SysML for embedded automotive systems: SysCARS methodology

Jean-Denis PIQUES

Valeo

Group Electronic Expertise and Development Services
14 avenue des Béguines, F-95892 Cergy-Pontoise Cedex
Tel: +331 34 331 751 – e-mail: jean-denis.piques@valeo.com

Abstract: This paper gives an overview of the five years of Valeo experience in deploying a Model Based System Engineering (MBSE) approach for mechatronic automotive embedded systems and products. The different stages are described, from initial studies, language and tool benchmarking up to the last returns of experience on industrial projects. Particular emphasis is put on describing the SysCARS methodology which gives, not only a precise mapping of System Engineering work items to SysML artefacts, but also the sequence of modeling activities to be performed. It is shown how the SySCARS methodology has been implemented as a SysML profile, based on a powerful "workflow-driven" mechanism, which helps the user during the modeling process. Finally it is presented how interoperability is ensured with the tools already in place for requirements management and control design.

Keywords: Model-Based System Engineering (MBSE), System Modeling, SysML, System Engineering, SysCARS

1. Introduction and overview

1.1. Motivations

During design and validation stages of automotive products, increasing complexity of technical systems, global organizations, business models and safety regulation (ISO 26262) requires higher formalization efforts than in the past. Standard System Engineering processes (ISO 15288, IEE 1220,…) are proven solutions to achieve the high level of quality targeted. These methodologies have been successfully used particularly in aerospace and railway transportation industries. However, the implementation of these processes with a traditional document centric approach leads to a huge effort in updating the documentation when customer change requests continuously occur; which is particularly the case for incremental development cycles involved in the automotive industry.

The Model Based System Engineering (MBSE) approach is a key lever for the automotive industry to cope with all these issues, while improving agility and R&D efficiency on innovative products. Indeed, the model is used as a (semi-)formal description of the product requirements shared by all project stakeholders, and as the unique source for on-demand automatic documentation generation.

1.2. Main lessons learned

Although SysML has become the de facto standard for MBSE, a supporting methodological background was and is still mandatory. The related Valeo experience is presented in **Chapter 2**, starting from the formalization of System Engineering processes and methods and ending with the development of a specific customization, to cope with the weaknesses of the current tools.

The SysCARS methodology [1], which is summarized in **Chapter 3**, defines the sequence of SysML diagrams and artefacts to be released, in order to implement the engineering process. However, pilot projects have shown that this guideline was not sufficient and consequently other critical issues have been addressed.

A major issue is the adoption of SysML existing modelling tools which are too complicated for non software engineers, providing no guidance on which diagram and artefact to use among overloaded GUIs. To support adoption and deployment control, a workflow driven approach is described in **Chapter 4** and is implemented by a Valeo profile, including ergonomic macros for the Artisan Studio modeler.

Moving from a document centric approach to model based engineering should also ensure the formal coupling with requirement management tools. **Chapter 5** addresses these aspects, defining a strategy regarding traceability checks and connection to dedicated tools such as DOORS and Reqtify.

Also to facilitate adoption and due to weaknesses of SysML compared to discipline modeling / simulation tools, SysCARS supports synchronization of structural diagrams. This feature is described in **Chapter 6** and is used to perform behavioural studies in legacy tools such as Simulink.

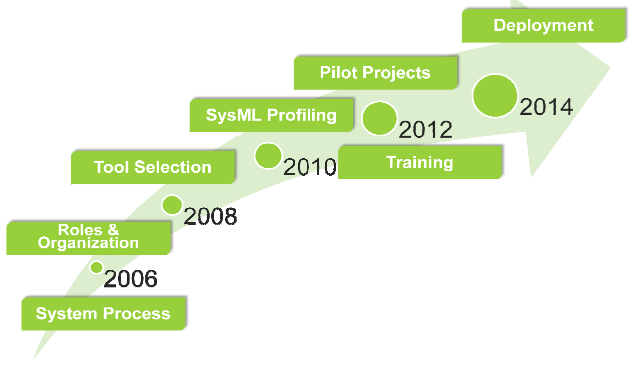


Figure 01: Model-Based System Engineering at Valeo

2. Valeo experience of using SysML

Valeo experience of using SysML and related works were initiated five years ago.

2.1. Process definition

Until recently, the low complexity of the automotive products historically manufactured by Valeo didn't require any System Engineering approach. This is the reason why neither System Engineering standard processes nor the related techniques and tools were precisely known.
The new Valeo strategy focusing on products minimizing the CO2 emissions of cars, naturally leads to the development of high added value complex systems. It was clear that the traditional processes, methods and tools were no longer adequate and a breakthrough was necessary to build a real System culture inside the Valeo group.
A transversal working group was put in place, consisting of representative experts from the relevant business groups of Valeo. This group, called "System & Product Technical Focus Group" was helped by external consultants familiar with System Engineering practices deployed in other industries such as aerospace and railways transportation. Based on these mixed competences, the working group defined engineering processes inspired from international standards (ISO 152888, IEE 1220, EIA 632, …) but totally adapted to Valeo's mindset and automotive constraints. The cornerstone of this referential was a document entitled "System Development and Validation Process", describing the System Engineering process as adapted to Valeo culture, with examples and hints to make it understandable by people not familiar with this domain. It was also completed with guidelines and templates of work products.

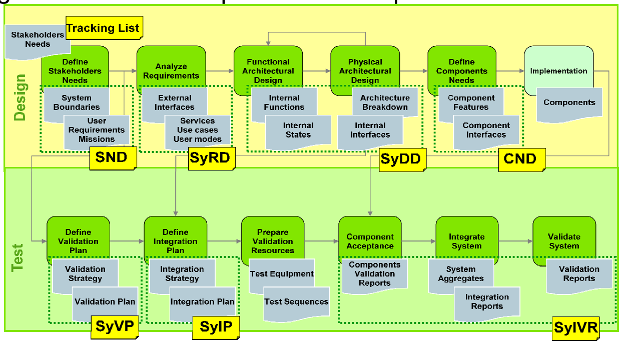


Figure 02: Valeo System Engineering process

2.2. Role definition and organization

In the field implementations of the System Engineering métier were also very different depending on the needs of the various Valeo Business Groups. In many situations, the System Engineering activities were not at all identified and there were no System team explicitly in charge of defining the best product architecture trade-off, prior to implementation level activities (software, hardware, mechanics). The system design was then entrusted to one of the implementation teams and generally not really formalized.
To cope with these issues, the "System & Product Technical Focus Group" defined a generic mapping of System Engineering activities to typical roles and responsibilities well established inside Valeo organization. This mapping was based on generic job descriptions generally used in the System Engineering community (e.g. system architect, requirements engineer, product manager, …). Starting from these indications, each Product Group has the ability to define customization rules to put in place the System organization best fitted to the constraints of its product line.

2.3. Tool selection

Some members of the "System & Product Technical Focus Group" were convinced that implementation of System Engineering processes using a document centric approach was unrealistic in the automotive domain, with continuously changing input requirements and very short time to market.
Therefore, investigations were performed in the Powertrain Systems Business Group, on methods and tools that could substantially help in deploying effective System Engineering processes. Considering that the key point is performing architecture design not managing requirements, the focus was put on investigating architecture modeling tools and not on requirements engineering ones.
The first stage of the tool selection process was to choose between tools with proprietary approaches and those based on the SysML language. Even if specialized architecture modeling tools (e.g. CORE) could have very interesting features and user-friendly GUI, the tools built upon the SysML language were preferred, due to their higher potential for evolution and interoperability. Indeed, the SysML language benefits from inputs from the whole System Engineering community and has the huge advantage of being standardized by OMG. Moreover, while still suffering from insufficiencies, the XMI interchange format offers the opportunity to migrate (most of the) SysML data from one tool to another tool, if required by the industrial constraints. Last but not least, learning SysML is now generally integrated into the training courses of engineers; which will make newly graduated engineers immediately efficient in their first professional environment.
The second stage of the tool selection process was to choose the SysML modeling tool best adapted to Valeo's expectations. After a pre-selection, based on answers to a questionnaire sent to SysML tool vendors, a detailed benchmark was performed on the three emerging SysML modelers.

Both the pre-selection questionnaire and the detailed benchmark were based on the same weighted criteria:

- **Exchanging data with existing tools**: Particular focus was put on exchanging data with requirement management tools (DOORS, Reqtify, …) and automatic documentation generation - Synchronizing architecture design descriptions with Simulink was also expected.
- **Ergonomics and generic features**: Stress was put on configuration management and control of user access and rights - Ergonomics and model readability were also mandatory expected properties - Ability to customize the interface and the workflow (ergonomic profiling) to make the easier to use, was also a strongly expected property for deployment.
- **System Modeling specific features**: Emphasis was put on the ability to check SysML language correctness, with contextual help to assist the user - Simulation internal to the SysML tool and autocoding from UML were not mandatory features.
- **Technical and methodological support**: Technical and methodological support from the tool vendor were considered as particularly important for efficient deployment.
- **Cost of deployment**: Lower cost of deployment was (of course) wished for, on the basis of high-end floating licences for System architects and low-end standalone licences for other System stakeholders.
- **SysML standard conformity**: Conformity to SysML 1.1 specification (October 2008) and later evolutions was mandatory.

Two tools emerged from the selection process: Artisan Studio from Atego and Rhapsody from IBM. Artisan Studio was preferred because it was best adapted to Valeo's intended use and ranking criteria. It is important to notice that at the time when the benchmark was performed (2009), open-source alternatives were not considered as mature and reliable enough for an industrial deployment.
Finally, the last stage of the tool selection process was to verify in detail the features of the selected tool, by implementing a wide scope example (i.e. powertrain management system).

2.4. SysML tool profiling

Despite the high potential of the selected SysML modeler, it was identified from the very beginning that tool customizations would be mandatory prior any efficient usage by generalist System Architects. Two concurrent approaches were then competing:

- Either developing a Domain Specific Language (DSL) completely masking the underlying SysML language, by using Valeo's own terminology and semantics specific to automotive embedded systems,
- Or keeping the original SysML syntax while providing a guided approach for using efficiently the right SysML diagram at the right analysis stage.

