Modeling Embedded Systems Using SysML

A thesis
Presented to
The Electrical and Electronic Department

by

Carlos Ernesto Gómez Cárdenas

Advisors:
Philippe Esteban
Jean-Claude Pascal
José Fernando Jimenez

In Partial Fulfillment
of the Requirements for the Degree
Master in Electronic Engineering and Computers

Electrical and Electronic Department
Universidad de Los Andes
July 2009

Modeling Embedded Systems Using SysML

Approved by:

Philippe Esteban
Jean-Claude Pascal
José Fernando Jimenez, Advisors

Date Approved ______________________

Table of Contents

List of Tables **vi**

List of Figures **vii**

I Introduction **1**

II Methods **3**

2.1 Definition . . . 3

2.2 Evolution in the embedded system design methods . . . 3

2.3 Platform-Based Design . . . 6

2.4 MCSE Method . . . 9

2.5 Model-Based System Design . . . 11

2.6 HiLeS Method . . . 13

2.7 Model Driven Architecture . . . 14

2.8 Conclusion . . . 16

III Tools Used in Embedded Systems Design **17**

3.1 Platform-based Tools . . . 17

3.1.1 Industrial Tools . . . 18

3.1.2 Academic Tools . . . 22

3.1.3 Summary, Strengths and weaknesses in Platform-based Tools 26

3.2 Model Based System Design and Model Driven Architecture . . . 31

3.2.1 Commercial Tools . . . 31

3.2.2 Open Source Tools . . . 33

3.3 HiLeS Method tool . . . 34

3.3.1 HiLeS Designer 34
3.4 Conclusion 34

IV Proposed Methodology **36**

4.1 Methodology definition 36
4.2 EIA-632: Processes for Engineering a System 37
4.2.1 Acquirer Requirements 38
4.2.2 Other Stakeholder Requirements 39
4.2.3 System Technical Requirements 39
4.2.4 Logical Solution Representations 39
4.2.5 Physical Solution Representations 40
4.2.6 Specified Requirements 40
4.2.7 Building block 40
4.3 System Modeling Language (SysML) 41
4.3.1 Behavioral Diagrams 41
4.3.2 Requirements Diagram 42
4.3.3 Structural Diagrams 42
4.4 Methodology 43
4.4.1 Requirement Analysis 44
4.4.2 Functional and Architectural Analysis 47
4.5 Conclusion 63

V Examples **65**

5.1 Skating Manager System 65
5.1.1 Stakeholder Requirements 65
5.1.2 Applying the proposed methodology 68
5.2 System Display Selector 84
5.2.1 Methodology applied by Airbus 84

5.2.2 System Description 85
5.2.3 Proposed Methodology applied to Airbus Example 87
5.2.4 Applying HiLeS Formalism 89
5.2.5 SD Selector System-HiLeS Model Verification 97
5.3 Conclusion 101
VI Conclusions and Future Work **103**
References **106**

List of Tables

1 Industry tools which reach the Complete Platform-based design concepts.(**DDF:** Dynamic Dataflow, **DE:** Discrete Events **FSM:** Finite State Machine **SDF:** Syncronous Dataflow **TLM:** Transaction Level Model) 27

2 Strengths and weaknesses in industrial tools. 28

3 Academic tools which reach the Complete Platform-based design concepts. 29

4 Strengths and weaknesses in Academic tools. 30

5 Cases to evaluate the requirements 18 and 19 of the SD selector. . . . 98

List of Figures

1 Capture-and-Simulate Methodology. 5

2 General design flow in ESL. 6

3 Platform-Based Design. 7

4 MCSE Method. 10

5 Development phases in MBSD proposed in [BCC$^+$00]. 12

6 V Methodology proposed by Hamon [Ham05]. 13

7 MDA Model transformation. 15

8 CoFluent Studio in the system design flow [Cal93]. 19

9 MILAN architecture [PL01]. 25

10 Relationship between the methodology parts. [Est08]. 37

11 Processes defined in EIA-632. [EIA99]. 38

12 Building block concept. 41

13 Relationship between SysML and UML. 42

14 SysML diagram groups. 43

15 Stakeholder Requirements Package Structure. 45

16 Connection “Derive Requirement”. 46

17 Relationship between requirements and use case. 48

18 Services and ports definition. 50

19 Ports use. 50

20 BDD of a block composition. 51

21 Constraints blocks Example. 52

22 Parametric diagram Example. 53

23 Attach constraints to a block. 53

24 “Meter” unit definition. 55

25 Functional and Architectural Analysis Packages distribution. 55

26 Relationship between requirements, use case and blocks. 56

27 Scenario in level 0. 57

28 System structure representation in level 0. 58

29 Scenario in the level 1. 59

30 System structure representation in level 1. 59

31 Methodology validation using MDA concepts. 61

32 Logical solution to physical solution mapping in the proposed methodology using MDA concepts. 62

33 Methodology summary. 63

34 The skating course. 66

35 A skater times. 67

36 Package distribution in the SysML project. 69

37 Referee Functional Requirements. 70

38 Referee Performance Requirements. 71

39 Referee Interface Requirements. 71

40 Referee Environment Requirements. 72

41 Skater Functional Requirement. 73

42 Race Rules requirements. 73

43 Derived System Technical Requirement from Stakeholders requirements. 74

44 Technology requirement. 74

45 Skate Manager System use case diagram. 75

46 Register Race Info scenario in the Level 0. 76

47 Manage Race scenario in the level 0. 77

48 Connection between requirements and use cases. 77

49 Ports, interfaces and services definition for Skater Manager System. . 78
50 Data type definition used in the constraints definition. 79
51 Skater Manager System Constraints defined in a BDD. 80
52 Parametric diagram for the Identification Time Constraint. 80
53 Block Definition Diagram in Level 0. 80
54 Manage Race scenario in the level 1. 81
55 Relationship between Use Cases, Blocks and Requirements. 82
56 Block Diagram Definition in Level 1. 83
57 Internal System Representation in Level 1. 83
58 Airbus Methodology 85
59 General architecture SD selector 86
60 Functional requirements of Generalities from Crew's Action Processing. 88
61 Crew's Action Processing Use cases diagram. 88
62 Part of a scenario in the Crew's action processing. 90
63 SD Selector Level 0 using HiLeS. 91
64 SD Selector Level 1 Representation 92
65 SD Selector Level 2 Representation. 93
66 SD Selector Level 3 Representation. 94
67 "Valid X Key" behavioral description using HiLeS. 95
68 Requirement algorithm represented in HiLeS 96
69 Environment levels using HiLeS. 97
70 Action Key representation using HiLeS. 98
71 Marking in our "Valid X key" unit. 99
72 Simulation of "Valid X key" Scenario 1, press a key during 2 sec. . . . 100
73 Simulation of "Valid X key" Scenario 2, press a key during 48 sec. . . 101
74 Simulation of Scenario 3, press three times C/B Key. 102

CHAPTER I

Introduction

The complexity in embedded systems has been increased in the last years. New heterogeneous systems which combine different domains are more common. Aircrafts, automobiles, cell phones, medical equipments are systems examples where domains such as electronics, communication, software, mechanics, physics, mathematics and medicine are part of the systems development today.

The markets demand to increase the functionality of the present systems, to mix domains, to anticipate the errors in the system development process and to reduce the delay of time to market in order to minimize design and production costs. Systems such as cell phones have passed from the simple functionality to talk with other people to new functionalities have been added such as SMS, music, videos, Internet, GPS and others, increasing the complexity, adding new systems and reducing their size.

In this work, we propose a methodology to design embedded systems. This methodology is based on EIA-632 standard, which gives the process to the development and production of a system. We use the object-oriented approach to develop each step in the design process proposed by the EIA-632 standard and we create a method based on SysML for its development.

