*Report*



**OpenES Project**

***Project Full Title:***

**“Open ESL Technologies for Next Generation Embedded Systems”**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | WP no. | | Deliverable no. | Lead participant |
| **WP1** | | **D1.2.2.b** | **ST** |
| **System modeling and uses cases v2** | | | | |
| Prepared by | | **ST: Alain CLOUARD, Sebastien REVOL (editor) CEA: Arnaud CUCCURU TCS: Aurélien BERHAULT CISC: Ralph WEISSNEGGER** | | |
| Issued by | | **Sebastien REVOL (ST)** | | |
| Document Number | | **OpenES/WP 1/D1.2.2.b** | | |
| Nature  Dissemination Level | | **Report**  **CO** | | |
| Submission Date | | **2014-10-03** | | |
| Due Date | | **2014-09-30** | | |
| **Project co-funded by eureka-logo.jpg CATRENE Programme (2013-2016)** | | | | |

© Copyright 2013 STMicroelectronics srl., Thales Communications & Security and Thales Research & Technology SA, NXP Semiconductors, Synopsys, CEA-LIST, Magillem Design Services SAS, CISC Semiconductor GmbH, Docea Power, Vector Fabrics B.V., University of Technology Eindhoven, UJF-Verimag, European Electronic Chips & Systems design Initiative.

**History of Changes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ED. | REV. | DATE | PAGES | REASON FOR CHANGES |
| AP | 1.0 | 2013-10-09 | 10 | Initial version |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

* The “Revision” is the document version number:
* V0.5 – project internal review
* V0.7 – modifications after project internal review
* V1.0 – initial draft released to the consortium
* V2.0 – final version delivered to the CATRENE review.

**<config services=`ExtConfigService`>**

**<output path='${outputPath}/Deliverable\_1\_2\_2b.docx' />**

**<param key='model' value='${inputPath}'/>**

**</config>**

**<config services=`ExtLogger`/><drop/>**

**<context model='${model}' />**

<bookmarks>

<alias source='reqID' target='[req.getReqID()/]'/>

</bookmarks>

Contents

[1 Introduction 4](#_Toc369093560)

[2 General formatting rules 5](#_Toc369093561)

[3 Empty Chapter 8](#_Toc369093562)

[4 References 9](#_Toc369093563)

[A. Nothing interesting here 10](#_Toc369093564)

# Introduction

The objective of WP1 activities is to define the modeling primitives that are part of the OpenES System Modeling Kit (OSMK), each task focusing on a particular modeling concern: the system level modeling and use case descriptions for T1.2, the extra-functional properties for T1.3, and the subsystem definition and integration aspect for T1.4.

Task 1.2 is mostly dealing with the modelling primitives required to enable the development of a system-wide model. It addresses the following concerns:

* Concern 1: The appropriate high level modelling language to represent all required system information and avoid multiple capture of the same information.
* Concern 2: The representation of system use cases in order to provide a link to the verification activities executed in WP3.
* Concern 3: Model refinement and connection to model generation techniques developed in WP2. They are considered to connect to traditional design flows and tools.

This document is part of Deliverable 1.2.2.b, and focuses on Concerns 1 and 3 mentioned above. It is the result of regular and fruitful interactions between WP1 partners, on the basis of weekly teleconferences. In addition to this report, Deliverable 1.2.2.b also includes a UML profile and some model examples, which are further described in section 3 of this document. These models are used to illustrate how the OSMK can be used to address OpenES requirements related to System Level Modeling aspects (D1.1).

In section 2, the document starts by stating important definitions related to modeling abstraction levels considered by the OSMK. In section 3, the various model examples provided by WP1 partners are introduced. In section 4, the document reports on the satisfaction of OpenES requirements, with references to model examples of section 3, and explanations about how this examples demonstrate that the requirements are satisfied. It is important to note that section 4 has been automatically generated by applying the flow described in OpenES deliverable D1.2.3.a (System Level model refinement and generation).

# OSMK Abstraction Levels

As explained in the introduction of this document, the objective of WP1 activities is to define the modeling primitives that are part of the OSMK, each task having in mind a particular aspect: the system level use cases description for T1.2, the extra-functional properties aspect for T1.3 and the subsystem definition and integration aspect for T1.4.

However, this separation of concerns (chosen to structure the WP1 activities) is orthogonal to an “abstraction-level-based” classification that naturally appears in a design flow. For instance, the “Y” chart defined in the first WP1 activities (requirements definition) is proposing a first specification of the OpenES design made of a sequence of refinement steps. It is depicted in Figure 1 below.

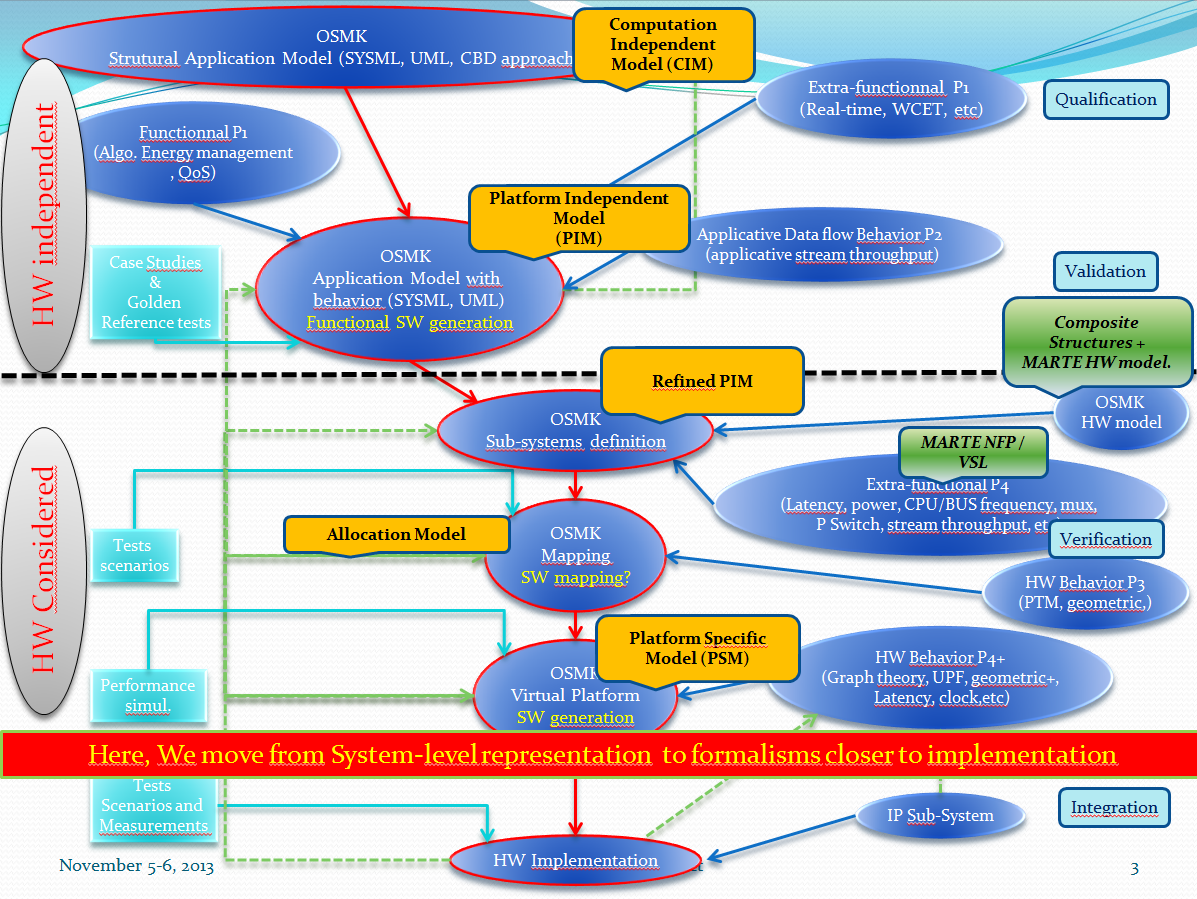


Figure 1 Abstraction level positionning in OpenES Design Flow

For each of those steps, the different aspects addressed by the different tasks (Use cases, NFP, subsystems) have to be considered in connection with abstraction-level concerns. A clear definition of what those abstraction levels are about is required, and it has to be shared between all the WP1 tasks.

Notice that the abstraction levels concepts used in OpenES flow are taken from Model Driven Architecture methodolgies : CIM (Computation Independent Model), PIM (Platform Independent Model), PSM (Platform Specific Model) etc. However, we realized that those concepts had no real formal definitions, and needed to be better specified in the context of OpenES. In in Figure 1, showing how those concepts are positionned in OpenES design flow (orange boxes). The next sections provide definitions for each abstraction level considered in the OpenES design flow.

## Computation Independent Model (CIM)

The CIM aims at providing a System Level view, mainly focusing on its functional structure. It does not specify any information on how the functionality will be implemented. In particular, there is no hardware or software identification. Moreover, this kind of model is not precise enough to be executed.

Despite there is no explicit or implicit model of computation, the CIM can include system level use cases showing synchronous and asynchronous communications between the different functional blocks.

The CIM can also be composed of several structural refinements, which can be done through a hierarchical decomposition of functional blocks.

We consider that CIM is too abstract to specify any extra-functional property.

In addition to this, we also include in CIM the UML representation of requirements, allowing then to link those requirements to UML model elements.

Identified diagrams for the representation of CIM aspects:

* UML Composite Structure with ports and connectors for the functional structure description
* UML sequence diagrams / use case diagrams / communication diagrams for the use cases descriptions
* SysML requirement diagrams for requirements

## Platform Independent Model (PIM)

PIM includes CIM capabilities, with additional behavioral models like UML state machines and activities. Moreover, we consider that functional and extra functional properties can be expressed at this level of refinement, with the use the MARTE Value Specification Language and UML constraints.

PIM are not necessarily fully executable models: only some parts of the full system can be detailed more precisely. If the behavior is not directly expressed in the model, it can include references to existing implementation language specific code (C/Matlab/SystemC etc.)

It implies that the different behavioral parts of the system can be defined with heterogeneous Models of Computation. In that context, the goal of OSMK is NOT to provide different methodologies to describe behavior for each model of computation, but rather focusses on the integration aspect, to provide a way to encapsulate and express interactions between heterogeneous executable functionalities.

Identified diagrams:

* In addition to CIM diagrams: UML State machines and activities for behavior, or equivalent textual notation with ALF. UML *Opaque behavior* can be used to reference external implementation specific languages behaviors, possibly using some annotations (as defined in the OSMK::Annotation sub-profile, see section 3.1).
* SysML constraint blocks can also be used for equation-like specifications.

## Refined PIM

This abstraction level has been explicitly identified to fit with the “OSMK Sub-systems definition” concept. It consists in a functional decomposition of the Platform Independent Model with a granularity detailed enough to allocate each of its block onto a single hardware or software execution resource. In other words, a PIM functional block can’t be allocated on several execution resources. It should be split before into the different sub-functionalities, and their communication interfaces should be identified before mapping them onto different execution resources.

## Platform Specific Model (PSM)

The PSM encompasses several aspects. It should first contain elements that will describe the execution platform, including Hardware and Software execution resources. On hardware side, the model can contain low level details, like registers and memory map information. On software side, the execution platform description can specify OS specific information, like tasks, scheduling algorithms or middleware services.

Complementarily to those platform description aspects, the PSM can contain a new refinement of the PIM model where platform independent functional components are transformed into platform specific components, with explicit references to the execution resources services.

This PSM part can be partially generated from the **Allocation Model** described below**.**

Identified diagrams:

* MARTE profile proposes several chapters to describe HW and SW execution platforms. In addition to that, ST ESL profile is proposing complementary concepts allowing the description in UML of IPXACT related concepts like HW communication protocols identification and registers descriptions.
* Other language-specific UML profiles can be part of PSM, like the UML for SystemC profile which allows expressing detailed SystemC concepts at UML level, thereby simplifying code generation aspect.