The second approach was preferred because Valeo's maturity on Model Based System Engineering processes was not estimated to be sufficient to define its own DSL, and also because using original SysML syntax is an efficient way to take benefits from the community of users, and in particular from young engineers already familiar with SysML.
Consequently, in a first step, the so-called SysCARS ("System Core Analyses for Robustness and Safety") methodology was developed to define a precise mapping between the sequence of System Engineering activities to be performed and the SysML modeling artefacts and diagrams to be used. In a second step, the SysML tool was customize to implement the SysCARS methodology, thanks to the "profiling" mechanism available in the tool. The SysCARS methodology is described in more detail in chapter 3, with its underlying workflow-driven implementation in chapter 4.

2.5. Pilot projects

The SysCARS methodology and the related SysML profile have been validated and optimized thanks to the pilot projects carried out during the last three years. These different projects allowed the coverage of a wide spectrum of problematics:

- Different kinds of product lines (e.g. combustion engine management systems, electrical and hybrid vehicle subsystems, steering column lock systems, electrical power steering systems, traction control systems, wiping systems), with different preferential modeling viewpoints,
- Different project typologies, from advanced studies focused on user requirements capture and architecture trade-off analyses, up to industrial projects focused on customer requirements traceability,
- Different System organizations.

Of course, these pilot projects led to the improvement of the SysCARS methodology and of its implementation inside the SysML tool. They have also contributed to a better definition of the role of the System Architect, as not just limited to requirements management but also including the completion of trade-off analyses necessary for product architecture optimization.

2.6. Training

Among the issues faced on the pilot projects, the main one was the slow learning curve of automotive engineers, due to the complexity of SysML modeling environments. The SysCARS methodology and the

related SysML workflow-driven profile partially solved the problem, but an efficient training course remains a mandatory pre-requisite.
The Valeo internal training course is divided into three main modules:

- **System Engineering basics**: This training is dedicated to acquiring background knowledge on System Engineering and related standard processes, methods and tools.
- **SysCARS methodology**: The objective of this training is to present SysCARS methodology concepts independently from any tool implementation, making comparisons with well known methods traditionally used for functional analysis.
- **SysCARS practice with SysML**: This training is dedicated to acquiring practical skills in using the SysML Valeo profile on a case study covering the whole scope of the SysCARS methodology.

After that, trained people are also helped on their initial project, by means of on the job training. In this context, the first step is to build the skeleton of the SysML model with the trainer. The latter also periodically reviews the model at different maturity steps and provides assistance for tricky tasks, such as connection to existing requirement management tools or configuration for automatic documentation publishing.

2.7. Deployment

Since the end of 2012, the model-based SySCARS methodology have been used at an industrial level for designing an electrical power steering system, with start of production planned for 2014. Other industrial applications are planned to start this year, in other product lines.

3. SysCARS methodology overview

SysCARS (System Core Analyses for Robustness and Safety) is a Valeo methodology which provides a practical help for system designers on how to perform the sequence of System modeling activities with SysML. However, its methodological background also makes sense independently from any tool implementation.

3.1. SysCARS principles

SysCARS methodology added value consists in:

- Selecting a subset of SysML diagrams and artefacts to be used in a convenient and pragmatic way (leading to the optimization of the learning curve),
- Providing defined semantics related to diagrams meaning and rules for verifying model consistency,
- Defining an obvious diagram sequence which ensures modeling efficiency regarding company processes,
- Implementing stereotypes and templates for automatic documentation generation at each stage of the process,
- Taking into account coupling constraints with other processes or tools such as Reqtify (from Dassault Systèmes) for requirement traceability or Simulink (from The Mathworks) for functional modeling

The current methodology [1][2] is therefore targeting the optimum trade off for Valeo deployment and it is built from existing state of the art. It does not claim for any theoretical novelty, while having merged relevant best practices from existing approaches, such as EIRIS methodology [3][4]. This implementation is also taking maximum benefits from available features of the selected SysML tool, namely Artisan Studio from Atego.

3.2. SysCARS workflow

The overall System Engineering process begins by analyzing the project context, considering the system to be developed as a black box, and then successively goes deeper into the details until specifying internal component (or system element) features. More precisely, the SysCARS methodology is divided into five major phases:

- ***Stakeholder needs definition***
- ***Requirements analysis***
- ***Logical architecture design***
- ***Physical architecture design***
- ***Components needs definition***

The related SysCARS workflow is described below.

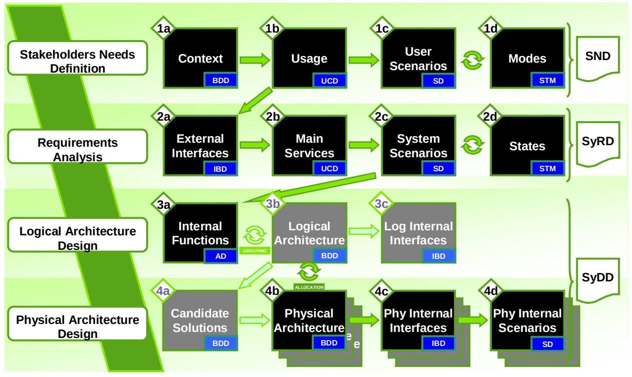


Figure 03: SysCARS methodology

For clarity purpose, the process and the sequence of activities are described in a pure sequential way. However, in practice, different steps could be performed simultaneously, with iterative and mutual refinements.
Moreover, each phase systematically ends with:

- Traceability analysis, to check the consistency and completeness of activities performed and artefacts created,
- Automatic generation of a document making a synthesis of the activities performed (*SND*:

Stakeholder Needs Document, *SyRD*: System Requirement Document, *SyDD*: System Design Document, *CND*: Component Needs Document).

The last stage (Component Needs Definition) has not been represented, because it is mainly an extraction of component artefacts from the physical architecture, producing one specification for each component.

On the [figure 03], the kind of diagram used at each step is given by its SysML acronym attached to the related activity: Block Definition Diagram (*BDD*), Internal Block Diagram (*IBD*), Use Case Diagram (*UCD*), Sequence Diagram (*SD*), STate Machine diagram (*STM*), Activity Diagram (*AD*)

Lessons learned on pilot projects have shown that in most situations it makes sense to bypass the elaboration of the logical breakdown and to directly allocate internal functions onto the physical architecture blocks. Indeed, physical architectures are very often frozen because resulting from carry over products, and therefore the investigation of several candidate solutions is not necessary.

Consequently, two kinds of optimized workflow have been defined depending on the project typology:

- **SysCARS-XS** (eXtended Stream): For innovative products, the whole set of activities of the [figure 03] are performed, and in particular the investigation of several physical architectures and trade-off analyses.
- **SysCARS-CS** (Core Stream): For carry over products, the activities represented by grey boxes on the [figure 03] are not performed

In the following of this chapter, for a clarity purpose, only the SysCARS-CS simplified workflow is presented. It is also important to notice that the names of paragraphs below are the same as those used in the workflow diagram presented at chapter 4.

3.3. Stakeholder needs definition

Probably the most important step in a system development process is collecting initial needs to secure the goals that the system under development is to pursue.

The key steps of this phase are:

- Identify all the stakeholder needs,
- Define the boundaries of the system and external actors involved,
- Identify and describe the operational use cases,
- Identify the user level operating modes,
- Link the stakeholder requirements to the operational use cases.

At this stage, all the analyses are performed from the system external user point of view, the system being considered as a black box. The output of this phase is the "Stakeholder Needs Document" (*SND*), which makes a synthesis of all the activities performed.

3.3.1. Stakeholder needs elicitation (REQ)

All individuals and organizations that may have an interest in the system are the potential source of requirements and therefore should be identified prior to all other activities. The key point is that stakeholder needs should describe the services expected by the system user, and not how the system will fulfill these needs.

The sources of stakeholder needs will be managed outside of the SysML model, within requirements documents or specific databases. It is particularly important to capture mission-level performance requirements and measurements of expected performances that will be used later to select the best one among the candidate solutions.

The next step is to import stakeholder needs (with all their relevant fields) into mirroring SysML requirement objects (with same identifiers). A gateway mechanism, such as those implemented by Reqtify, is required to perform a mono-directional synchronization (from external data to SysML) in case of change of source data.

Because the standard SysML requirement format is quite limited, the extension mechanism of stereotypes is used to add new specific attributes (i.e. *tags*) to keep track of extra information resulting from analyses performed during elicitation. A particularly important *tag* attached to requirements at elicitation stage is dedicated to classifying requirement into one of the three following categories: user related, system related or component (i.e. system element) related. This value conditions at which modeling level the requirement will be later covered.

3.3.2. Context analysis (BDD)

The system context diagram represents the direct environment of the system and gives initial information about the system boundaries and the interactions between the system and external systems and users.

The first step is to identify the different stages of the system lifecycle, from manufacturing to recycling. For each stage of the system lifecycle, one SysML block definition diagram is declared to model the associated operational context.

The system itself appears in the center of the diagram as a single black box SysML *block*. The next step consists in representing all currently known interacting partners, using SysML *actor* objects. An *actor* is not necessarily a concrete individual or system, but a role played by an outside element in interaction. Then, interactions between *actors* and the system are represented as SysML *association* relationships. The purpose is to identify basic information helpful to determine the services requested from the system embedded in its environment, and not to give technical details of these services. Constraints on these services are documented in the description field of the *association* relationships.