Our methodology uses the MDA concepts to do initial model verification using the transformation from SysML model to HiLeS [Jim00] formalism. In HiLeS, we can execute a formal logical verification extracting the logical behavior represented in Petri Nets and to analyze formally using analyzer tools such as TINA [BRV04]. Also, we can verify by simulation, generating a virtual prototype in VHDL-AMS from the HiLeS model.

The content of this work is the following: In the chapter 2, we give a review in the evolution of embedded system design, we explain some methods used in the system design today. In the chapter 3, we present the tools used to implement the methods exposed in the chapter before. In the chapter 4, we introduce the EIA-632 standard, the model language SysML and we present our methodology. In the chapter 5, we show two examples where the proposed methodology was applied. Finally, we give some conclusion and future work in this topic in the chapter 6.

CHAPTER II

Methods

In order to make the design of a complex system, it is necessary to use a method which will guide the steps to conceive a specific system. In this chapter, the term "method" is explained, because it will be used throughout all this work. Also, it will show the evolution of the methods in the design of complex system. We continue describing different approximations to design complex system in order to have a good idea what is the present problem in the design. Finally, we explain the base methodology of our work, MDA, which is strongly used in the development of software system.

2.1 Definition

A method is defined as the techniques applied inside a design process by the system designers; which is "how" the design task process works [Est08]. Following this definition, in the design of complex system there is a history involved.

2.2 Evolution in the embedded system design methods

Since the 1960's, the development of embedded systems was a sequential designing process, starting from the requirements that each system had to reach. The software designers developed algorithms which represented the different system's behaviors and created the system's specifications where their application had to run. However, these specifications were not complete, because they did not take into account many features that the software designers were not able to see until after the physical prototype is made. Afterwards, the software designers handed over the specifications to the hardware designers in order to design and build a physical prototype that would fit the software designer needs.

Hardware designers began building a block diagram of the system, following the given specifications. When the block diagram was finished, they manually transformed it to a gate level. In this level they could simulate and validate if the behavior of their design reached the specifications. This transformation was time-consuming, because they must simulate and verify each specification. Also, the verification was possible only when the hardware design achieved the gate level; therefore the designer had to go down in different abstraction levels without the designer's verification in each step of the process, simply because the verification could not be made in another level.

Once the verification was finished, the hardware designers built a physical prototype for the software team. Often this system prototype did not fit the software team expectations, due to the ambiguity and the incompleteness of the original specifications. Thus, the software team implemented their applications onto the prototype and tried to do their best effort to achieve the system real specifications and get the final system prototype. In the Figure 1, it is drawn this methodology, called "Capture-and-Simulate" in [Gaj07].

When the technology improved and the logic synthesis arose, the design method changed. Hardware designers could describe the system's specifications with boolean equations and they could transform these equations to gate level using a specific tool. With these new methods and tools, the hardware designers could verify their designs in the boolean equation description and re-verify them on gate level. This methodology is called "Describe-and-Synthesis" in [Gaj07].

During 1990's, the boolean equations description was changed for languages which rose the abstraction level to the register transfer level (RTL). Hardware Design Languages (HDL), such as VHDL and Verilog followed this new hardware description. These languages have been the most used until today and they are called "Synthesis-based" in [Gaj07].

Since the late 1990's, the system complexity has risen, therefore system designers have had to increase the abstraction level of their designs. They began to create

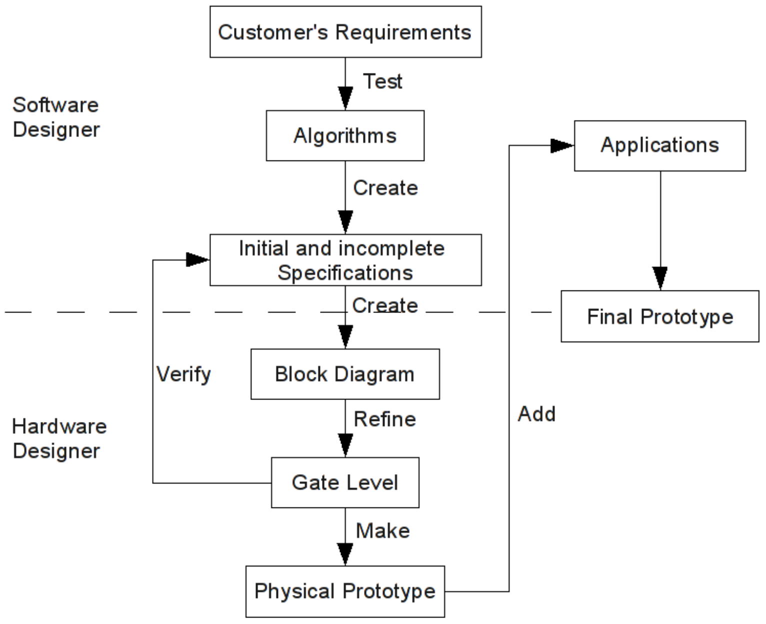


Figure 1: Capture-and-Simulate Methodology.

system models using generic languages to describe the system's specifications. These models describe only the behavior and the basic structure of the system without taking into account what will be hardware and what will be software.

Languages as C, C++ and Java were used by the hardware and software designer to create the behavioral model of these systems. However, this approximation was limited, because in this kind of language certain hardware concepts were not present, such as concurrency and the notion of time [HS07]. Hence, it was necessary to extend these languages in order to complete the hardware concepts. Languages such as SystemC [16608] and SpecC [Dom02] are examples of this work.

Concepts such as "Transaction Level Modeling" or TLM [Mon07], which are above RTL, are rising. Their focus is to separate the computation and the communication in the system design to help reusing models. SystemC follows this concept.

Having available this new option, the designer can develop systems in a higher level of abstraction and these systems can be executed to verify their behavior and the fulfillment of their specifications and constrains. This new process is called Electronic System Level (ESL) [Bai07] design (figure 2). The ESL solved an important

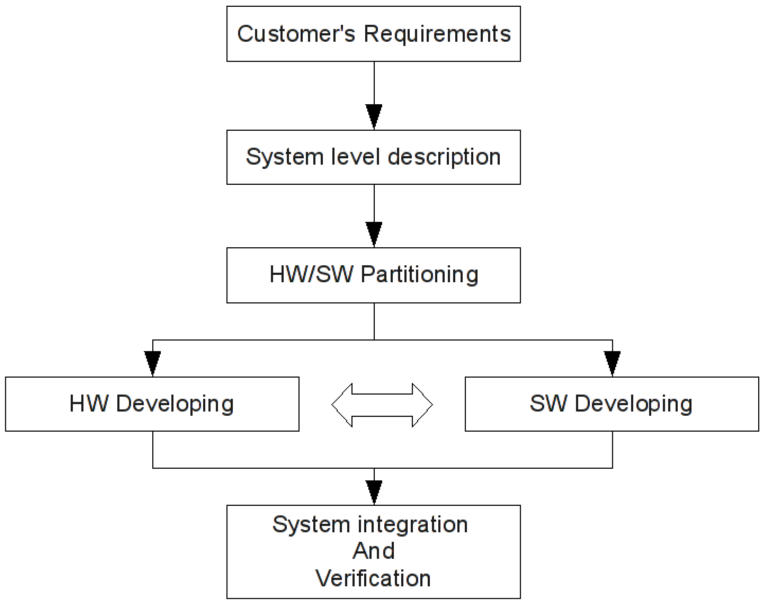


Figure 2: General design flow in ESL.

part of the design problem, describing the behavior of the system without deciding which part will be hardware and software. Nevertheless, it helps to decide about the partitioning and to have an early virtual prototype in order to develop, almost at the same time, the software and hardware after the partitioning. This virtual prototype helps to verify the hardware and software developments according to the specifications. A good method example is the platform-based design.

2.3 Platform-Based Design

Nowadays, according to the market demands, it is important to reduce the time in the systems design. Therefore, it is necessary to move towards re-using design methods and to develop alternative ways to transform the system specifications to re-use designs.