## Allocation Model

The allocation model is an intermediate step between the refined PIM and the full PSM. It expresses how refined platform-independent functional components can be allocated onto HW or SW execution resources. It implies that part of the PSM already exists, to identify and reference those execution resources. It can be the input of extra-functional properties analysis tools to verify if a given mapping will allow meeting expected NFP constraints. It can also be the source of code generation or model transformation to obtain detailed platform-specific application model.

Identified diagrams:

MARTE and SysML profiles are both proposing allocation concepts. They can by several ways, including dedicated dependencies (arrows in structural diagrams) or in allocation tables.

# WP1 models presentation

This section is presenting the different models delivered with this report. Some of those models have been made specificly for this task as generic representative models, and some other, mainly from CISC and TCS, have been extracted from their current OpenES WP4 case study.

In any case, each of those models have been made according the OpenES Modeling Kit (OSMK) guidelines. Notice that all those models are part of this deliverable and can be viewed with Papyrus tool. The install procedure is decribes in the Appendix A of this document.

## OSMKProfile

The OSMK profile is a UML profile that mostly relies on the UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), and the System Modeling Language (SysML) profile. The OSMK profile is used within OpenES for the design of system-level models. The architecture of the profile is depicted in Figure 2.

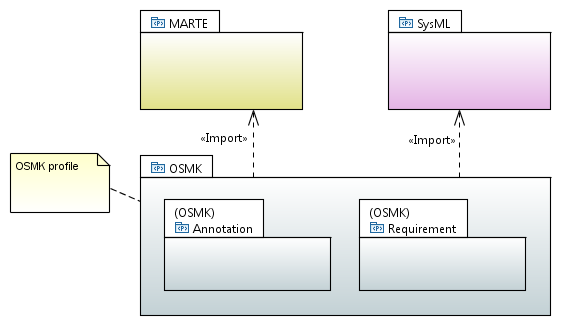


Figure 2. OSMK Profile Architecture

By relying on UML 2, and by importing MARTE and SysML profile, the OSMK profile implements the conclusion of the OpenES report about modelling language selection:

* Architecture and functional modelling: OSMK requirements call for the ability to decompose system level models into blocks, with explicit connection points. These multiple blocks can be assembled through appropriate connections so that their composition (and the global system behaviour emerging from this composition) provides a model of the global system. OSMK requirements also call for the ability to describe blocks hierarchically, where a block may itself be composed of other blocks, connected together, and so forth. These modeling requirements are addressed by UML Composite Structures. The term Composite Structure refers to the ability of UML classes (or components) of having explicit connection points called Ports, with explicit provided/required interfaces (which formalize a contract between each component and its environment), as well as having an internal structure composed of parts, assembled together with connectors. UML interfaces of ports have a client/server flavor (i.e., possibility to expose operations and/or receptions for signals), which means that components communicate together with communication actions such calling an operation or sending a signal. MARTE and SysML introduce data-flow oriented extensions to UML ports. Instead of calling an operation or sending a signal, one can simply write a value on a port, this value being propagated to other connected components (like in classical data-flow oriented languages). Note that the semantics of UML composite structures are being formalized in the context of PSCS (Precise Semantics of UML Composite Structures PSCS [5] ), an ongoing OMG standardization effort partially funded by OpenES. This specification also includes annexes about semantics of MARTE and SysML extensions, globally providing a strong and formal basis for the OpenES design flow.
* Characterization of system-level models with extra-functional aspects: OpenES requirements related to extra-functional aspects call for the ability to enrich system-level models (i.e., individual components, assembly of components, connections, etc.) with extra-functional information, in support of analysis and design activities such as simulation for architecture exploration, requirements traceability, etc. This information can be used at different steps of the design flow, and can represent constraints on the actual design (e.g., a specific task shall be performed in less than a certain amount of time, the global system shall consume less than a certain amount of energy while operating), as well as actual characterization of system elements that are known (e.g., they have been computed / estimated / measured on actual IPs) and that must be considered for the design choices (e.g., a bus is able to support a given bandwidth). These requirements are addressed by the normative MARTE Non-Functional Property (NFP) framework. More details about these aspects are given in deliverables related to task 1.3.

In addition to the modelling capabilities brought by UML, MARTE and SysML, the OSMK profile introduces specific mechanisms for modelling of Annotations and Requirements. They are depicted in Figure 2 by packages Annotation and Requirement:

* **The OSMK Annotation sub-profile** provides a very generic and open mechanism to annotate UML\* model elements. This mechanism has been proposed as a flexible solution to OpenES requirement 5 (“OSMK must provide a way to identify and categorize the application and architecture models”). It simply relies on collections of key/value pairs, which can be attached to any model element, for any purpose. From a methodological perspective, the mechanism however encourages reuse and sharing of information, since keys are not “free” strings, but literals formally defined as part of Enumerations. These Enumerations can then be shared by partners through common model libraries, avoiding the specification of inconsistent keys and problems in the automation of the flows. Deliverable 1.2.3.a describes how this annotation mechanism has been used for the generation of section 4 of this document.
* **The OSMK Requirement sub-profile** enables the definition of requirement traceability models which are orthogonal to both requirement models and models that provide elements satisfying those requirements. By orthogonal, we mean that the definition of a satisfaction relationship between a requirement and a model element can be done in a separate (i.e., orthogonal) model, without any pollution in the source models. We explain in Deliverable 1.2.3.a how this mechanism can be used for collaborative management of requirement traceability, and how it has been used for expressing the traceability relationships between OpenES requirements and model examples described in the following sections.

## Requirements Model

The Requirements Model is a SysML representation of OpenES requirements delivered in D.1.1 at the beginning of the project. It is composed of two main parts which have been fully generated thanks to model transformations implemented by ST (cf deliverable D1.2.3.a for more details on that flow).

The first part contains the SysML requirements themselves. As shown in Figure 3 they are represented as UML classes, stereotyped with the <<SysML::Requirements::Requirement>> stereotype. The ID and the text of each requirement is contained in the SysML Requirement ‘id’ and ‘text’ tagged value. If a comment was specified in the excel table, a corresponding UML comment was also generated. Moreover, we also used the OSMK annotation mechanism introduced in [OSMK profile](#_OSMKProfile) to associate each reaquirement to a given WP1 task.

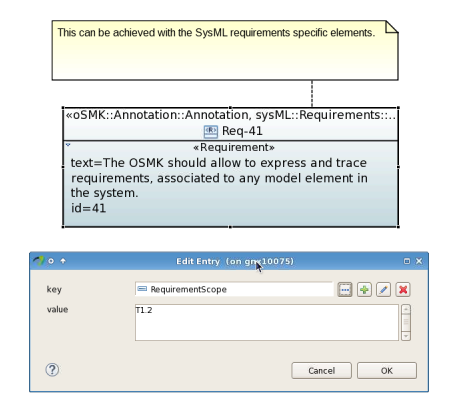


Figure 3 Example of OpenES requirement

The second part of this requirements model is the traceability package. It contains SysML <<Satify>> relationships between SysML requirements and examples coming from all the other models presented in this document. Each links express the fact that the referenced example is satisfying the referenced requirement. This part is generated by a “merge process” described in D1.2.3a, gathering into this single model traceability information contributed by each partner in their respective traceability model accompanying their example models.

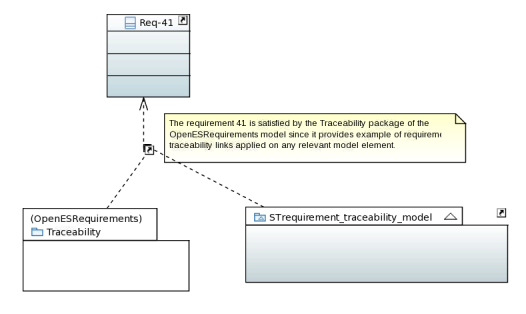


Figure 4 Example of "Satisfy" relationship

Last, Papyrus also allows to represent SysML requirements in a specific Excel-like Table view which provides a good overview of all the requirements and their “satisfiedBy” relationship. Notice that unlike excel, this table is fully dynamic : if name of a model element is modfied, or if a “satisfy” relationship is added in the model, the table is immediately updated and systematically up to date.

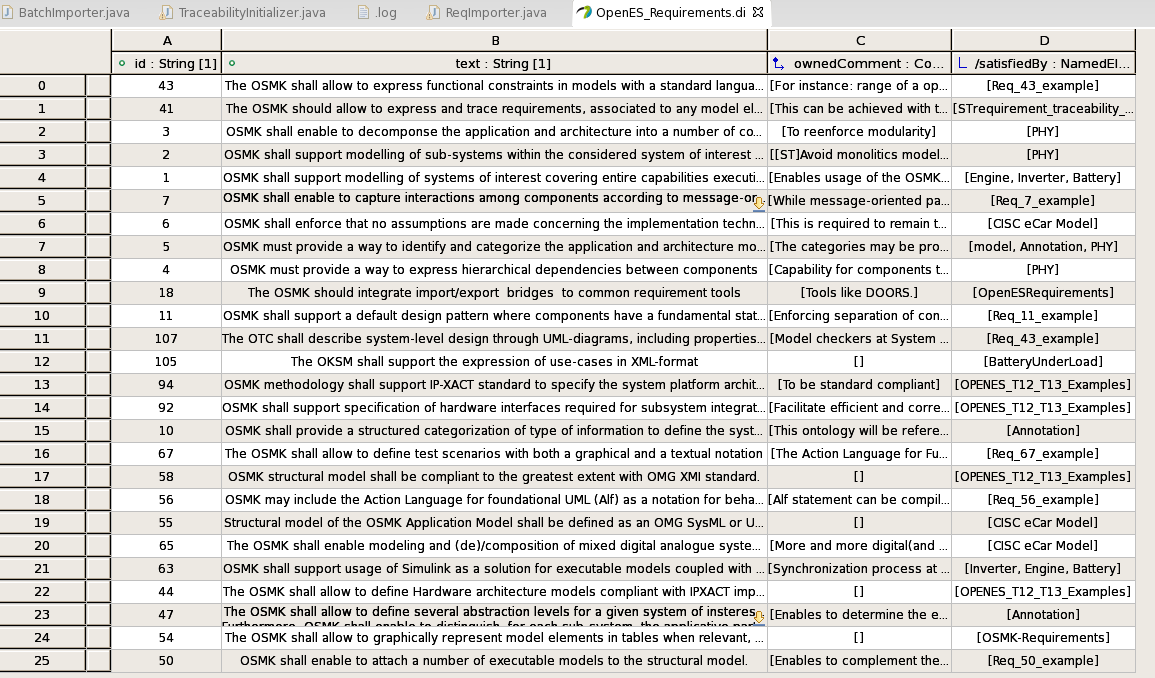


Figure 5 SysML requirement view

## Common Model

This model composed of 3 main diagrams.

The first class diagram has been created to define the main elements composing the system.

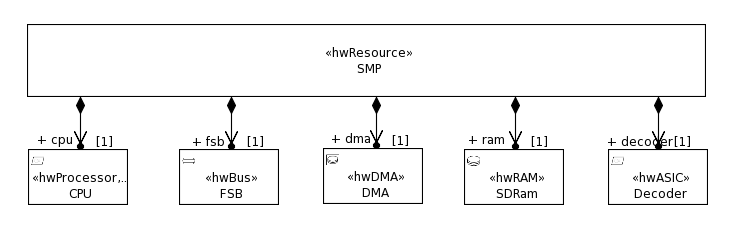


Figure 6 Class diagram

Each box is a class, that will indeed represent an Hardware component. The association between each class provides a first overview of containment relationship which will be much more explicit in the following composite structure diagram.

Notice that each class has been stereotyped with a stereotype coming from MARTE::MARTE\_DesignModel::HRM (Hardware Resource Modeling) profile.