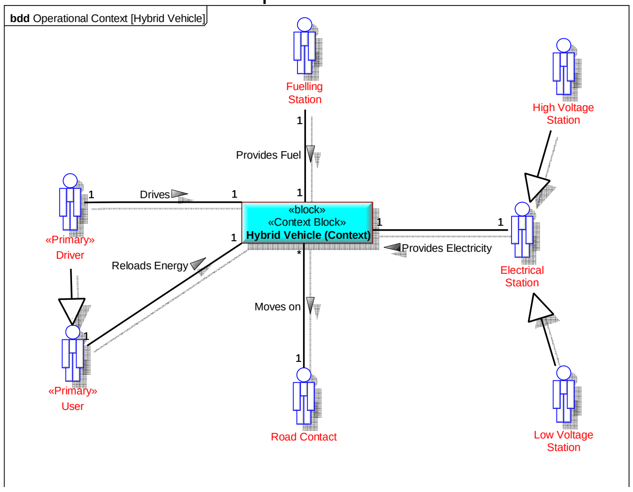


Figure 04: Operational context diagram

Even though, defining the context diagrams may seem obvious, in practice, searching for actors can lead to very fruitful discussions for defining responsibilities between the different stakeholders.

3.3.3. Context scenarios identification (UC, SD)

Context *use cases* represent the services expected by the system users (people or other systems); which means that they will be key input elements for the requirement analysis stage. Indeed, context *use cases* will help to refine stakeholder expectations and therefore identify system requirements in greater details.

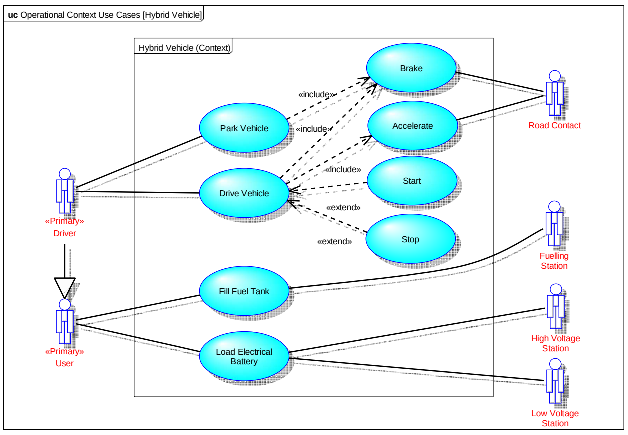


Figure 05: Context use case diagram (user level)

Context *uses cases* will be identified starting from the context diagrams, asking what the *actors* want of the system, especially with regard to their roles and incoming information flows. More precisely, a *use case* always refers to at least one *actor*; it is started by an external trigger and it ends with a user result. Moreover, as many use case diagrams as stages of the system lifecycle will be described.
In fact, a *use case* can be seen as a group of scenarios performed by the same main *actor*, with the same starting point and leading to the same ending point. These scenarios describe sequences of interactions and actions, beginning with the same pre-condition (trigger) and ending with the same post-condition (result); the pre-condition and the post-condition corresponding to modes (i.e. user *states*) in the user mode state machine mentioned in the next chapter.
The scenarios are described using SysML sequence diagrams. The interactions inside scenarios are declared as context *events*, also used later to define transition conditions between *states* of the user mode state machine. It is particularly important to notice that each sequence diagram established here is primarily aimed at identifying the system interactions. The sequence diagram will be further refined, at requirement analysis stage, to identify functions performed by the system.

3.3.4. User modes identification (STM)

A mode characterizes a situation in the system life for which a specific expected behavior can be defined. It represents a state invariant of the system from the external user point of view (i.e. regarding the service given to the user and not how this service is performed by the system).

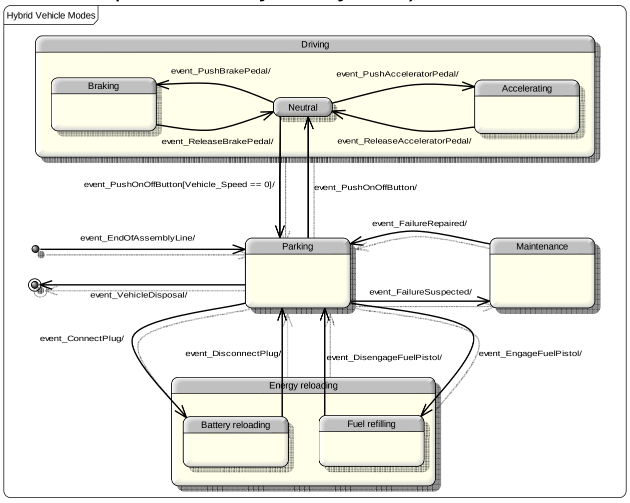


Figure 06: User modes state diagram

The objective is to derive a unique state machine aggregating all the modes (*states*) and main transitions (*events*) identified in the context scenarios. Therefore, establishing the user mode state machine is an iterative process tightly coupled and interleaved with the identification of context scenarios described in the previous chapter. The purpose is to describe the behaviours involved in all the context scenarios, factorized into a unique state machine. This state machine is owned by the *context block* associated to the whole black box system.

3.3.5. Context traceability checking (REQ)

We remember that stakeholder (or initial) requirements refer to statements that define the expectations from the system in terms of mission objectives, environment, constraints and

measurements of performance, from the system user point of view. In order to make sure that all stakeholder needs are covered by the context use cases, respective traceability links have to be established between SysML *use cases* and *requirements*. As described in the traceability data model presented at chapter 5.3, all *requirements* that are characterized as user related shall be covered by corresponding context *use cases*, and linked together by *derive* relationships.
Traceability analyses are performed to verify the model completeness, using *requirements tables* and *traceability matrices*, which are specific features of the Artisan Studio tool.

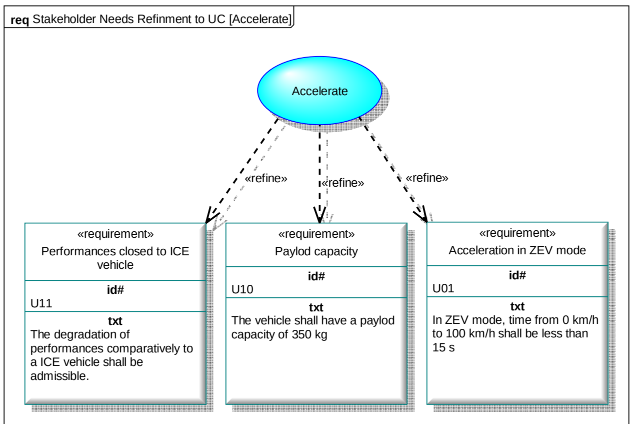

req Stakeholder Needs Refinment to UC [Accelerate]
Accelerate
«refine»
«refine»
«refine»
«requirement»
Performances closed to ICE vehicle
id#
U11
txt
The degradation of performances comparatively to a ICE vehicle shall be admissible.
«requirement»
Paylod capacity
id#
U10
txt
The vehicle shall have a paylod capacity of 350 kg
«requirement»
Acceleration in ZEV mode
id#
U01
txt
In ZEV mode, time from 0 km/h to 100 km/h shall be less than 15 s

Figure 07: Requirement traceability diagram

3.3.6. Stakeholder needs document

The last step is to launch the automatic generation of a document, making the synthesis of all the modeling activities performed during the "stakeholder needs definition" stage. This document is entitled "Stakeholder Needs Document" (*SND*).

3.4. Requirements analysis

The objective of the requirement analysis phase is to analyze the inputs previously collected, in order to move from a problem statement to an abstract solution.
The key steps of this phase are:

- Describe precisely the interfaces of the system with external actors,
- Develop and refine the system use cases,
- Identify the system level operating states,
- Develop and refine the system requirements into external function and interface descriptions,
- Link system functions and interfaces to the system requirements.

At this stage all analyses are performed from system designer point of view, the system still being considered as a black box. The output of this phase is the "System Requirement Document" (SyRD), which summarizes all the activities performed.

3.4.1. External interfaces identification (IBD)

To keep track of analyses previously performed at context level, it has been decided to define the *system block* used afterward as a specialization of the *context block* studied at the previous stage (i.e. stakeholder needs definition stage).
The objective of the system interface identification step is to give more details on the interaction flows between the *actors* and the system (always seen as a black box). The system physical external interfaces are described using internal block diagrams, where are represented the *system block* and all the interacting *actors*.

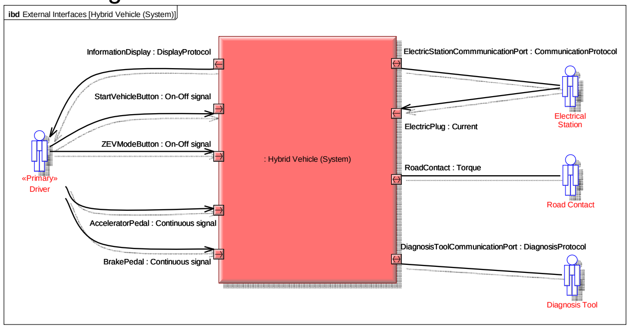

ibd External Interfaces [Hybrid Vehicle (System)]
InformationDisplay : DisplayProtocol
StartVehicleButton : On-Off signal
ZEVModeButton : On-Off signal
«Primary» Driver
AcceleratorPedal : Continuous signal
BrakePedal : Continuous signal
: Hybrid Vehicle (System)
ElectricStationCommmunicationPort : CommunicationProtocol
Electrical Station
ElectricPlug : Current
RoadContact : Torque
Road Contact
DiagnosisToolCommunicationPort : DiagnosisProtocol
Diagnosis Tool

Figure 08: External interfaces description

To specify the kind of admissible data flow, a type indication shall be associated with each *port*, using SysML *item types* or *flow specifications*. Several internal block diagrams are defined to describe the different contexts of use. Moreover, it is also possible to define several internal block diagrams for the same context of use, each diagram corresponding to a specific kind of interface (e.g.: mechanical, electrical, data processing buses…). This is particularly interesting to ease information sharing with involved disciplines and also to avoid overloaded interface diagrams.