A proposal was made in [SV07], and it is to separate the system functionality from the platform where the functionality will be implemented. This process is called “Platform-based Design” (PBD). The figure 3 shows the basic design of this

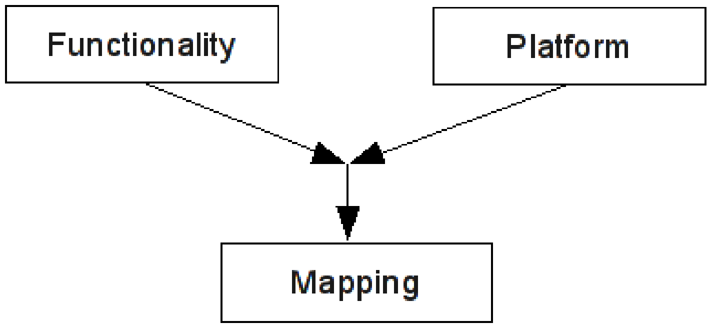


Figure 3: Platform-Based Design.

designing process. It begins with the customer's requirements: Functional and non-functional requirements. Functional requirements correspond to the "what", what the system has to do. The non-functional requirements match up to the "which" and "how", how the system's behavior has to be and which the system's size has to be. These requirements will be transformed to specifications and constrains in the system designer terms.

- **Functional Side:** The system designers develop the system's functionality, where it is expressed the behavior of the system. Usually this is modeled on algorithms, using different languages and there is not distinction between hardware and software design.

  The software and the hardware designers have their own approaches in describing the system's functionality. The hardware approach uses languages originally created by software designers, such as C, C++ and its derivation such as SystemC and SpecC. An additional approach came from model-based designers, which use UML as a basis to create a new profile that can be used in the embedded system design. Examples of this tendency are UML for SoC [MM05] and recently SysML [LdOFV07].

  On the other hand, the software approach which includes hardware concepts creates languages such as Esterel, Lustre and Signal. Other approaches are models of computation, where the system is graphically represented using

Dataflows, Finite State machines, Petri nets, etc. These models can be heterogeneous (a mix between different domains such as analogous and discrete) or homogeneous.

- **Architectural Side:** The architecture represents the structure where the system's functionality will be implemented. It is made using an architecture's library, a component's library, the system's constraints and the system's specifications. With these elements, the designer can use a preexisting architecture and then change the components which do not fulfill the constraints, and probably adding other components to achieve the wished functionality.

  On this subject, there are again two approaches: Software and Hardware approaches. In the software approach, the architecture description languages (ADL's) are used; in [MT00] it is possible to find a complete survey about these languages. Another approach is UML and its extensions. In contrast, the hardware approach has languages such as HDL and more recently TLM, which is used in many architecture design tools.

- **Mapping platform result:** When the designer has the functionality and the possible architecture where the system will be implemented, he maps these models using different techniques to find the best performance. In this step, the designer decides which functionality will be implemented in the hardware and which will be solved in the software, to achieve the wished performance.

  A traditional mapping technique is to force the functionality model and the architectural model to match each other in an iterative way, it means that the functionality is mapped on a candidate architecture and if the constraints are not reached, then the designer has to change either the functionality or the architecture. When the change is made, the system design is tested again and the process starts all over again until the systems' specifications are reached. This technique is not optimal, because it does not use any optimization method to find the best way to match functionality onto architecture.

  In [SV07] it is proposed to use a unique mathematical formalism in order to express both models in equations, and also the functionality, the architectural and the constrains, to be able to build a cost function and to apply an optimisation method to make an automatic mapping. In [CFDBSV$^+$05] the idea of the PC architecture continues, in order to create standard architectures for specific fields. The main idea is to apply specific applications using Application Platform Interfaces (API).

2.4 MCSE Method

The Electronic System Design Methodology (MCSE in french) follows the main features of Platform-Based Design. It splits the system functionality from the platform where it will be implemented. By simulation, the platform configuration is evaluated in order to satisfy the functional constraints [Cal93].

This method is decomposed in three stages:

- Timed-Behavioral modeling
- Platform modeling
- Architectural exploration and performances study

In the Figure 4, it is shown the design flow of this approach.

- **Timed-Behavioral modeling:** In this stage the system specifications are transformed to a timed-behavioral model. This model is represented in SystemC, so it can be verified by simulation. This stage has three steps:
  - *Algorithms analysis:* The specifications represented in algorithms are studied. These specifications are analyzed, identifying the level of granularity required for the model, the elementary computational blocks, the data flow and the data structures.
  - *Application model creation:* The analysis made in the last step, is structured in order to create an application model. An application model consists of elementary functions and communication links. Functions are described with a behavioral model including the needed computational

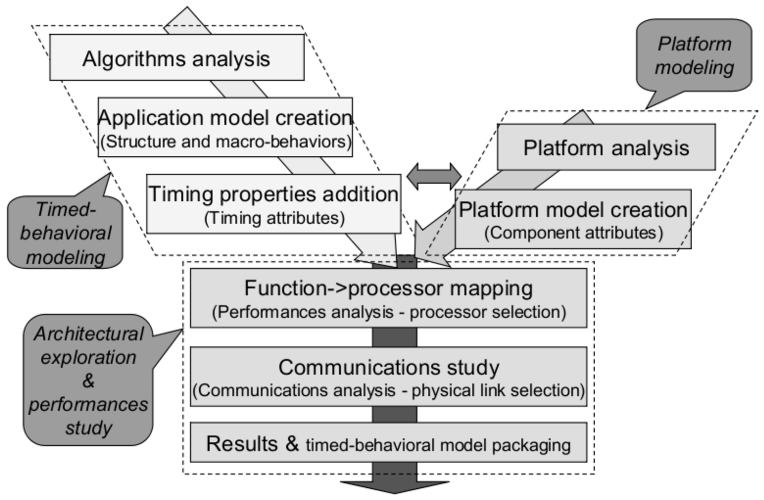

Algorithms analysis
Application model creation
(Structure and macro-behaviors)
Timing properties addition
(Timing attributes)
Timed-behavioral modeling
Platform modeling
Platform analysis
Platform model creation
(Component attributes)
Function->processor mapping
(Performances analysis - processor selection)
Architectural exploration & performances study
Communications study
(Communications analysis - physical link selection)
Results & timed-behavioral model packaging

Figure 4: MCSE Method.

blocks. These functions are linked by communication links. This application model is also a SystemC model, so it is possible to verify this stage by simulation.

- *Timing properties addition:* In this step the time attributes are added to the behavioral model inside the application model in order to get a timed-behavioral model.

- **Platform modeling:** In this stage a platform model is created. The modeling of the platform is built in an abstract level which allows certain performance evaluations, such as message level. There are two steps involved in this stage:
  - *Performance Analysis:* The computational units are identified; their features are defined, as well as the communication links between them.
  - *Platform model creation:* A graphical model of the physical architecture is built. This model contains the computational units which are necessary to run the application model. This model is featured by a set of attributes.

- **Architectural Exploration and Performances Study:** Finally, this stage allows mapping the application model elements to the platform model elements. In this process, there is an exploration of the design space and it is studied the system performances. In this stage there are 3 steps:

  - *Function-to-processor mapping:* By experience, the designer does the first partitioning software/hardware. This activity is to map functions to processors inside the SystemC environment. The result is a timed-behavioral model influenced by the processors constrains. Such as the mapped model is also a SystemC-based model, it can be evaluated by simulation.
  - *Communication study:* The influence of the communications links between processors is studied. Modifications and variations are done in order to achieve a wished performance.
  - *Results production:* This step provides a SystemC virtual system architecture which details the specification collected in the two steps developed before; and it is traduced to the execution performance for the application.

This method shows the problem which is concerned to the experience of the designer. The partitioning between software and hardware is done initially by experience and later according to the simulation results, the partition or the selection of the components is modified.

