On Reference Architectures

Daniel KROB[1], Adrien ROQUES[2]

April 29, 2024

Keywords

Abstraction, Architecture, Family of systems, Reference architecture, System

Abstract

A reference architecture is a key architectural asset for the systems engineer, that intends both to simplify its work and to guarantee the exhaustiveness of the coverage of its system analyses. This paper proposes formal definitions for a reference architecture, based on the theory of abstract interpretation that formalizes the notion of abstraction, which cover respectively large and small-scale families of systems. These definitions are illustrated by automotive, building and mining cases studies.

Acknowledgements

We would like to thank the many CESAMES' current and former collaborators who contributed to the various industrial case studies that supported our work: Sevag AINEJIAN, Cécile BEYSSAC, Augustin CURLIER, Yann DECRE, Jérôme GUI, Loic LE SAUCE and Antoine MOUTTAPA. Special thanks are also due to the anonymous referees who helped us to strongly improve this paper.

[1] INCOSE Fellow – Center of Excellence on Systems Architecture, Management, Economy & Strategy (CESAMES) –France – daniel.krob@cesames.net

[2] Center of Excellence on Systems Architecture, Management, Economy & Strategy (CESAMES) – France – adrien.roques@cesames.net

1 – Introduction

Model-based systems engineering (MBSE) fundamentally deals with system models, positioned at the core of the systems engineering & architecting processes (cf. [3], [10], [13], [22], [27], [28], [35], [37]). Succeeding in better defining the precise contours of such system models and the generic methods to build them, is thus both a huge scientific challenge, due to the fact that the theoretical foundations of system modeling still remain to be consolidated, and a huge economic challenge, since eliminating the factors of degradation of quality & performance of complex industrial systems is linked to the capacity of efficiently constructing exhaustive & relevant models of such systems.

In order to improve the development of these system models in industrial contexts where system complexity – in particular functional – is only increasing, having effective systems engineering tools is clearly key. From a system architecting perspective (cf. [12], [23], [27], [36]), such an objective leads to the question of capitalizing on *reference architectures,* that is to say – as considered all along this paper – on predefined standardized generic representations of behavioral & structural features of a family of systems (cf. [26]) within their environment, that one can use, adapt or modify in order to deploy systems architecting & engineering activities without starting from a blank sheet.

In this last matter, note that the MBSE point of view that we took here, led us to a restricted approach to the notion of reference architecture, compared to broader definitions (cf. [6], [32]), which may include for instance vision statements, guidelines, templates or even generic implementation codes, that one can find in a quite rich literature. The concept of reference architecture was indeed especially developed in the ISO 42024 and 42042 standards (cf. [17], [19]) within the recent ISO 420XX series (cf. [15], [16], [17], [18] and [19]). Reference architectures were also similarly explored by the ISO 15704 standard from an enterprise architecture perspective (cf. [14]).

Note that reference architectures in systems engineering can probably be seen as the equivalent of a well-established practice in software engineering, namely "design patterns" (cf. [4]), which allow to optimize the time of design and construction of a software architecture. The same types of ideas, such as system architectural construction patterns, generic & reusable system views or system architecture heuristics, were indeed progressively disseminated in the systems engineering domain (cf. [2], [5], [7], [20], [27], [40]). In the same line, the creation of the INCOSE architecture working group (cf. [12]) can be seen as a good measure of the importance that the notion of systems architecture has nowadays taken in our domain. Examples of application fields such as automation, embedded systems, high-tech, medical equipment or satellite constellations, where such system architectural approaches have been empirically implemented with success are provided for instance in [8], [11], [21], [24] and [38].

It happens that a reference architecture in our MBSE meaning is especially a valuable aid to the systems engineer in the initial phase of preparation of a system design & modeling work: it indeed allows to quickly identify, on one hand, the relevant use cases, functions and components that shall be specified, but also, on the other hand, the set of external systems, functions and components interfaced with the system of interest, which became a highly non-trivial problem due to the complexity of modern complex architectures. The core purpose of a reference architecture is therefore both to simplify the work of the systems engineer & modeler and to guarantee the exhaustiveness of the coverage of the system analyses & models that one has to manage.

One shall moreover note that a reference architecture is of course not a tool to support detailed design: it indeed remains a macroscopic object which specifies at a quite high level of abstraction the whole scope of a given systemic perimeter from operational, functional and structural perspectives. As a consequence, a reference architecture is also key, due to its generic nature, to structure the business architecture (cf. [29]) of the information system that supports the development process of a

complex industrial system, i.e. the PLM (Product Lifecycle Management) information systems, which are used by system designers & manufacturers to collaboratively store and exchange technical data during a system development process (cf. [25] or [31] for good overviews on this topic).

The main challenge of our paper was to try proposing mathematical foundations, based on the abstract interpretation theory of Cousot & Cousot, initially developed for software engineering applications [9], on which one could construct an unambiguous framework for reference architectures from the MBSE point of view taken here. Abstraction – in the formal meaning of abstract interpretation [9] – allows indeed to naturally capture two key properties of reference architectures, i.e. their ability to support the lineage from generic to less generic and from logical to concrete, that are pointed out in most of the relevant standards (cf. [14], [19]). Last, but not least, our theoretical approach was also validated in practice, by following a methodology consisting in abstracting a number of practical case studies that had a good level of genericity, based on the systems architecting framework that we developed along the last decade (cf. [22]), which allowed us to obtain robust and effective definitions for a reference architecture from a model-based systems engineering perspective.

Note finally that the main role of the mathematical formalism introduced in this paper is to provide unambiguous semantics to the systems architectural concepts that are discussed. We are however not directly playing with it, as it is, since it may be inconvenient from a practical perspective. The key idea – as already used in our CESAM framework[22] – is indeed to allow the working systems architect to work safely, protected by rigorous fundamentals, without knowing them. Our formalism is moreover only a first step towards sound foundations of reference architectures. Other formal frameworks, such as for instance the model synchronization paradigm of Batteux, Prosvirnova and Rauzy in [1], which propose to live with the possible discrepancies between a generic model and its concrete instances, under the condition that they are traced and agreed, may for instance complement our approach. In the same way, it would be clearly interesting to explore modular mechanisms to generate reference architectures in our MBSE meaning, based on relevant patterns and compositionality principles.

2 – Preliminaries

2.1 – CESAM Systems Architecture Framework

As a matter of fact, each integrated system S can always be analyzed from three complementary and distinct perspectives that give rise to three complementary generic *systems architectural visions*, that is to say the operational, functional and structural visions, each of them grouping different types of systemic models, as defined below (see [22] for more details):

- *Architectural vision 1 – Operational vision:* the operational vision of S groups 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 of S groups all system-level models describing the input/output dynamics of S, without making reference to any concrete implementation. Such functional models are thus abstractly modeling the behaviors of S.
- *Architectural vision 3 – Structural vision:* the structural vision of S groups the system-level models of S, constructed by composition of its lower-level models associated with its logical components. Such structural models are thus describing the structure of S.

Note that the operational and functional visions in this approach refer to the same classical systems engineering concepts (cf. [13]), when the structural vision is in fact an abstraction of the classical logical and physical visions (cf. [12]). These visions, complemented with a number of sufficient & necessary

standard analysis axes (expected features, states, static elements, dynamic behaviors and flows), derived from a formal definition of a system by means of generalized transfer functions (cf. [22]), are the cornerstones of the so-called CESAM systems architecture framework, as presented in Table 1, that we will use throughout this paper in order to illustrate our approach.

Architectural visions	Expected Features	States	Static elements	Dynamic behaviors	Flows
Operational vision	Needs	Operational contexts	Use cases & external interfaces	Operational scenarios	Operational flows
Functional vision	Functional requirements	Functional modes	Functions & functional interfaces	Functional scenarios	Functional flows
Structural vision	Structural requirements	Technical modes	Components & structural interfaces	Structural scenarios	Structural flows

Table 1 – The CESAM systems architecture framework [22]

Note that other systems architecture frameworks do exist such as the one implicitly used in the INCOSE handbook [13], or introduced within the ISO 42010 standard [15], or also provided by all the systems-of-systems-oriented frameworks such as DoDAF, MoDAF, NAF and nowadays UAF (see [33]). From an historical perspective, one may also cite Penalva, who was probably the very first author to introduce, in 1993, a systems architecture framework – denoted SAGACE – dedicated to the design of general systems [33]. However, all these frameworks have a lot of similarities and, at one moment, one must choose one, when doing systems architecture. The choice of the CESAM framework [22] was here done due to its sound formal fundamentals, while hidden to the working systems architect.