3.4.2. System scenario refinement (SD)

The objective of this stage is to refine context level scenarios, in order to identify main services or functions the system shall perform. Therefore, this activity is similar to a classical external functional analysis.
To keep track of analyses previously performed at context level, it has been decided to keep context sequence diagrams intact and to clone them, in order to obtain initial system sequence diagrams. The same approach is adopted between context use case diagrams and system use case diagrams. The system sequence diagrams are then complemented by the functions to be performed by the system (always seen as a black box), after having replaced the *context block* by the *system block*. As shown on the figure below, the functions are modeled as SysML *operations* attached to the (lifeline of the) *system block*.
The interactions inside scenarios are declared as existing context *events* or new system *events*, used

afterward to define transition conditions between *states* of the system state machine. The starting point and ending point of each scenario will also correspond to states of the system state machine.

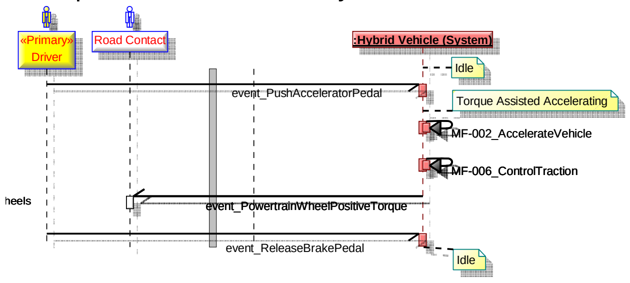


Figure 09: Scenario description (system level)

3.4.3. System states identification (STM)

The objective of this step is to describe the behaviours involved in all the system scenarios, factorized into a unique state machine. This state machine is owned by the *system block* describing the whole black box system.

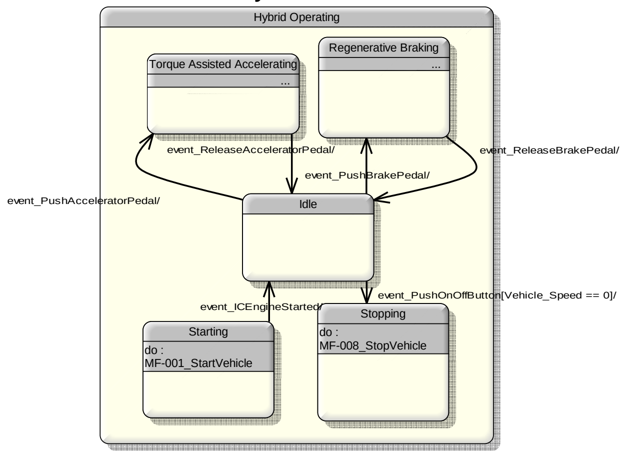


Figure 10: Sub-state of the system state machine

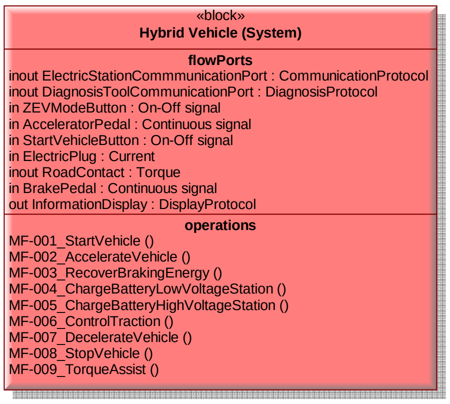
The system state machine is not necessarily only a refinement of the user mode state machine, as possibly new sub-states or suppressed states and even a completely different structure may be defined. Practically, establishing the system state machine is an iterative process tightly coupled and interleaved with system scenarios refinement described in the previous chapterOnce the transitions and states of the system state machine are well defined, a particularly important step is to define in which states are triggered the main functions identified in the system scenarios. This is simply done by calling the related *operations* with the *Do property* of the corresponding *states*. The system state machine then becomes the central element of the system model, allowing the simulation of its behaviour for validation purpose.

3.4.4. System requirement traceability checking (REQ)

As it will be discussed at chapter 5, system level requirements are all described inside the SysML model and not rewritten into an external (textual) requirements repository. As often as possible, the SysML model artefacts (e.g. *operations, ports, states*) are directly used as "requirements"; their description field being written in a requirement-like way. Only non functional system requirements are modeled by SysML *requirements* (at the exception of system related functional requirements coming from stakeholder inputs). Non functional requirements concern constraints (including performance target) related to existing model artefacts, such as *operations*, *ports*, *states* or to the *system block* itself. Therefore, we can identify two categories of "traceability" links:

- **Implicit traceability**, when there exists a strong dependency between two model artefacts (e.g. *operation* owned by the *system block*, *port* owned by the *system block*)
- **Explicit traceability**, when a *satisfy* relationship has been declared between a model artefact (i.e. *operation, port, state, system block*) and a non functional *requirement* of the same level or when a *refine* relationship has been declared between a system level *requirement* and a user level *requirement*.

During the requirement analysis process, implicit traceability links have been generated, while the *system block* has been automatically populated with all *operations* and *ports* declared in the different diagrams. Moreover, *operations* are also linked to *sequence diagrams* where they have been defined and to the *states* which call them.


«block»
Hybrid Vehicle (System)

flowPorts
inout ElectricStationCommmunicationPort : CommunicationProtocol
inout DiagnosisToolCommunicationPort : DiagnosisProtocol
in ZEVModeButton : On-Off signal
in AcceleratorPedal : Continuous signal
in StartVehicleButton : On-Off signal
in ElectricPlug : Current
inout RoadContact : Torque
in BrakePedal : Continuous signal
out InformationDisplay : DisplayProtocol

operations
MF-001_StartVehicle ()
MF-002_AccelerateVehicle ()
MF-003_RecoverBrakingEnergy ()
MF-004_ChargeBatteryLowVoltageStation ()
MF-005_ChargeBatteryHighVoltageStation ()
MF-006_ControlTraction ()
MF-007_DecelerateVehicle ()
MF-008_StopVehicle ()
MF-009_TorqueAssist ()

Figure 11: System requirements as block properties (implicit traceability)

Explicit traceability links shall be declared to ensure that all user level *requirements* are covered by system level *requirements* (*derive* relationship) and that all system level *requirements* are covered by model artefacts (*satisfy* relationship) of the same level. These relationships are declared either in requirement diagrams or directly in the object database. Justifications related to coverage or refinement are logged as comments inside the description fields of the requirements diagrams.

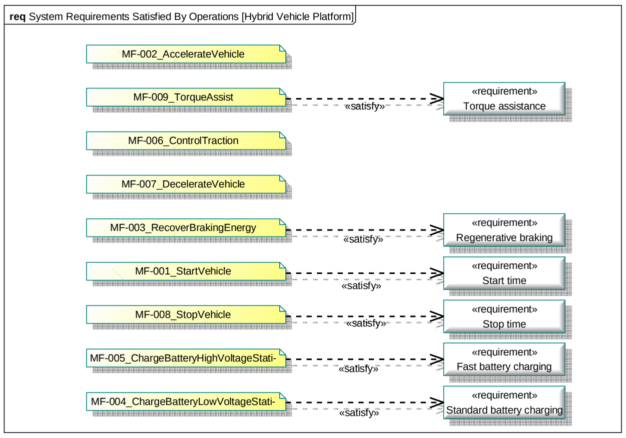

req System Requirements Satisfied By Operations [Hybrid Vehicle Platform]
MF-002_AccelerateVehicle
MF-009_TorqueAssist
«satisfy»
«requirement» Torque assistance
MF-006_ControlTraction
MF-007_DecelerateVehicle
MF-003_RecoverBrakingEnergy
«satisfy»
«requirement» Regenerative braking
MF-001_StartVehicle
«satisfy»
«requirement» Start time
MF-008_StopVehicle
«satisfy»
«requirement» Stop time
MF-005_ChargeBatteryHighVoltageStati-
«satisfy»
«requirement» Fast battery charging
MF-004_ChargeBatteryLowVoltageStati-
«satisfy»
«requirement» Standard battery charging

Figure 12: Traceability to system requirements (explicit traceability)

Traceability analyses are then performed to verify the model completeness, using *requirements tables* and *traceability matrices*, which are specific features of the Artisan Studio tool.

3.4.5. System requirements document

The last step is to launch the automatic generation of a document, making the synthesis of all the modeling activities performed during the "requirements analysis" stage. This document is entitled "System Requirements Document" (*SyRD*).

3.5. Architecture design

The objective of the architecture design phase is to describe how the system will be internally structured to perform the expected features. Within the framework of the SysCARS-CS simplified workflow presented here, the physical architecture is directly elaborated taking into account implementation technologies. Logical architecture design activities are limited to identify internal functions and there is no intermediate logical architecture.

The key steps of this phase are:

- Identify the set of internal functions to be provided by the system elements (or components),
- Describe how these internal functions are activated depending on the system state,
- Define a physical architecture capable of performing the required internal functions,
- Allocate coherently the internal functions to physical components,
- Develop and refine the components physical interfaces and interactions,
- Develop and refine the related components requirements,
- Evaluate the measurements of effectiveness of the physical architecture.

At this stage all the analyses are made from system internal point of view, the system being considered as a white box.

The modeling elements developed are included in the "System Design Document" (*SyDD*), which makes a synthesis of all logical and physical architecture design activities. The output of this phase is a also a set of "Component Needs Documents" (*CND*), which correspond to specifications for the components (or system elements) to be implemented.

3.5.1. Internal functions identification (ACT)

The objective of this step is to provide details on the internal behavior of the *operations* owned by the *system block*. Therefore, the kind of task performed is similar to a classical internal functional analysis.

For this analysis, a top level activity diagram is attached to each *operation* of the *system block*, in order to describe how the corresponding main function is implemented by internal technical functions. This description may involve several layers of activity diagrams. The activity diagrams use data flow and control flow representations in a hierarchical decomposition to work out internal activities that should be performed. The lowest level *activities* (namely leaves activities) of this hierarchy represent calls (*call-operation-actions*) to internal functions modeled by elementary *operations*. The rule to end the hierarchical decomposition is that every identified elementary *operation* can be assigned to a unique system element (or component) of the physical architecture.

