. constructional) dimension of S. Such a property implies therefore that operational architecture is connected both to functional and constructional architectures of a given system. As a matter of fact, the geometry of a given system's environment influences for instance directly the geometry of the considered system, without any connection with functions. The same situation also holds for most of the physical properties of the environment.

On the other hand, one must also notice that there may be feedbacks from the constructional vision onto the functional vision and/or the operational vision and/or from the functional vision onto the operational vision. The choice of a specific technology at constructional level may indeed typically induce functions that were not directly requested. In the same way, the choice of a specific function at functional level may allow new services that were initially not designed. As another example, just remember that nobody could imagine the creation of an entire new world of new services thanks to the apparently simple Internet functionality!

2.3.4 Organization of a System Model

We are now in position to derive the first consequences of the CESAM framework on the structure of a system model. We are indeed now aware of two dimensions of any system, the first one being provided by the three architectural visions used to model a system, the second one being simply given by the abstraction / grain level on which a given system may be analyzed. On this last point, we shall just recall that any integrated system can be naturally analyzed on the different levels of abstraction provided by its integration hierarchy, that is to say at system, sub-system, sub-sub-system, etc. levels.

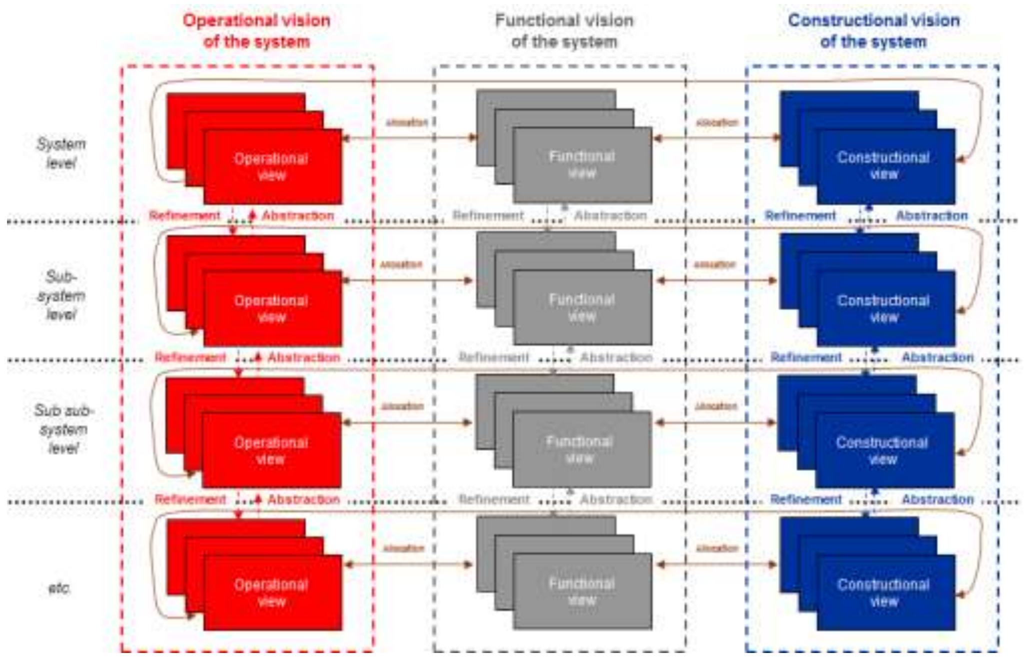

Figure 16 – Organization of a system model

Hence any system model can be naturally organized into a matrix where the different system views are classified according to their architectural vision and the level of analysis within the system integration hierarchy, as depicted in Figure 16. Note that the horizontal relationships between these views is allocation or implementation as explained in the previous sub-section, when the vertical relation is refinement[25] when going from a high-level

[25] Abstraction & refinement are core mechanisms for systems architects, especially when creating architectural hierarchies. A quite frequent problem in architecture is indeed the excessive number of objects generated by a step of an architectural analysis. In order to handle them effectively and to achieve a real global understanding, we typically have to cluster and to

view to a lower-level view and abstraction when doing the converse. On one hand, refinement is clearly the right tool when one wants to precisely analyze a problem. On the other hand, abstraction is crucial for being able to define an architectural strategy without being lost in an ocean of details.

2.4 CESAM Systems Architecture Pyramid

2.4.1 The Three Key Questions to Ask

As discussed in the previous section, any system can be analyzed from an operational, a functional and a constructional perspective. In order to achieve such analyzes in practice, one must simply remember that one shall just ask three simple questions in a particular order to cover these different architectural visions:

- *Key operational question:* WHY does the system exist?[26]
- *Key functional question:* WHAT is doing the system?[27]
- *Key constructional question:* HOW is formed the system?[28]

One usually summarizes these different questions in the *CESAM Systems Architecture Pyramid*, which is a simple graphical way to represent a system, presented in Figure 17 below.