For instance the class CPU has been stereotyped “HWProcessor”. It allows to specify additional attribute in the Properties/Profile view as shown below

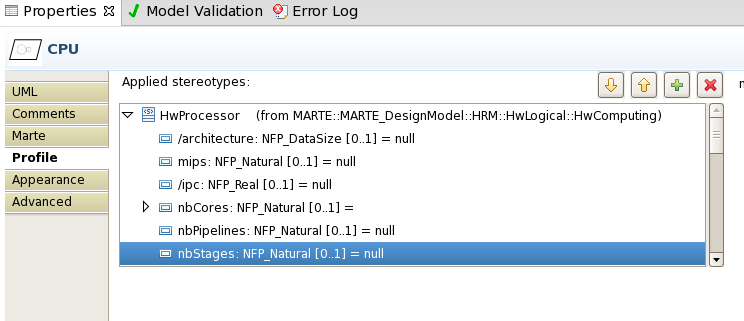


Figure 7 HWProcessor stereotypes details

In addition to MARTE/HRM stereotypes, a specific stereotype coming from ST ESL profile has also been applied to add on each component IPXACT related information missing from MARTE and Standard UML

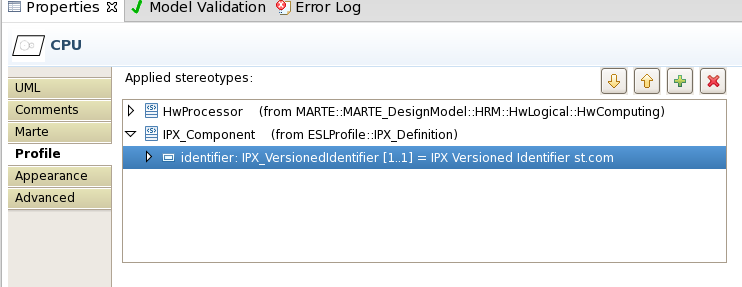


Figure 8 IPX\_Component Stereotype details

The second diagram defined in the model is a composite structure diagram. In that diagram, the containment relationship expressed between the *SMP* system and the CPU, FSB, DMA, SDRam and Decoder are much more explicit. Moreover ports of components instances are also present, as well as connectors between ports.

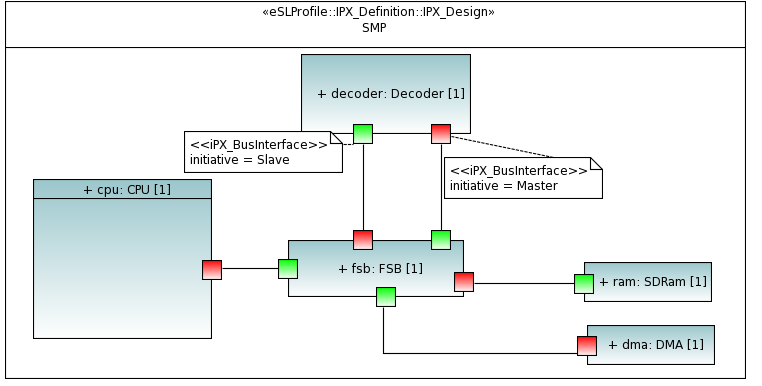


Figure 9 Composite structure of the system

Notice that a specific stereotype called *IPX\_Design* is applied on the SMP system. It identifies this specific class as an IPXACT design, allowing for instance to associate the IPXACT-specific identifier (Vendor, Library, Name , Version).

Another stereotype called *IPX\_BusInterface* has been applied on each component ports. Unlike UML, MARTE or SysML specialized ports, it proposes the IPXACT specific concept of *initiative*, which can be *Master, Slave or System*. It is different from in/out/inout ports, since in reality a Master or Slave IPXACT bus interface is a macro port composed of a set of in/out/inout ports. It is providing an higher abstraction level, adapted to System Level descriptions.

Thanks to those additional annotations, and as a reference for further developments that will be realized in Task 2.2, we have developed a custom script allowing the generation of the equivalent IPXACT description of each component and of the SMP design.

The generated IPXACT descriptions are part of this delivery and can be found in the “IPXACT” folder included in the archive.

The figure below show the resulting design displayed in Magillem GUI.

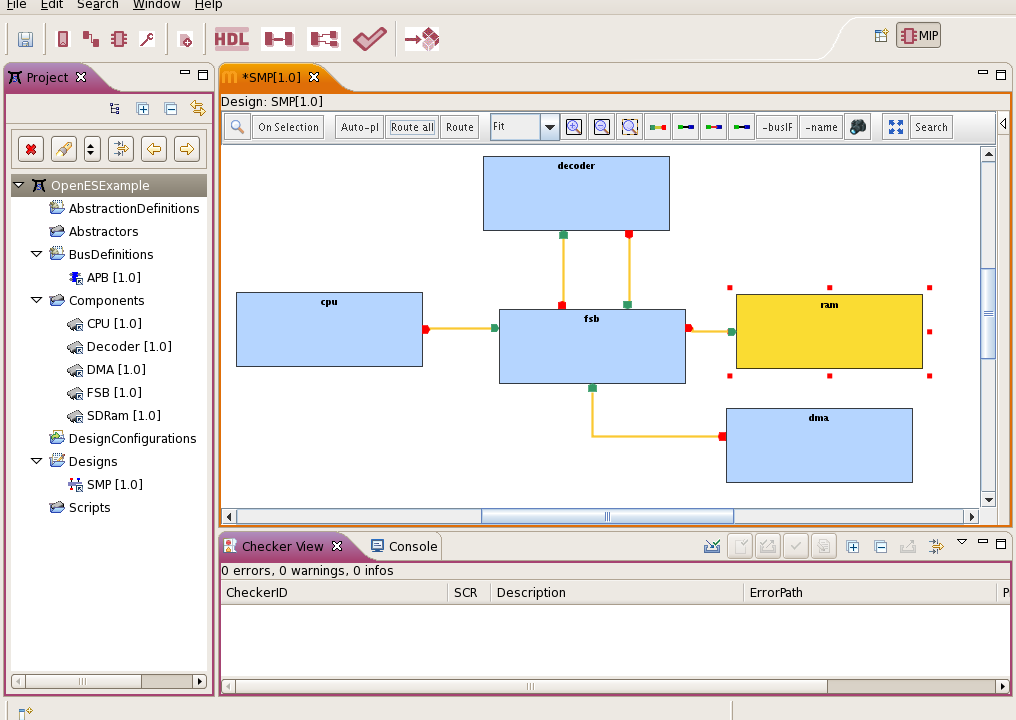


Figure 10 Magillem view of the generated IPXACT descriptions

We can see in *“Components”* folder of the left panel that each Class with a IPX\_Component stereotype allowed to generate an IPXACT component. As mentioned above an IPXACT Design has also been generated from the SMP Class having the IPX\_Design stereotype.

The last diagram show in the picture below is a sequence diagram. It has been introduced to describe a typical use case of the system.

We can see a lifeline corresponding to each component implied the use case. Notice that the communication bus *FSB* is not part of this use case. It would have implied to transform each interaction between the components by a read(address, data)/write(address, data) transaction.

Instead of that, we have introduced high level UML operations describing the service offered by the components. For instance, the DMA is proposing a “*startTransfer*” and a “*configure*” operations, while the decoder is proposing a “decode” high level function.

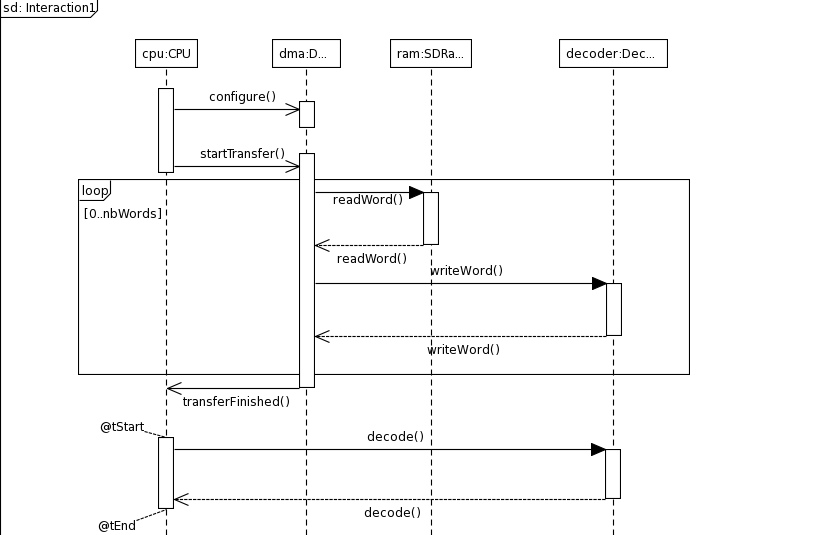


Figure 11 Decode Use case sequence diagram

## CEA Model

CEA examples contributed by CEA are decomposed into several models. The global architecture is depicted in Figure 2.

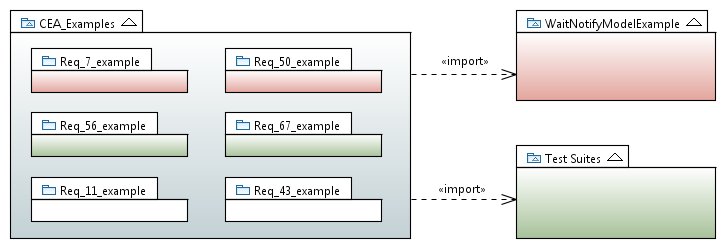


Figure 12. Architecture of CEA model examples

The main model is represented by the big package on the left hand side of the figure. It is called CEA\_Examples. Packages represented in white (Req\_11\_example and Req\_43\_example) are examples actually contributed by this model. Packages represented in red (Req\_7\_example and Req\_50\_example) reuse material provided by model WaitNotifyModelExample. Packages represented in green (Req\_56\_example and Req\_67\_example) are designed by reusing material from model Test Suites. The following sections provide an overview for each of the model. Details on how these models actually demonstrate that OpenES requirements are covered are given in section 3.

### Wait Notify Model Example

The Wait Notify Model Example includes a simple application model, consisting of two processes (A and B, as depicted in Figure 3) communicating and synchronizing according to a simple wait/notify protocol. This model is executable according to the normative semantics of PSCS. The screenshot behind Figure 3 indeed shows an execution snapshot where A is about to send a Notify signal, and B is actually waiting for it. This example is used to demonstrate that OSMK is able to cover message-based communications (Requirement 7) as well as executable modelling (Requirement 50).