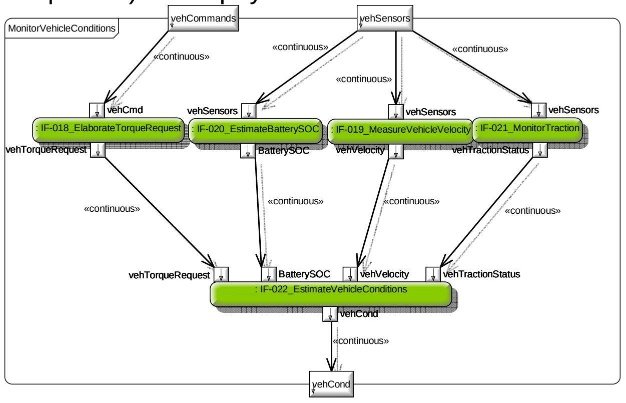


Figure 13: Lowest level activity diagram describing system internal functions

The key point is that the upper level activity diagrams describing the internal behavior are triggered by the system state machine; which will be an interesting and mandatory property for later execution of the system model.

3.5.2. Physical architecture definition (BDD)

The focus of the physical architecture design phase is on the allocation of elementary *operations* to components (or parts) of a physical architectural structure. This structure may result from a previous trade-off study or be a legacy architecture resulting from many years of experience of a product line. It is

the reason why the elaboration of an intermediate logical architecture (independent from any technology choice) is most of the time useless.
The partitioning criteria used for allocation of internal functions to components should reduce the impact of requirements and technology changes and more effectively address key issues such as performance, reliability, efficient re-use of COTS, maintainability, security and cost.At model level, an internal physical *block* is declared for each component (or part) of the physical architecture, and this block owns the elementary *operations* which were allocated to him. To keep track of analyses previously performed at system level, it has been decided to define the upper level *physical (system) block* used afterward as a specialization of the *system block* studied at the previous stage (i.e. requirements analysis stage). Then, a block definition diagram is used to describe the physical architecture, i.e. the compositional relationships between the upper level *physical (system) block* and its constitutive physical *blocks*.

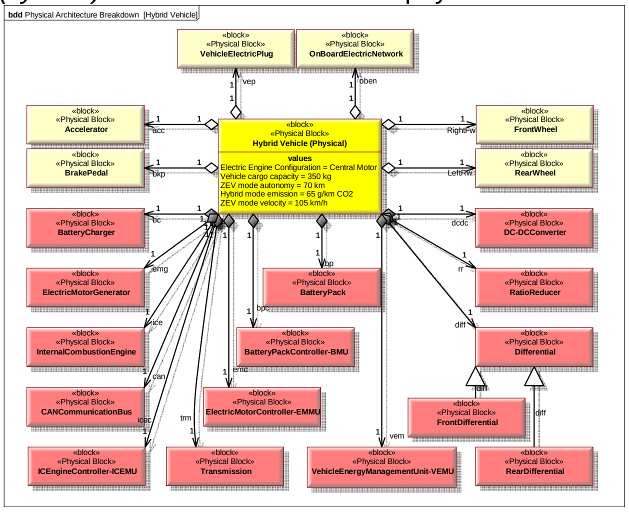

bdd Physical Architecture Breakdown [Hybrid Vehicle]
«block» «Physical Block» VehicleElectricPlug
«block» «Physical Block» OnBoardElectricNetwork
«block» «Physical Block» Hybrid Vehicle (Physical)
values
Electric Engine Configuration = Central Motor
Vehicle cargo capacity = 350 kg
ZEV mode autonomy = 70 km
Hybrid mode emission = 65 g/km CO2
ZEV mode velocity = 105 km/h
«block» «Physical Block» Accelerator
«block» «Physical Block» BrakePedal
«block» «Physical Block» FrontWheel
«block» «Physical Block» RearWheel
«block» «Physical Block» BatteryCharger
«block» «Physical Block» DC-DCConverter
«block» «Physical Block» ElectricMotorGenerator
«block» «Physical Block» BatteryPack
«block» «Physical Block» RatioReducer
«block» «Physical Block» InternalCombustionEngine
«block» «Physical Block» BatteryPackController-BMU
«block» «Physical Block» Differential
«block» «Physical Block» CANCommunicationBus
«block» «Physical Block» ElectricMotorController-EMMU
«block» «Physical Block» FrontDifferential
«block» «Physical Block» ICEngineController-ICEMU
«block» «Physical Block» Transmission
«block» «Physical Block» VehicleEnergyManagementUnit-VEMU
«block» «Physical Block» RearDifferential
vep
oben
acc
bkp
RightFw
LeftRw
bc
dcdc
emg
bp
rr
ice
bpc
diff
can
emc
icec
trm
vem
1

Figure 14: Physical architecture

3.5.3. Physical internal interfaces identification (IBD)

The objective of the internal physical interface description step is to provide more details on the interaction flows between the internal *physical blocks*, using internal block diagrams. Physical interfaces between internal *physical blocks* are represented by *ports* which can be connected either to other internal physical blocks or directly to external interfaces of the upper *level physical (system) block*. To specify the kind of admissible data flow, a type indication shall be associated with each *port*, using SysML *item types* or *flow specifications*.

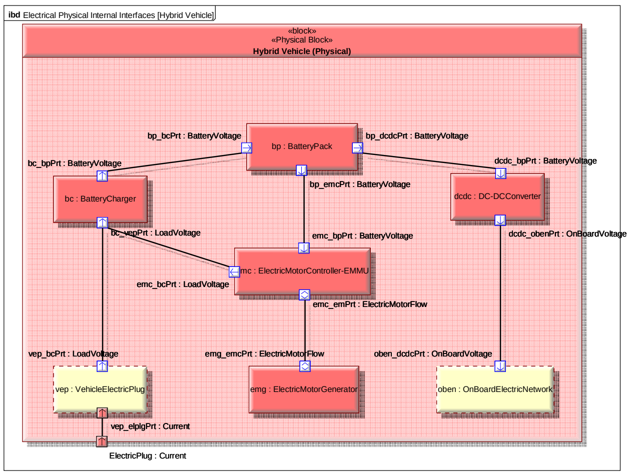

ibd Electrical Physical Internal Interfaces [Hybrid Vehicle]
«block» «Physical Block» Hybrid Vehicle (Physical)
bp_bcPrt : BatteryVoltage
bp : BatteryPack
bp_dcdcPrt : BatteryVoltage
bc_bpPrt : BatteryVoltage
dcdc_bpPrt : BatteryVoltage
bp_emcPrt : BatteryVoltage
bc : BatteryCharger
dcdc : DC-DCConverter
bc_vepPrt : LoadVoltage
emc_bpPrt : BatteryVoltage
dcdc_obenPrt : OnBoardVoltage
emc : ElectricMotorController-EMMU
emc_bcPrt : LoadVoltage
emc_emPrt : ElectricMotorFlow
vep_bcPrt : LoadVoltage
emg_emcPrt : ElectricMotorFlow
oben_dcdcPrt : OnBoardVoltage
vep : VehicleElectricPlug
emg : ElectricMotorGenerator
oben : OnBoardElectricNetwork
vep_elplgPrt : Current
ElectricPlug : Current

Figure 15: Physical internal interfaces description

To avoid information overload on the same diagram and to make communication to a specific team easier, several internal block diagrams will be described, each diagram corresponding to a specific kind of interface (ex: mechanical, electrical, data processing buses, …).

3.5.4. Internal scenarios definition (SD)

The focus of black-box sequence diagrams described at system level was on the identification of the system main functions. Because some physical components may require significant refinement to address discipline-specific concerns, it may be necessary to establish white-box sequence diagrams focusing on the collaboration between the different internal components. This activity is not systematically performed and is only reserved for particularly critical scenarios. A white-box internal scenario reveals internal *physical blocks* on a same sequence diagram. It allows checking that the sequential activation of the elementary functions (*operations* owned by internal *system blocks*) is consistent with the main functions (*operations* owned by the upper level *system block*) expected at system level.
Moreover, a physical internal component may include a state machine as part of its specification, if it has significant state-based behavior

3.5.5. Physical architecture traceability checking (REQ)

The same considerations as those done at chapter 3.4.4, regarding implicit and explicit traceability links, also apply for internal *physical blocks* and related *requirements*. Therefore, the traceability checking is performed in the same way as at the requirement analysis stage.
During the architecture design process, implicit traceability links have been generated, while the internal *physical blocks* have been automatically populated by all elementary *operations* and *ports* declared in the different diagrams. Moreover, elementary *operations* are also indirectly linked to

the *states*, which call the activity diagrams in which they appear. Regarding explicit traceability links, they shall be declared to ensure that all system level *requirements* are covered by component level *requirements* (*derive* relationship) and that all component level *requirements* are also covered by model artefacts (*satisfy* relationship) of the same level. These relationships are declared either in requirement diagrams or directly in the object database. Justifications related to coverage or refinement are logged as comments inside the description fields of the requirements diagrams.

Traceability analyses are then performed to verify the model completeness, using *requirements tables* and *traceability matrices*, which are specific features of the Artisan Studio tool.

The results from engineering analyses done on the physical architecture are also capitalized inside the model. These information often referred to as measurements of effectiveness (MoEs), are incorporated into the SysML model as value properties attached to the upper level *physical block* describing the physical architecture. The estimations of MoEs result from specific analyses performed with appropriate tools such as modeling and simulation environments and involve different analysis objectives (performance, robustness, safety, cost…).

3.5.6. System design document and component needs documents

The last step is to launch the automatic generation of a document, making the synthesis of all the modeling activities performed during the "architecture design" stage. This document is entitled "System Design Document" (*SyDD*).

The physical architecture model results in the specification of the components to be implemented by each specific discipline (e.g. hardware, software, mechanics, …). As it is necessary to isolate relevant information for each team in charge of developing components, there is a need for as many specification documents as there are internal *physical blocks* inside the physical architecture. These documents called "Component Needs Documents" (*CND*) are also automatically generated, making the extraction of all the artefacts attached to the internal *physical block* under consideration and filtering any confidential or unnecessary information.