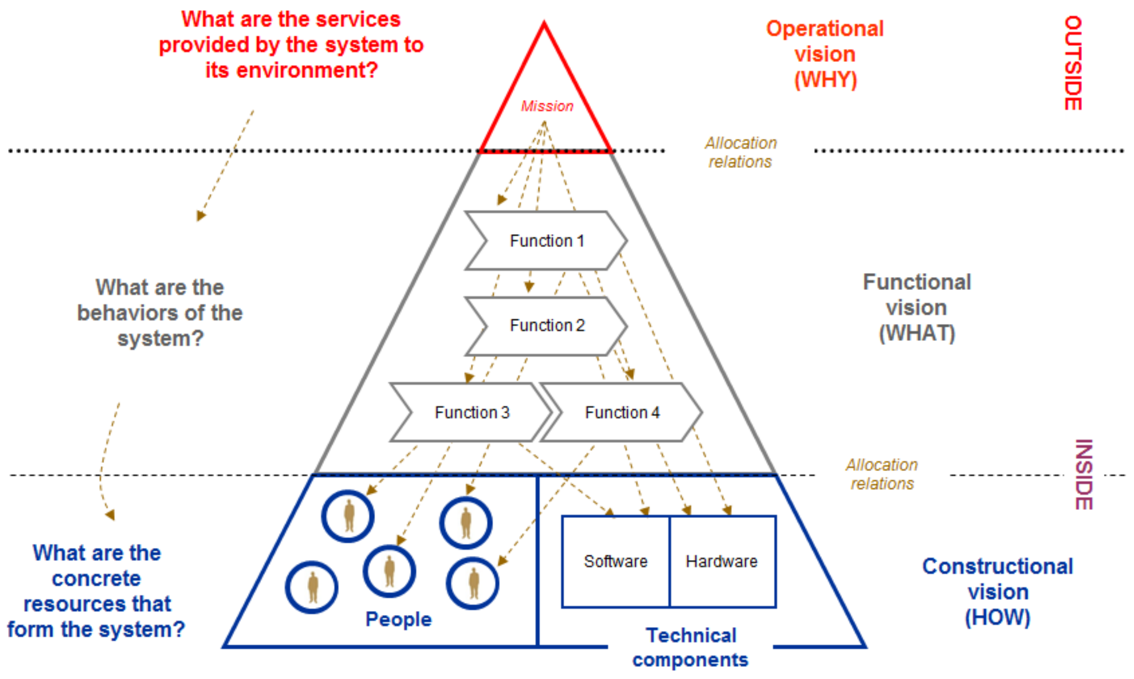

Figure 17 – The CESAM systems architecture pyramid

Be however careful *not to manage successively*, that is to say one after the other, these three types of analyses which should largely overlap in practice. At some point, it is indeed just impossible to reason operationally

synthesize them into abstract objects. This architectural abstraction activity can be achieved by partitioning the considered objects in clusters of "similar" weakly inter-dependent objects, then clarifying systematically the key characteristics (goal, function, feature, etc.) of each group and naming each group consequently. Such a process will naturally lead to architectural hierarchies such as mission, functional or product breakdown structures as introduced in the previous subsections.

[26] Or more precisely what are the services provided by the system to its environment?

[27] Or equivalently what are the behaviors/functions of the system?

[28] Or in other terms, what are the concrete resources that form the system?

without any vision of the concrete solution that will answer to the operational architecture. A good systems architecting practice is clearly to manage in parallel the three architectural analyses at the same time, but not at the same grain of analysis. Organizing the systems architecting process in that way will allow passing from technical-oriented to value-oriented system design strategies.

2.4.2 The Last Question that Shall Not Be Forgotten

The three previous questions are however not the only ones that one should ask in practice in the context of complex systems design & development. The last fourth key question, unfortunately often forgotten in real systems contexts, refers to the product/project duality. It simply consists in asking which person – in other terms "Who?" – is the project counterpart of the different product elements described in the three architectural visions of the product, that is to say:

- WHO owns each architectural element of the system?

This question shall therefore be declined according to the three architectural visions as follows:

- *Operational perspective:* who are the stakeholders of the system?
- *Functional perspective:* who is in charge of the functions of the system?
- *Constructional perspective:* who is responsible of the components of the system?

Asking these different questions is clearly fundamental from a systems architecting perspective. First it is just impossible to capture the right needs without deeply interacting with the stakeholders of the system of interest. Secondly, we may recall that the robustness of a system design is directly correlated to the fact that all transverse functions are managed. Third one cannot imagine influencing the design of a system without involving all its functional & constructional system owners. In this matter, we shall also recall that the role of a systems architect is usually to manage a complex socio-dynamics implying all these different actors, which cannot be done without perfectly understanding their personal motivations, their synergies and their antagonisms with respect to a given system design & development project. We are here again in the "who" – and not the technical – sphere.

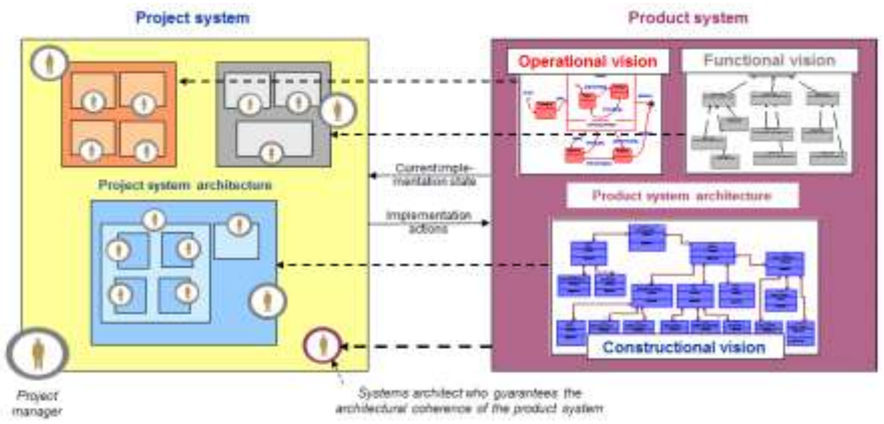
Project system
Product system
Operational vision
Functional vision
Project system architecture
Current implementation state
Implementation actions
Product system architecture
Constructional vision
Project manager
Systems architect who guarantees the architectural coherence of the product system

Figure 18 – Alignment of the project system architecture with the product system architecture

Note finally that the "who" question is also crucial in the construction of the project organization. A good project architecture indeed results from the mapping of all architectural elements of a given product system into the project system, where they shall be put under a single responsibility. Figure 18 illustrates this alignment principle on the electronic toothbrush example: the boxes appearing on the project system side are for instance modeling the project teams that correspond there to the first levels of breakdowns of the different architectural views which are provided on the product system side.

2.5 More Systems Architecture Dimensions

Architectural visions are however not the only architectural dimensions of a system. We shall now introduce several new dimensions that can be used to refine each architectural vision.

2.5.1 Descriptions versus Expected Properties

One must now recall that there exists two complementary ways of specifying a system. The first one refers to *descriptions*: in this specification mode, one explicitly[29] describes the behavior and structure, either of the system of interest (if one reasons functionally or constructionally) or of its environment (if one reasons operationally).