2.2 – Abstraction

Abstraction is a key mechanism in systems engineering & architecting. A system modeling process can for instance be seen as a permanent back-and-forth between, on a first hand, an abstraction activity that constructs a formal model of a real-world concrete system– using the methods & tools of systemic analysis – and on a second hand, an experimentation activity where the structure or behavior of the considered concrete system, as provided, derived or predicted by its model, are checked against those really observed in reality. In the same way, integration of systems is a mechanism that can be formally interpreted in terms of abstraction, as described in [22].

It happens that the mathematical theory of abstract interpretation gives a precise definition of the concept of abstraction, introduced in a software engineering context by Cousot & Cousot in [9] and presented here below, that also perfectly works in systems engineering and contributes therefore to the construction of rigorous algebraic & logical foundations of our domain (see also [30]),

Definition 1 – Abstraction. Let C be an ordered set, called a concrete set, and let A be another ordered set, called an abstract set. A pair (α, χ) of functions, respectively between C to A and A to C, is then a pair of *abstraction / concretization* functions if and only if the following properties are respected:

$$\forall\, c \in C,\ c < \chi(\alpha(c)) \quad \text{and} \quad \forall\, a \in A,\ \alpha(\chi(a)) < a \qquad (1).$$

In other words, properties (1) do mean that if one abstracts a concrete element and concretizes the obtained abstraction, it must dominate – with respect to the concrete order on C – the initial concrete element, and if one concretizes an abstract element and abstracts the obtained concretization, it shall be dominated– in the sense of the abstract order on A – in the initial abstraction.

This formal notion of abstraction is in fact permanently used – usually without knowing it – in systems architecting when constructing *systemic hierarchies*. An operational, functional or product breakdown structure is indeed just a tree of use cases, functions or logical components whose arrows can be interpreted as abstractions in the previous meaning. In order to illustrate this last fact, let us consider the simple example of a functional breakdown structure where a function F is broken down into the functions F1 and F2 as illustrated in the left hand-side of Figure 1. To connect with Definition 1, one shall just introduce an abstract set A consisting of the part of the set { F } and a concrete set C consisting of the parts of the set { F1, F2 }, both naturally ordered by inclusion. The abstraction function α consists then to map the empty set in C onto the empty set in A and all other elements of C on { F }, when the concretization function χ consists in mapping the empty set in A onto the empty set in C and { F } onto { F1, F2 }. One can indeed easily check that the two characterizing properties (1) are then fulfilled.

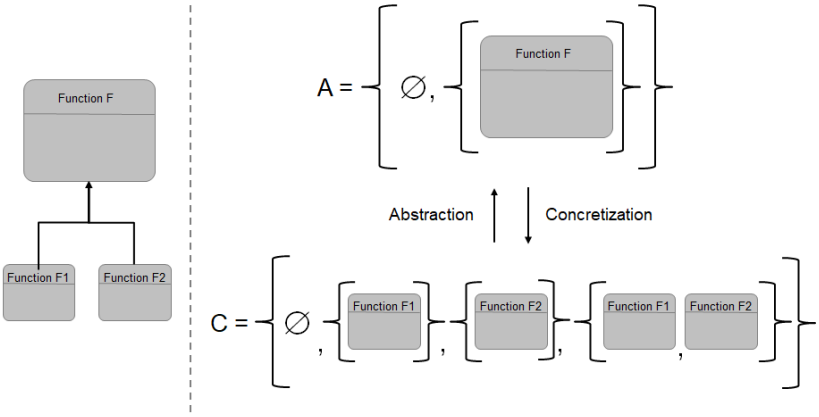


Figure 1 – Example of a simple architectural abstraction & concretization mechanism

Our example showed therefore that system *decomposition* can be interpreted in terms of abstraction in the meaning of Definition 1. Interpreting F1 and F2 in the previous example as functions that are just partially contributing to F illustrates then that system *aggregation* (in the SysML meaning) can also be interpreted in the same way as a form of abstraction, as formalized by Definition 1. More generally, the same formalism can be employed in order to capture *generalization* and *instantiation* of systems: for instance, the abstraction / concretization mechanism introduced in the above example also works if we are considering a generic function F, such as "provide energy", and two instances F1 and F2 of F, such as "provide electricity" and "provide fuel".

It is now an easy mathematical exercise to give a formal meaning to the notion of *system abstraction* that we will use widely through this paper. We shall indeed say that a system S abstracts a system R as soon as one can find a series of abstraction / concretization functions that connect the architectural items (use case, function, component) of S and R, but also the associated operational, functional and structural flows. We refer to the next sub-section for a corresponding illustrating example.

2.3 – Families of systems

Reference architectures, as introduced in this paper, are associated with *families of systems*, that is to say sets of similar systems, sharing a number of operational, functional or structural commonalities. One can indeed architect such families of systems (cf. [26]), based on a generic system that abstracts – in the meaning of Definition 1 – all the systems of a given family.

To illustrate this last notion, let us consider a simple family of systems F = { S1, …, S8 }, consisting in a series of eight soft drinks, as shown in Figure 2. Each system S_i of the family F consists here of the same cork C and the same glass bottle B, but of a specific liquid L_i and of a specific colored label LB_i. One can

then abstract – in the meaning of section 2.2 – this family of systems by introducing a generic abstract system S – that has no actual existence, but is key for dealing with a single object, instead of a series of different systems – consisting of the cork C, the glass bottle B, a generic liquid L and a generic colored label LB. The associated abstraction function α between F and S consists then to map C onto C, B onto B, each L_i onto L and each LB_i onto LB, when the reverse concretization function χ consists in mapping C onto C, B onto B, L onto { L_1, … , L_8 } and LB onto { LB_1, … , LB_8 }. Abstraction, in the meaning of abstract interpretation [9], as defined in Definition 1, allows therefore to easily formalize what is a family of systems considered from a systems architecture perspective.



Figure 2 – Example of a family of systems consisting here of a series of soft drinks

3 – General Definition of a Reference Architecture

3.1 – What May Be a Reference Architecture?

Based on the CESAM systems architecting framework and the notions of abstraction & generic system as introduced in section 2, a reference architecture can now be defined as follows.

Definition 2 – Reference architecture (general definition). Let F be a generic system that abstracts a set R of concrete systems. A *reference architecture* associated with F – or equivalently with R – is then provided by the following items:

- a set A of systems architecture views that cover the operational, functional and structural visions of the generic system F,
- an instantiation mechanism that allows to derive from A the concrete operational, functional or structural architectures C(r) of any system r of R.

Note that this new definition deals with abstraction in the meaning of Definition 1: the underlying abstraction function is then here just the function that maps the concrete operational, functional and structural architectures of a system r of R onto A, when the dual concretization function is nothing else than the instantiation mechanism requested by Definition 2, which can therefore be formally defined as any mapping from A to C(r) that respects the axiomatics of Definition 1.

The first key idea captured by Definition 2 is that a reference architecture intends fundamentally to be a representation of the invariant elements / aspects (whether operational, functional or structural) of a given family R of concrete systems. One can then generate variants – either the existing ones, as synthesized by R, or new ones – by making variability choices within these invariants. The second key idea of Definition 2 is that it supports a mixed top down / bottom up approach for dealing with variant systems engineering within the scope of a certain family of similar systems.

The generic structure of such a reference architecture is illustrated in Figure 3 on an automotive example. We can see that the corresponding reference architecture is made of generic operational, functional and structural views which must abstract an instantiation perimeter which aims to be as

wide as possible: in our automotive case, the instantiation perimeter is indeed formed for instance by crossing, on the one hand, many types of vehicles (low cost, thermic, electric, hybrid, autonomous, family, etc.) and, on the other hand, many car manufacturers, knowing that their vehicles are not based on the same architectural choices & assumptions at all system architectural levels.

