An Advanced Engineering Framework experimented on a R AE Electric Vehicle case

F Colet, S Chabroux, J. Matta

An Advanced Engineering Framework experimented on a R&AE Electric Vehicle case

F. Colet (Renault), S. Chabroux, J. Matta (Knowledge Inside)

Abstract

This article describes modeling activity experimented on an Advanced engineering case of Zero Emission Vehicles at Renault. A key advantage of our approach is that system architecture and requirements management at all the stages of the system life cycle are managed in a unique data model and unique database. It reviews conceptualization and production process of a complex system. It presents a spectrum of activity modeling techniques, ranging from a widely used systems engineering diagram, to continuous flow modeling. The techniques include use case definition, requirements elicitation, system architecture definition and finally Electric and Electronic architecture. The article also describes refinements of modeling activity using arKItect© tool.

Keywords: system modeling, project process, use case, requirement and architecture modeling, model in the loop, fault injection

1. Introduction

Mastering complex systems design remains difficult. This is due to many issues including language, methods, tools diversity. Indeed, modeling systems for the purpose of setting a description, an explanation or a rational meets the same problems.

In fact, many complex systems descriptions in the industry are spread over many models, formats and documents (consider the 100 000 documents related to a Nuclear Plant according to Anne Lauvergeon – Le Monde 02/09/2009). Resulting heterogeneous system descriptions are aggregated in normative frameworks that overlook many important aspects of project, process and product management.

Normative language constraints may even be a hindrance to modeling properly complex systems. They most likely follow specific objectives and need long learning curves (e.g. UML, SysML or more recently BPMN / BPML).

Conversely, non normative frameworks (e.g. Microsoft Visio) allow a representation of domain and company specific idiosyncrasies whatever they relate to functional, technical or organizational items. However, system integrity checks clearly miss in such frameworks and more generally the ability to manage the links between objects and processes constituting a system.

However, complex systems are characterized by a significant number of relations of different kinds. Their management compels a very important investment during their design (requirement management, risk management, verification & validation …) but also during their whole life cycle.

So there is a need for a common framework that would be sufficiently flexible to be used by a variety of teams and competencies and at the same time sufficiently structured to support different stages of system life cycle, providing a numerical continuity. This has been the main topic of investigation for Samuel Boutin in his team during 10 years in the context of the automobile industry. arKItect©, a software edited by Knowledge Inside, is the result of this work.

This technology is applied today in an advanced engineering project of Zero Emission Vehicles at Renault. In the sequel, we shall call it “the AE-EV project”. The project team has set up a process and system description framework inherited from Renault engineering best practices and also from standard methods, e.g. functional analysis. The main advantage of such an approach resides in the integration of knowledge provided by a variety of teams and competences in a structured manner. This knowledge is extracted from various existing documents (design documents, meeting minutes, test results, analysis …) and also directly entered by arKItect© users.

Following our approach, system architecture description becomes the backbone for the system knowledge and a reference for capitalization, further reuse, and system design and integration. In this paper, we present the implementation and return on experience of this program.

This article will first expose the arKItect© concept before presenting the AE-EV project conceptualization process and associated use cases. Then, our Model Based System Engineering approach will be introduced with requirements and architecture definitions. Finally, we give an example of future improvement on how the system modeling activity can be an input to Electric and Electronic architecture and embedded software architecture, including real-time aspects.

2. arKItect© concept

arKItect© is a tool for defining Diagram Domains Specific Languages (DSL) and using them in different applicative domains. At the time three, major DSL have been successfully defined and used within arKItect©: embedded systems, information systems, infrastructure. The definition of a DSL is established in [6]. A Diagram Domain Specific Language is made up of three layers: D2, D1 and D0 layer.

At the D2 level, the Diagram DSL represents the language for describing any type of diagrams. It is solution - and platform - independent and contains all criteria understandable by a user who wants to specify diagrams. This level is problem-oriented for specifying diagrams.

At the D1 level, a Diagram DSL instance describes a type of diagram. It contains the view model description for producing a type of diagram that is model elements to be displayed with their layout properties. This description respects the language defined by the Diagram DSL. This level contains all data for generating tools producing diagrams.

At the D0 level, we have diagrams expected by end-users in their modeler.