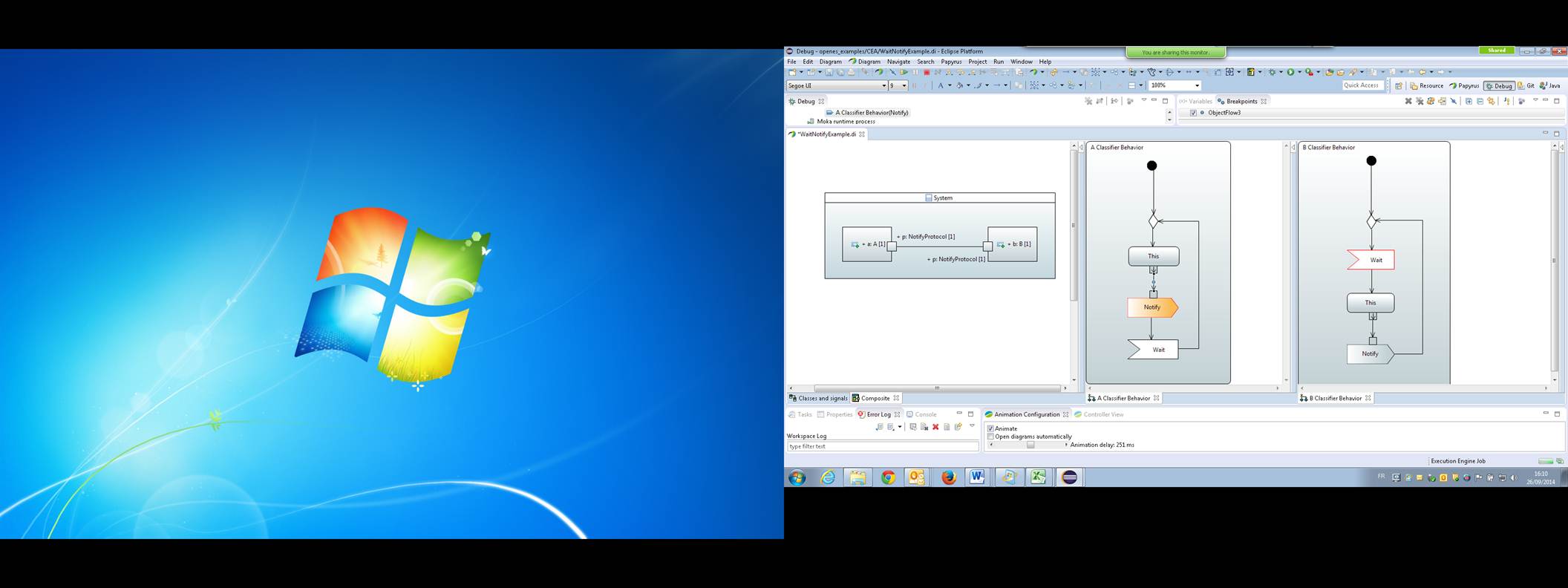


Figure 13. Structural and Behavioral aspects covered by the Wait Notify example

### PSCS Test Suite

The PSCS Test Suite model is an excerpt of the normative PSCS test suite, which has been mostly contributed by CEA in the frame of the ongoing PSCS standardisation process. This suite of test cases can be used by tool vendors to demonstrate conformance to this specification, since each test case is defined as a combination of executable model elements. The test suite covers several key aspects of PSCS semantics, as depicted by the four packages of Figure 4 (Instantiation, communication, invocations through ports, destruction).

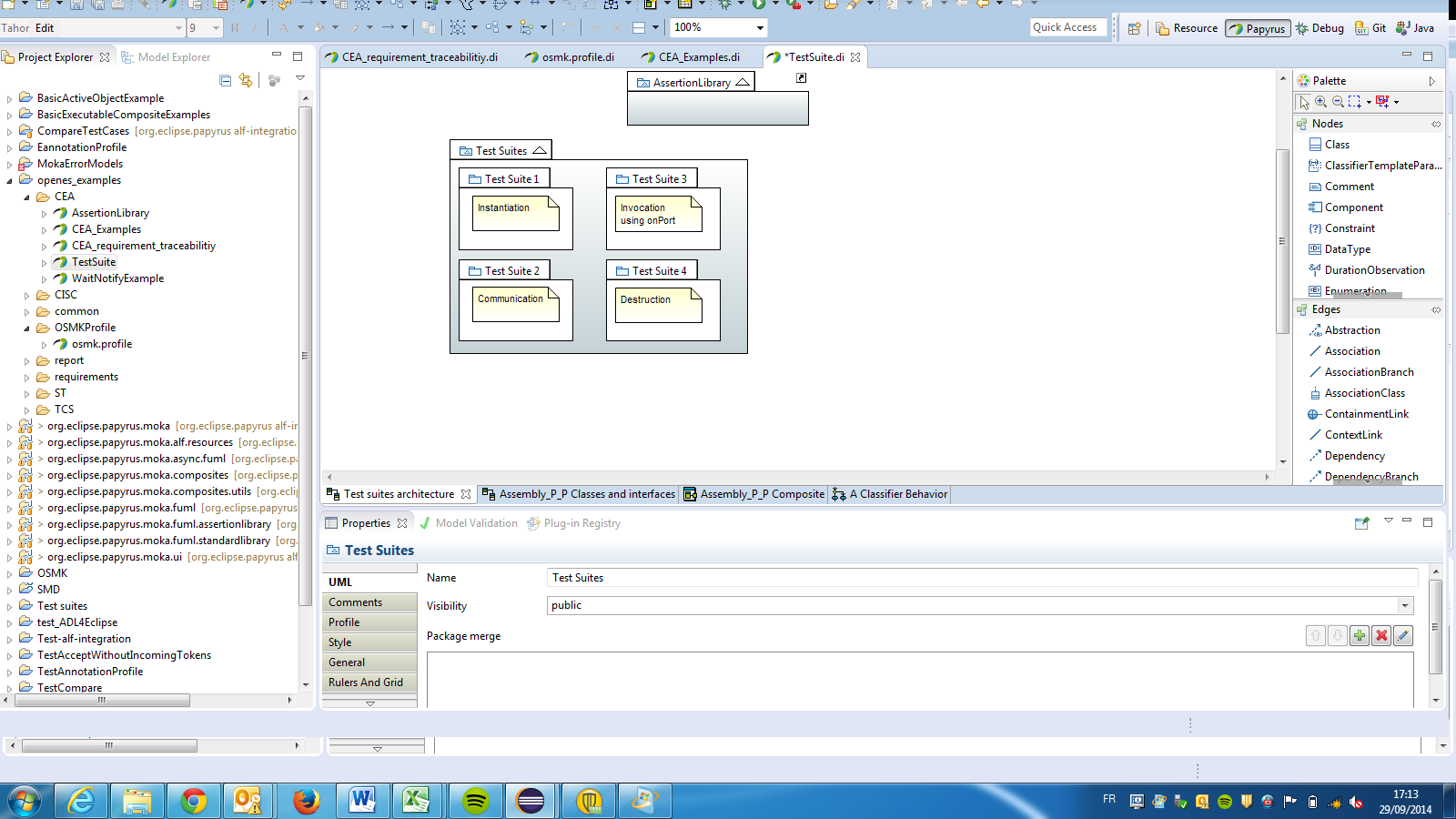


Figure 14. Architecture of the PSCS Test Suite model

Test cases consist in combinations of structural and behavioral elements, whose executions shall verify some assertions. Assertions are specified using basic functions specified in a library, which is itself an executable UML model. A tool may demonstrate compliance with this specification if no assertion fails when executing the test cases. Structural aspects are depicted with appropriate structural diagrams (classes and composite structures), while behavioral aspects are either graphically using activity diagrams, or textually using Alf. Since PSCS (as a subet of UML) and Alf are part of OSMK, this model example demonstrates the capability to satisfy both requirements 56 (inclusion of Alf in OSMK) and 67 (definition of test scenarios by combining graphical and textual notations)

### Other examples

Packages Req\_11\_example and Req\_43\_example include examples demonstrating the coverage of OpenES requirements 11 and 43. The example related to requirement 11 demonstrates the capability of the OSMK to represent state machines. This is illustrated in Figure 5.

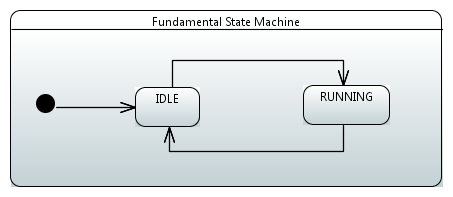


Figure 15. Representation of a foundamental state machine, as requested by Requirement 11

The example related to requirement 43 illustrates the usage of OCL and UML constraints, to constrain the range of possible values on the parameters of an Operation. It is depicted in Figure 6.

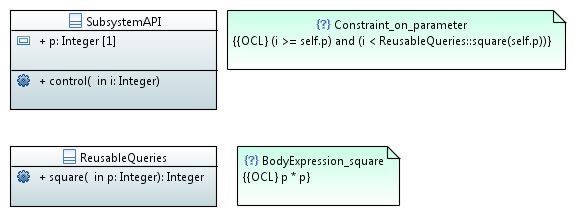


Figure 16. Constraining parameters of operations with OCL, as requested by requirement 43

## CISC Model

The eCar model in UML illustrates the essential component-based construction capabilities of the OpenES modeling framework. Below is a diagram illustrates the major electrical parts of a modern car. As the diagram illustrates the car combines digital and analog components in a single system.

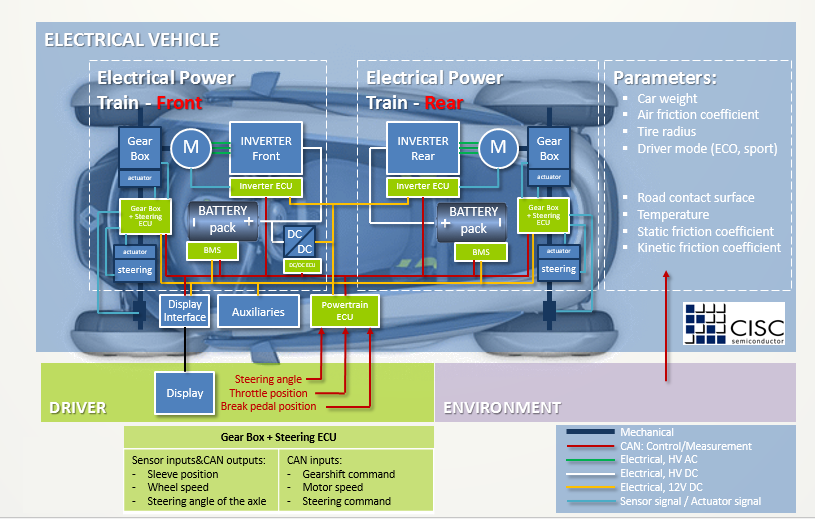


Figure 17: CISC Electrical vehicle

The UML modeling techniques allow abstracting from details, yet capturing essential structural and behavioral properties of the system. In particular, the class diagram of the eCar resource is shown in the figure below, which is followed by a structural diagram.

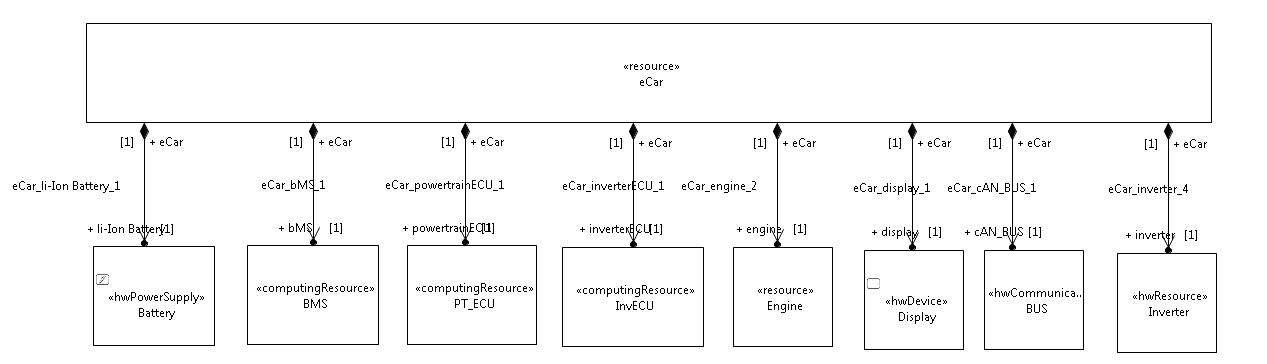


Figure 18: eCar class diagram

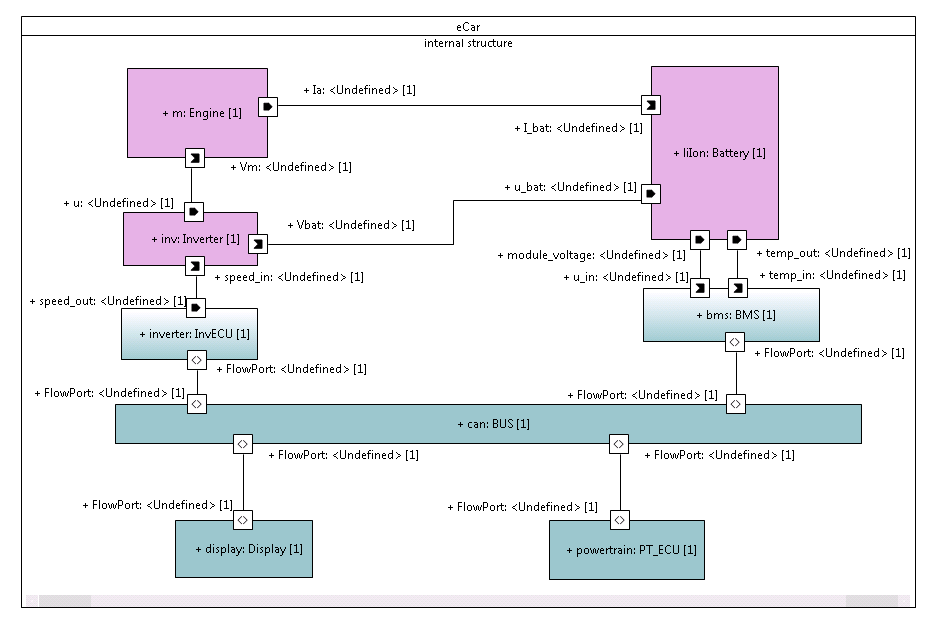


Figure 19: Structural diagram of eCar

Note, that this UML model illustrates the ability to specify analog components, e.g. the Battery. This ability is stated by OpenES requirements #65 and #66. Another remarkable feature of the OpenES UML modeling framework is the sequence diagram. This diagram captures interactions between components through component ports and defines operation of the system. Below is a sequence diagram for the eCar model, specifying system’s operation when the driver interacts with the speed pedal. Sequence diagrams also allow assigning for example timing constraints to events, which later can be verified in simulation.