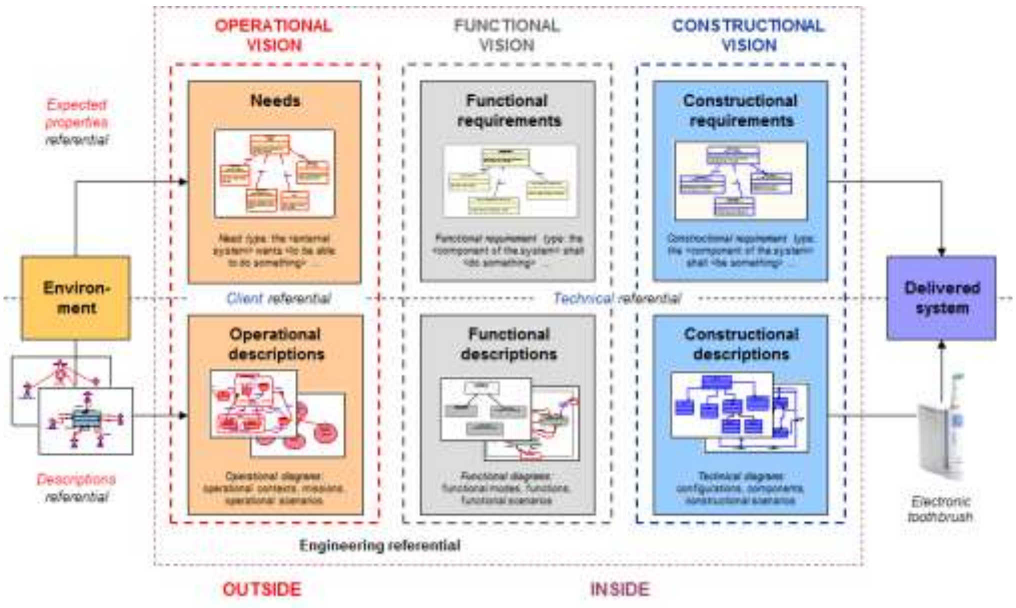
OPERATIONAL VISION
FUNCTIONAL VISION
CONSTRUCTIONAL VISION
Expected properties referential
Needs
Functional requirements
Constructional requirements
Environ-ment
Client referential
Technical referential
Delivered system
Operational descriptions
Functional descriptions
Constructional descriptions
Operational diagrams: operational contexts, missions, operational scenarios
Functional diagrams: functional modes, functions, functional scenarios
Technical diagrams: configurations, components, constructional scenarios
Descriptions referential
Electronic toothbrush
Engineering referential
OUTSIDE
INSIDE

Figure 19 – Descriptions versus expected properties

The second way deals with *expected properties*: one is now not explicitly describing a system, but rather stating the – operational, functional and constructional – properties, expected/intended[30] to be satisfied by the system. Note these expected properties are usually called *requirements* in systems engineering. This gives rise to the six different – but altogether exhaustive – specification modes that are presented in Figure 19:

- For descriptions: operational descriptions, functional descriptions, constructional descriptions,
- For expected properties: needs[31], functional requirements, constructional requirements.

From a theoretical perspective, one can equivalently specify any system by using either descriptions or expected properties. However, these two modes of specification are just not equivalent at all from an engineering effort perspective. On one hand, descriptions are indeed well adapted to define and to synthetize the behavioral & structural dimensions of a system. On the other hand, expected properties can be efficiently used to capture the performances of a system. But the converse is totally false: behavioral and structural breakdowns & interactions are painful to specify as expected properties when performances are difficult to specify using descriptions. As a good practice, an efficient & optimal system specification shall mix descriptions (reserved for defining behavioral

[29] This is why descriptions are considered as specifications *in extension*.

[30] This is why expected properties are considered as specifications *in intention*.

[31] We will use here the term “need” instead of “operational requirement”, even if they are equivalent. We indeed prefer to reserve the term “requirement” for functional & constructional purposes in order to strictly separate the domain of the question (expressed with needs) and the domain of the solution (stated with requirements).

& structural elements and their dynamics) and expected properties (reserved for performances). This trick allows drastically reducing the requirements volume in a specification file, thus improving its readability.

2.5.2 Descriptions

Descriptions can be separated in four different views, each of them modeling a different dimension of a system. States are first modeling time. Static elements are then depicting the core objects of each architectural vision when dynamics are describing their temporal behavior. Flows are finally consolidating the exchanges involved in these dynamics. One should also note that all these system views are both exhaustive – they allow modeling completely a system – and non-redundant – each view provides a specific perspective which is not covered by the other views – due to the mathematical foundations of our system architecting framework.

2.5.2.1 States

A *state T* associated with a given system S is modeling a period of time, that is to say a set consisting of one or more intervals of time, where the system S can be analyzed in a homogeneous way from the perspective of a given architectural vision. A state can be usually specified by its initiation and termination events, which are both modeling phenomenons occurring instantaneously, i.e. at a specific moment of time.

In each architectural vision, a key temporal analysis indeed consists in breaking down into various states the timeline of a system, from its birth to its death. In such analyses, one can then model the usual temporal behavior of a system as a succession of states in which lies the system. There are therefore logically three different types of states for a system S, depending on the considered architectural vision, defined as follows:

- Operational states are called *operational contexts*: an operational context for S is a period of time OC(S) characterized by the fact that external interactions of S during OC(S) do only involve a certain fixed set of stakeholders or external systems of its environment;
- Functional states are called *functional modes*: a functional mode for S is a period of time FM(S) which is characterized by the fact that the behavior of S during FM(S) can be modeled by only using a certain fixed set of system functions;
- Constructional states are called (technical) *configurations*: a configuration for S is a period of time TC(S) – usually identified with the involved components – characterized by the fact that the structure of S during TC(S) does only consist of a certain fixed set of system components.

Let us end by providing the standard representation of states – in most modeling languages – which are usually modeled by means of oval shapes, as one can see in the below Figure 20.

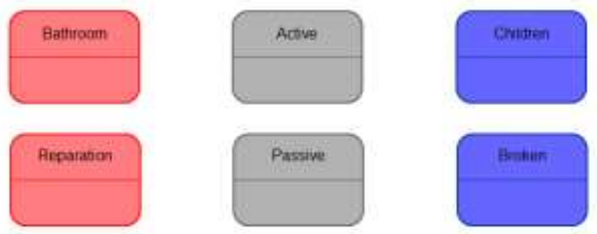

Figure 20 – Standard representations of states

2.5.2.2 Static Elements

A *static element* with respect to a given system S refers to an input/output mechanism associated with S from the perspective of a certain architectural vision. We are using the term "static element" to emphasize that this new system description does not focus on the temporal dynamic of the involved input/output mechanism, but provides just a non-temporized definition of such a mechanism without eliciting its "algorithm".