So arKItect© is a framework to develop system development tools and the three levels, D2, D1 and D0 are supported by the framework together with a user administration tool allowing granting a variety of access rights to the different users. arKItect© is funded on an evolution of the object model supporting a different approach to flows representation and handling. We call it a relational object model because flows between objects are first class citizens. The very nice theoretical added value of our object model is that all the DSL are built upon 6 features, in comparison to more than 20 for UML or other derived languages. This allows a very short learning cycle for the tool, no more than one day is necessary to understand the framework and work on a simple model. Of course, the more diagrams are added, the more difficult it is to master a DSL.

A very important feature is that arKItect© is a visual tool. It is commonly said that a picture stands for thousand words. We believe so. arKItect© can communicate with Doors. It is also interfaced with Simulink© from Mathworks for the purpose of generating models for simulation. W.r.t SysML and similar languages based upon the OMG standard, it is possible to define the SysML diagrams in arKItect© although in most of our projects, only a few UML/SysML like diagrams (e.g. activity diagram) may be needed. Conversely, the user can add many diagrams that are not already standardized and that prove to be useful for each project. Building new diagrams in arKIitect© is very easy. Indeed, no diagram is predefined in the tool if you start a model from scratch without reusing a predefined meta-model. Rebuilding UML like diagrams is quite easy, but many new diagrams can be useful, e.g. dynamic architecture diagrams as described below in the paper.

Our experience is that given the application domain and the size of the project, the environment needed for development and the way people will work can change a lot.

Last but not least, the tool is open, a python script is available and can be executed on each object or a diagram. This feature is used to import/export Microsoft Office documents and is typically used for documents generation.

So arKIitect© is a meta language definition system, a diagram definition system and modeling tool. The language is formed of a very compact syntax allowing to specify UML SysML diagrams and many others as needed for DSL design.

3. The AE-EV project process

Many systems engineering standards were produced over the last few decades. These standards have been increasingly harmonized in the past few years and have evolved into more widely accepted international standards. The ISO IEC 15288:2002 standard “Systems Engineering – System Life Cycle Processes” (see figure 1) is the reference standard at Renault for the development and the deployment of systems engineering. A review of the ISO 15288 standard leads to the conclusion that systems engineering is very broadly defined. Most systems engineering handbooks only describe the engineering process(es), while the ISO 15288 describes the processes of the entire system life cycle, including Agreement, Entreprise and Project processes. This standard overlaps with the better known standard ISO 9001:2000 for Quality Management.

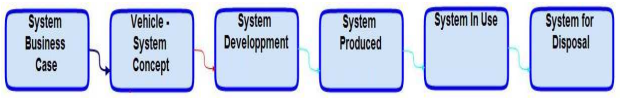

Figure 1: Product life cycle

A major feature of our approach is that the tool allows to design at the same time the product lifecycle and engineering process specifications (Figure 1 to 4) and to link the process steps with the modeling activity for the system.

The Engineering process of system conceptualization step is composed of three subprocesses: requirements elicitation, functional analysis and design see figure 2 below.

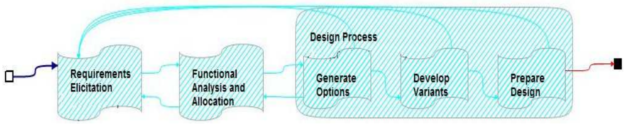

Figure 2: Conceptualization process

The AE-EV project starts off with a problem - Zero Emission Vehicles - that requires a solution. This is one of the stakeholder requirements and at the same time it's already a performance requirement attached to the vehicle function. To be able to properly manage the complexity of the ZE Vehicle, a top-down approach to the engineering process is essential. The conceptualization process is repeated at the different levels of the system structure, till the refinement of the requirements leads to ready for use components or adaptations of such components. This approach is extensively described in the INCOSE guideline [2].

The engineering process whereby the design to be produced is defined, is followed by the production process. During this phase, the system is physically produced and validated.

As for the conceptualization process, the production of complex systems is carried out through the integration of the different system layers. That's why we rather call this step the integration step, see Figure 3. In contrast to the previous step, the integration process is carried out bottom-up. It consists of system or subsystems implementation and validation to determine whether the system or subsystems behave as designed and as required meeting the expected performance requirements.

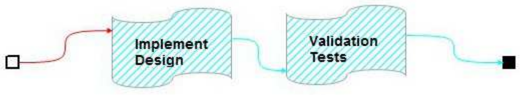

Figure 3: Integration process

Below Figure 4 presents the AE-EV project main step process inherited from Renault engineering best practices and also from standard system engineering approach as described above.