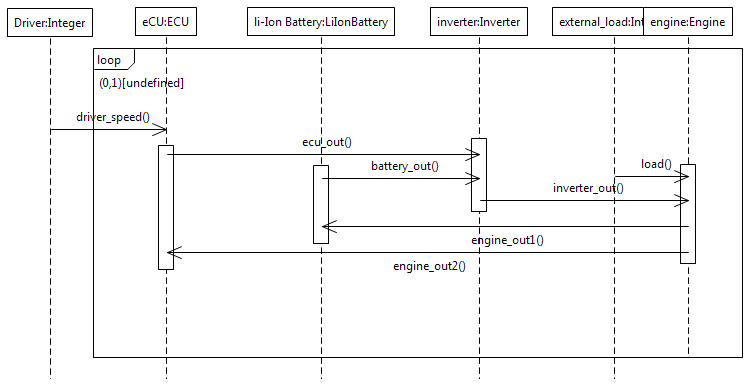


Figure 20: Sequence diagram of eCar

## TCS Model

The Figure 7 represents the actual design flow of SDR case study. There, only system and software models are described.

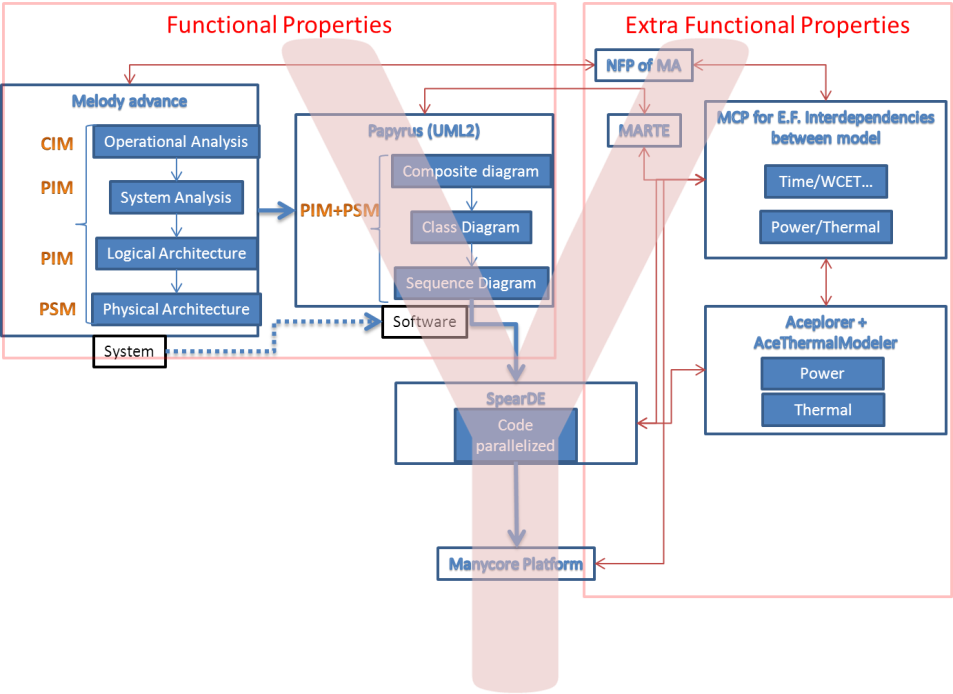


Figure 21: SDR case study.by the OSMK.

The Melody Advance and Papyrus tools are respectively used for representing System and Software models.

### Melody Advance models

The goals are to represent the case study following the Arcadia methodology. It is decomposed into 4 parts:

* Operational Analysis : Beyond and before requirements, the goal of the Operational Analysis is to model the business view of an operational domain. It is not focused on the system to be develop but on its global environment.

Operational Need Analysis is more and more important in complex systems engineering: many major customers require or impose this analysis as mandatory part of the delivered "architecture".

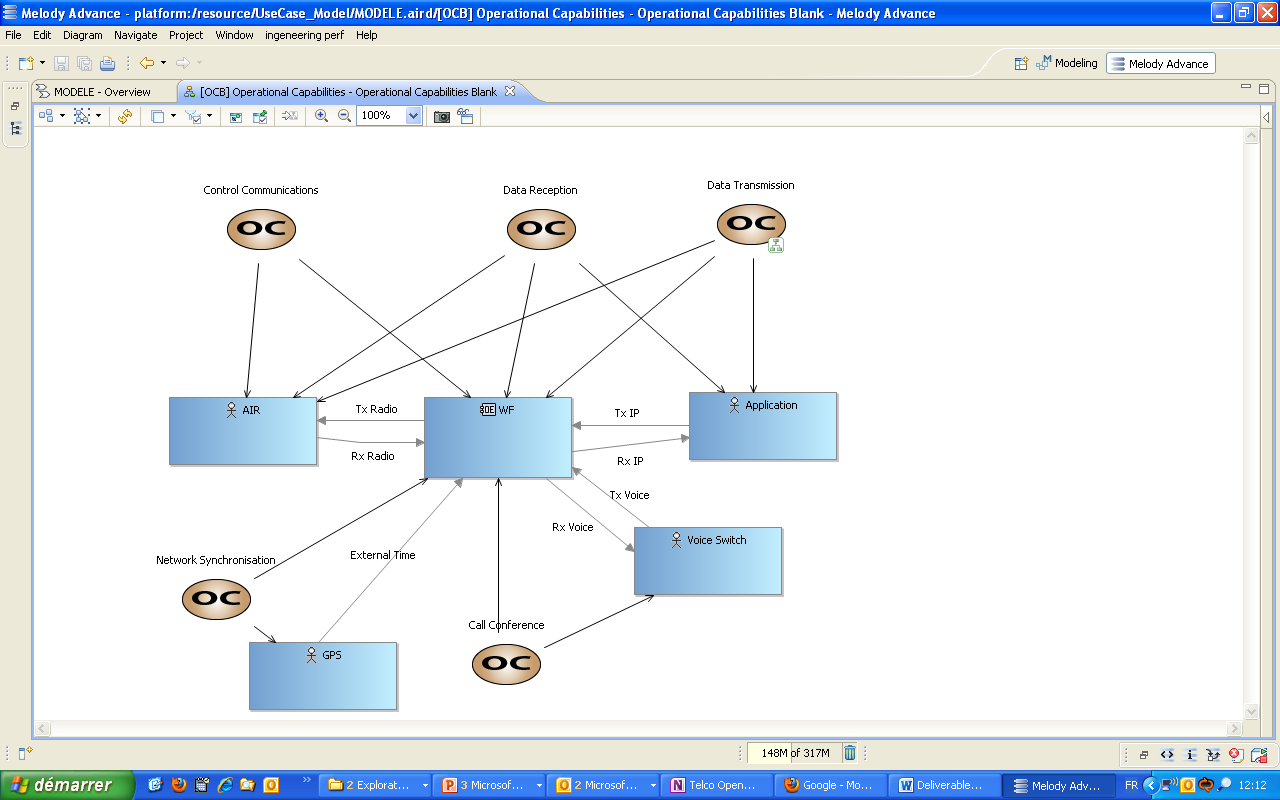


Figure 22: SDR case study, one of the Melody Advance operational analysis view

* System Analysis: The System Analysis level is used to model the system viewed like a black box. It clarifies what the system is expected to do and what its boundaries are.

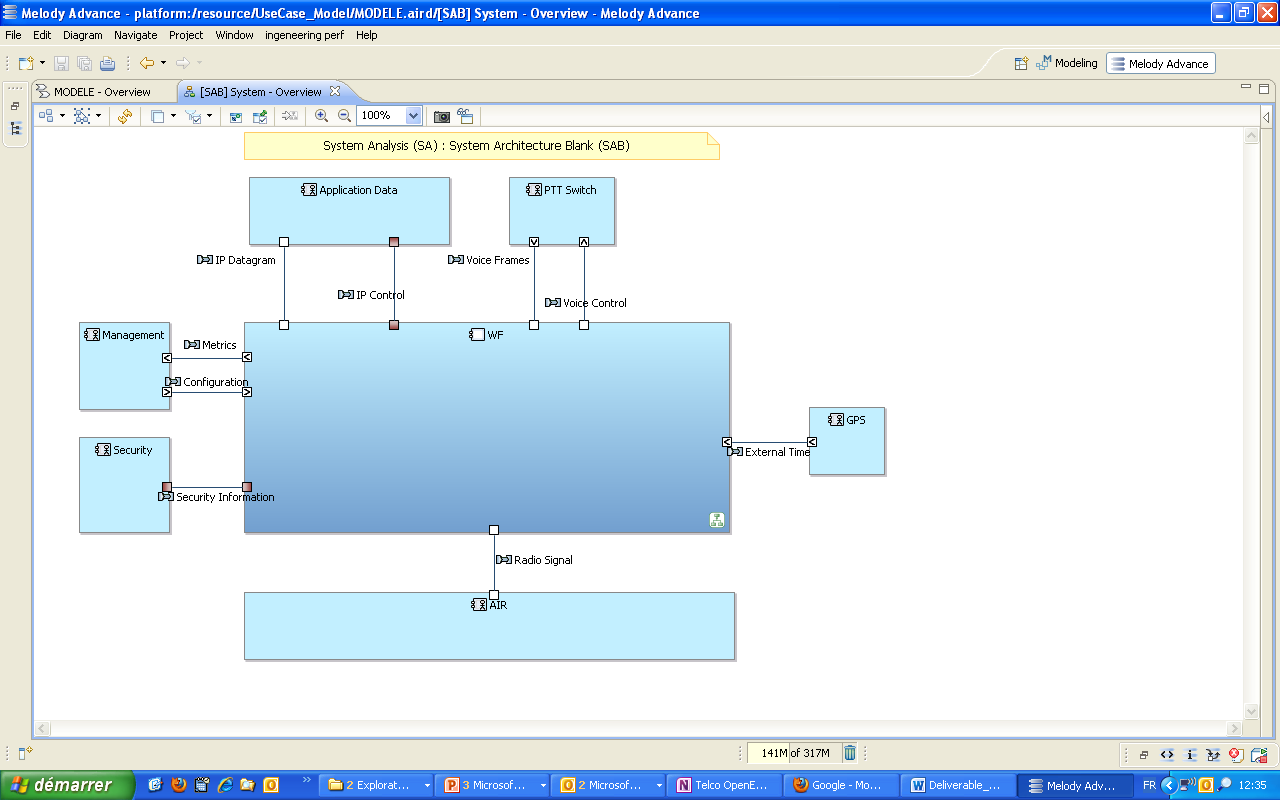


Figure 23 :SDR case study, one of the Melody Advance system analysis view

* Logical Architecture: The Logical Architecture is used to model the system viewed like a white box. The Logical Architecture is a justified breakdown of the system or software, compliant not only with functional issues, but also with all main non-functional concerns susceptible to impact the breakdown.