There are therefore logically three different types of static elements for a given system S, depending on the considered architectural vision, which are defined as follows:

- Operational static elements are called *missions*: a mission of S is defined as an input/output behavior of the environment Env(S) of S, involving both S and other external systems (missions are not expressed at the level of the system of interest, but at the level of its environment);
- Functional static elements are called *functions*: a function of S is defined as an abstract implementation-independent input/output behavior of S, which is therefore achieved by only involving S (functions do only refer to the system of interest);
- Constructional static elements are called *components*: a component of S formally refers to a concrete implementation-dependent intrinsic input/output behavior of S and is therefore naturally identified to a concrete part of S (components do only refer to the system of interest).

We may also provide the standard representations of the three different types of static elements – in most modeling languages – which are usually modeled by circles for missions, ovals for functions and boxes for components, as one can see in the below Figure 21.

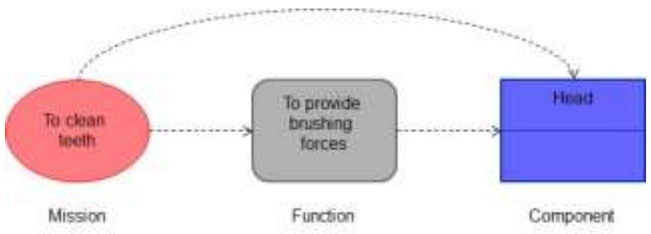

Figure 21 – Standard representations of static elements

Note that static elements can occur at different abstraction levels that do also correspond to different integration levels, resulting in aligned abstraction and integration hierarchies. Hence it is always crucial to specify how different types of static elements are connected altogether by such relationships. The standard representations of these abstraction / integration relationships are provided by the below Figure 22.

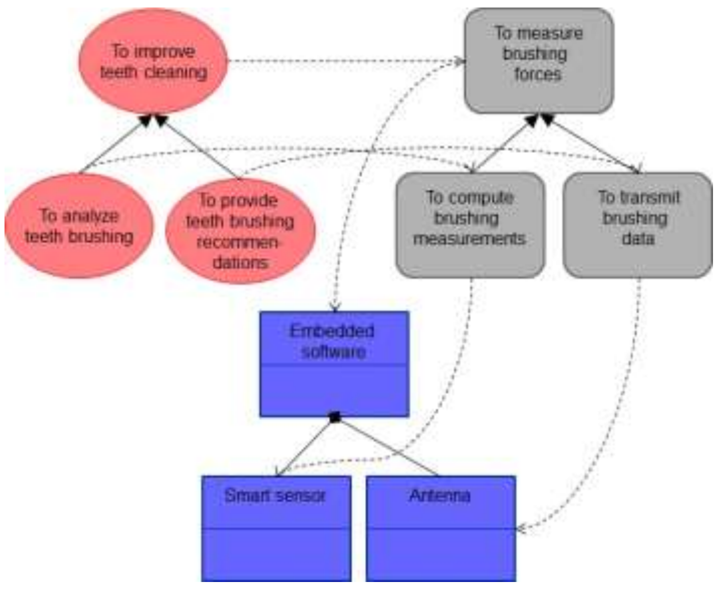
To improve teeth cleaning
To measure brushing forces
To analyze teeth brushing
To provide teeth brushing recommen-dations
To compute brushing measurements
To transmit brushing data
Embedded software
Smart sensor
Antenna

Figure 22 – Standard representations of integration relations between static elements

For the sake of completeness, one also needs to explicitly represent the full *integration mechanism* that relates the different static elements of lower level that are abstracted by a static element of higher level. Figure 23 below shows an example of standard representation of such an integration mechanism – where oriented arrows labelled with an exchanged flow represent interfaces– between different constructional components of the same level of abstraction. Similar representations do also exist with functions or missions.

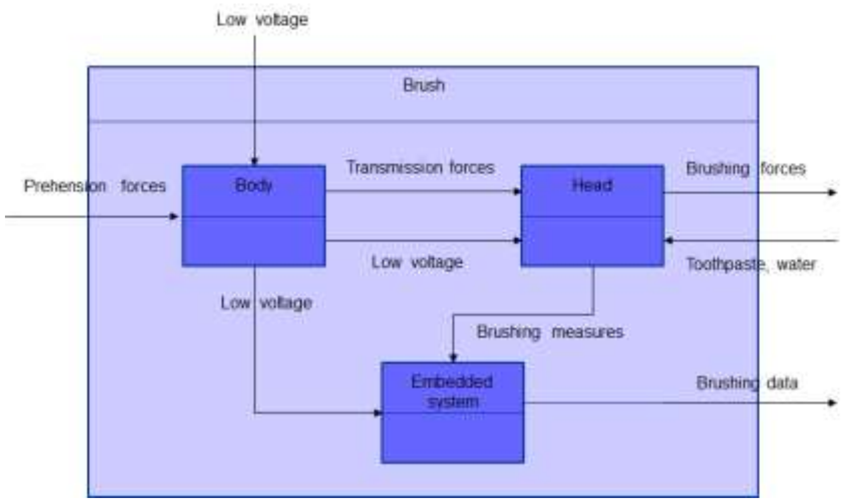

Figure 23 – Interfaces standard representation

2.5.2.3 Dynamics

The *dynamic* of a static element of a system S refers to a possible temporal behavior or equivalently to an algorithmic description of such a temporal behavior. Dynamics are completely crucial if one wants to precisely specify the behavior of any system or any system mission, function or component.

There are therefore logically three different types of dynamics that one can associate with a system, depending on the considered systems architecture vision, that is to say operational, functional and constructional dynamics, which are defined as follows for a given system S:

- Operational dynamics are called *operational scenarios*: an operational scenario of S is an algorithmic description of the interactions that do exist between the system (considered as a black box) and the various external systems of its environment,
- Functional dynamics are called *functional scenarios*: a functional scenario of S is an algorithmic description of the interactions existing, on one hand, internally between the functions of S and, on the other hand, externally with the external systems of the environment of S,
- Constructional dynamics are called *constructional scenarios*: a constructional scenario of S is an algorithmic description of the interactions existing, on one hand, internally between the components of S and, on a second hand, externally with the external systems of the environment of S.