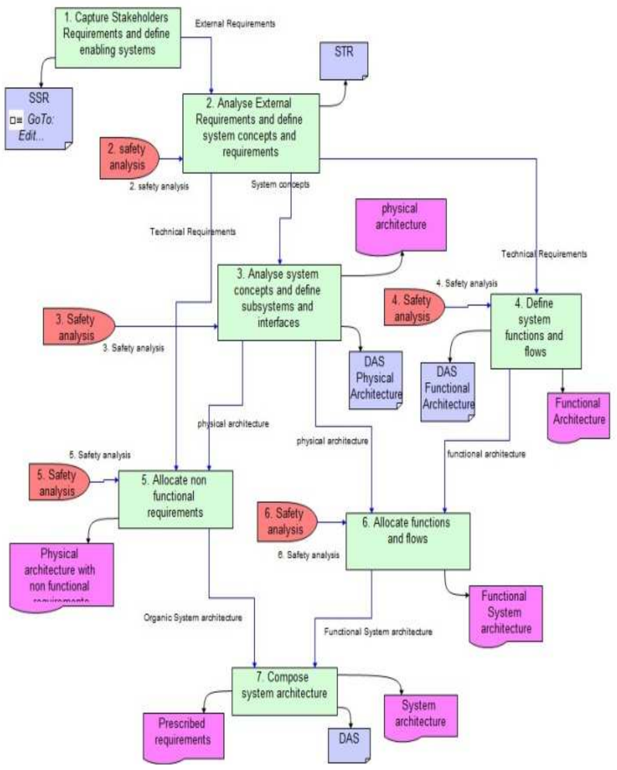
1. Capture Stakeholders Requirements and define enabling systems
External Requirements
STR
SSR
GoTo:
Edit...
2. Analyse External Requirements and define system concepts and requirements
2. safety analysis
2. safety analysis
System concepts
Technical Requirements
physical architecture
Technical Requirements
3. Analyse system concepts and define subsystems and interfaces
4. Safety analysis
4. Safety analysis
4. Define system functions and flows
3. Safety analysis
3. Safety analysis
DAS Physical Architecture
DAS Functional Architecture
Functional Architecture
physical architecture
physical architecture
functional architecture
5. Safety analysis
5. Safety analysis
5. Allocate non functional requirements
6. Safety analysis
6. Allocate functions and flows
6. Safety analysis
Physical architecture with non functional requirements
Functional System architecture
Organic System architecture
Functional System architecture
7. Compose system architecture
Prescribed requirements
DAS
System architecture

Figure 4: System engineering data management process

In this graphical project view, green boxes represent the project main process step. The sequence of those steps is schematized by blue arrows. On some steps, parallel activities (red boxes) come in and allow us to produce artifacts (pink boxes) and to generate associated documents (blue boxes) for example System Technical Requirements (STR) and Data Architecture System (DAS) in Renault project process.

4. Use case

A commonly admitted idea about the design of Electric Vehicles is that it could be seen as just the electrification of a traditional vehicle equipped with a thermal engine. From the outside of the vehicle this is not completely false; the car will have wheels, a body… But the Electric Vehicle has to reconsider one of its life cycle phases from a different point of view: the energy refuelling. Until now this phase was not completely described with the exception of GPL / GNV due to safety issues mainly.

For the energy recharge, the electric vehicle has to comply with several standards like IEC 61851 (under revision) describing the different charge modes, IEC 62196 for the physical interface and new communication requirements.

The main interest of this study of the charge is that it considers the vehicle as a sub system as well as the charge spot and therefore it implies that a main system exists that includes the spot, the vehicle and all the necessary means to fulfil the charge like cable, plugs and communication.

Several use cases have to be taken into account. For each case, the system will not have the same behaviour and the customer will not interact in the same way.

The Use case diagram Fig. 5 provides a high-level description of the usage requirements for the EV system. This diagram is composed of four object types. The grey boxes are subsystems. Each subsystem of the complete system includes use cases (ellipse boxes). Use cases are linked between them by actions (black arrows). The actor (white box) executes first level actions leading to subsequent behavior in other subsystems.