The Logical Architecture is expected to be stable against further engineering steps and detailed design choices (unlike the physical architecture)

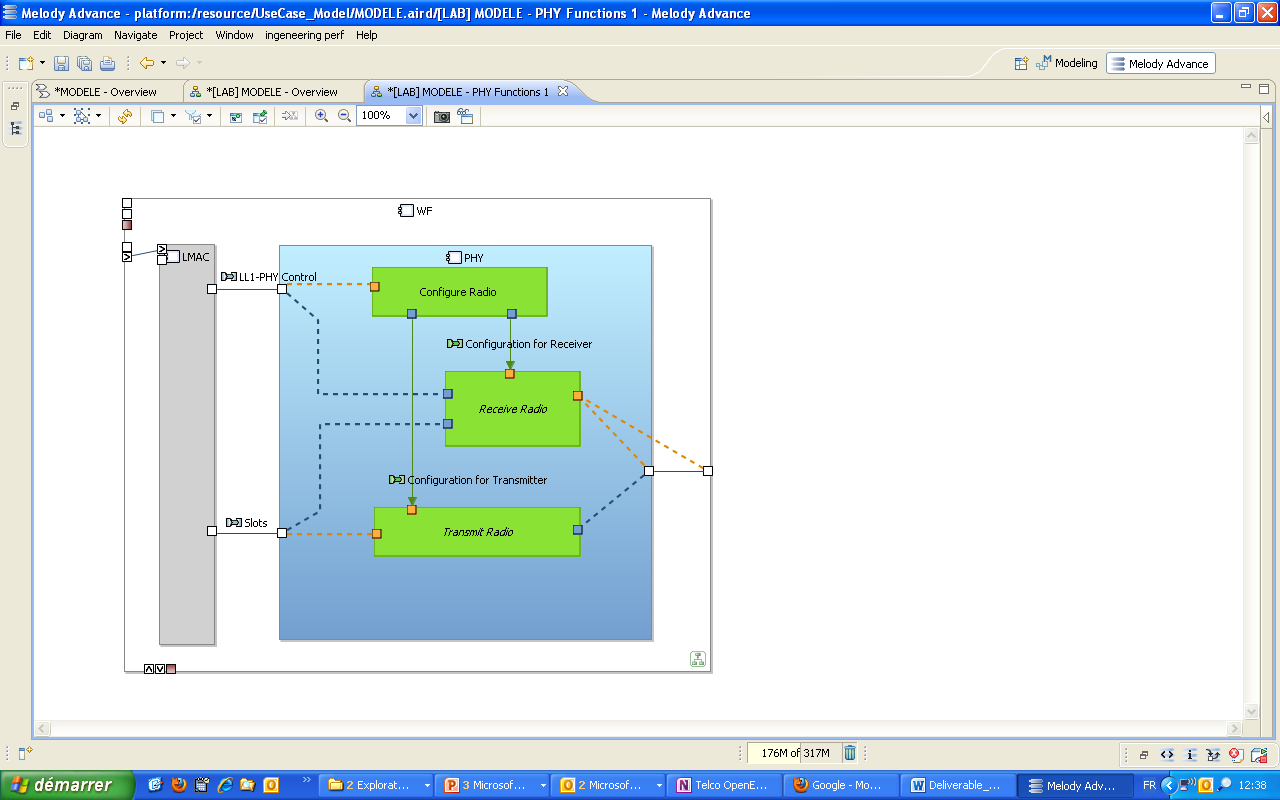


Figure 24: SDR case study, one of the Melody Advance logical architecture view

* Physical Architecture: The Physical Architecture is a refinement and enrichment of the Logical Architecture, dealing with technological and development constraints, through an appropriate organisation and the use of "architectural patterns" or templates to rationalise development.

The physical architecture identifies all components that will be produced and integrated.

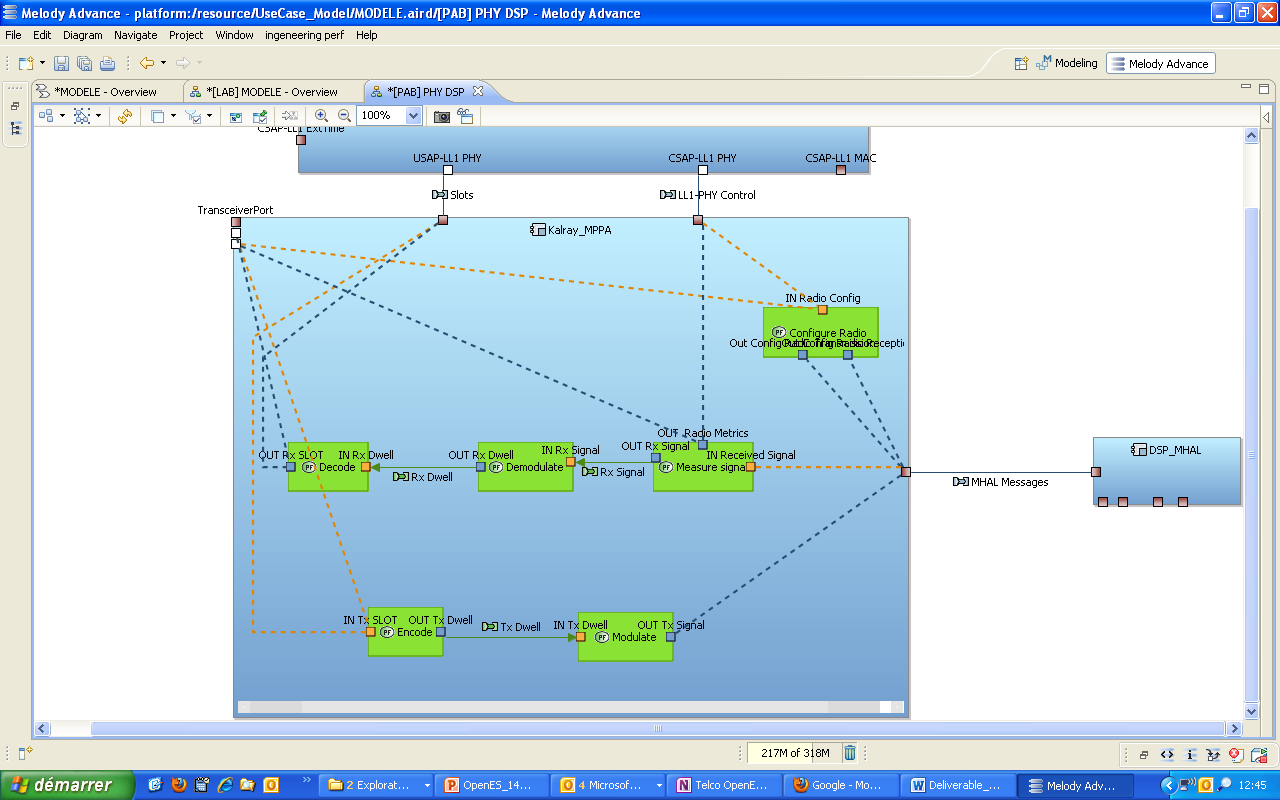


Figure 25: SDR case study, one of the Melody Advance Physical architecture view

### Papyrus Model

This model is used to represent the software level of the SDR case study. Basically, it retakes information form Logical Architecture and physical architecture provided by Melody Advance. Then, communication is refined.

The logical and physical architectures are represented into Component and composite diagrams, see the Figure 12.

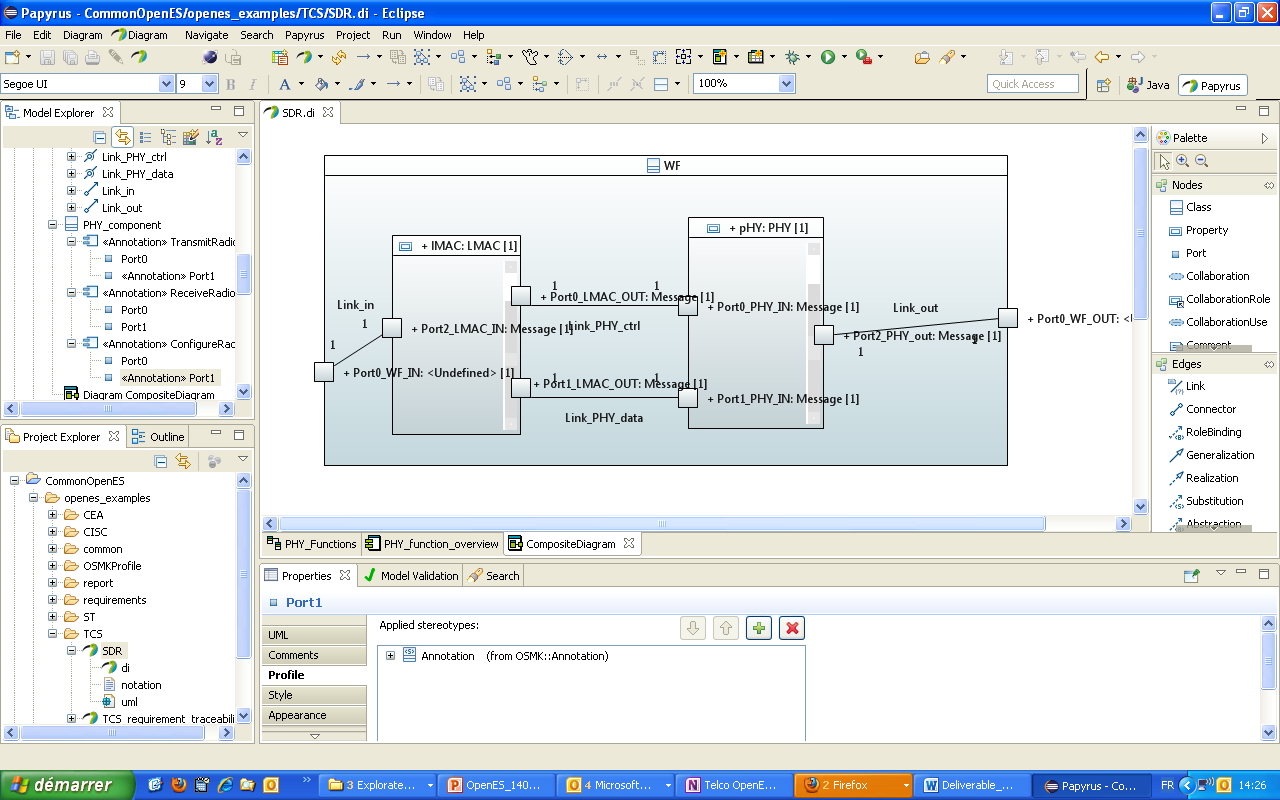


Figure 26: Example of Papyrus links with Melody Advance viewpoints

Then each component are refined into class diagram which is a structural representation of the SDR case study.