3.6. Modeling difficulties encountered with SysML 1.2

During the different modeling stages, some difficulties have been encountered with SysML 1.2.

First of all, we would like to have a unified semantics for interfaces. Unfortunately there is no relationship between the *ports* defined on Internal Block Diagrams and the *pins* used on Activity Diagrams. For each *port* declared for external interfaces, it was necessary to create one corresponding *pin* with the same name and type. To keep track of the similarity of the two artefacts, a *trace* relationship was declared between them.

Moreover, readability issues appear on Internal Block Diagrams when the number of *ports* and *connectors* becomes important. This problem was partially solved by declaring composite flows as often as possible, thanks to *flow specifications*. Nevertheless, routing efficiently *connectors* remains problematic. In this area, inspirations from tools like Simulink would be welcome, for example by adding higher level constructs such as *virtual buses* or *Goto-From* connections, in order to avoid crossing connectors. Including automatic routing features would also be a valuable evolution.

In the context of the full SysCARS-XS workflow, difficulties were experienced when dealing with architecture alternatives and particularly for functional to physical allocation. We would like that the same *operation* could be declared only once and be owned by different *blocks*, each one being related to an architecture alternative. Unfortunately, it was necessary to clone each *operation* representing the same function, as many times as there were alternatives to explore.

Weaknesses of the XMI interchange format are well known and particularly the impossibility of exchanging diagrams. However, we were very surprised to notice that some basic information were not exported (e.g. descriptions fields of some artefacts or characters different than ASCII ones). Therefore, it remains very difficult to transfer properly modeling descriptions to other modeling tools or environments such as Simulink.

4. Workflow-driven approach

4.1. A specific SysML profile

GUIs of SysML existing tools remain too complicated for a non software specialist, who is the targeted audience for System Engineering. Indeed, SysML user interfaces provide confusing and unneeded features from the UML world. Very often, UML and SysML artefacts and diagrams are mixed without any possibility for the user to limit to a pure SysML scope. Moreover, no guidance is provided on the relevant diagram to be used and on the correct ordering of operations.

To cope with these drawbacks, a specific ergonomic profile (thereafter referred to as "Valeo Profile") has been developed, introducing the concept of workflow-driven approach. The basic idea behind the workflow-driven approach is to provide the System engineer with a step by step help throughout the SysCARS engineering workflow. Moreover, at each step of the workflow, only relevant features and diagrams are available in a simplified GUI.

The mechanisms of the workflow driven approach are detailed in the chapters below.

4.2. Workflow diagram navigation

When creating a new model with the Valeo profile, this model directly opens on a pre-defined "workflow diagram". The "workflow diagram" is the central element of the Valeo Profile, defining the sequence of modeling activities to be performed in accordance with the SysCARS methodology. In fact, the workflow diagram is simply a statechart diagram, where states and super-states respectively correspond to elementary activities and main stages of the SysCARS methodology. No more than one elementary state can be active at one moment; i.e. only one kind of elementary activity should be performed. On the workflow diagram represented below, the active state is highlighted in blue.

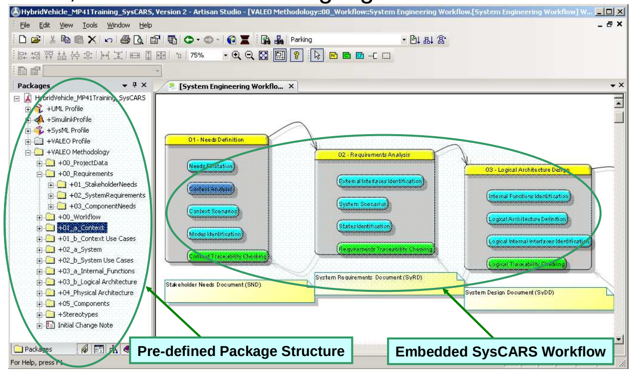

Pre-defined Package Structure
Embedded SysCARS Workflow
01 - Needs Definition
02 - Requirements Analysis
03 - Logical Architecture Design

Figure 16: Valeo profile GUI overview

It is possible to navigate the states of the workflow diagram and to select the workflow commands available: "Next Step", "Previous Step", "Go to step…". Then the modeling step is changed accordingly.

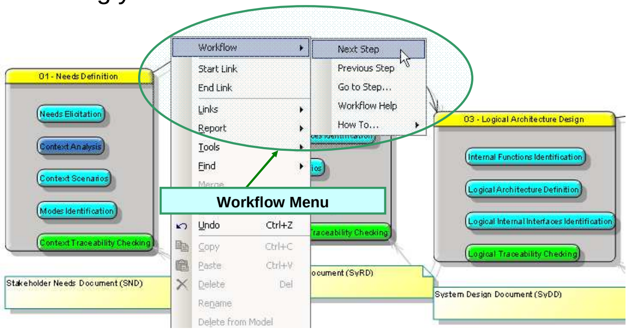


Figure 17: Valeo profile navigation

A second kind of navigation mechanism is available from the workflow diagram. Right-clicking on each state allows to reach the diagrams summarizing the results of this modeling step. The relevant diagrams should have been attached as associated diagrams once created.

The implementation of the workflow in the profile is not frozen but configured using dedicated XML files. This option enables further evolutions on the SysCARS workflow.

4.3. Pre-defined package structure

When creating a new model with the Valeo Profile, this model is also provided with a pre-defined package structure. This package hierarchy is directly correlated to states and super states of the workflow diagram, which in turn correspond to stages and steps of the SysCARS methodology.

However, the user is free to organize differently artefacts and diagrams within a different package structure.

As previously, the pre-defined package structure is not frozen but configured using a dedicated XML file.

4.4. GUI features defined by workflow state

The current active state of the workflow diagram is used to monitor the look and feel of the SysML modeler, in order to provide the user only with the features required at this step of the system modeling process. Consequently, command menus available in the object browser and toolbar menus on diagrams are both customized differently in each state of the workflow diagram.

The diagram below clearly shows the level of simplification on command menus reached by the Valeo Profile.

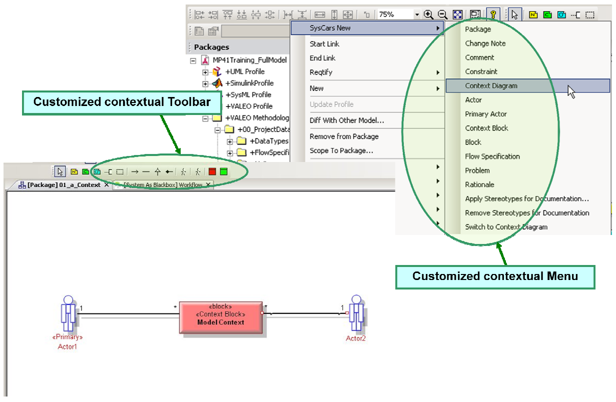

Customized contextual Toolbar
Customized contextual Menu
SysCars New
Package
Change Note
Comment
Constraint
Context Diagram
Actor
Primary Actor
Context Block
Block
Flow Specification
Problem
Rationale
Apply Stereotypes for Documentation…
Remove Stereotypes for Documentation
Switch to Context Diagram

Figure 18: Customized menus

In the object browser window, the "SysCars New" command menu displayed when right-clicking an existing SysML object, is customized individually for each type of SysML artefact and diagram. In case the user wishes to have access to the classic features of SysML, he can select the standard "New" command menu.

In the graphical window, buttons available on each diagram toolbar are also customized depending on the workflow diagram active state.

The GUI features are evolutionary and configured from two dedicated XML files, one for the package browser command menus and one for the diagram toolbars.

4.5. Stereotypes for documentation

Documentation in a format that is easily comprehensible by a broad range of stakeholders remains an effective way to validate and communicate system design information. The first thing to do is to precisely define the expected document format and contents by creating a corresponding template for the publishing tool. The same document template will be re-used on different projects, without any modification. Then, thanks to the publishing feature of the SysML tool, automatic document generation can be run on demand, to collect relevant data from the SysML model, without any special effort.

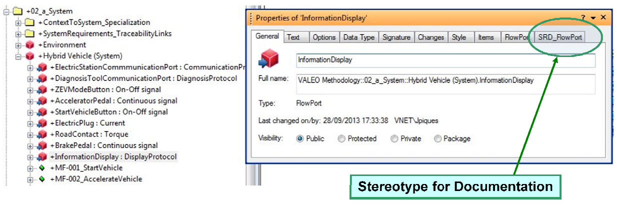
Furthermore, separation between modeling data and document templates enables versatile customisation either to generate generic outputs or to address specific customer process.

The organisation of the documentation is also based on the workflow diagram breakdown. One particular kind of document (with related template) is defined for each workflow diagram super-state, in order to make the synthesis of modeling activities performed within this stage:

- ***SND*** (Stakeholder Needs Document) for Stakeholder needs definition stage,
- ***SyRD*** (System Requirements Document) for Requirements analysis stage,
- ***SyDD*** (System Design Document) for Logical and Physical architecture design,
- ***CND*** (Components Needs Document) for Components needs definition stage.

SysML artefacts and diagrams created when being in a given super-state of the workflow diagram are automatically attached with stereotypes indicating that they should appear in the document associated with this super-state. The names of these stereotypes are built with the name of artefact or diagram, prefixed by the name of the target document (e.g: *SND_requirement*). It is also possible to manually apply documentation stereotypes when artefacts should appear in multiple documents

The only thing left to do is to load into the publishing tool the pre-defined documentation template related to the workflow super-state to be documented, and then to launch documentation rendering. Diagrams and artefacts appearing in the final document are automatically filtered depending on their documentation stereotypes, i.e. on the stage of the workflow where they have been created.