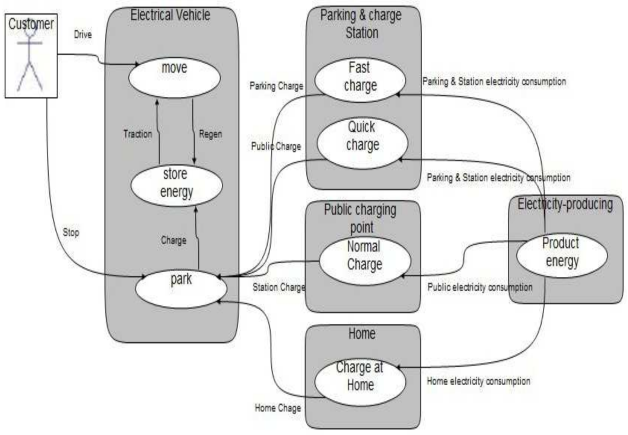

Figure 5: EV use case diagram

For example, in the use case of charge at home, the customer will plug the cable into his car on one hand and into the wall socket on the other hand, and the charge will start at a low power (typically 3 to 6kW) or eventually a clock will manage the start of the charge and its end. But the customer will not stay nearby its car during the whole charging process. At the opposite, the quick A.C. charge at 43kW will mainly be used to partially recharge the battery and provide the autonomy needed by the customer to finish his trip. The customer will for this case only plug the cable which is attached to the spot into the socket of his car and will stay the several minutes requested to fill the battery.

5. System requirement modeling

The objective of the requirements analysis process is to translate the stakeholders' requirements into measurable system requirements and functions. The requirements for the functions of the systems to be designed determine what the system must be able to do and must be functionally specified. At the same time the limitations, such as environmental factors and regulations, are also addressed.

Requirement diagram captures requirements hierarchies and requirements derivation, and the "satisfy" and "verify" relationships allow relating a requirement to a model element that satisfies or verifies the requirements.

Let's consider the specific case of IEC 61851 (Fig. 6) where we have design specification and systems requirements on the charging mode. For example, the Mode 3 charging definition in this standard is: *connection of the EV to the A.C. supply network (mains) utilizing dedicated EVSE where the control pilot function extends to control equipment in the EVSE, permanently connected to the A.C. supply network (mains)*.

Each green box represents a requirement applied to a system or a subsystem (white or grey boxes). The requirement object includes attributes: identifier, description, source, requirements availability, performance,

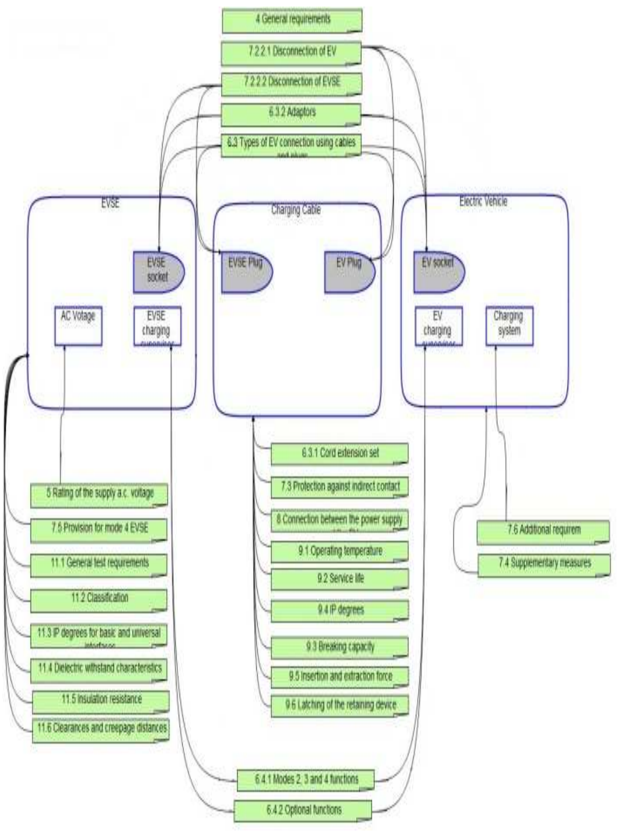
4 General requirements
7.2.2.1 Disconnection of EV
7.2.2.2 Disconnection of EVSE
6.3.2 Adaptors
6.3 Types of EV connection using cables
EVSE
EVSE socket
AC Votage
EVSE charging
Charging Cable
EVSE Plug
EV Plug
Electric Vehicle
EV socket
EV charging
Charging system
5 Rating of the supply a.c. voltage
7.5 Provision for mode 4 EVSE
11.1 General test requirements
11.2 Classification
11.3 IP degrees for basic and universal
11.4 Dielectric withstand characteristics
11.5 Insulation resistance
11.6 Clearances and creepage distances
6.3.1 Cord extension set
7.3 Protection against indirect contact
8 Connection between the power supply
9.1 Operating temperature
9.2 Service life
9.4 IP degrees
9.3 Breaking capacity
9.5 Insertion and extraction force
9.6 Latching of the retaining device
7.6 Additional requirem
7.4 Suppementary measures
6.4.1 Modes 2, 3 and 4 functions
6.4.2 Optional functions

Figure 6: IEC 61851 requirements

In addition to requirements diagram, for complex system study, we may need to define in details some behavior. In some instance, the sequence diagram may be used to show the interactions between objects in the sequential order that those interactions occur. The Fig. 7 illustrates the charging sequence in a station.