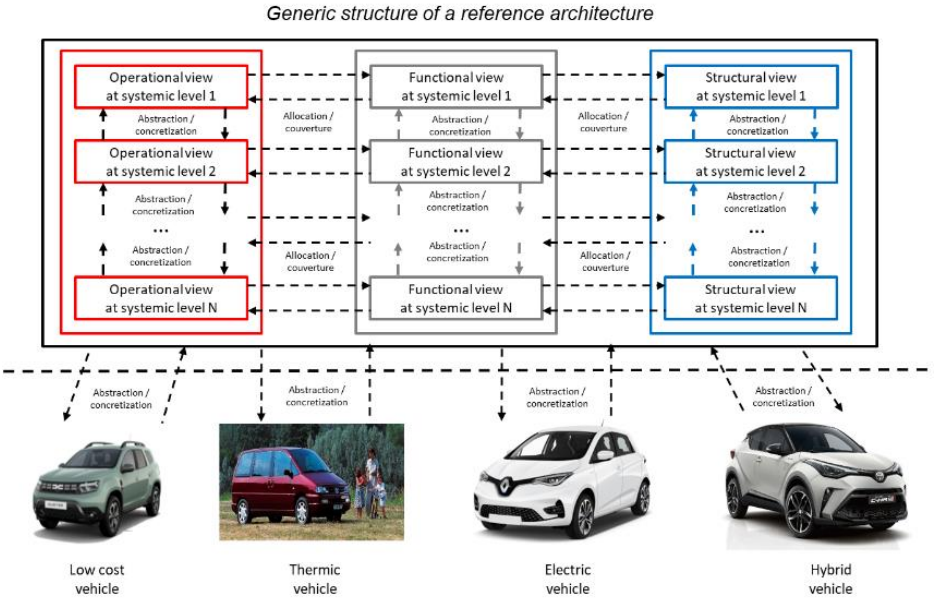


Figure 3 – Definition of a reference architecture illustrated in an automotive context

To be more specific, Figure 4 provides an example of a generic functional systems architecture, again for an automotive context, in the line of the seminal work of Ubrich et alli, who proposed in [38] a reference functional systems architecture for the software kernel of an automated vehicle (that corresponds to the central "manage strategic decisions" function in the functional architecture which is presented here). This reference functional architecture (cf. [22] for more details) is organized in the following functional layers, each of them grouping structurally independent functions:

- **Layer 0 – Protect during crash:** this functional layer is reduced to a single high-level function that protects passengers during crash, which is a safety function which reflects at functional level the passive safety mechanisms that are implemented inside a vehicle.
- **Layer 1 – Manage energies:** this functional layer groups all the functions that manage the vehicle energies, i.e. fuel, electricity, vacuum, torque and movement. One can decompose it into two main functions, which respectively manage energy production (e.g. provide torque) and transformation of energy in movement (e.g. generate lateral movement).
- **Layer 2 – Control extended vehicle:** this functional layer groups all the functions that control the vehicle motion and trajectory. One can decompose it into three main functions which respectively manage the sensing of the environment (e.g. measure vehicle speed), take the strategic decisions (e.g. decide braking) and act on the environment (e.g. brake). Note that these functions are key for autonomous driving and connected vehicle management.
- **Layer 3 – Manage user interfaces**: this functional layer groups all the functions that manage the different user interfaces that are offering value-added services to the vehicle users. One can decompose it into three main functions that cover respectively the management of the mechanical user interfaces (e.g. open / close the vehicle), the visual user interfaces (e.g. provide vehicle status) and the comfort user interfaces (e.g. provide heat).

Note that the construction of this example was not easy. We indeed needed to find and guarantee the associated correct functional decoupling (segregation of functions by coherent functional perimeters and according to criticality, scalability, etc.), to identify and verify the consistency of the structuring functional chains, to guarantee traceability with the operational and structural views of the same level, etc., while ensuring at the same time the satisfaction of the key objective of a reference view, namely its ability to abstract a very large scope of applications within a given industrial domain.

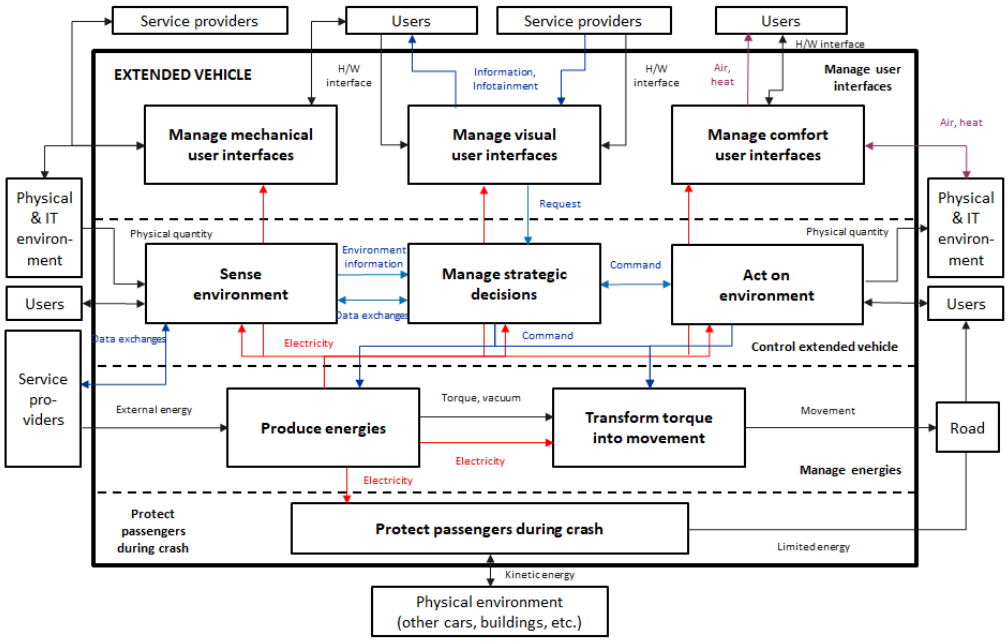


Figure 4 – Example of a reference functional systems architecture of level 1 in an automotive context

As shown by Figure 3, a reference architecture appears as a complex multi-dimensional object. Each of the architectural views that it contains indeed involves many elementary objects, ranging from ten for high-level views to several hundreds, or even thousands, for low-level views. Moreover, one has to connect these views between them, both within a given architectural vision (vertical axis in Figure 3), but also between architectural visions (horizontal axis in Figure 3), using abstraction / concretization relations in the meaning of Definition 1, which generates interconnection matrices that can contain tens of thousands, or even many millions for low-level architecture, of relationships that one needs to manipulate when designing a reference architecture in a given field.

But this is only one aspect of the difficulty: the reference architecture must also be successfully instantiated to all relevant systems within a given application context, typically low-cost, thermic, hybrid and electrical vehicles from a large number of car manufacturers in our automotive example. This ability to abstract many concrete architectures – which is the reversed facet of the difficulty that we just mentioned – is moreover the major criterion of success that we used to validate the relevance of a given reference architecture in practice.

As one can see designing a reference architecture at system level is never an easy task due to the intrinsic combinatorial complexity of such an architecture and the many internal consistency issues that must be continuously overcome during its design, on which is also grafted a pragmatic problem of validation, the reference architecture making sense only by its ability to be instantiated, a property that shall also be verified, which requires confronting it with many concrete architectures.

Finally, it should be noted that the problem of building a reference architecture includes a fundamental unknown: it is indeed not possible to know a priori what is the level of systemic depth – denoted N in

Figure 3 – at which one shall descend in the systemic hierarchy of the family of systems which is abstracted by the reference architecture. One of the problems that arises when building a reference architecture is thus to identify the value of N, which has moreover a real scientific value since it measures the maximum level of abstraction covered by a systemic approach in a given context.

Note that Definition 2 works for most of the application cases on which we could check it, such as aircrafts, automotive electrical & electronic systems, railways or smartphones (see for instance [22] for the aircraft, railway and smartphone cases; a part of a smartphone reference architecture is also provided here in Figure 5). As a matter of fact, it appeared that the maximal level of abstraction on which we could maintain a reference architecture for all these examples was 3: this last figure seems to be a "magic" number in this matter, at least for medium or mass production systems, which is the common characteristic of all these examples (we will indeed see in the forthcoming section 4 that this result does not anymore work for other types of systems with more diversity).