Note that these three types of scenarios are syntactically quite similar since they are all describing an exchange algorithmic. The only difference comes here from the semantics and the nature of the underlying exchanges that are described by these different types of scenarios.

Let us now end by addressing the question of the standard representations of the different types of dynamics that we introduced, which are typically given – in most modeling languages – by *sequence diagrams*. Figure 24 depicts this formalism with an example of operational dynamic description or operational scenario, here the initiation of a toothbrush use where one specifies the interactions existing between all involved stakeholders and the electronic toothbrush system. Sequence diagrams do indeed provide an efficient and classical formalism for representing distributed algorithms, which typically correspond to the generic situations that occur in a dynamic system context (see [5] for more details).

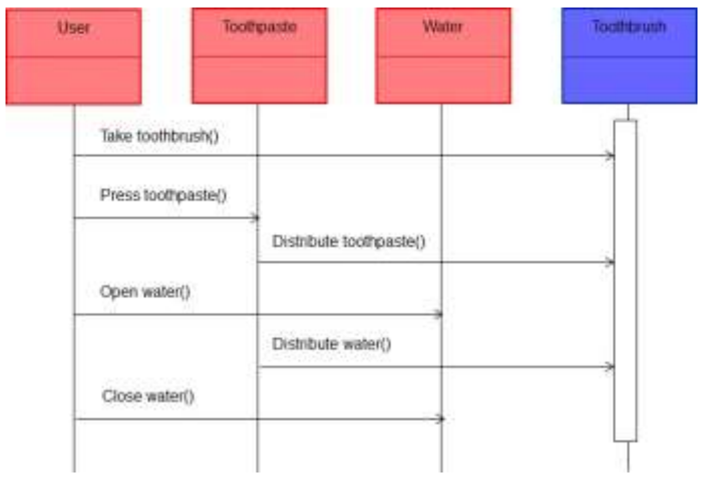

Figure 24 – Standard representations of an operational dynamic

For the sake of completeness, we also provide an example of constructional scenario that can be found in Figure 25 below which represents the exact constructional counterpart of the previous operational scenario. Functional scenarios are represented exactly in the same way, components being just replaced by functions. Note that the difference between a functional or constructional scenario and an operational scenario is that the environment is a black box in the first situation, when it is the case of the system in the second situation.

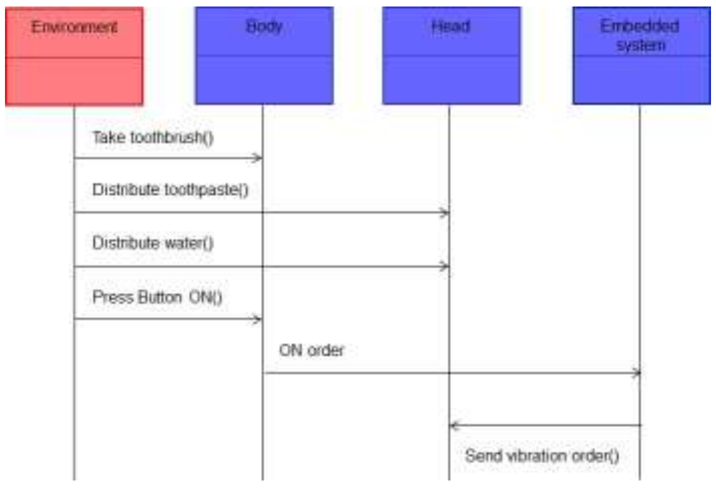

Figure 25 – Standard representations of a constructional dynamic

2.5.2.4 Flows

A *flow* associated with a system S models an object – matter, energy, data, information, etc. – which is exchanged either externally between the system and its environment, or internally between two functional or constructional elements of the system.

There are therefore logically three different types of flows for a given system S, depending on the considered architectural vision, which are defined as follows:

- Operational flows or objects: an *operational flow or object* of S is an object that is exchanged between S and its environment, i.e. between S and one of its external systems,
- Functional flows or objects: a *functional flow or object* of S is an object which is an input or an output of one of the functions of S,
- Constructional flows or objects: a *constructional flow or object* of S models a concrete object which is an input or an output of one of the components of S.

Flows are usually stated using only names referring to concrete objects exchanged between various systems. Table 1 below illustrates the different notions of flows with some examples for an electronic toothbrush.

Flow types	Flow	Nature
Operational Flows	Toothpaste	Matter
	Brushing data	Data
Functional Flows	Low voltage	Energy
	Brushing measure	Data
Constructional Flows	Electricity	Energy
	Brushing pressure signal	Data

Table 1 – Examples of flows for an electronic toothbrush

Let us end by providing in the below Figure 26 the standard representations of the three different types of flows – in most modeling languages – which are modeled by "objects" or "blocks" in the usual meaning given to this concept in object-oriented modeling (see [5] or [7]).

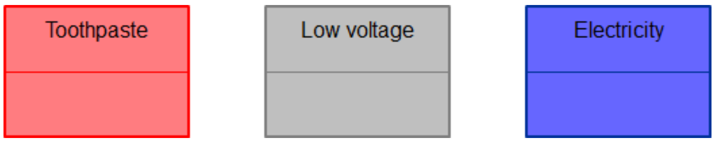

Figure 26 – Standard representations of flows

2.5.3 Expected Properties

Expected properties are formally speaking *logical predicates* related with the system of interest. An expected property is thus nothing else than a Boolean function[32], i.e. a function P that maps a system on TRUE or FALSE, depending whether the property that P models is satisfied or not by the system:

$$P: S \longrightarrow P(S) \in \{ \text{FALSE}, \text{TRUE} \} .$$

This consideration allows avoiding confusing expected properties with their constitutive elements. An expected property indeed typically expresses that a given system shall behave or be structured with a certain external or internal performance[33]. The involved behavioral or structural elements & performances shall thus not be mixed with the property, since they are just not logical predicates.

[32] This is thus also true for needs and requirements according to the definitions that follow.

[33] In other terms, expected properties do express the performances that shall be satisfied by the system of interest or by its environment, depending on whether one deals with the functional/constructional or operational visions.