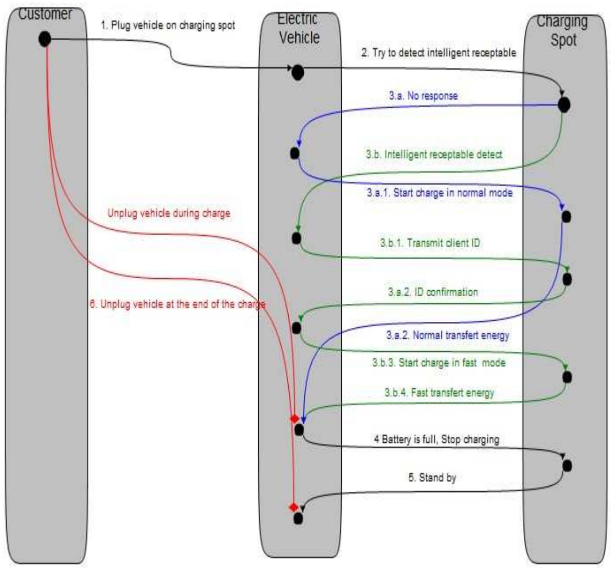
Customer
Electric Vehicle
Charging Spot
1. Plug vehicle on charging spot
2. Try to detect intelligent receptable
3.a. No response
3.b. Intelligent receptable detect
3.a.1. Start charge in normal mode
3.b.1. Transmit client ID
3.a.2. ID confirmation
3.a.2. Normal transfert energy
3.b.3. Start charge in fast mode
3.b.4. Fast transfert energy
4 Battery is full, Stop charging
5. Stand by
Unplug vehicle during charge
6. Unplug vehicle at the end of the charge

Figure 7: Charge sequence diagram

The sequence number three gives an exclusive choice between two sequences referenced “3.a” and “3.b”. arKItect© allows us to characterized sequence flow by various colors. The result is better readability of the two alternatives which converge to the sequence number four “Battery full charge, Stop charging” or interrupt by the sequence “Unplug vehicle during charge”.

arKItect© allows us to generate a compact view of this sequence diagram, Figure 8. This new view, activity diagram, is obtained automatically by collapsing the three package objects (grey boxes). The numbering sequence allows us to keep the sequential order.

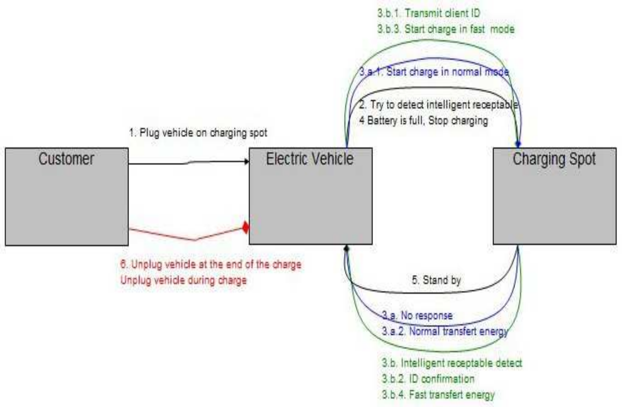
3.b.1. Transmit client ID
3.b.3. Start charge in fast mode
3.a.1. Start charge in normal mode
2. Try to detect intelligent receptable
4 Battery is full, Stop charging
1. Plug vehicle on charging spot
Customer
Electric Vehicle
Charging Spot
6. Unplug vehicle at the end of the charge
Unplug vehicle during charge
5. Stand by
3.a. No response
3.a.2. Normal transfert energy
3.b. Intelligent receptable detect
3.b.2. ID confirmation
3.b.4. Fast transfert energy

Figure 8: Charge activity diagram compact view

6. System architecture modeling

6.1. Functional architecture

The objective of the functional analysis and allocation process is to transform the functions of a system into subsystems. Internal block diagrams provide a simple overview of the internal functionality and signal flow of a device. It allows us to model functional architecture.