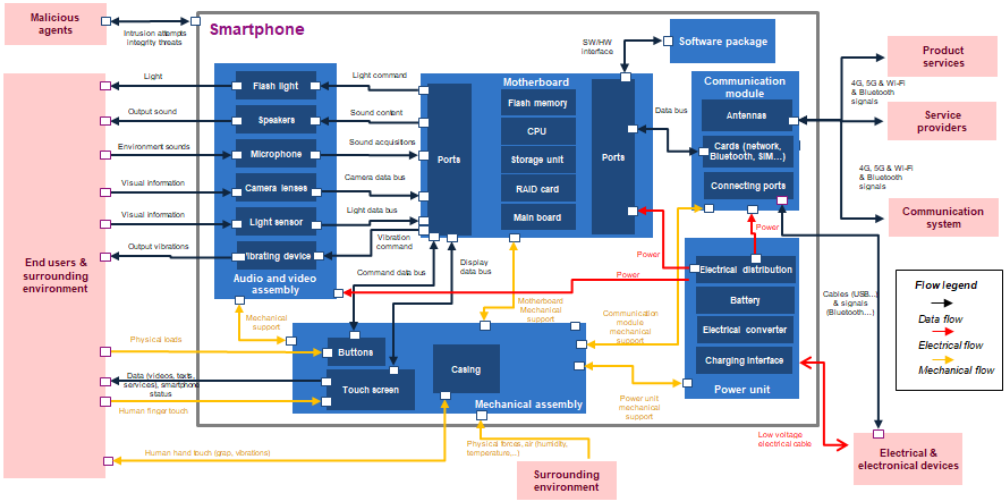


Figure 5 – Reference structural systems architecture of level 2 for a smartphone

3.2 – Case Study: Automotive Reference Architecture

In order to illustrate both what is a reference architecture and how to construct it, we will dig in the automotive case study that we initiated here above, by providing a complete reference architecture for a generic automobile vehicle family of systems. The chosen systemic perimeter is the extended vehicle, in the meaning of the ISO 20077 standard, in order to be as covering as possible. It will be recalled in this regard that this perimeter covers both the vehicle itself and all the off-board information systems with which the vehicle communicates, in particular to ensure the evolutionary and corrective maintenance of the on-board systems that a vehicle contains.

Our work is based on freely accessible public data in the literature that we first had to collect and then analyze. The data collection phase was complicated because the public information is dispersed among a large number of sources, which are generally not consolidated. Moreover, a first observation could quickly be done: many consolidated public information can serve as entry points for abstracting the use cases and the components of a vehicle, which allows to get reference operational and structural architectures, but this is not the case for functions. The only information that we had in this last matter was the high-level functional architecture of an extended vehicle, presented in Figure 4. We therefore had to deal separately with functional architecture.

For operational and structural architecture, i.e. for use cases and logical components, we had to adopt different methods, depending on the nature of the captured information, as described below:

- For use cases, our starting point was the thesis of S.M. Zoepf (MIT), who already proceeded to a first consolidation of information at this level [41] and also provided many bibliographical references on this subject that we analyzed. As this work was already a bit old, we also studied the commercial documentation of several car manufacturers (i.e. Mercedes, Nissan, PSA, Renault, Tesla) to identify the commercial use cases that they are proposing to their customers, knowing that these use cases are just the face visible to the customer of a vehicle architecture, from a system perspective. By consolidating all of this information, we were able to arrive at a first "raw" list of automotive use cases that we could consider as reasonably exhaustive.
- For components, the starting point was the more or less well-organized lists of components that can be found in the literature (see for example [39]). We supplemented these lists with lists from the car maintenance websites of the major car manufacturers that we mentioned above, which allowed us to obtain a first "raw" list of car logical components that we could again consider as reasonably exhaustive.

This first phase of capture thus made it possible to obtain "raw" lists of automotive use cases and logical components which served as a starting point for our abstraction work. But it also broke the initial naïve plan that we imagined, due to the lack of functional information that we discovered. We will see later how we managed to overcome this initially unforeseen, but real, difficulty.

In a second step, we then carried out a meticulous work of clustering and abstraction of these data to bring out their genericity. This work was first done for the operational and structural visions. At this stage, what became apparent, was that the reference architecture did not have to go lower than three levels of abstraction because the diversity of the implementation solutions emerged at the fourth level of abstraction (e.g. a crash protection system has a natural place in a generic reference architecture, but as soon as we want to break it down more finely, one comes across objects which depend on the implementation and are therefore no longer generic). The resulting reference operational & structural architectures – obtained up to level 3 – are provided in Figure 6 and Figure 7.

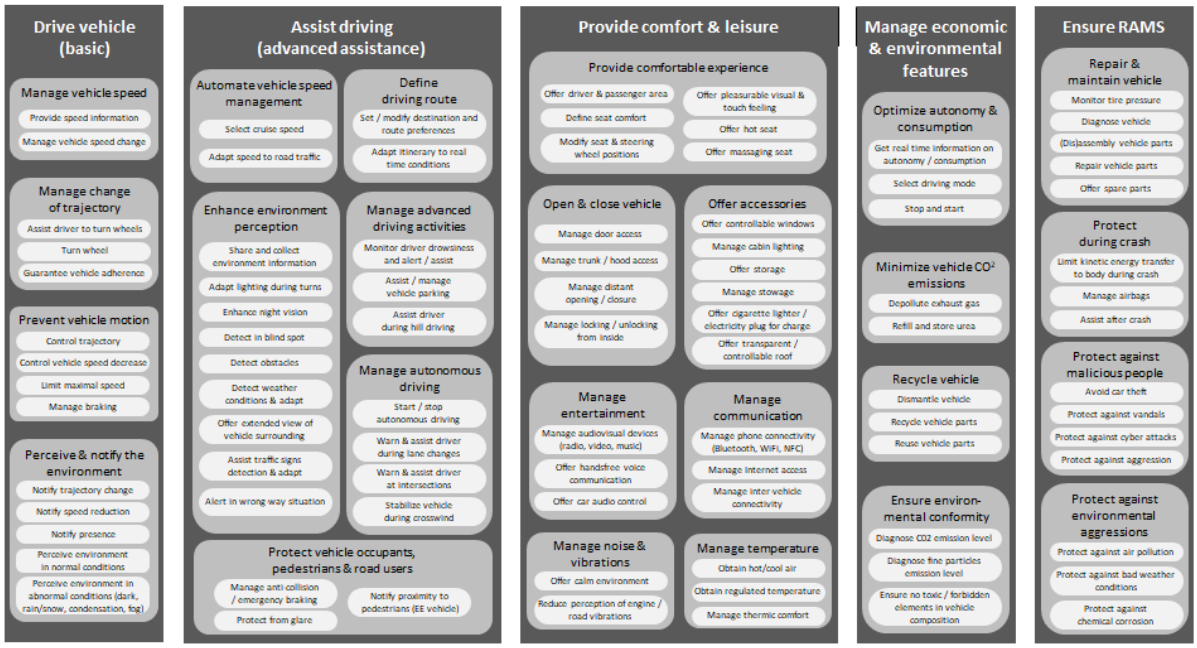


Figure 6 – Reference operational architecture up to level 3 for an extended vehicle