+02_a_System
+ContextToSystem_Specialization
+SystemRequirements_TraceabilityLinks
+Environment
+Hybrid Vehicle (System)
+ElectricStationCommmunicationPort : CommunicationPr
+DiagnosisToolCommunicationPort : DiagnosisProtocol
+ZEVModeButton : On-Off signal
+AcceleratorPedal : Continuous signal
+StartVehicleButton : On-Off signal
+ElectricPlug : Current
+RoadContact : Torque
+BrakePedal : Continuous signal
+InformationDisplay : DisplayProtocol
+MF-001_StartVehicle
+MF-002_AccelerateVehicle
Properties of 'InformationDisplay'
General
Text
Options
Data Type
Signature
Changes
Style
Items
FlowPor
SRD_FlowPort
Stereotype for Documentation

InformationDisplay
Full name: VALEO Methodology::02_a_System::Hybrid Vehicle (System).InformationDisplay
Type: FlowPort
Last changed on/by: 28/09/2013 17:33:38 VNET\Jpiques
Visibility: [✓ Public] Protected Private Package

Figure 19: Documentation stereotype example

5. Requirement management

5.1. Distributed requirement management

Speaking about requirements in general may lead to adopt wrong requirement management tooling solutions. In fact, initial needs are iteratively refined during the engineering process, producing different levels of so-called requirements, corresponding to very different kind of information. Typically these requirements can be classified in three categories:

- **User requirements** describe the expected services from the end user point of view.
- **System requirements** define the features of the system necessary to fulfill its mission.
- **Component requirements** specify the internal constitutive parts necessary to implement the expected features.

Therefore, believing that a unique tool has the capability to address efficiently these three layers of information is incorrect. On the contrary, a pragmatic approach adopted at Valeo is to take benefits from tools optimised for each field and to make them collaborate efficiently.

Another common mistake is to confuse two categories of requirements related tools:

- **Requirement definition tools** are containers of requirements (or any modeling artefacts used for specification).
- **Requirement traceability tools** do not define any requirements but have the ability to analyze requirements from requirement definition tools, and to verify properties of traceability links.

A tool of the second category (e.g. Reqtify) can therefore be used as a gateway to optimize collaboration between tools of the first category (e.g. DOORS, SysML Artisan Studio, Simulink, …), for synchronizing interface requirements and producing the whole traceability analysis. Another interesting property of this scheme is its ability to let people working with their discipline specific tools (e.g. Simulink for control design).

Classical requirement management approaches assume that all requirements shall be written in natural language inside a centralized database (typically DOORS). Then, SysML modeling artefacts are only considered as intermediary by-products that need to be finally re-written into textual requirements. This process makes sense in the aerospace or railway transportation fields were certification procedures are document-centric by nature. However, in the automotive area, without any constraints from certification procedures, a pure model-centric approach is far more efficient.

In the Valeo approach, maximum benefits are taken from expressive power and semi-formal verification capability of the SysML modeling language. Consequently, requirements or requirements-like artefacts produced during system modeling activities are not reformulated in natural language into an

external centralized database. On the contrary, the model itself becomes the central reference, and the automatically generated documentation only an illustration of this reference. This philosophy is also used at implementation level, where requirements or more exactly requirements-like artefacts remains embedded into discipline specific native models (e.g. Simulink models, for control design).
All the above mentioned principles are summarized on the figure below, showing the typical mapping of the tools used at Valeo.

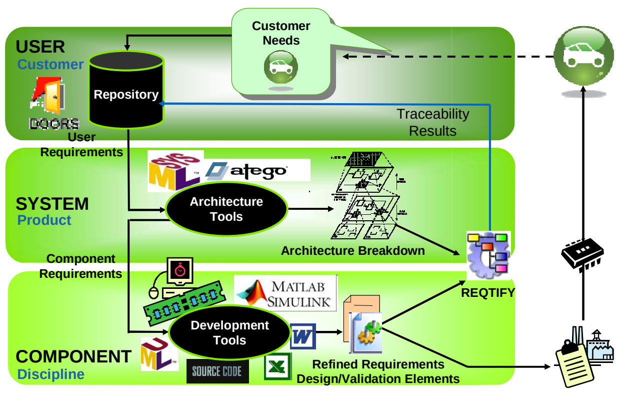


Figure 20: Distributed requirement management

This approach optimises the requirement management effort because requirements are distributed among the tool locations where they have been defined, at each stage of the engineering process. As a counterpart, the consistency of the distributed storage must be supported by powerful traceability tools, with efficient mechanisms for synchronizing requirements at the interfaces between modeling layers.

5.2. Standardized interchange formats

The interface between the system modeling tool and the implementation discipline ones (e.g. hardware, software, mechanics, …) is a critical issue. The system modeling tool providing the component specifications (*CNDs*) for the different disciplines, it is crucial to avoid loss of information and manual reworking of exchanged data. Among the possible alternatives, the approaches independent from the tools are preferred to those using tool-dependent interaction protocols (e.g. specific APIs). From this point of view, a file-based exchange mechanism based on neutral format or standard interchange format is a good answer.
Preferred relevant interchange standards are:

- ***RIF/ReqIF*** (Requirement Interchange Format) to exchange requirements between requirement management or traceability tools,
- ***XMI*** (XML Metadata Interchange format) to exchange system models artefacts between SysML tools (with possible extension to other modeling and simulation tools).

The maturity levels of these two standards are very different. *RIF/ReqIF* format is now mature enough to allow roundtrip exchange with customers on Valeo industrial projects, as illustrated by the figure below. On the other hand, *XMI* format suffers from many weaknesses, not being capable of exchanging diagram contents and some important properties of SysML objects.

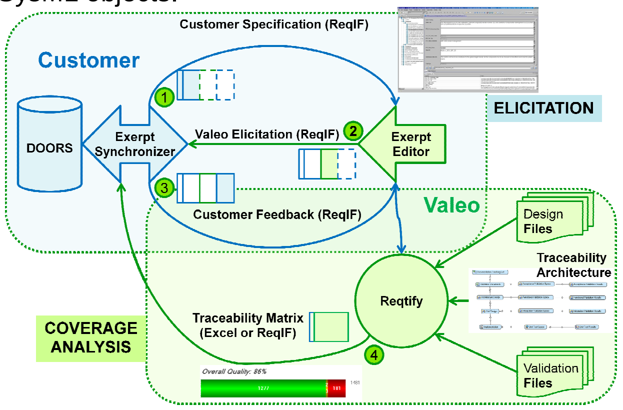

Customer Specification (ReqIF)
Customer
ELICITATION
DOORS
Exerpt Synchronizer
1
Valeo Elicitation (ReqIF)
2
Exerpt Editor
3
Customer Feedback (ReqIF)
Valeo
Design Files
Traceability Architecture
Reqtify
COVERAGE ANALYSIS
Traceability Matrix (Excel or ReqIF)
4
Validation Files

Figure 21: Requirements exchanges and analyses

5.3. User requirements in external repositories

The initial stakeholder requirements (namely user requirements) remain captured in text specifications external to the SysML modeling tool, as in the classical approach. Typically, these specifications are stored in a DOORS database but may also be described using classical word processing or table editing softwares. The combination of the Reqtify gateway and of Artisan Studio modeling tool provides a mechanism to import external text requirements by creating mirroring SysML requirements directly into the SysML model and to later maintain these data synchronized. In fact, three kinds of synchronization mechanisms are available:

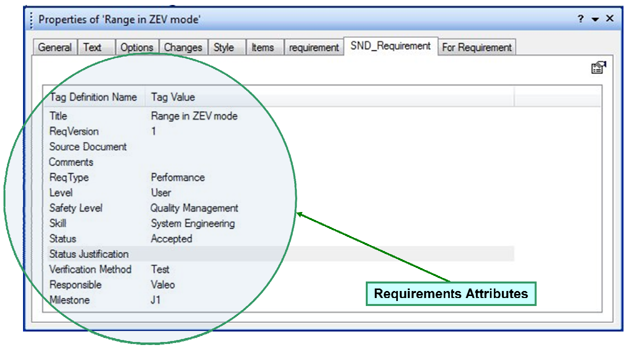
- Synchronization with a DOORS database,
- Synchronization with any kind of requirement file captured with Reqtify,
- Synchronization with Excel files (feature added by the Valeo Profile).

The SysCARS modeling activities performed to analyze stakeholder needs can lead to propose updates to the user requirements baseline. However, the textual requirements are formally updated and controlled in their native external requirement repository and changes are propagated to the SysML model thanks to the synchronization mechanism.

5.4. System and component requirements inside the SysML model

Requirements produced during SysML modeling activities are not reformulated in natural language into an external centralized repository. As a consequence, system and component level requirements are located inside the SysML model,

taking benefits from internal traceability with other model artefacts.
The standard SysML *requirement* object being mainly limited to an identifier and a description field, it has been necessary to add complementary attributes, for efficient requirement handling. The figure bellow shows these additional fields added by the Valeo profile, using *tag definitions*.


Properties of 'Range in ZEV mode'

General | Text | Options | Changes | Style | Items | requirement | SND_Requirement | For Requirement

Tag Definition Name	Tag Value
Title	Range in ZEV mode
ReqVersion	1
Source Document	
Comments	
ReqType	Performance
Level	User
Safety Level	Quality Management
Skill	System Engineering
Status	Accepted
Status Justification	
Verification Method	Test
Responsible	Valeo
Milestone	J1

Figure 22: Stereotyped requirements attributes

As already stated, the use of SysML *requirements* is limited to non functional requirements. Indeed, as often as possible, requirements are represented by SysML artefacts attached to *blocks*; typically by *operations*, *ports*, and *states*. More than avoiding reformulating model artefacts into textual requirements, this approach also saves the cost of declaring traceability relationships between structural elements and related requirements.

5.5. SysCARS traceability model

The traceability model adopted in the SysCARS methodology has been pragmatically defined taking into account the features of the SysML modeling tool and the kind of verification that could be later performed.