It is also important to remember that one can analyze in practice a given system S from an external or from an internal perspective. This consideration leads to the key distinction between needs & requirements, which are the two first main types of expected properties on S:

- **External perspective[34] – Need:** a *need* with respect to S is a property that is expected or imposed by the environment Env(S) of S, expressed in the language of the environment, i.e. only referring to operational descriptions & performances,
- **Internal perspective – Requirement:** a *requirement* on S is a functional or constructional property that shall be satisfied by S, expressed in the language of the system, i.e. only referring to functional or constructional descriptions & performances.

One shall of course understand that these definitions are fundamentally relative to a given system. A need with respect to a system S is indeed a requirement on the environment Env(S) of S. In the same way, a requirement on S is also a need with respect to a subsystem of S. One should hence always remember to point explicitly out the system of interest to which refers any need or requirement.

Note also that we used here above a voluntarily different terminology, depending on the external or internal perspective that we may take with respect to an expected property. A good system architecting practice indeed consists in strictly separating the domain of the *question* – expressed using needs – from the domain of the *solution* – expressed by means of requirements. In other terms, needs shall be only reserved to model questions, when requirements shall model the associated answers. This point is crucial since many engineering issues are due to the fact that stakeholders are often expressing their "needs" in an intrusive way, that is to say in terms of requirements. This bad practice both limit the ability of the designers to propose better alternative solutions and prevent them from knowing the real needs hidden behind requirements, which may ultimately lead to bad solutions from an end-user perspective, even if fitting perfectly to their requirements.

Note finally that requirements on a system S can be of course refined into two sub-types depending on whether one is dealing with functional or constructional visions:

- **Functional requirement:** a *functional requirement* on S is a property satisfied by the behavior of S, expressed in its functional language, i.e. functional descriptions & performances,
- **Constructional requirement:** a *constructional requirement* on S is a property satisfied by the structure of S, expressed in its constructional language, i.e. constructional descriptions & performances.

To write down properly these different types of expected properties, one shall use the standard statement patterns that are provided in Table 2.

Need	The "external system"[35] shall "do something / be formed of something" with a certain "operational performance" in a given "operational context".
Functional requirement	The "system" shall "do something"[36] with a certain "functional performance" in a given "functional mode".
Constructional requirement	The "system" shall "be formed of something"[37] with a certain "constructional performance" in a given "configuration".

Table 2 – Standard statement patterns for needs and functional & constructional requirements

[34] External perspective is here a synonym of operational vision.

[35] Or equivalently a stakeholder of the system.

[36] "Do something" shall always refer here to an existing function of the considered system.

[37] "Be formed of something" shall always refer here to an existing component of the considered system.

Table 3 below illustrates the three previous different types of expected properties – written in accordance with the above standard statement patterns – on an electronic toothbrush. In this example, the proposed need was derived first into a functional requirement that was derived then into a constructional requirement. One can thus immediately trace back here the last constructional choice to the associated service provided to the end-users, which is useful – especially for analyzing the stakeholder value brought by a given technical decision – since this end-user service cannot be seen at constructional level.

Need	End users shall get a positive feeling when efficiently cleaning their teeth.
Functional requirement	The electronical toothbrush shall display an encouraging message within 1 second when cleaning performance has been met.
Constructional requirement	The electronical toothbrush shall have a user interface of 2.5 cm x 1cm in each of its configurations.

Table 3 – Examples of expected properties per architectural vision

Note finally that the previous types of expected properties are formally *complete* due to the mathematical foundations of the CESAM framework. In other words, one can always specify any system in intension by *using only* needs, functional requirements and constructional requirements, without anything else.

There is in particular no need to introduce the concept of "non-functional requirements" that exists in many other architectural referential (cf. [9], [10] or [11]) and does often refer to "ity" properties of a system such as availability, maintainability, operability, safety, reliability, security, etc. All these properties can indeed easily be expressed in terms of needs. To be more specific, let us take the example of a maintainability property for a given system which would probably be expressed by most engineers by stating $M \equiv$ "the system shall be maintainable". However, "be maintainable" can clearly not be considered as an internal function of a system since it does not refer to any input/output behavior, but rather to a permanent status of the system. Property M is therefore neither a functional, nor a constructional requirement[38], nor a need[39] within our framework. It has thus absolutely no status at all, which shows that it is probably just a bad specification! The good way of expressing the property M is then just to identify the hidden stakeholders which are behind – here maintenance teams – and to understand what are expecting these stakeholders. In our example, this would lead us to reformulate M by stating instead $M' \equiv$ "the maintenance teams shall maintain the system with a certain performance" which is now obviously a need since it expresses an expectation from the environment of the system.

2.6 CESAM Systems Architecture Matrix

We are now in position to introduce CESAM Systems Architecture Matrix, which is presented in Table 4 below. This matrix is just a synthesis of the different architectural dimensions that we introduced. It indeed presents all the types of views that allow to exhaustively describe any system, classified according to:

- a first axis of classification corresponding to the three *architectural visions*, that is to say the operational, functional and constructional visions,
- a second axis of classification corresponding to *behaviors*, that is to say the conjunction of *expected properties* and all *descriptions,* i.e. states, static elements, dynamics and flows.

One will of course always have to find the good balance between expected properties and descriptions when specifying a system. CESAM Systems Architecture Matrix is thus only an help to be sure that all dimensions of a system were considered during its modeling, but it does not provide – neither CESAM systems architecting

[38] Since it also obviously does not directly refer to a property of the components of the system.

[39] Strictly speaking, it indeed only refers to the system and not to its environment.

method does – an automatic specification mechanism for systems. Systems architecture indeed remains an art where expertise, experience and competency of systems architects are clearly fundamental.

Visions	Expected properties	Descriptions			
		States	Static elements	Dynamics	Flows
Operational vision	Needs	Operational contexts	Missions[40]	Operational scenarios	Operational flows or objects
Functional vision	Functional requirements	Functional modes	Functions[41]	Functional scenarios	Functional flows or objects
Constructional vision	Constructional requirements	Configurations	Components[42]	Constructional scenarios	Constructional flows or objects

Table 4 – CESAM Systems Architecture Matrix

To concretely illustrate this last notion, let us now provide an example of a partially completed CESAM Systems Architecture Matrix for the electronical toothbrush.