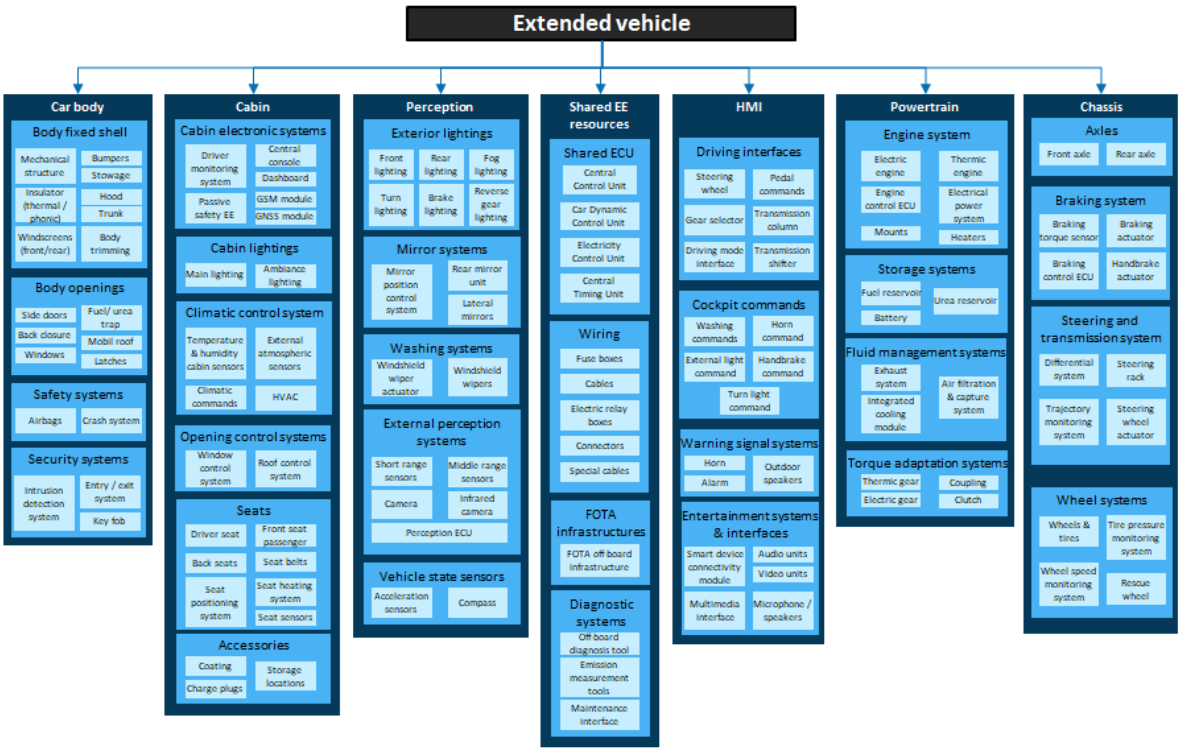


Figure 7 – Reference structural architecture up to level 3 for an extended vehicle

At this stage, we could assume that the target reference architecture, that we were looking for, had the structure with three levels of abstraction which is described in Figure 8, where we colored the systems architecture views for which we had been able to propose a start of organization.

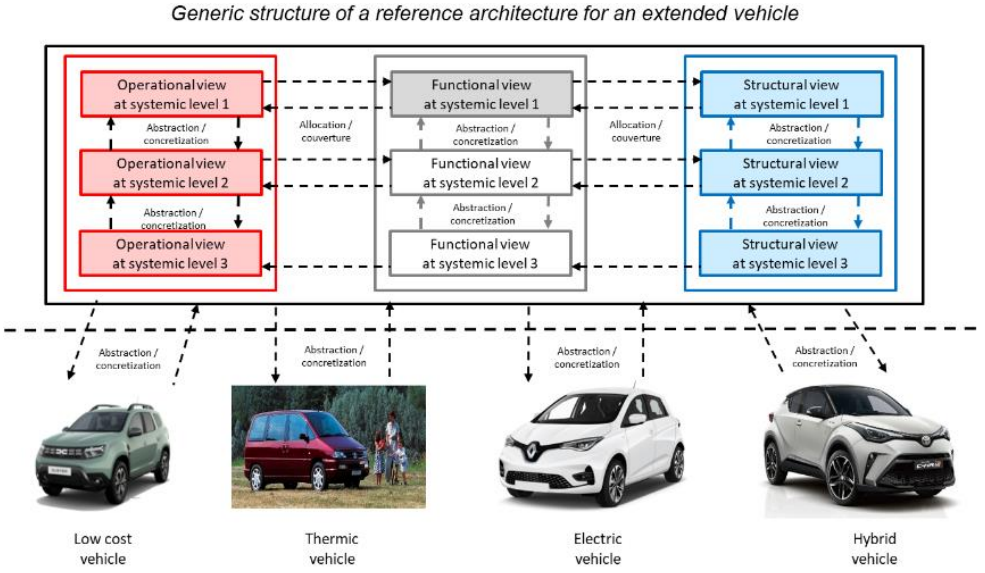


Figure 8 – Intermediate state of our construction of a reference architecture for an extended vehicle

The next phase consisted in "filling the holes" by identifying the level 2 (resp. level 3) functions which make it possible to implement the level 2 (resp. level 3) use cases and which are implemented by level 2 (resp. level 3) logical components, the key tool used here being of course functional analysis. The resulting reference functional architecture is illustrated on Figure 9. Once this work was done, we have a first version of the target reference architecture, but given its mode of construction, we could only

be sure of one thing, namely that it was not correct and likely contained many inconsistencies. But it still had the merit of existing and was thus used as a starting point for a next stage of internal consistency checks, which consisted of identifying all inconsistency issues and gaps in our architectural analyses, and completing them, while keeping a holistic vision, the important thing being of course the overall consistency of the reference architecture that emerged from this process.

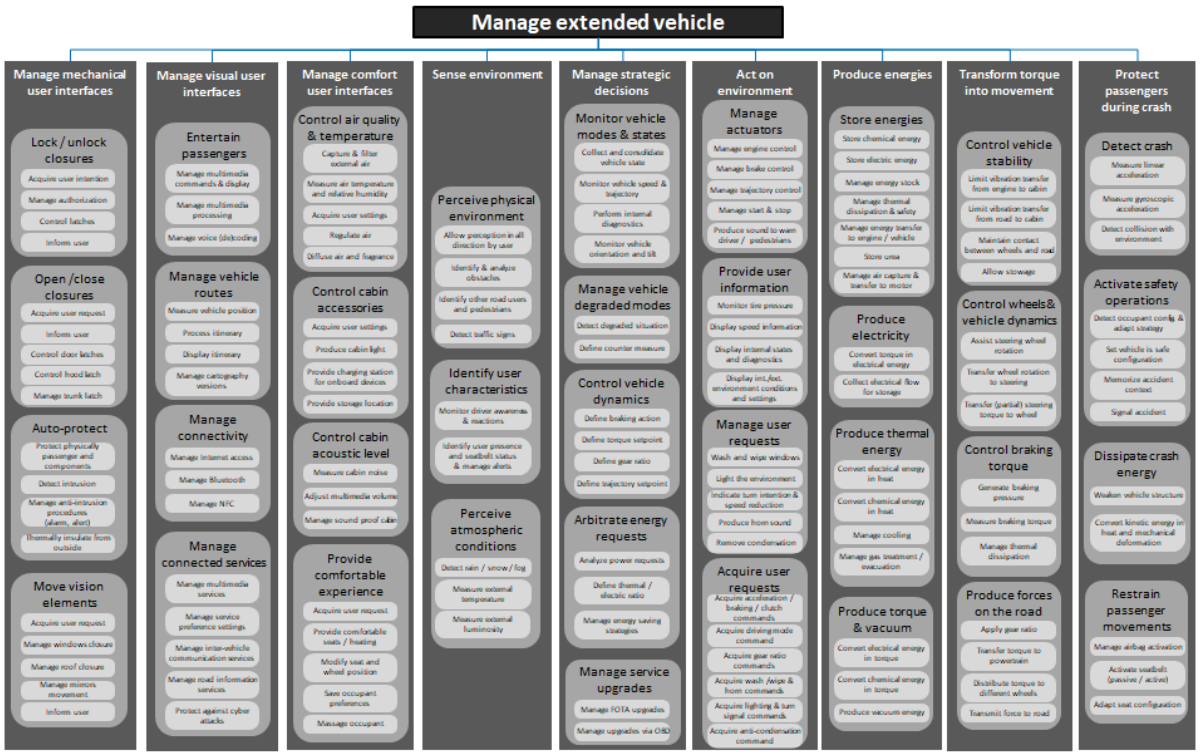


Figure 9 – Reference functional architecture up to level 3 for an extended vehicle

The following phase of our work consisted therefore in carrying out numerous cross-analyses to verify the solidity of the first architectural views that we brought to light and of their interrelationships. This work is difficult to formalize because there is nothing linear about it: it is made up of a multitude of small checks, where one must ensure compliance with the abstraction/concretization relationships between the different architecture views built and often going to retrieve information on this or that point and, at the end, arrive at a first version of a coherent overall reference architecture.

Once such a first version of potential reference architecture was obtained, we then moved on to a more advanced analysis phase where the key tool is the traceability matrix: the idea is, now that we passed the where-nothing-is-stable phase, to verify one by one the relationships existing between the different objects contained in the architectural views. This requires the construction of traceability matrices between, on the one hand, all use cases and all level 1-2-3 functions and, on the other hand, between all functions and all level 1-2-3 logical components, which are both 150 x 150 arrays in terms of order of magnitude (see Figure 10). This traceability analysis raises the question of the detailed relationships existing between the different objects of our reference architecture and makes it possible to identify the last errors and correct them by guaranteeing the exhaustiveness of the analysis. Note finally that, since our approach was a bottom-up one, based on existing operational & structural data, the validation criteria mentioned in section 3.1 is of course here fulfilled by construction. At the end of this stage, we thus achieved a reasonably robust reference architecture for an extended vehicle, as summarized in Figure 6, Figure 7 and Figure 9.