2.5 Model-Based System Design

The Model-Based System Design is a method which uses as basis a system engineering process to design a complex system. In [BCC$^+$00], the authors used the system engineering process, defined in the standard IEEE 1200, to implement their method inside the process' activities by the development of a complex system based on models. In the figure 5, it is depicted the development phase in [BCC$^+$00]. The IEEE 1200 standard follows a top-down design flow.

- **System Definition:** This phase corresponds to the analysis and requirements baseline validation of the IEEE 1200 activities. In this phase are defined the system requirements and they are integrated to a system model. The system

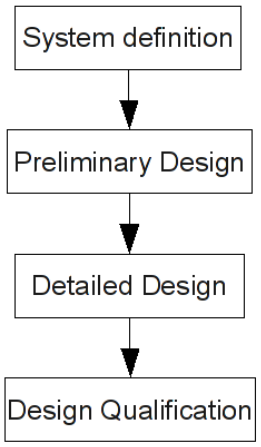


Figure 5: Development phases in MBSD proposed in [BCC+00].

model that is created in this phase has to be executable in order to evaluate the system requirements, in other words, to verify the system model.

- **Preliminary Design:** The system is split into subsystems. The system model descends to a more detailed description level and gives a guide to the component distribution. Also in this phase each subsystem model has to be executable.

- **Detailed Design:** Subsystems are detailed in this phase. The model of the subsystems is mapped to components, afterwards executable models are created which follow the behaviour of the components mapped of the subsystems. These executable models verify if the detailed subsystem models fulfil the system specifications. Also in this step, other constraints are chosen such as size, weight, colour, etc.

- **Design Qualification:** The model tests are compared to the manufactured components' tests. The complete verification is done in this phase. Models are updated after the integration and they are verified against the specifications.

Based on the conception MBSD, Estefan in [Est08] shows a survey of methodologies that implement it, such as Telelogic Harmony-SE, INCOSE Object-Oriented Systems Engineering Method (OOSEM), IBM Rational Unified Process for Systems

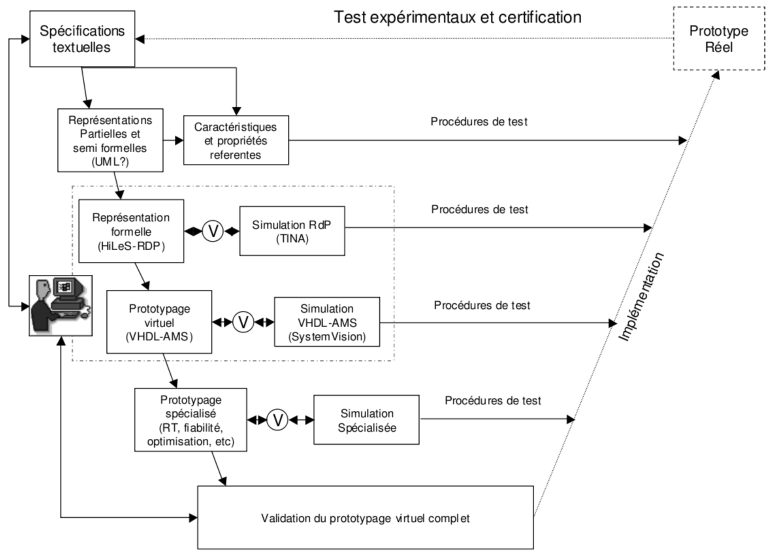

Spécifications textuelles
Test expérimentaux et certification
Prototype Réel
Représentations Partielles et semi formelles (UML?)
Caractéristiques et propriétés referentes
Procédures de test
Représentation formelle (HiLeS-RDP)
Simulation RdP (TINA)
Procédures de test
Prototypage virtuel (VHDL-AMS)
Simulation VHDL-AMS (SystemVision)
Procédures de test
Implémentation
Prototypage spécialisé (RT, fiabilité, optimisation, etc)
Simulation Spécialisée
Procédures de test
Validation du prototypage virtuel complet

Figure 6: V Methodology proposed by Hamon [Ham05].

Engineering (RUP-SE), Vitech Model-Based System Engineering (MBSE) Methodology and State Analysis (SA). The methods implemented in these methodologies are presented in the different steps of the system design, starting from the customer requirements and all the treatment necessary to conduce to system model and later to a physical system. PBD is only oriented to the phase of the system design, how to do the partitioning and get a real system.

The two methods, PBD and MBSD, are complement each other. PBD can be used to descend from high level description to a low component description, as well as MBSD proposes in each phase, but MBSD does not explain how that is made.

2.6 HiLeS Method

Hamon proposes a method to develop complex systems in its PhD thesis [Ham05]. It is a classical V methodology whose steps are represented in figure 6 and its work was in the formal representation of the system, the virtual prototype and its verification by formal analysis and simulation.

He suggests a previous transformation from textual specifications to a semiformal representation in UML. Once the designer has the system described in UML, it can be transformed to a formal representation called HiLeS; a formalism based on Petri Nets [CL08]. The logical representation of the system can be verified formally by formal analysis using a tool called TINA[BRV04] and the behaviour of the whole system is verified by simulation transforming the system model to VHDL-AMS and later using tools such as System Vision [Gra09] to simulate. The system model represented in VHDL-AMS is called "Virtual prototype" and it will be used as a specification to the suppliers in order to get a physical prototype.

2.7 Model Driven Architecture

Model Driven Architecture (MDA) is an OMG approach to system development. MDA uses models to understand, design, construct, deploy, operate, maintain and modify systems. Its main purpose is to split the functionality of the system from the platform where the system functionality will be implemented. This helps to the portability, the interoperability and the reusing of the system design [OMG03].

Dividing the functionality and the platform, it is possible to refine the functionality in different levels of abstractions. Each level is transformed using a transformation rule from one level to another. MDA opens the possibility to do an intermediate transformation to verify the system behaviour using a specific platform, such as an executable language (Java or VHDL).

The main elements in MDA are:

- **Computation Independent Model (CIM):** This model is used to describe the system domain. It is useful to understand the requirements of the system and also, it is a communication link between the customer and the designer. This model does not have any system solution.

- **[Platform Independent Model (PIM):** This model describes the behaviour of the system which is independent from the platform where it will be implemented. This model can be refined in different degrees of detail, descending on design abstraction levels.

- **Platform Specific Model (PSM):** It is the model where the PIM will be

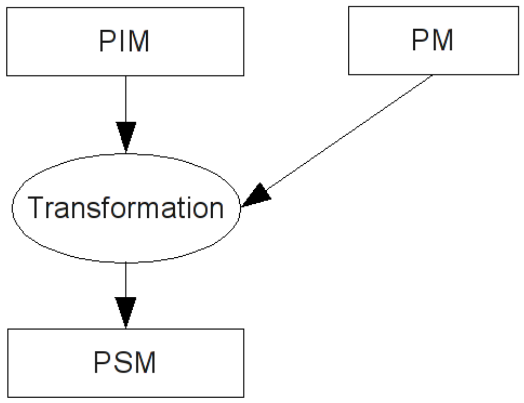


Figure 7: MDA Model transformation.

implemented on a platform. This model allows verifying the PIM in different ways, e.g. by simulation.

PIM, PSM and the platform model (PM), where the PIM will be implemented, need to create a metamodel which will be the "template" that allows to create a model either PIM or PM. In order to map the concepts from PIM to PM it is necessary to define transformation rules that relate the PIM's metamodel concepts to PM's metamodel concepts. A transformation engine is used to make these transformations and generate a PIM. In the figure 7 is shown the Model transformation.

MDA is clearly related to PBD, where PIM is the functionality model and PM is the same Platform model. The mapping platform can be considered as the PSM in MDA. Vincentelli said that there are similar concepts between both approaches, but he cleared that MDA does not make a good description on how PM or PIM have to be implemented [SV07]. From this point of view, MDA is only a process which gives support to designing systems, but it does not describe "how" that can be implemented. Hence, PBD can be an implementation way.

There are other works which try to implement the MDA concepts, such as MDA approach for the classical "Y-chart" Design [BDDM03] and MDA applied to embedded systems dedicated to process control [HB03].