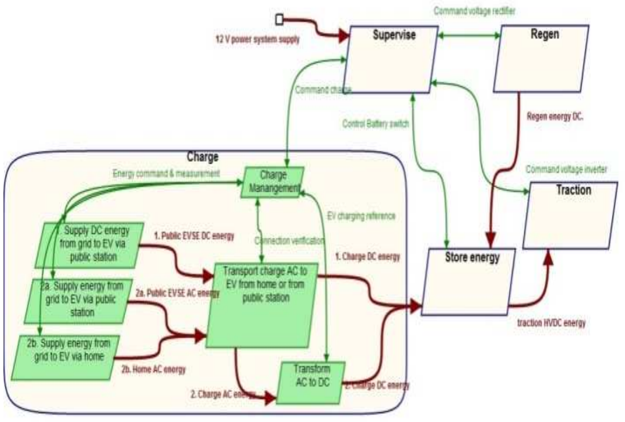

Figure 9: EV Functional architecture

Above, the figure 9 represents the architecture of energy management function. This function is decomposed in five macro functions (white parallelogram boxes). In this view, the macro function “Charge” is expanded to highlight internal subfunctions (green parallelogram boxes). So arKItect© allows us to represent graphically a hierarchical system architecture. The signal flows exchanged between functions and subfunctions are of two types in this example. The red one represents high voltage signal charge and green one represents low voltage signal command.

To conclude on the conceptualization process, we have all system definition data, requirements, use case, and the system architecture, in only one model. We do not have to support links between several tools during this project step.

6.2. System architecture

The last step of conceptualization process is the design. During this phase, the subsystems are actually developed in accordance with the functional analysis. In other words: a solution-independent subsystem is transformed into a physical solution-based subsystem.

In parallel of functional architecture, we use one more time internal block diagram to define an organic architecture.

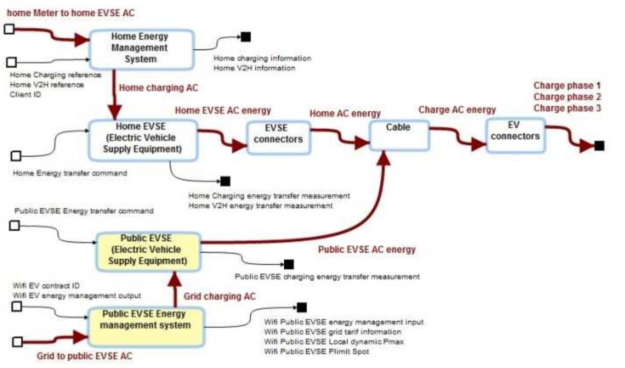
home Meter to home EVSE AC
Home Energy Management System
Home charging information
Home V2H information
Home Charging reference
Home V2H reference
Client ID
Home charging AC
Home EVSE AC energy
Home AC energy
Charge AC energy
Charge phase 1
Charge phase 2
Charge phase 3
Home EVSE (Electric Vehicle Supply Equipment)
EVSE connectors
Cable
EV connectors
Home Energy transfer command
Home Charging energy transfer measurement
Home V2H energy transfer measurement
Public EVSE Energy transfer command
Public EVSE (Electric Vehicle Supply Equipment)
Public EVSE AC energy
Public EVSE charging energy transfer measurement
Wifi EV contract ID
Wifi EV energy management output
Grid charging AC
Public EVSE Energy management system
Wifi Public EVSE energy management input
Wifi Public EVSE grid tarif information
Wifi Public EVSE Local dynamic Pmax
Wifi Public EVSE Plimit Spot
Grid to public EVSE AC

Figure 10: Charging spot organic architecture

The example of charging spot organic architecture Figure 10 has the same properties that EV functional architecture Fig. 9: hierarchical architecture definition and flow characterisation. In this view, boxes schematized system, red arrows represent high voltage flow and black arrows are data flows.

7 Future improvements of the Renault model

7.1. Electric and Electronic (EE) architecture

In the system architecture, we eventually can find components and functions corresponding to software components. However, supporting systems like ECUs, Electrical distribution, data bus don't need to be specified at this stage as they do not participate to the functional objective of the system.

So the next step in system design is the definition of the EE architectures and 3D integration. EE architecture includes ECUs definition, networks definition (including electrical distribution). The ECU definition includes allocation of signal conditioning and of software (Autosar) components.