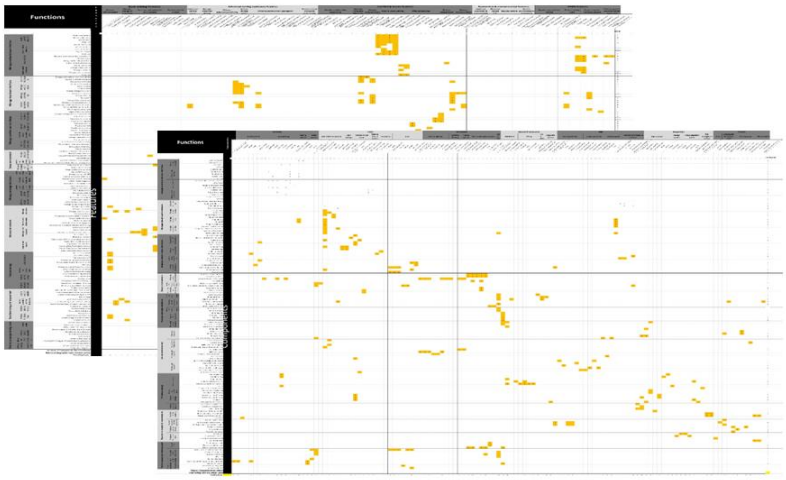


Figure 10 – Use cases to functions and functions to components traceability matrices[3]

4 – How to Deal with Small-Scale Systems?

4.1 – Case Studies: Industrial Reference Architectures

When trying to find the "right" definition of a reference architecture, we however faced a number of cases in which Definition 2 did not work. All these "bad" cases were dealing with families of systems of small scales, that is to say systems which only exist in small amounts, with moreover only a few available data and a lot of individual differences between each system of the considered families, which reduces the ability of achieving proper architectural abstractions at system level. Despite this issue, we were however able to create reference architectures, as we will see, in this other situation.

Let us now present the small-scale cases that we explored. They were four of them, namely airports, buildings, nuclear plants and mines. Airports and buildings can be classified in the same category, since they both deal with civil engineering systems, when nuclear plants and mines can be analyzed as enterprises, which form another – quite different – category of analysis due to the fact that it requires to deal with enterprise architecture practices (cf. [29]). We will here discuss only one case study in each of these categories, since it appeared that the difficulties are exactly the same – with respect to defining a reference architecture – in each of them.

The first interesting case to discuss is the "building" case. We worked here in partnership with an international civil engineering leader which brought us a large variety of case studies, allowing us to manage an efficient abstraction work and achieve relevant reference architectures. The key problem that we faced here was the quite huge diversity of buildings existing in practice (e.g. individual houses, apartment buildings, factories, stadiums, etc.), relying on very different systems architectures, which prevents having a completely generic systems architecture approach for the construction sector, as it is the case, for example, in the automotive or aeronautical industries, where all vehicle and aircraft systems can be described in the same way at a certain level of abstraction, which then opens up the possibility of constructing generic systemic models, as we saw in section 3.2 for automotive. Given this intrinsic difficulty, we choose to concentrate on a restricted perimeter for our work, i.e. collective housing, because it is already quite wide and allows to properly address the issues related to reference architectures in the field of construction.

[3] The detailed corresponding material is freely available and can be communicated at demand by the authors.

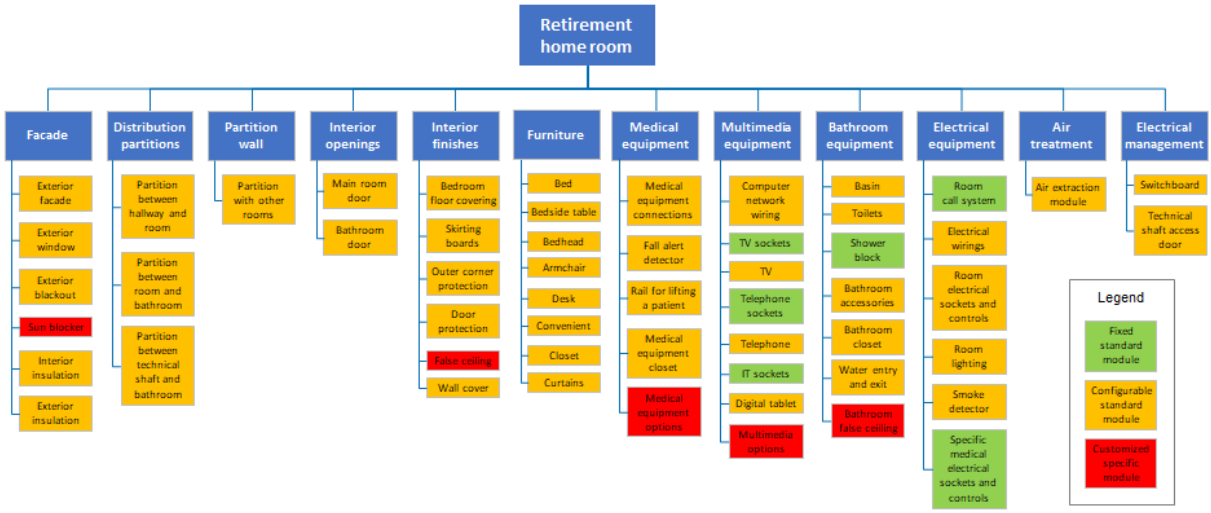


Figure 11 – Reference structural architecture at level 2 for a retirement home room

The last difficulty with which we were confronted in the context of reference architectures for buildings was to take into account the great variability of the logical components of a building, the latter having naturally the objective of meeting as well as possible the needs of the various stakeholders (architects, customers, end-users, etc.). We have thus introduced this diversity into our reference architectures for buildings, by favoring the design of modular reference architectures, made up of modules ranging from completely fixed modules on a whole range of buildings of the same type, to modules that are specifically customized for customer needs, via intermediate standard parametrized & configurable modules. An example of such a modular reference architecture for a retirement home room is provided in Figure 11. Note finally that on most of the case studies on which we worked on building case studies, we were not able – contrarily to the large-scale cases mentioned in section 2 – to go further than the second systemic level within the corresponding reference architectures.

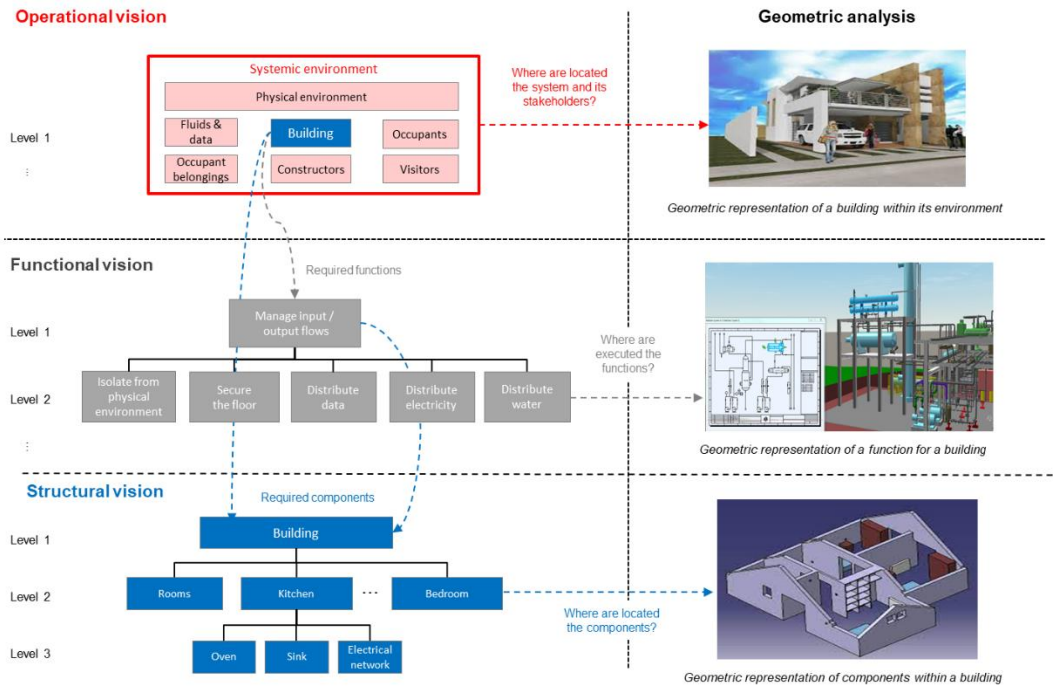


Figure 12 – How to connect geometry within a systems architecture approach

Last, but not least, the importance of geometric modeling for buildings naturally led us to work on the extension of our systems architecture framework (cf. section 2.1 and [22]) in order to include in it the geometric dimension. At this level, we had to resolve a crucial semantic problem: is geometry a new systemic vision in its own, or just a new analysis axis, transverse to the systemic visions? The second solution appeared to be the right one, since each usual systemic vision can have its own geometric representations. We can indeed represent geometrically each operational, functional or structural view of a system, which rather gives geometry the status of a specific attribute of each architectural vision and not of a new systemic vision, as illustrated in Figure 12. As a consequence, it seems therefore difficult to extend the notion of reference architecture to geometrical architecture.