2.8 Conclusion

In this chapter, we defined the method conception, we gave a brief review in the evolution of embedded system design methods, we explained the Platform-based Design, MCSE method, Model-Based System Design, HiLeS method and MDA. We showed some similarities between the different described methods and we could see that the tendency in system design is the partition between functionality and architecture, and subsequently make the mapping between them. Also, we could appreciate that increasing the abstraction level in the design is a contemporary tendency, in order to reduce the complexity of the systems and to easily verify the model according to the requirements.

In the next chapter, we will present the tools used in the methods described in this chapter, and we will do a classification according to the development group, its features, and its reachability.

CHAPTER III

Tools Used in Embedded Systems Design

Part of the complex systems methodologies are the tools which make easier the development of a system design. Each methodology has its own classification of tools, according to what tool fulfills the requirement inside the methodology and who is the developer of the tool, the academia or the industry. In this chapter, we show the classification of the tools used in embedded system design according to the methods described in the last chapter, its developer group, its main features, weakness and strengths of each tool inside a methodology context.

3.1 Platform-based Tools

There are different tools, both in the industry and the academy which follow part of the Platform-based principle. In [DPSV06], Densmore et al. make an excellent classification list of these tools, we extract the main tools which use this design method and we give a more detailed description of these tools. We based our work in their web pages and available articles. The authors divide the tools in three groups, according to its functionality:

- *Single:* Tools which only develop a part of the platform-based method, that means just Functional model or Architectural model or Mapping model.
- *Pair:* Tools which develop a combination of two parts of the platform-based method, that means, Functional with Architectural, Architectural with Mapping and so on.

- *Complete:* Tools which develop each part of the platform-based in the same environment.

We are interested in the tools which reach the requirements of the complete group, thus we chose the tools which follow this principle in the industry and the academy world. These tools offer a unique environment to model the functional and the architectural part, and it is also possible to make the mapping between the two models in the same tool. In the following sections, we make a summary of each Industrial and academic tool chosen in [DPSV06], we show a summary table which sums up the main features of these tools and finally we show another table which exposes the strengths and weaknesses of them.

3.1.1 Industrial Tools

3.1.1.1 *Cofluent Studio*

This tool is a system design environment based on the SystemC language that covers the timed-behavioral modeling and system architecting steps according to MCSE methodology [Cal93]. This tool helps the designer to convert system specification to a virtual system which is verified by simulation.

The tool has three modules:

- **Timed-Behavioral Modeling:** In this module, the designer creates and validates an application model starting from a well defined system specification. The application model has different refinement levels, starting from a behavioral model without taking into account the time, ending into a timed model.

- **System Architecturing:** This module allows exploring multiple platform, different hardware/software partitioning and allocation strategies. Non-functional constraints such as performance, real-time constraints and costs are taken into account. The result of this module is a SystemC Virtual System, which is the system application mapped in a platform with a previous performance analysis that follows the non-functional constrains of the system. This virtual System is used to start the development of the hardware and the software part.

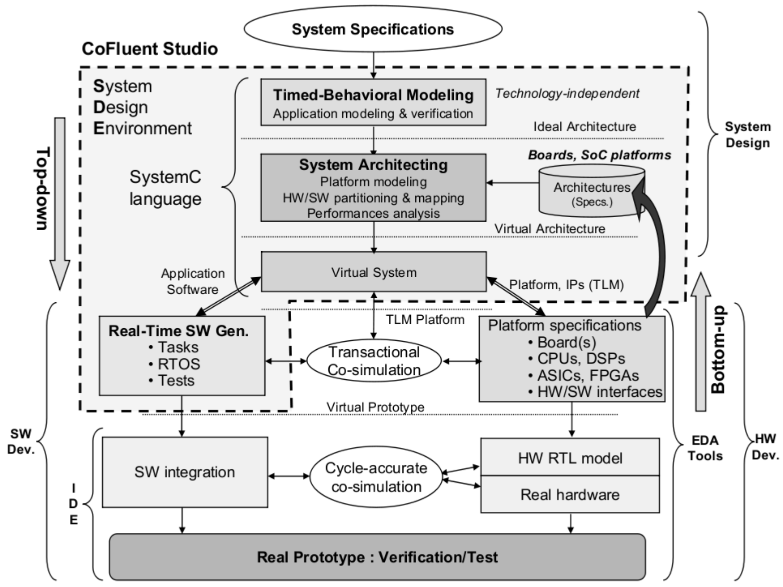


Figure 8: CoFluent Studio in the system design flow [Cal93].

- **Real-Time Software Generation:** Using this module, the designer can generate the C code of the software part from the virtual system.

In the figure 8 is pictured the different modules presented in this tool. The pointed area is covered by CoFluent Studio.

3.1.1.2 ***MLDesigner***

This tool is a MLDesign Technologies product. It is a system-level simulation modeling platform which integrates both system level modeling areas (architecture and function), and simulation domains in a single tool [STF+03]. The modeling domains that this tool supports are: Discrete event, Dynamic Dataflow, Synchronous Dataflow, Continuous Time/Discrete Event, Finite State Machine, Boolean Dataflow and Higher-Order Function. The domains can be used ether in a homogeneous (to use the same domain to model a system) or heterogeneous (to use different domains to model a system) environment.

The base of this tool is Ptolemy, and its graphic representation is based on the Ptolemy language called Ptolemy Tool command Language (Ptcl).

A system model is represented by hierarchical blocks called modules. A module contains primitives and other modules. A primitive is a single function of the model which is defined ether by a C++ code form or by finite state machine model. Each module and primitives has ports which are connected by visual links called relations [MT07].

Using this tool, the designer can build a complex system starting from a high abstraction level such as mission-level or operational level and descending for System level design (computer architecture, communication network design, etc) arriving to a functional level design where is expressed the algorithm design, the implementation and the partition software/hardware.

The tool is not linked to a specific methodology, therefore it is possible to apply a methodology which describes the system in high level (behavioral and structural) and split it in subsystem following a detailed description when the designer descends into different abstraction levels.

From the point of view of Platform-based design, using this tool, it is possible to describe the system functionality separated from the platform where the system will be implemented using the same language, in this case, Ptolemy. Manually, the designer can map the functionality to the platform by experience, for instance, and to see the mapping result on the simulation of the resulting system model.

The tool helps to simulate whatever behavior that the designer can build, but we cannot find if it has the functionality to help in the mapping process.

3.1.1.3 ***Visual Sim***

This tool is a Mirabilis Design product. It is a block-oriented system model design environment, similar to Ptolemy and MLDesigner, where the designer can create a hierarchical system design in different abstraction levels. Each block inside of a system model is called SmartBlock and it can represent hardware, software, networking components at queuing, performance, transaction and cycle-accurate levels of abstraction [Inc03]. It uses different model domains, Mirabilis calls them “simulation kernel” and they are Discrete event, Synchronous Data Flow, Finite State Machine and Continuous time. Visual Sim has an extended library of components such as

CPU, caches, bus and memory ready to use in the system model, reducing the time to design.

A designed system in Visual Sim can be described in three parts: Architecture, Behavior and Workload. Architecture is the representation of the components interconnected, the behavior represents the actions that will be performed by the system and the workload is the transactions that traverse the system such as network traffic. The mapping between behavior and architecture is performed using virtual execution [AM06].

Mirabilis proposes a top-down methodology, which consists of three main steps: Requirement analysis, Functional Analysis and Architecture Analysis. These steps are supported by the System Analysis. Each step can be modeled, detailed and simulated in Visual Sim.

Visual Sim does not have enough documentation to evaluate the real advantage to use this tool compared to MLDesigner because its block-model approach is similar. It is clear that it has an extended library to build system models and the facility of import and build models in different language such as C, C++, Java, SystemC, Matlab, etc. and maybe this is the unique advantage that we can find in this tool.