This step description is based on other experiences than the EV project. It corresponds to further works that can be addressed once the system analysis will be completed.

In the next sections, we will use vehicle air conditioning system to illustrate this modeling phase.

The figure 11 represents the four ECUs (red border boxes) use for a vehicle air conditioning system. They make low voltage sensor signals acquisition and controls low voltage actuators signals. Those signals are representing by red arrows. ECUs communicate between them by network frames (yellow arrows).

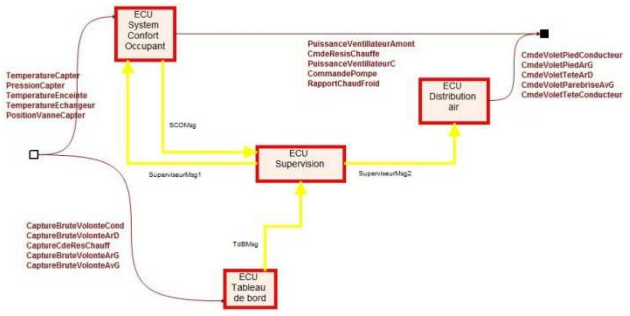

Figure 11: EE architecture view

This view is created within arKItect© and corresponds to a stage where software functions and signal conditioning drivers have already been allocated. Such a step can be completed only after static software and dynamic software architecture are completed.

7.2. Static software architecture

In the previous diagram, we have identified ECUs that host software functionalities. In this subsection and the following we explain activities prior to allocation: static and dynamic software architecture design.

The main step of the software static architecture definition is the modeling the interface between functions and their organisation in layers. The figure 12 shows details of the intereaction between applicative software components for a simplified climate control system. Each control (grey boxes) of this layer consumes and products data flow from/to the others controls of this layer or other one.

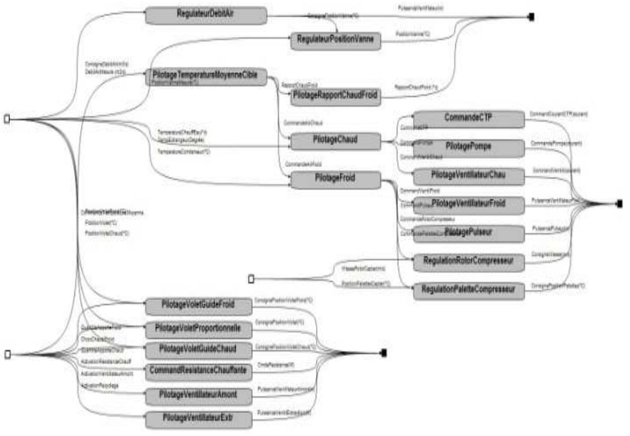

Figure 12: Applicative software architecture

Of course data flow must be specified with a physical type (e.g. meter), a software type, a range and step and a conversion function between software and physical type.

The advantage of a display of the static architecture is to verify that all consumed flows are produced and conversely that all produced flows are consumed. It's also efficient to see quickly the interaction between functions and document impact analysis and failure propagation or analysis.

7.3. Dynamic software architecture

The dynamic architecture specifies what is executed and when. The figure 13 illustrates this modeling. The events (white boxes) activate the execution of software components (grey boxes). The controls have been indentifying in the previous static diagrams.

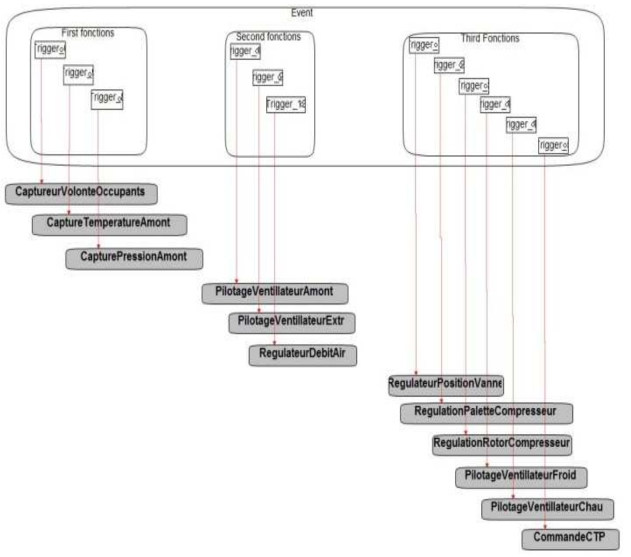

Figure 13: Dynamic software architecture

First functions could be a task as well as Second Functions and Third Functions. Each task contains a few function calls. Function calls are represented as red arrows in the diagram.