The second case that we shall discuss is the "mine" case. This new case study made it possible to open up a new dimension, namely that of enterprise architecture (cf. [29]). A mine – or more precisely a mining company – has the particularity of being, above all, an organizational system coupled with a technical system, where enterprise architecture extends / replaces systems architecture.

We worked here within a partnership with two international mining companies – a diamond mine and an iron mine on two different continents – through regular workshops with each of them, consisting in analyzing the scientific & technical literature that we captured during the numerous interactions that we had with them, in order to release relevant systems architectural views of their respective mines on various perspectives. The two mining examples that we had the chance to deal with, helped us to abstract the information we collected and achieve a good reference architecture for a mine, as partially illustrated in Figure 13. Note that again, it was not possible to go further than the second systemic level if one wants to maintain genericity.

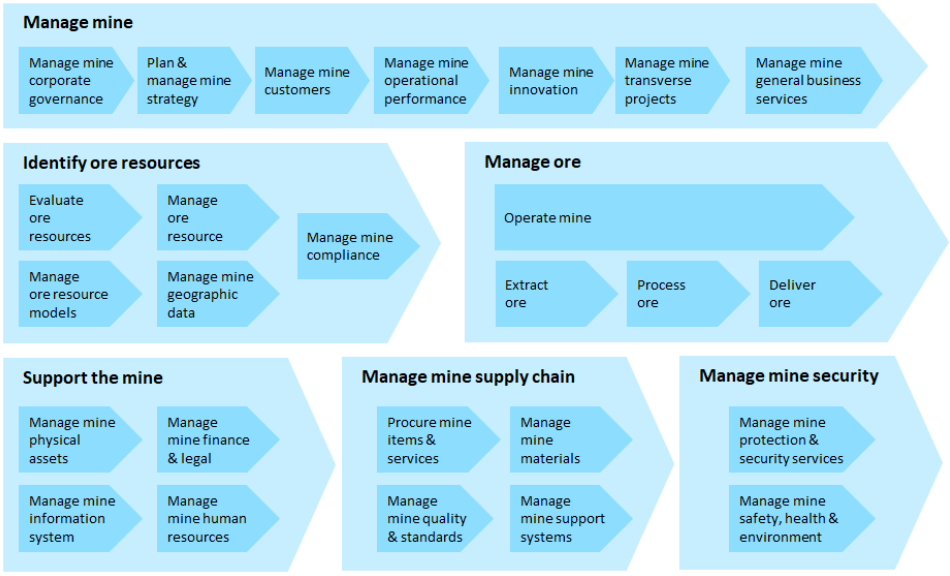


Figure 13 – Reference business systems architecture at level 2 for a mine

In this matter, we were also led to extend our systems architecture framework (cf. section 2.1 and [22]) in order to take into account the two classic dimensions of an enterprise architecture, i.e. the organizational and IT dimensions on which rely any modern organization. Figure 14 summarizes how our generic system analysis framework should be complemented and adapted in order to capture the most important specific features of enterprise architecture.

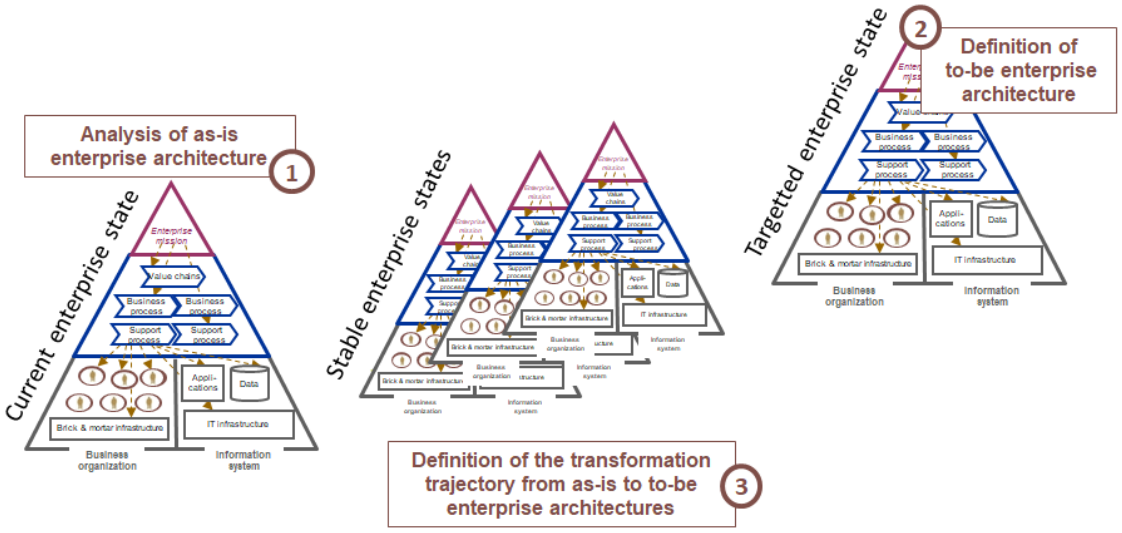


Figure 14 – Overview of our enterprise architecture framework

The main evolutions of our systems architecture framework (cf. [22]) consisted in segregating the enterprise systems architecture views into two coupled sets, one dedicated to the organizational subsystem of an enterprise and another one dedicated to its information subsystem, which led us to segregate, in the enterprise case, all system concepts – use cases, functions and components – into organizational and information system concepts: business processes are then for instance nothing else than the functions of the organization, which explains why Figure 13, which presents the reference business architecture of a mine, can be seen as a reference architecture, here of functional nature for the organizational subsystem of a mine. Another key point that we shall mention is the introduction of the notion of transformation trajectory to capture the deformable nature of organizational systems, which naturally leads to introduce the concepts of transformation state and transformation requirement. The overview of our enterprise architecture framework is illustrated on Figure 14. As explained above, this new framework is nothing else than an instantiation & adaptation of our general systems architecture framework, as presented in section 2.1, to an enterprise system.

4.2 – Another Concept of Reference Architecture

The analysis of the various case studies mentioned in section 4.1 showed that Definition 2 did not work in these contexts and obliged us to propose a new conceptualization, as presented in Definition 3. The main issues that we indeed faced here, were, either pragmatic considerations (lack of data to construct relevant generic systems architectures), or much deeper reasons (difficulties of abstraction and/or concretization). This new restricted definition for a reference architecture is based on the notion of logical architecture, which shall be seen as a synthesis between functional and structural architectures. Using this new definition allowed then us to take into account all the small-scale case studies that we managed, which validates it in practice.

Definition 3 – Reference architecture (restricted definition for small-scale families). Let F be a generic system that abstracts a set R of concrete systems. A *reference architecture* associated with F – or equivalently with R – is then provided by the following items:

- a set L of logical systems architecture views of the generic system F in full generality,
- an instantiation mechanism – in the same meaning than in Definition 2 – that allows to derive from L the concrete logical architectures of any system of R.

Figure 15 illustrates this new framework with do-it-yourself systems, for which reference architectures in the meaning of Definition 3 can actually be built. Note that the maximal level N of abstractions involved in such a reference architecture, as mentioned in Figure 15, appears in practice to be equal to 2 for small-scale and 3 for large-scale systems, as discussed previously.