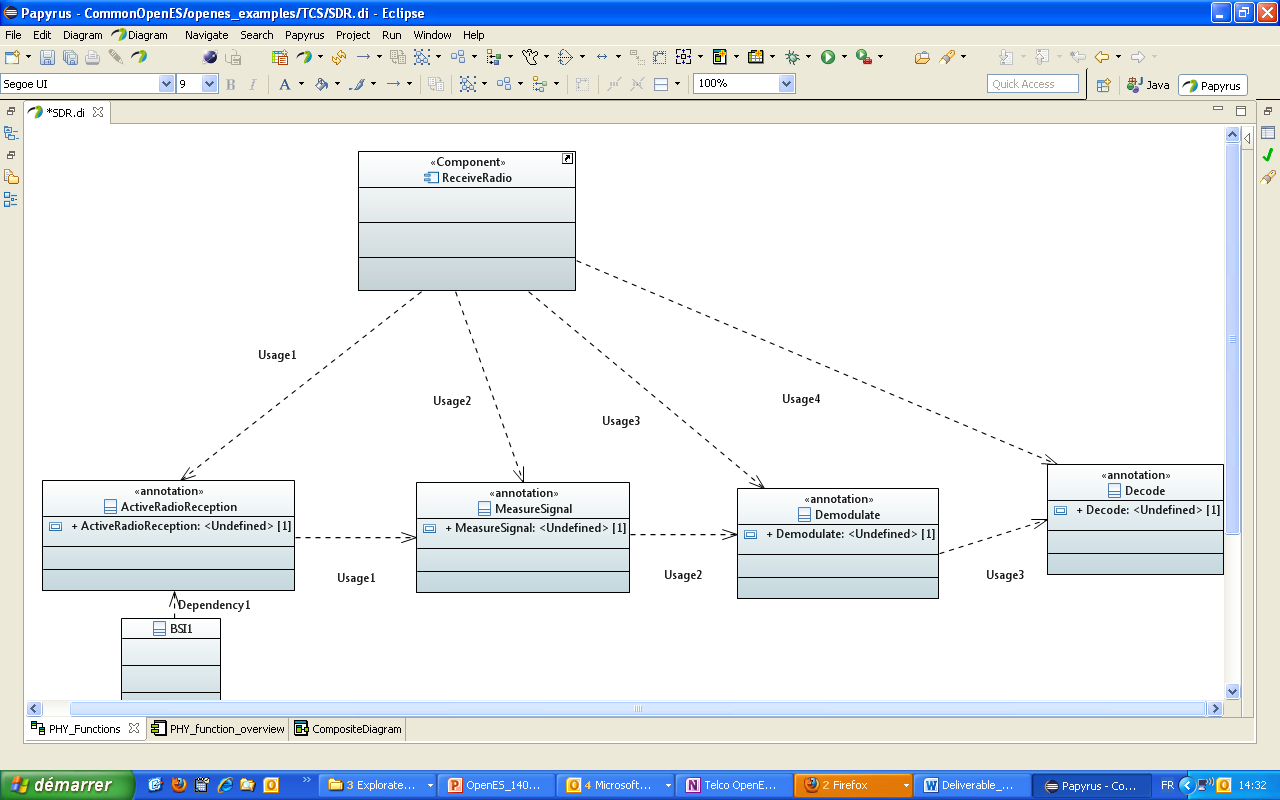


Figure 27: Example of class diagram of the first level of the component "Receive Radio"

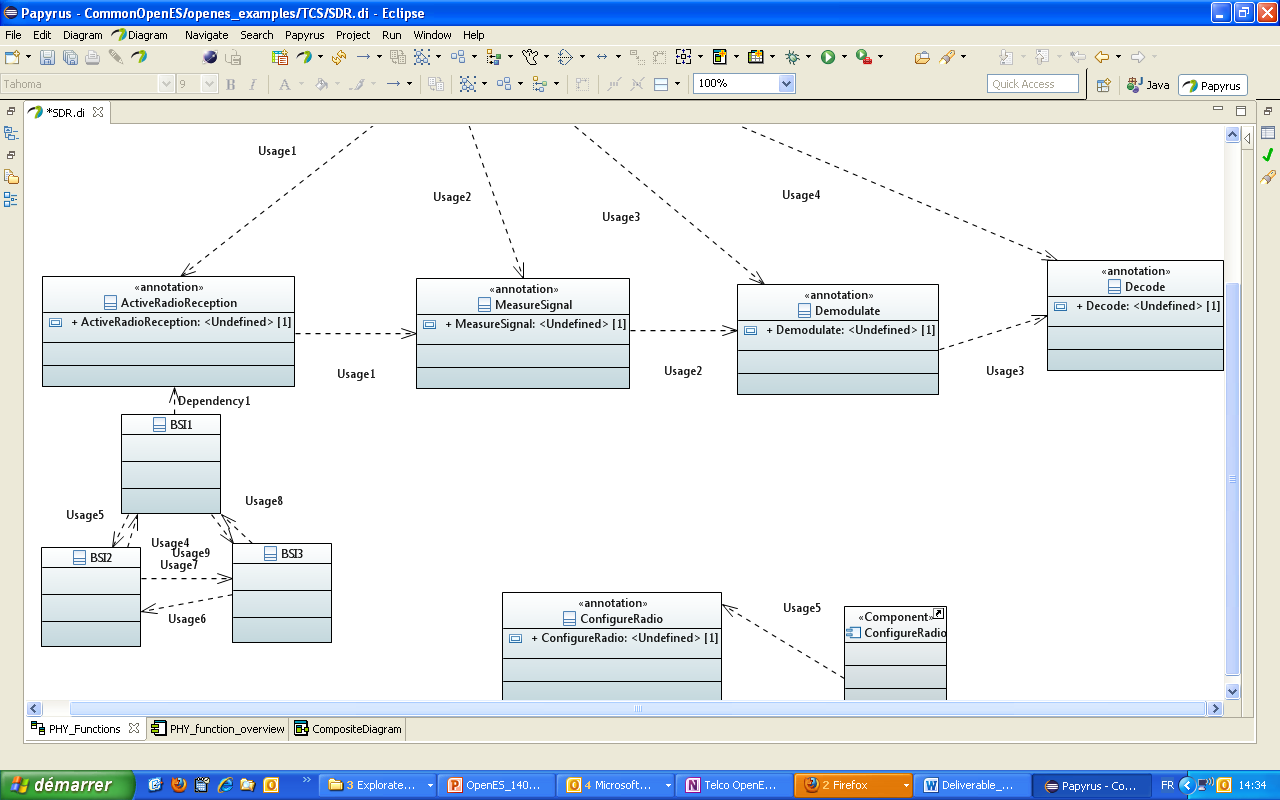


Figure 28: Example of class diagram of the first level of the class "ActiveRadioReception"

Then, activity diagram is developed in order to write the reception scenario of the SDR case study.

### Link between Melody Advance and Papyrus

There is no automatic model transformation between Melody Advance and Papyrus modeller tools. However, a link to trace Extra functional properties are considered into the others tasks T1.3 and T1.4.

Moreover, by this case study, the OSMK proves its modularity by allowing if needed to attach a specific modeller (Melody Advance) into the OSMK (papyrus) for the application model.

## ST Model

ST mainly contributed to the [Requirement Model](#_Requirements_Model) and the [Common Model](#_Common_Model). In addition to those models like each contributing partner, ST also filled his own “Requirement traceability model”. This model is relying on the OSMK::Requirement::Satisfy stereotype to express satisfaction link between a requirement and the contributed example satisfying the requirement.

The picture below provides an example of how the model was filled.

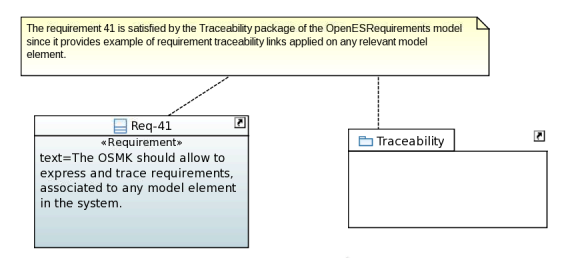


Figure 29 Partner specific requirement traceability model example

The content of each partner traceability model has been then collected and transformed into SysML traceability relationship as described in [Requirements Model](#_Requirement_Model).

# OSMK requirements traceability

The goal this last section is to provide a demonstration that T1.2 specific requirements have been addressed and illustrated by at one of the example models delivered with this report. Hence the remaining part this section is fully generated from the “traceability” models contributed by each of the partners (cf D1.2.3a for the flow description).

The first section contains a summary table, whose generation was possible thanks to the “RequirementScope” OSMK annotation, and coming from comments specified in the input excel table.

Then, each of the satisfied requirement is reminded in this document, with link to the satisfying model and the reason why we considered that it was satisfied.

## Requirement analysis summary

<fragment name=’getModelLink’><arg name=’abs’ type=’Abstraction’/><arg name=’uri’ type=’String’/>[if (uri.contains('CISC'))][CISC Model](#_CISC_Model)[elseif (uri.contains('OSMK'))][OSMKProfile](#_OSMKProfile)[elseif (uri.contains('requirements'))][Requirement Model](#_Requirement_Model)[elseif (uri.contains('CEA'))][CEA Model](#_CEA_Model)[elseif (uri.contains('TCS'))][TCS Model](#_TCS_Model)[elseif (uri.contains('ST'))][ST Model](#_ST_Model)[elseif (uri.contains('common'))][Common Model](#_Common_Model)[/if]</fragment>

<gendoc>

|  |  |  |
| --- | --- | --- |
| **Requirements satisfied in this deliverable** | [getSatisfiedRatio()/] | [for (req : Class | getSatifiedReqs ())separator (‘,’) ] [[req.getReqID()/]](#reqID)[/for] |
| **Requirements that will be satisfied by task T1.3** | [getT13Ratio()/] | [for (req : Class | getT13Reqs ())separator (‘,’) ] [req.getReqID()/][/for] |
| **Requirements that will be satisfied by task T1.4** | [getT14Ratio()/] | [for (req : Class | getT14Reqs ())separator (‘,’) ] [req.getReqID()/][/for] |
| **Requirements that are out of OSMK scope** | [getNotOSMKRatio()/] | [for (req : Class | getNotOSMKReqs ())separator (‘,’) ] [req.getReqID()/][/for] |

## Satisfied requirements

[for (req : Class | getSatifiedReqs ())]<drop/>

### Requirement [req.getReqID()/]

Requirement definition :

*“[req.getReqText()/]”*

[if(not req.getReqComment().oclIsUndefined())]<drop/>

**Comment :** [req.getReqComment()/]

[/if]<drop/>s

Satisfaction traceability :

[for (abs : Abstraction | req.getSatisfyLinks())]

[getTextIntro(i)/] satisfied in [abs.getModelLink(abs.getSatisfyingModel())/] by :

<list><drop/>

[for (elem : NamedElement | abs.getSatisfyingElements())]<drop/>

* [elem.getSatifyingElementText()/]

[/for]

</list>

**Rational :** [abs.ownedComment->asSequence()->at(1).\_body/]

[/for]<drop/>

[/for]<drop/>

</gendoc>

# Conclusion

The models and this document that constitute the deliverable D.1.2.2b of task 1.2 provide two different aspects of the OpenES modeling methodology :

* The Section 2 of this document defines the important abstraction levels identidied as necessary to perform a model-based Electronic System design flow. It provides the methodological “skeleton” of the OpenES design flow.
* Then the model examples contributed by T1.2 partners provide concretes answers for each questions asked by the project requirements in deliverable D1.1. The elaboration of those examples was source of many exchanges between partners, and it will guarantee the consistency of the different case studies that will be continued in the Work Package 4 of this project.

The remaining step to fullfil the objectives of WorkPackage 1 is to propose the same kind of examples for the requirements specific to Task1.3 (Functional and Non Functional Properties) and Task 1.4 (Substem Interaction Modeling).

All those guidelines will be put in practice in the different WP4 OpenES case studies, and will be the source of the automated refinement flows that will be implemented as part of Task 2.2.

# References

1. "Project Full Proposal". OpenES – Open ESL Technologies for Next Generation Embedded Systems. Project Full Proposal for Project: (CA703 OpenES)
2. [UML] OMG: The Unfied Modeling Language (UML), version 2.4.1 (2011), <http://www.omg.org/spec/UML/2.4.1/>
3. [MARTE] OMG: UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, version 1.1 (2011), <http://www.omg.org/spec/MARTE/1.1/>
4. [SysML] OMG: The System Modeling Language (SysML), version 1.3 (2012), <http://www.omg.org/spec/SysML/1.3/>
5. [PSCS] OMG: Precise Semantics of UML Composite Structures (PSCS), second revised submission (2013). To appear.