Diagram should be read from right to left corresponding to the order of calls.

All the functions in this diagram are executed at the same rate.

It is useful to represent the flows between functions in parallel with the call diagrams in order to avoir wrong schedule. Such diagrams are of course supported in our approach.

7.4. ECU design

As part of static architecture, the designer shall identify the basic and applicative software functions. The Basic software shall also be split into electrical layer (converting words in micro controller registers into current or voltage measure) and physical layer (converting said current or voltage into a physical measure). Then drivers and signal conditioners implement the interaction between basic software and hardware.

Below, the Figure 14 represents this software decomposition in several layers. Each layer (white boxes) exchange data flow (black arrows) with the other layer. The driver software layer makes sensor low electrical signal (red arrows) adaption to data flow and in the other way use data flow to drive actuator low electrical signal. The network software layer received and transmitted frame messages (yellow arrows).

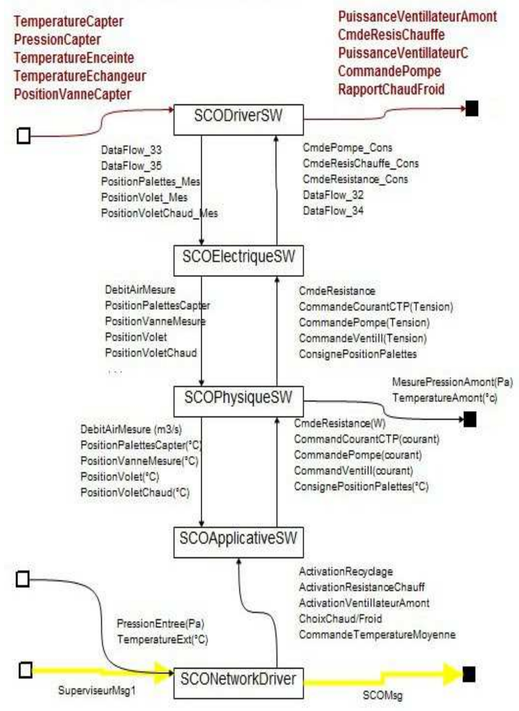

Figure 14: Software layers architecture

This completes the explanation of diagrams supporting the EE architecture design.

8. Conclusion

One of the most powerful features offered by this type of tool is to have all the life cycle phases of the System Engineering process on one single model offering therefore traceability from the early requirements to the system architecture whereas most commercial tools are only focussing on one or two phases.

While evaluating SysML, Renault also looks at other approaches that share the same type of data model but provide simplified and specialized views.

The interest of a tool like arKItect© is that it offers an intuitive interface and more flexible views that most SysML editors do.

Renault is currently working on the definition of so called "architecture frameworks" that define all necessary and indispensable views that a project team must produce. This will be the subject of another publication.

In the paper we have presented the design activities at the concept and development stage of the Renault AE-EV project. In fact, it is also possible to address simulation, validation and safety related issues in arKItect but this is clearly out of the scope of the present paper. We relate to the SASHA paper in the same conference for a short explanation of related activities with arKItect.

9. References

[1] ISO/IEC 15288. Systems and software engineering

[2] Guideline Systems Engineering for Public Works and Water Management, Ministery of Water Management, The Nederlands, May 2008 Second edition

[3] ISO 61851 Electric vehicle conductive charging system

[4] ISO 62196 Plugs, socket-outlets, vehicle couplers and vehicule inlets – Conductive charging of electric vehicles

[5] ISO 26262. International Organization for Standardization. Road Vehicles functional Safety. Standard under development

[6] Toward Families of QVT DSL and Tools, Benoît Langlois, Danierl Exertier, Ghanshyamsinh Devda, Thales Research & Technology, DSM forum, 2005

[7] System architecture, tools and modeling for safety critical automotive applications – the R&D project SASHA, publication, ERTS 2010

10. Glossary

AE-EV:	Advanced engineering Electric Vehicle
BPML:	Business Process Modeling Language
BPMN:	Business Process Modeling Notation
DSL:	Domains Specific Languages
ECU:	Electronic Control Unit
EE:	Electric and Electronic
EV:	Electric Vehicle
EVSE:	Electric Vehicle Supply Equipment
SysML:	System Modeling Language
UML:	Unified Modeling Language