3.1.1.4 ***System Studio***

This tool, developed by Synopsys, allows building, simulating and analyzing complex digital signal processing (DSP) systems in a system-level design environment [Inc08].

The description of a DSP system can be split in functional and structural models. The functional model is defined in a hierarchical block-oriented environment where the description of the lower blocks is made in C/C++. The designer can used the different model domains offered by this tool to describe the functional model of his system. The available model domains are Dynamic Data Flow and Finite State Machine. Each domain can be mixed in the model in order to generate the behavior of the wished system. In order to model the architectural part of the system, this tool uses SystemC language and the designer can build an system architecture graphically by drag-n-drop hierarchical blocks described in SystemC.

System Studio has a library focused to different DSP systems domain, such as

Broad band access (ADSL, DOCSIS cable modem), 3G wireless (CDMA, EDGE), other wireless (Bluetooth, IEEE 802.11, GSM), digital video (MPEG-4), and others.

System Studio gives the tools to analyze the system model performance when the functionality of the system is mapped in an architecture proposed by the designer. We do not find any information that this tool helps in the decision process of the partition hardware/software.

3.1.2 Academic Tools

3.1.2.1 ***Metropolis***

This is a tool developed by the University of California in Berkley and it follows strictly the Platform-based Method proposed by the same university. The base of Metropolis is a metamodel where there are defined the main concepts: process and medium. These concepts correspond to the separation of the model in actions (processes) and communications (mediums). Each process has associated a sequential program called thread. In order to communicate each process, each one of them has ports, and each port is specified with an interface. The interfaces are implemented by mediums which allow the division between actions and communication, getting to the designer the facility to reuse previous models [BWH$^+$03].

With the Metropolis metamodel, the designer can represent the functional modeling, the architectural modeling and the candidate platform gotten from the mapping between functional an architectural modeling. Hence it uses the same language to describe behavioral and structural representation, the metamodel helps to do a fast mapping between the components of each representation.

The expression of the non-functional features, such as time, power and size is described in quantity managers. Execution process constraints are represented in the form of logical formulas in order to restrict the execution of the model.

Metropolis offers a framework, which provides to the designer verification, simulation and synthesis of its models.

3.1.2.2 ***Ptolemy extension as Codesign Environment (PeaCE)***

This tool is developed by Seoul National University and it is focus on multimedia system design, however the authors say that the tool can be used in other domains. PeaCE follows a traditional methodology to design embedded system and its functionality can be combined with other more specialized tools in each design step by generation of XML files [HKL+07].

The designer begins the system design, building the system level specification using extended Dataflow model and Finite State Machine. In order to simulate the model, PeaCE generates a code C from the model, compiles it and executes it.

When the system model is verified, the designer proceeds to explore the possible architecture candidates. The designer can use the pre-build platform which are available in the PeaCE's data base, or it can build a new one, using the available components in the library.

Once the platform is build, the designer can map the system functionality to the components. In this step, the designer decides the partition Software/Hardware and the component selection. PeaCE helps in this task performing a schedulability analysis, and calculating the performance of each functional block contained in the system model and the communication between different blocks.

After the partition is completed, PeaCE co-simulates the system to generate memory traces from the processing elements. With this information, the designer can explore bus architectures and select the best option. When the system is completed, it can be verified in PeaCE by simulation or using other third tool.

Verified the virtual system solution, PeaCE allows generating C code for processor core and VHDL code for FPGA implementation.

PeaCE follows the platform-based approximation and defines clearly how each step can be implemented.

3.1.2.3 ***Model Integrate Computing (MIC) and Milan***

MIC is an environment created by Vanderbilt University to build software systems. However, the group, which develops this environment, uses MIC to create a tool to design embedded system. This tool is called Milan [PL01].

The environment MIC focuses on the formal representation, composition, analysis, and manipulation of models during the design process. It places models in the center of the entire life-cycle of systems; including specification, design, development, verification, integration, and maintenance following an Model-based development approach [fSIS09].

MIC is composed by the following tools:

- Generic Model Environment (GME): Tool allows creating Domain-specific models and program synthesis environments.
- Model Management tool suite
- Model Transformation tool suite
- Design Space Exploration tool suite

With these tools, the design can create an own tool to design a system in a specific domain following its own methodology. An example is Milan [PL01].

Milan is a tool based on MIC to design embedded systems. This tool has the architecture which is depicted in the Figure 9. A system in Milan is composed by three models: Application model, Constraint model and Resources model. The Application model represents the functionality of the system, Resources model describes the components where the functionality of the system will be implemented. Constraint model is used to characterize the components' limitation in connection level, power consume, etc. Also, this model represents the non-functional requirements that the system has to complete, such as power, performance, size, etc. The language to define these three models is defined in GME.

Milan uses the MIC engine transformation to convert the models expressed in GME to standard languages such as C, Java, VHDL. These languages can be interpreted by specialized-tools, such as Matlab and SimpleScalar, to simulate the model built in Milan. The simulation results are returned to Milan to verify the model behavior.

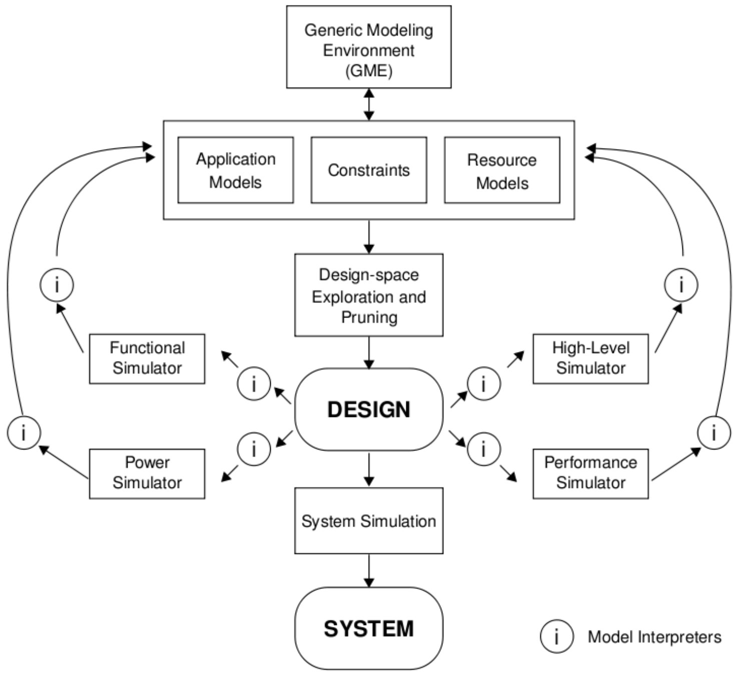


Figure 9: MILAN architecture [PL01].

In order to reduce the exploration time when the designer is deciding the partition Software/Hardware, Milan uses the MIC Design Space Exploration to reduce the component selection space, in order to map the elements of the application model to the resource model.

There are many applications based on MIC, they are available in [fSIS09]. The group, which developed Milan, has not continued its development.

3.1.2.4 ***Artemis***

Artemis is a workbench which contains a set of tools which allows modeling applications and SoC-based architectures at high level of abstraction, mapping the applications on the architecture and estimating performance numbers through co-simulation of application and architecture models [Pim05]. Its domain focus is multimedia solutions.

The main tools of this workbench are:

- Compaan

- Laura
- Molen
- Sesame

Sesame is the base of the Artemis workbench. This tool allows choosing an specific-domain platform architecture (Architecture template) and building an application specification in a sequential program. The platform architecture is instantiated to an architectural model, while the application specification is transformed to a concurrent model in Kahn Process Network (KPN) using Compaan. Between the architecture model and the application model, there is a mapping model which provides the performance result of the mapping between the application model elements and the architecture model elements. This performance result achieved Laura and Molen tools. Laura is used to map the KPN diagram in the architecture and to generate code (Verilog, VHDL or C) to co-simulate and to give performance numbers. Molen is used to calibrate the performance of the architecture model.