[ESL] UML profile for ESL (Electronic System Level) : PhD thesis report, 2008 - Sébastien REVOL. Modélisation de système sur puce (SoC) matériel/logiciel en SystemC/TLM et approche pour la transformation de modèles UML vers TLM.

1. UML modelling environment setup

In the context of Task 1.2,1.3,1.4 to fulfil the requirements mentioning the main System Level modelling environment should be open and free, we have chosen to rely on the Eclipse-based UML modelling tool named Papyrus ([https://www.**eclipse**.org/**papyrus**](https://www.eclipse.org/papyrus/)/). This tool is the official UML editor of the eclipse community, and its project leader is the CEA laboratory involved in several tasks of OpenES (T1.3 leader). Moreover, this tool is also used for products modelling and development at STMicroelectronics.

To setup this tool, several steps are required, which will be detailed further:

* Download a base eclipse installation
* Install Papyrus and additional MARTE Profile
* Install additional ESL profile (from ST) for IPXACT related notations
* Import the model project in Papyrus

## Eclipse download and install

To run the latest version of Papyrus (1.0.1) delivered as plugins on the official Eclipse repository, it is first required to install a base bundle of Eclipse Luna.

Several distributions are available for the official eclipse website : <http://www.eclipse.org/downloads/> . The most appropriated distribution, already including some required dependencies is named “*Eclipse Modelling Tools*”.

It can be directly downloaded from this page: <https://www.eclipse.org/downloads/packages/eclipse-modeling-tools/lunasr1>, choosing the download link according the target OS architecture to which the package is intended to be installed (Linux/Windows/Mac, 32/64 bits).

Once downloaded, just unpack the archive.

Eclipse can be directly started with the ***eclipse/eclipse.exe*** executable contained in the target directory.

Notice that a Java Runtime Environment is required to run eclipse. If not present on the target machine, it can be downloaded and installed from: <http://www.java.com/fr/download/>

## Papyrus install

Papyrus is not present by default in the modelling package. Once this eclipse bundle is downloaded, installed and started, you can install additional plugins from eclipse user interface with the following steps described below. Notice that it requires eclipse to connect to Internet. If you use a proxy, you should configure it accordingly.

* Step0 (optional): if you connect to internet through a proxy, configure proxy information from Window ->Preferences -> General ->Network connections
* Step1: click on Help -> Install new software (Figure 1)

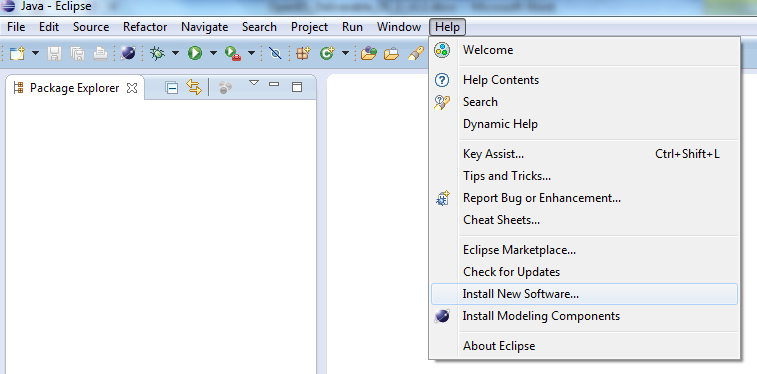


Figure 1 : Install new Software menu

* Step2 : Select Kepler main repository

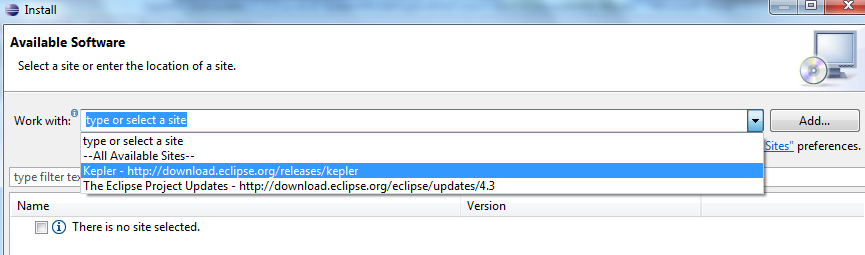


Figure 2 Select Luna Repository

* Step3 : Browse and Select Papyrus in Modelling sub project and click “next”:

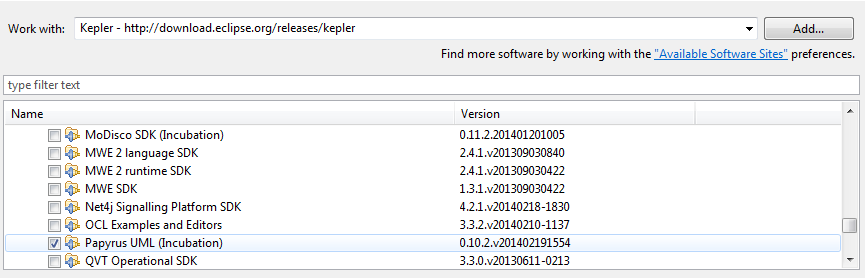


Figure 3 Select Papyrus in Modelling subproject

* Step4 : Accept licence agreement (Eclipse Plugin Licence), install bundles and restart eclipse.
* Step 5 : Installation of Papyrus extensions (MARTE profile): after eclipse restart, enter in workbench and right click on help-> install Papyrus additional components. Select MARTE and click on finish. Restart eclipse.

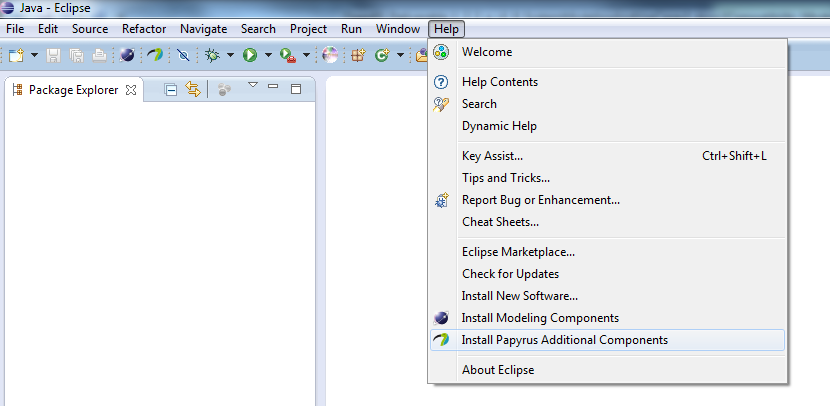


Figure 4 Install Papyrus Additional Components

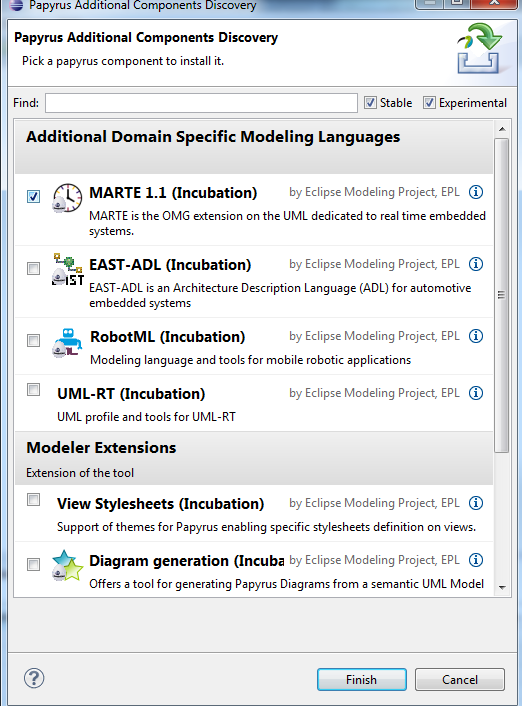


Figure 5 Select MARTE and click on finish

## Install additional ESL profile (from ST) for IPXACT related notations

As an input to the Task T1.2, ST is providing an in house UML profile allowing to describe IPXact related information. This profile is packaged in an additional eclipse plugin that should be installed in eclipse installation. It can be simply done by the following steps:

* **Copy** the archive **com.st.eslprofile\_1.0.0.jar previously delivered in D1.2.2a** in the **eclipse/dropins** directory.
* **Restart eclipse**.

## Import the model in the workspace

The model of this deliverable is packaged as a zipped eclipse project. It can be easily imported into eclipse with the following procedure:

* Click on File->Import

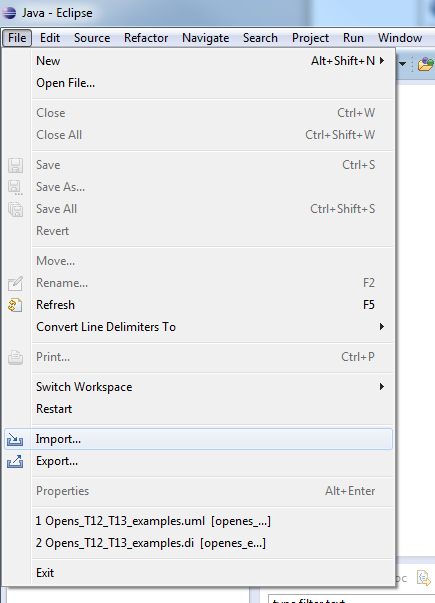


Figure 6 Click on File -> Import

* Select General/Existing Project into workspace

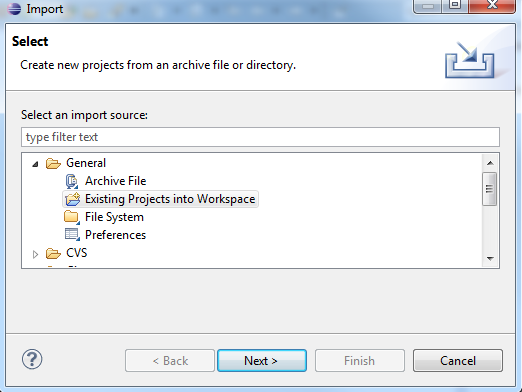


Figure 7 Select General/Existing projects into Workspace

* Select **Archive File** checkbox (! Important, don’t select ‘root directory’!) and browse to the OpenESExamplesD122a-D13a.zip archive. Then click on finish.

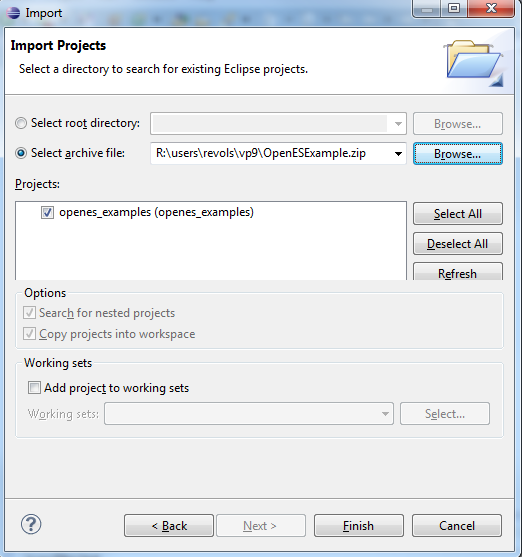


Figure 8 Zipped project selection

* Activate the Payrus perspective : click on window -> open perspective ->others and select Papyrus

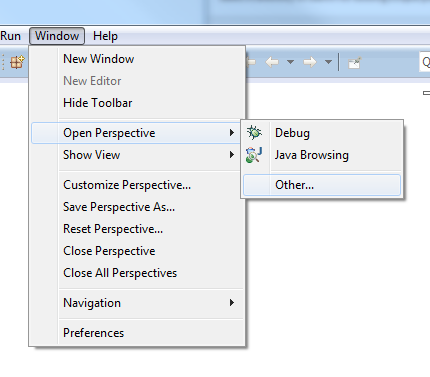


Figure 9 Path to activate Papyrus perspective

* Open the model named Openes\_T12\_T13\_examples contained in the the top left view,in the openes\_examples project.