Visions	Expected properties	Descriptions			
		States	Static elements	Dynamics	Flows
Operational vision	End-users want to have less than one cavity in average per 5 years due to teeth brushing	Teeth Brushing	Brush teeth	Teeth brushing scenario	Toothpaste
Functional vision	The electronic toothbrush shall produce a brushing force of 0.5 N in automatic mode	Automatic mode	Produce a brushing force	Brushing force production scenario (functional)	Brushing force
Constructional vision	The electronic toothbrush shall have a removable head in children & adult configurations	Children configuration	Head	Brushing force transmission scenario (concrete)	Electricity

Table 5 – Example of a CESAM Systems Architecture Matrix for the electronical toothbrush

One can now understand why system modeling is so unintuitive. If one completes CESAM Systems Architecture Matrix by adding system abstraction / integration levels, one may indeed understand that a system model looks much more to a cube than to a matrix, as depicted on the below Figure 27 that represents CESAM Systems Architecture Cube, the 3D-version of the 2D CESAM Systems Architecture Matrix. One can thus understand that it is easy to be lost in such a multi-dimensional world! This is therefore probably one of the most important difficulties of complex systems architecting and modeling.

[40] Including descriptions of all integration mechanisms involving missions.

[41] Including descriptions of all integration mechanisms involving functions.

[42] Including descriptions of all integration mechanisms involving components.

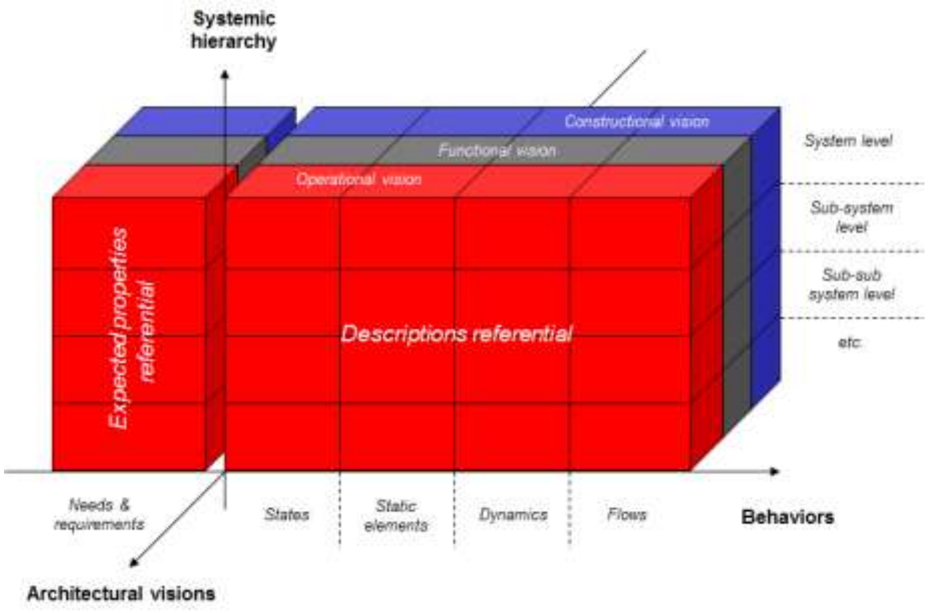

Figure 27 – CESAM Systems Architecture Cube

Note also that the three first description types – states, static elements and dynamics – are the most important since the last one – flows – is just a dedicated synthesis, focused on exchanges, which consolidates information that can already found in the views associated with dynamics. Restricting the CESAM Systems Architecture Matrix to these three first description types lead us thus to a simpler matrix – the so-called CESAM 9-views matrix[43] – which provides the minimal number of descriptions to construct when "modeling" a system. An example of such CESAM 9-views matrix is provided below on the electronical toothbrush case study.

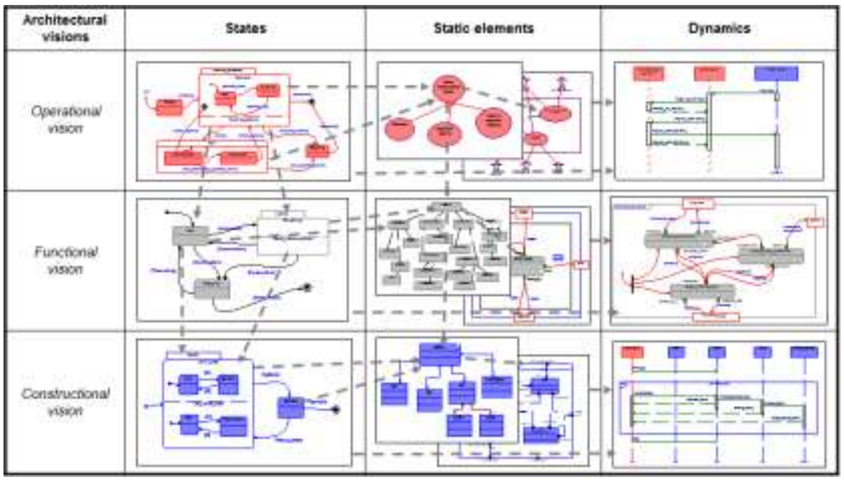

Figure 28- Example of a CESAM 9-views matrix for an electronical toothbrush

At this point, note finally that CESAM Systems Architecting Method is nothing else than a certain way of moving in the CESAM Systems Architecture Matrix, starting from the knowledge of all use cases associated with a system, up to arriving to a quite precise vision on all constructional scenarios of the system.

[43] This terminology was invented by V. Vion, former chief systems architect at PSA Peugeot Citroën (now Stellantis).

3 Conclusion

Time to Develop a Systems Architecting Leadership

Mastering the system modeling framework contained in this pocket guide is clearly a first necessary important step for a systems architect. However, systems architecting cannot and shall not be reduced to systems modeling. The objective of systems architecting indeed consists in helping system development teams to make rational & shared optimal choices in complex environments. Therefore, the main difficulty of systems architecting is not to master modeling in its own, but rather to create a common and shared vision – involving all project actors – around the system in development with the help of system models.