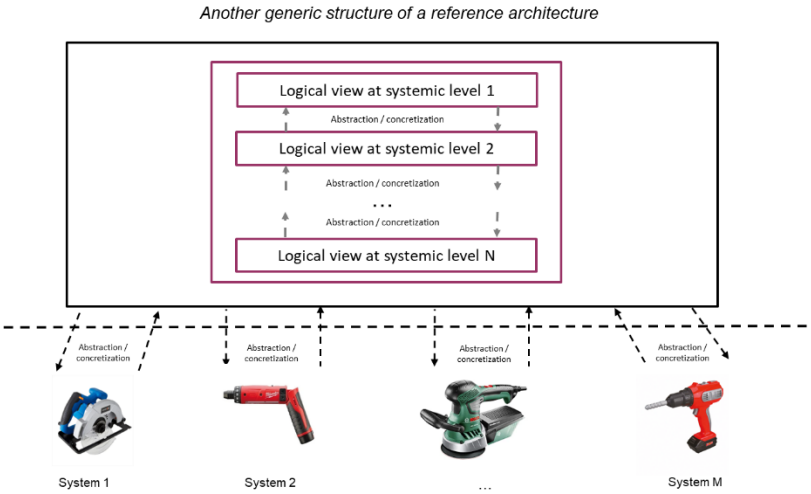


Figure 15 – Another definition of a reference architecture illustrated in a do-it-yourself tool context

5 – Conclusion

In this paper, we presented proposals for clear semantics of a reference architecture, since such systems architectures are precious assets for any systems engineer and architect. The proposed general definition appeared to work on a quite important number of concrete large-scale examples, so we do believe that it is reasonably sound. However, only experience can give us a certain level of confidence in such a definition, which is the main lesson of the different case studies that are discussed in this paper. Our work shall therefore be seen as a contribution that has still to be checked.

Another interesting byproduct is the emergence of the "magic" number 3 in the context of reference architectures. In all examples of large-scale products that we were able to address, it indeed seems that 3 is the maximal level of abstraction where one can establish a reference architecture, which would be consistent with the famous "7x7x7" cognitive rule, i.e. a system model can only be understood by an human being if it contains at most 3 systemic levels and 7 elements per level (see [22] for more details). For the small-scale systems, diversity reduces this limit to 2 in practice. This information is of course of pure pragmatic nature, but it may be helpful in practice. However, one may ask whether it is not a deep fundamental property of systems which is highlighted in this way.

6 – References

[1] Batteux M., Prosvirnova T., Rauzy A., *Model Synchronization: A Formal Framework for the Management of Heterogeneous Models*, Model-Based Safety Assessment, Y. Papadopoulos, K. Aslansefat, P. Katsaros, M. Bozzano, Eds., 157-172, Springer, 2019.

[2] Beale D., McKinney D., Oosthuizen R., Smith G., Watson M.D., *System Engineering Heuristics for Complex Systems*, INCOSE International Symposium 2022, Detroit, 293-312, 2022

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

[4] Buschmann F., Henney K., Schmidt D.C., *Pattern-oriented software architecture: On patterns and pattern languages*, John Wiley and Sons, 2013.

[5] Bussemaker, J. H., Ciampa, P. D., *MBSE in architecture design space exploration,* In Handbook of model-based systems engineering, 1-41. Springer, 2022.

[6] Cloutier R.,1, Muller G., Verma D., Nilchiani R., Hole E., Bone M., *The Concept of Reference Architectures,* Systems Engineering, 13 (1), 14-27, 2009, doi:10.1002/sys.20129

[7] Cook, D., Schindel, W. D., *Utilizing MBSE patterns to accelerate system verification*. Insight, 20 (1), 32-41, 2017.

[8] Corns, S., Gibson, C., *A Model-based Reference Architecture for Medical Device Development*. In INCOSE International Symposium, 22, (1), 2066-2075, 2012.

[9] Cousot P., Cousot R., *Abstract Interpretation,* Symposium on Models of Programming Languages and Computation, ACM Computing Surveys, 28 (2), 324-328, 1996.

[10] de Weck O., *Strategic Engineering – Designing systems for an uncertain future*, MIT, 2006.

[11] Doornbos R., Marincic J., Vasenev A., Wesselius J., *Distilling Reference Architectures in the High-tech Equipment Industry, Insight*, INCOSE, 2023

[12] INCOSE, *Architecture working group*, https://www.incose.org/communities/working-groups-initiatives/architecture

[13] INCOSE, *Systems Engineering Handbook, A guide for system life cycle processes and activities*, INCOSE, 5th edition, 2023.

[14] ISO, *ISO 15704 - Enterprise modelling and architecture*, 2019

[15] ISO, *ISO/IEC/IEEE 42010 - Software, systems and enterprise – Architecture description*, 2022

[16] ISO, *ISO/IEC/IEEE 42020 - Software, systems and enterprise – Architecting processes*, 2019

[17] ISO, *ISO/IEC/IEEE 42024 - Software, systems and enterprise – Architecture fundamentals*, Under development

[18] ISO, *ISO/IEC/IEEE 42030 – Software, systems and enterprise – Architecture evaluation framework*, 2019

[19] ISO, *ISO/IEC/IEEE 42042 – Software, systems and enterprise – Reference architecture*, Under development

[20] Jackson, M., Wilkerson, M., Castet, J. F., *Exposing hidden parts of the SE process: MBSE patterns and tools for tracking and traceability*, In 2016 IEEE Aerospace Conference, 1-12, IEEE, 2016.

[21] Kelly, M. S., Jacques, D., Ayres, B., Cobb, R., Ford, T. *Using a CubeSat Reference Architecture for Accelerated Model Development and Analysis*, J. of Small Satellites, 10 (3), 1097-1108, 2021.

[22] Krob D., *Model-Based Systems Architecting – Using CESAM to Architect Complex Systems*, ISTE, Wiley, 2022.

[23] Kumar A., Anand E., Natarajan S., Dandekar A., *Architecture Analysis Methods*, Systems Engineering, 31 (1), 1377-1392, 2021

[24] Kumar A., Lokku D.S., Zope N., Natarajan S., *Value based System Architecting: Illustrated by Designing a Task Automation System*, INCOSE International Symposium, 26: 2211–2225, 2016

[25] Mahut F., *Intégration des approches PLM et SLM pour le développement et la gestion des Systèmes Produit-Service en contexte automobile : proposition méthodologique*, Thèse de doctorat, Université de Compiègne, 2019.

[26] Maier M.W., *Architecting a portfolio of systems*, Systems Engineering, 22 (4), 335-347, 2019.

[27] Maier M.W., Rechtin E., *The art of systems architecting*, CRC Press, 3d edition, 2009.

[28] Martin J.N., *Architecture Frameworks & Modeling*, INCOSE International Symposium 2011,

Denver, 21 (1), 1390-1390, 2014

[29] Martin J.N., *An Enterprise Systems Engineering Framework*, INCOSE International Symposium 2010, Chicago, 20 (1), 243-264, 2014

[30] Mordecai Y., Fairbanks J.P., Crawley E.F., *Category-Theoretic Formulation of the Model-Based Systems Architecting Cognitive-Computational Cycle*, Appl. Sci., 11 (4), 1945, 30 pages, 2021

[31] Mueen Mulla F., Kulkarni V.N., Gaitonde V. N., Kottursheta B. B., *PLM as a tool for collaboration in aerospace industries – A review*, AIP Conf. Proc., 2316 (1), 2021

[32] Muller G., *A Reference Architecture Primer*, University of South-Eastern Norway, Sept. 2020 - https://www.gaudisite.nl/ReferenceArchitecturePrimerPaper.pdf

[33] OMG, *Unified Architecture Framework® (UAF®)*, https://www.omg.org/uaf/

[34] Penalva J.M., *SAGACE method: the modelling of human designed systems*, COMETT'93, Rome, 1993

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

[36] SEBoK, System Architecture, https://sebokwiki.org/wiki/System_Architecture

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

[38] Ubrich S., Reschka A., Rieken J., Ernst S., Bagschik G., Gierkes F., Nolte M., Maurer M., *Towards a Functional System Architecture for Automated Vehicles,* ArXiv, 2017.

[39] Wikipedia, *List of auto parts*, https://en.wikipedia.org/wiki/List_of_auto_parts.

[40] Wu, Q., Gouyon, D., Levrat, E., Boudau, S., *A review of know-how reuse with patterns in model-based systems engineering*, In "Complex Systems Design & Management" (CSD&M 2018), E. Bonjour, D. Krob, L. Palladino, F. Stephan Eds., 219-229, Springer, 2018.

[41] Zoepf S.M., *Automotive features: mass impact and deployment characterization*, MIT, 2011.