3.1.3 Summary, Strengths and weaknesses in Platform-based Tools

In this section, we review the main features of the industrial tools. In the table 1 , we show the tools from the industrial world. They are the same tools found in [DPSV06]. They have its own focus on the system design and they use different abstractions to represent the functional, architecture and mapping models. Most of them use Model of Computation (Dataflow, Finite state machine, etc) to represent the models, but each one has a specific limitation. In the table 2 are expressed the strong and weak points of these tools.

In the table 3 , we show the tools from the academic groups. They generally do not use standard abstractions to model systems, instead they extend it to reach its own representations. In the table 4, it is shown the strong and the weak points of each academic groups.

Company	Tool	Abstraction						RTL Trans.	Alg. Repr.	Domain
		DE	DE/CT	DDF	SDF	FSM	TLM			
CoFluent Design	CoFluent Studio				X		X		C, C++, SystemC	Mix
MLDesign Technologies	MLDesigner	X	X	X	X	X				Mix
Mirabilis Design	Visual Sim product family		X		X	X	X	X	C, C++, Java, SystemC, Verilog/VHDL	Mix
Synopsys	System Studio			X	X	X	X	X	C, C++, SystemC	Mix

Table 1: Industry tools which reach the Complete Platform-based design concepts.(**DDF:** Dynamic Dataflow, **DE:** Discrete Events **FSM:** Finite State Machine **SDF:** Syncronous Dataflow **TLM:** Transaction Level Model)

Company	Tool	Strengths	Weaknesses
CoFluent Design	CoFluent Studio	- The tools is based on MCSE methodology, which is a Platform-based methodology. - The tool generates SystemC code from the models made. - The mapping action is made by drag and drop	- It has generic models to aluate the platform where the functionality will be implemented. - It cannot generate VHDL or Verilog Code.
MLDesign Technologies	MLDesigner	- The tool provides a large range of Model of Computation to model the functional and the architecture of the system.	- It cannot generate neither C/C++ Code nor VHDL/Verilog Code.
Mirabilis Design	Visual Sim product family	- The extensive library that the designer can reuse in order to build his own designs. - Possibility to use the tools by a simple internet browser. - HDL code generation.	- Few information available
Synopsys	System Studio	- Good library for Digital Signal Processing Systems - HDL code generation.	- Its focus is only on Digital Signal Processing Systems.

Table 2: Strengths and weaknesses in industrial tools.

Company	Tool	Abstraction	RTL Trans.	Alg. Repr.	Domain
U. of California, Berkeley	Metropolis	Its own model language, following the TLM concepts.		MoC	Mix
Seul National Univ.	Peace	Extended SDF, Extended FSM	X	C	Mix
Vanderbilt Univ.	MIC	DSML			Mix
University of Amsterdam, Leiden University and Delft Univ. Of Technology	Artemis	Khan process networks (KPN)	X	MATLAB, C++, Perl, SystemC	Media
U. of Southern California and Venderbilt University	Milan	SDF, DDF		C, Java, MATLAB, SystemC, VHDL	Mix

Table 3: Academic tools which reach the Complete Platform-based design concepts.

Company	Tool	Strengths	Weaknesses
U. of California, Berkeley	Metropolis	- It uses the same language to describe functionality and architecture. - Constrains can be expressed using its model language.	It is not clear how the mapping can be made.
Seul National Univ.	Peace	- It provides an easy way to integrate with other tools. (Using XML) - It guides the architecture exploration in order to make a good mapping. - HDL code generation.	-The mapping is made by testing and error correction.
Vanderbilt Univ.	MIC (GME, GreAT, Desert, UDM, OTIF)	- The designer can create its own language design and its own design flow. - It is possible to do a previous automatic exploration to reduce the platform selection space.	- The designer has to create the tool (tool chain).
University of Amsterdam, Leiden University and Delft Univ. Of Technology	Artemis (Compaan and Laura, Sesame, Spade)	- It allows transforming a sequential model into concurrent model. - Mapping action is made using multi-objective optimization.	- The tool's focus is Multimedia applications.
U. of Southern California and Venderbilt University	Milan	- They introduce the concept of Constrain Model in order to help to the mapping process. - The mapping exploration is made by using Ordered Binary Decision Diagram.	The development of this tool depends of the MIC support.

Table 4: Strengths and weaknesses in Academic tools.

3.2 Model Based System Design and Model Driven Architecture

There is not a complete tool which can develop all the steps of the methods proposed in Model Based System Design. Different methodologies are described in [Est08] and they use different tools to support them. Estefan relates Model Based System design (MBSD) with MDA based on a work done by Cloutier [Clo08], who said that the system engineering productivity increases 10-20% using of MDA, compared with existing system engineering methods.

Object Management Group (OMG) proposes to use standard language such as UML and SysML to represent the system's concepts in a model based on these languages and implement each step of the MDA method in these languages. There are many tools which help to create models in these languages and we only mention the most representatives, specially tools which support SysML, which is our language base of our work.

In [For09], there is a list of the tools which allow creating system models based on SysML. We classify them in two groups: Commercial and Open source tools. Commercial tools are tools developed by the industry which source code is property of the enterprise which develops the tool. Nobody can access to them, and the user has to pay to use it. In the other hand, open source tools are tools developed by the industry or academy, the source code is accessible by everybody and the user has not to pay to use it.

3.2.1 Commercial Tools

3.2.1.1 *IBM Rational Rhapsody and IBM Rational Tau*

Rhapsody is a tool which enables to the designer, system designer and software designer, to create real-time and embedded system model based on UML/SysML. This tool has the capability to do the traceability between the stakeholder requirements and the elements of the model using its link with tools such as DOORS. Rhapsody also has an environment to simulate the model behavior which allows to the designer to verify early their designs and also it allows validating them with the stakeholder in an easy way. This tools also helps to generate documentation automatically in different formats such as HTML, rich text format and Microsoft PowerPoint. Finally Rhapsody can generate code in C, C++, Java and Ada languages for 8, 16 and 64 bit applications [IBM09b].

Tau is other IBM tool which has the same purpose of Rhapsody, just its focus is the design of complex systems [IBM09a]. UML is its main graphical language, and also it has a SysML plugin, through this is not used too much. This tool, as well as Rhapsody, can be generate documentation, simulate and test models and link requirements with model elements. An important feature in this tool is the possibility to collaborate with teams and organizations.

3.2.1.2 ***MagicDraw***

MagicDraw is a No Magic Inc. tool which is used to model systems in UML mainly. This tool has a SysML plugin which is developed with the help of INCOSE, which is one of the SysML standard developers. MagicDraw has connections with Microsoft Excel, MATLAB and Mathematica in order to execute the simulation of the system model built in SysML using ParaMagic plugin [Inc09b]. As Rhapsody and Tau, it binds the requirements with the model elements, using Cameo DataHub [Inc09a]. This tools provides two profiles to extend the description that the designer can make in SysML: MARTE and SYSMOD. MARTE profile [OMG08b] is an extension of UML for the development of real-time and embedded systems, which can be used in SysML. SYSMOD is an extension proposed in [Wei08] to describe more detailed a model designed in SysML and it is used in a methodology also proposed in [Wei08]. MagicDraw has the capacity to generate Java code using Xholon [Pri09] and C/C++ code using SinelaboreRT [Sin09].

3.2.1.3 ***Enterprise Architecture***

Developed by Sparx Systems and, as MagicDraw, it is focus on the system modeling on UML and it has the possibility to use SysML using a plugin developed by them.

Unlike the previous tools, it only gives the elements to model in SysML, but it does not enable to generate specific code to simulate the behavior of the system or to link the requirements from other tools to the requirements diagram in SysML like MagicDraw.

This tool is very used in the software development, although in the case of complex system is too limited and it only can be used to document a system design.

3.2.2 Open Source Tools

3.2.2.1 ***Topcased***

Topcased is a toolkit dedicated to realization of critical embedded systems [Top09]. This tool aims to develop editors for specification, design and implementation of systems, to integrate formal-verification tools, and to generate code and documentation automatically to the implementation of systems [FV09].