This last point can only be approached through concrete system development projects with real experiences of consensus building. Understanding the "theory" is clearly not enough as soon as one is dealing with human relationships issues, which are in fact at the center of systems architecting practice. One shall indeed understand that one cannot manage a convergence protocol, in order to create a shared vision among project actors on a given topic, in the same way than a technical study or a prototype development. When dealing with people, mistakes or bugs are typically forbidden. Contrarily to a purely technical problem, human issues shall therefore always be managed with great care. All the complexity of the systems architect job is thus to be able to manage convergence protocols – with their inherent socio-dynamic difficulties – in complex technical environments.

To progress in systems architecting, the systems architect shall thus clearly develop its non-technical competency (consensus creation, workshop animation, meeting facilitation, etc.), which must be applied in technical complex contexts. In this matter, practice is fundamental. One can thus just not achieve developing leadership in systems architecting without confronting to the complexity of real development situations. In this matter, one may note that CESAMES is offering dedicated 6-to-10-month on-the-job trainings in systems architecting: such trainings are especially dedicated to that leadership construction through the concrete application of a full systems architecting process on a real system and the achievement of a complete systems architecture specification file (for more details, see the corresponding item in the training section on our website www.cesames.net).

CESAM Community

CESAMES and CESAM Community

CESAM Community, which supported the construction of this pocket guide, is managed by CESAMES non-profit organization which emerged in 2009 as a spin-off of the Ecole Polytechnique "Engineering of Complex Systems" industrial chair with the aim of disseminating systems architecting in academia & industry. To do so, CESAMES organizes awareness events all year long for the scientists and professionals to meet and share about complex systems. As an example, CESAMES organizes on a yearly basis the "Complex Systems Design & Management" (CSD&M) international conference series. Since 2010 in Paris and 2014 in Asia (Singapore and China), this event gathers each year more than 300 academic & professional participants, coming from all parts of the world. The CESAMES non-profit organization also manages thematic evenings and working groups, always with the same goal: increasing awareness about systems architecting methods and tools.

Thanks to these events, CESAMES has federated a significant international community of system engineers and architects who all share the same vision: systems architecture and engineering do represent a key factor of competitiveness for the companies.

To reinforce visibility and get more influence at a worldwide level, CESAM Community was established in 2017 by CESAMES. Its mission remains the same: organizing the sharing of good practices in enterprise and systems architecture and attesting the ability of practitioners to implement them through the CESAM certification, which groups today around 3,500 certified architects in the world, at various levels of seniority.

More precisely, CESAM Community works to:

- **Make architecture a key tool for business competitiveness** by promoting and disseminating its use in companies and by communicating the obtained results within the community;
- **Propose and develop the best practices of systems architecture in industry and services** through the creation of publications & guides and the sharing of returns on experience between systems architects and engineers during the events of the community;
- **Propose a generic architecture framework, but also offer adapted systems architecture frameworks**, specific to some industrial domains in order to facilitate the work of systems architects;
- **Facilitate access to the CESAM method and develop its use worldwide.**

References

[1] ANSI/GEIA, *ANSI/GEIA EIA-632 – Processes for engineering a system*, 2003

[2] Aslaksen E.W., *The changing nature of engineering*, McGraw-Hill, 1996

[3] Aslaksen E., Belcher R., *Systems engineering*, Prentice Hall, 1992

[4] Blanchard B.S., Fabricky W.J., *Systems engineering and analysis*, Prentice Hall, 1998

[5] Booch G., Jacobson I., Rumbaugh J., *The Unified Modeling Language Reference Manual,* Second Edition, Addison-Wesley, 2004

[6] de Weck O.L., Roos D., Magee C.L., *Engineering systems – Meeting human needs in a complex technological world*, The MIT Press, 2011

[7] Friedenthal S., Moore A.C., Steiner R., *A Practical Guide to SysML : the Systems Modeling Language*, Morgan Kaufmann OMG Press, 2012

[8] Honour E.C., *Understanding the value of systems engineering,* INCOSE 2014 International Symposium, Vol. 14, 1207–1222, Toulouse, June 20-24, 2014, France, INCOSE, 2004; accessible at http://www.seintelligence.fr/content/images/2015/12/ValueSE-INCOSE04.pdf

[9] IEEE, *IEEE 1220-2005 – Standard for Application and Management of the Systems Engineering Process*, Institute of Electrical and Electronics Engineers, 2005

[10] INCOSE, *Systems Engineering Handbook, A guide for system life cycle processes and activities*, INCOSE, January 2011,

[11] ISO/IEC/IEEE, *ISO/IEC/IEEE 15288:2015 – Systems and software engineering -- System life cycle processes*, May 2015

[12] Kossiakoff A., Sweet W.N., *Systems engineering – Principles and practice*, Wiley, 2003

[13] Maier M.W., Rechtin E., *The art of systems architecting*, CRC Press, 2002

[14] Meinadier J.P., *Ingénierie et intégration de systèmes*, Lavoisier, 1998

[15] Meinadier J.P., *Le métier d'intégration de systèmes*, Lavoisier, 2002

[16] NASA, *Systems Engineering Handbook*, 2007-edition, NASA*/SP-2007-6105, 2007*

[17] Sage A.P., Armstrong J.E., *Introduction to systems engineering*, Wiley, 2000

[18] Sillitto H., *Architecting systems – Concepts, principles and practice*, College Publications, 2014

[19] Simon H., *The Architecture of Complexity*, Proc. of the American Philosophica, 106 (6), 467-482, Dec. 1962

[20] The Open Group, *TOGAF® Version 9.1 – The Book*, The Open Group, 2011

[21] Turner W.C., Mize J.H., Case K.H., Nazemetz J.W., *Introduction to industrial and systems engineering*, Prentice Hall, 1978

[22] von Bertalanffy K.L., *General System Theory : Foundations, Development, Applications*, G. Braziller, 1976

[23] Wikipedia, *Enterprise architecture framework*, https://en.wikipedia.org/wiki/Enterprise_architecture_framework

[24] Wikipedia, OSI model, https://en.wikipedia.org/wiki/OSI_model

[25] Wikipedia, *Predicate*, https://en.wikipedia.org/wiki/Predicate_(mathematical_logic)

[26] Wikipedia, *Systems engineering*, https://en.wikipedia.org/wiki/Systems_engineering

[27] Wikipedia, *Systems theory*, https://en.wikipedia.org/wiki/Systems_theory