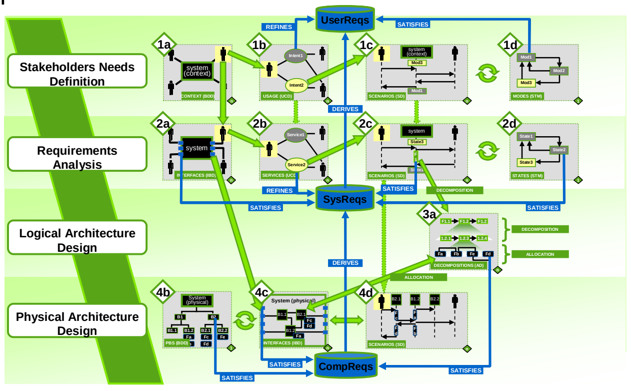


Figure 23: SysCARS traceability scheme

The main rules used for defining traceability relationships are the following:

- **Derive** is used between two levels of requirements,
- **Refine** is used between a *use case* or a *scenario* and the corresponding elicitated *requirement*,
- **Satisfy** is used between a model artefact (i.e. *state*, *port*, *operation*, *block*) and a related (non functional) *requirement*.
- **Trace** is used between two representations of the same item, either refined between modeling levels or reformulated at the same level

Refine and *Satisfy* relationships shall connect artefacts developed at the same modeling stage, while *Derive* and *Trace* relationships are also capable of linking artefacts from different levels.

5.6. Verification and validation of requirements

Verifications of requirement traceability are triggered throughout the whole system engineering process. In fact, two kinds of traceability analyses are performed:

- **Internal traceability analyses** between SysML model artefacts, directly generated using the SysML tool,
- **External traceability analyses**, between the distributed requirement repositories, done using a general purpose requirement traceability tool such as Reqtify.

Internal traceability analyses are the ending activities performed at each stage of the workflow to verify the model consistency (refer to green states of the workflow diagram, [figure 16]). They use *requirement tables* and *traceability matrices* to check the coverage of all requirements by appropriate model artefacts, in accordance with the traceability model presented at the previous paragraph. These matrices and tables are generated on demand at Excel format.
By parsing the SysML database, external traceability analyses (performed with Reqtify) can also automatically verify the consistency and the completeness of the model, in accordance with the SysCARS traceability scheme. The verifications are based on the analysis of SysML artefacts and related relationships and stereotypes, including coverage links to external documents or requirement repositories.

6. Coupling to control design tools

The issue of coupling a SysML tool to discipline related tools (and particularly simulation tools) is not studied in general but limited to coupling to control design environments, and particularly to Matlab/Simulink.

6.1. Specification rather than co-simulation

Some approaches promote to use the SysML model as an integration framework for building a whole executable system model, in order to analyze the dynamics of the system. To support this, the static system modeling environment must be upgraded by execution mechanisms, with closed connection to discipline specific simulation tools.

This way has not been chosen at Valeo's for several reasons:

- A higher degree of sophistication of the SysML environment would go against a wide adoption by (generalist) system engineers,
- Somehow, there is a contradiction between flat deep detailed modeling and the layered refinement approach promoted by system engineering,
- Simulation and co-simulation capabilities of SysML tools are quite limited compared to those of domain specific tools,
- For large scale system, a full integration simulated model is practically intractable.

The final objective being the verification and validation of the whole system model, a static verification of traceability properties, as discussed in previous paragraphs, has been preferred. The purpose is then to gain maximal confidence in the completeness of the intellectual progress which led to the physical architecture solution.
In a second time, as explained in the next paragraph, each component will be efficiently refined (and possibly simulated) independently in its discipline related development (and possibly modeling) environment, based on input data from the system model.

6.2. Transfer of structural descriptions to Simulink

The problem of collaboration between SysML and Simulink is not stated in terms of (co)simulation but rather in terms of efficiently transferring and synchronizing modeling data between both environments. The synchronization at architecture description level was proven to be an efficient way to transfer information between system engineering teams and control design teams.

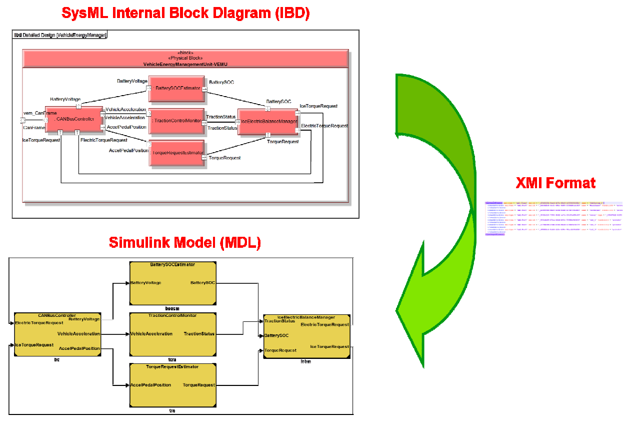


Figure 24: Synchronization between SysML IBD and Simulink MDL

As illustrated by the figure above, the approach selected was to transfer the IBD structural descriptions of control law components, from SysML towards Simulink. The resulting Simulink models, initially corresponding to empty structures are afterward refined, and control algorithms implemented, simulated and validated inside the Simulink modeling and execution environment.
Artisan Studio natively provides the main features required to synchronize and update SysML structural models and Simulink models: changes can be propagated in both directions. However, extensions in the existing mechanisms would be necessary for a full interoperability between both environments. The suggested evolutions are presented in the next paragraph. Moreover, using the XMI interchange format would be preferred to Artisan Studio proprietary mechanisms.

6.3. Mapping between SysML and Simulink structural artefacts

The table below presents the detailed mapping for an efficient synchronization of structural descriptions between SysML Internal Block Diagrams and Simulink Dataflow models. This mapping has been verified thanks to a mock-up implemented under Matlab/Simulink. Currently existing features of Artisan Studio are written in standard font, while suggested extensions are written with bold characters.

SysML	Simulink
Internal Block Diagram	MDL File
Block	Model Reference
	Sub-system
Flow port (in)	Inport
Flow port (in) + "control" stereotype	*Trigger port*
Flow port (out)	Outport
Flow port (out) + "control" stereotype	*Outport + Function-call*
Connector	Connector
Item flow	*Signal name – connector name*
Requirement with "Satisfy" link to a block or a port	*DocBlock inside the corresponding sub-system*
Block description	*DocBlock inside the corresponding sub-system*
Block + "Mux/Demux" stereotype	*Mux/Demux*
Statechart attached to a block	*Stateflow attached to a MDL*

Figure 25: Mapping between IBD and Simulink

The main mandatory evolution required is related to the ability to deal with Simulink events and not only with continuous flows. Indeed, events are systematically used to specify control flow mechanisms of algorithms. The solution selected was to add a "control" stereotypes to SysML *ports*, in order to make a distinction between control flows and data flows.
The ability to transfer names to Simulink connectors is also mandatory, because in most situations they are used as variable names by automatic code

generation tools. The selected solution was to use the name of item flows, but this is not completely satisfactory for exchanging Mux/Demux signals, and evolutions of SysML 1.3 (and later) in this sense would be welcome.
The possibility to transfer requirements from SysML to Simulink could be emulated by creating Simulink documentation blocks at the appropriate sub-system level.
It would be also potentially very interesting to transfer information related to the expected behaviour of the algorithm. For that purpose, SysML state machines could be translated into Simulink Stateflows. Limitation of the semantics of transferable state machines could be tolerated.

6.4. Necessary evolutions of XMI format

The efficient bi-directional synchronization between a SysML model and a Simulink model is not only a matter of tools. Indeed, there remains blocking issues due to limitations and weaknesses of the XMI format. Among possible good ideas, it would be welcome to attach GUID to objects to allow unique identification and synchronization between tools, or to systematically store object description fields (with possibly not only ASCII characters!).

7. Conclusion

As a conclusion, experiences from Valeo pilot projects and more recently from industrial projects have confirmed that the SysML language provides an adequate lever to extend the modeling practices to the area of System and Product Engineering. Valeo's experiences have shown that a successful approach requires a precisely defined modeling methodology (SysCARS) but also a solid training course and the sponsoring of the organization. Furthermore, the customisation of existing tools in a workflow driven mindset is mandatory. Further improvements remain necessary on commercial tools regarding ergonomics and interfacing with simulation and safety analyses tools. And last but not least, we are convinced that the sharing of a commonly agreed data model describing the System Engineering concepts (such as "functions" or an "interfaces") independently from any language or tooling solution, remains a key enabler for System modeling adoption.

8. References

[1] Eric Andrianarison, Jean-Denis Piques: *"SysML for embedded automotive Systems: a practical approach",* ERTS 2010.

[2] Eric Andrianarison, Jean-Denis Piques*: "SysML for embedded automotive Systems: lessons learned"*, ERTS 2012.

[3] Françoise Caron: *"Exigences et ingénierie système: Mise en œuvre avec SysML"*, EIRIS Conseil, 2008.

[4] Françoise Caron: *"A collaborative process based on systems engineering and mechatronics methods"*, 22th Annual International INCOSE Symposium, 2012.

9. Acronyms

AD	Activity Diagram
BDD	Block Definition Diagram
CND	Component Needs Document
COTS	Commercial Off-The-Shelf
DSL	Domain Specific Language
EIA 632	System Engineering Standard
GUI	Graphical User Interface
GUID	Globally Unique IDentifier
IBD	Internal Block Diagram
IEEE 1220	System Engineering Standard
ISO 15288	System Engineering Standard
ISO 26262	Automotive Functional Safety Regulation
MBSE	Model Based System Engineering
MDL	Simulink file extension
MoE	Measure Of Effectivness
MoP	Measure Of Performance
OMG	Object Management Group
REQ	REQuirement Diagram
RIF/ReqIF	Requirements Interchange Format
SD	Sequence Diagram
SND	Stakeholders' Needs Document
STM	STate Machine diagram
SyDD	System Design Document
SyRD	System Requirements Document
SysCARS	System Core Analyses for Robustness and Safety
SysML	System Modeling Language
UCD	Use case Diagram
XMI	XML Metadata Interchange