AdaCore, Airbus France, Anyware Technologies, Atos Origin, CNES, Laboratoire d'analyse et d'architecture des systmes, Communication & Systmes, EADS Astrium, ENSIETA, ESEO, ENSEEIHT, Ellidiss Technologies, Fdration de Recherche en Informatique et Automatique CNRS FR 2238, INSA, Institut de Recherche en Informatique de Toulouse, Institut National de Recherche en Informatique et en Automatique, MICOUIN Consulting for Innovative Systems Engineering, Laboratoire Modelisation Intelligence Processus Systmes (Universit de Haute Alsace), ONERA, Rockwell Collins, Sodifrance, Siemens VDO, Sogeti High Tech, TNI-Software, Tectosages, Thales Avionics, UFSC DAS, Universit Paul Sabatier are the partners in the developing of this project.

This platform has model editors such as UML, SysML, AADL, SAM which are used to describe the specification of a system. By transformation, the system model can be verified by other simulation or analysis tools using a pivot language such as FIACRE [Berthomieu08], which does the bridge between the model tools and the verification tools. The model can also be transformed to code, e.g. UML to C or UML to Java. Aditionally, there is a tool which made the transformation from the model to natural language in order to create the supporting documentation to the development of the system.

3.3 HiLeS Method tool

3.3.1 HiLeS Designer

This tool is developed by the Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) and the Universidad de los Andes. HiLeS Designer is the tool which supports the HiLeS method explained in the chapter before.

The objectives of HiLeS Designer are:

- Build system design models using HiLeS formalism.
- Generate automatically a virtual prototype in VHDL-AMS.
- Verify the system design behavior by simulation.
- Verify formally the logical system design by TINA.
- Reuse designs.

At the present, this tool is being redefined to follow the new conceptions in model driven design. The group in charge to develop this tool wants to add new features as the system design representation using general language model such as SysML and UML. In these languages, the designer can make the first description of the system and transforms the SysML or UML model to HiLeS in order to verify the model before to continue with the details of a system design. This is a path to give execution properties to the specification described in SysML and UML; it also allows the designer to make an early verification before to go down abstraction levels in its design. This transformation can also help to search quickly a physical system which matches with the features which are represented in the virtual prototype generated by this tool, especially if the physical system was designed using this tool.

3.4 Conclusion

In this chapter, we present the tools used in the design of embedded systems, their relationship with the methods exposed in the chapter before and their features, developers, strengths and weaknesses.

We can see that the tendency in these tools to create system model without having an execution language predefined. This is the main conception in MDA, where the system model is clearly independent from the platform, in this case, execution language. Also it is clear the independence between functionality and architecture; the same concept of PIM and PSM in MDA. These tendencies are reflected in the academic or open source tools which will be the future commercial tools; an example of this evolution is Ptolemy, where it is used in several commercial tools, such as MLDesigner and Visual Sim.

In the next chapter, we will present our methodology based on the EIA-632 standard. This methodology uses SysML as language to express the definition of the behavior and structure of a system design. We will describe each step of our methodology and we will give some illustrated examples to guide the reader in the application of our methodology.

CHAPTER IV

Proposed Methodology

In this chapter we present the methodology which we propose to use in the design of complex systems, using the standards EIA-632 and SysML. First, we explain what a methodology is, following the Estefan's definitions [Est08] and what its composition is. Second, we make a brief description of the standards EIA-632 and SysML. Finally, we present the methodology and we connect between our methodology, the standards EIA-632 and SysML.

4.1 Methodology definition

Estefan in his article extracts the definition of methodology from Martin [Mar96], who says that a methodology consists of the interaction of three elements: Process, Method and Tool.

- *Process:* A process is a logical sequence of tasks performed to achieve a particular objective. Here, it is defined "what" it is done in each task, but not "how".
- *Method:* It defines "how" the task is performed. In a method can be included different tasks to perform a single task of a process.
- *Tool:* It allows accomplishing the "how" and the "what". It is the instrument to help to apply a particular method following a specific process.

These three are the fundamental parts of a methodology and we show the relationship between them in the figure 10.

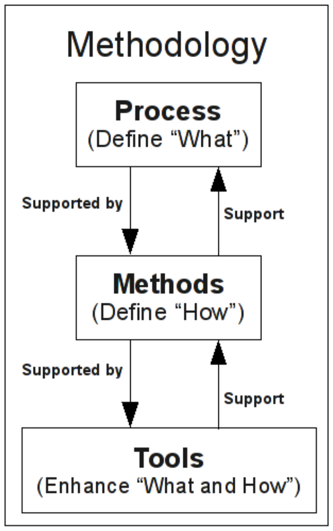


Figure 10: Relationship between the methodology parts. [Est08].

4.2 EIA-632: Processes for Engineering a System

The purpose of this standard is to provide an integrated set of fundamental processes to aid the developer in the engineering and reengineering of a system [EIA99]. The standard defines different processes to create a system, theses processes are listed in the figure 11.

We are interested in the "System Design" group, where the stakeholders requirements are transformed to Specification, Drawings and Models in order to manufacture the final product. This group has two processes: Requirements Definition Process and Solution Definition Process. Requirement Definition Process has three requirements:

14 - Acquirer Requirements

15 - Other Stakeholder Requirements

16 - System Technical Requirements

Where the transformation is defined from the stakeholder requirements into system technical requirements. The numeral corresponds to the requirement number in the standard.

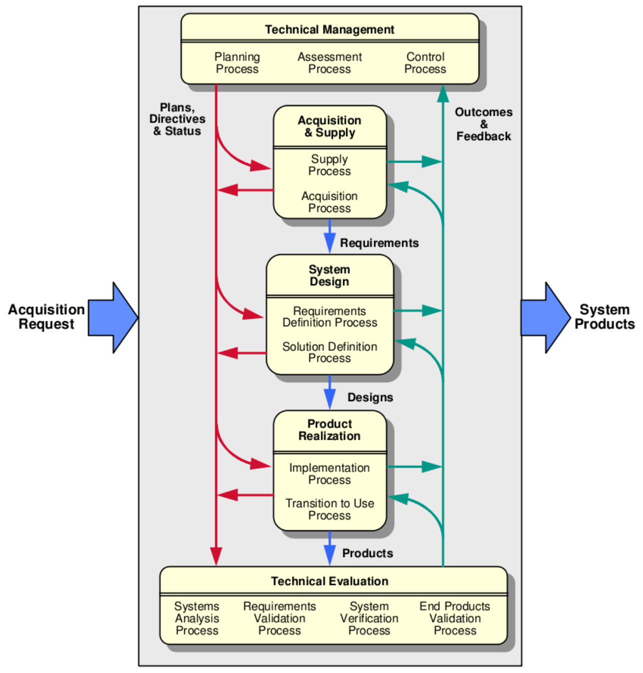


Figure 11: Processes defined in EIA-632. [EIA99].

Solution Definition Process has also three requirements:

17 - Logical Solution Representations

18 - Physical Solution Representations

19 - Specified Requirements

Where the transformation are defined from the system technical requirements into an acceptable system solution, described in models, drawings and specifications.

4.2.1 Acquirer Requirements

The acquirer requirements are the needs of customer, user or operator that the system has to perform. This requirement involved the requirements priority, the decomposition of complex requirements, and the record of them.

4.2.2 Other Stakeholder Requirements

These are the other requirements which can constrain the acquirer requirements, such as laws, technology, standards and so on. Additionally, these requirements can complete the existing requirements given by the stakeholders. The requirements' record is also included.

4.2.3 System Technical Requirements

In this requirement is analyzed the relationship between the acquirer requirements and other stakeholder requirements, its results are requirements which are unambiguous, complete, consistent, achievable and verifiable. This requirement proposes to create operational profiles, environment to use the system, how well the system has to answer, frequency to use, interfaces and functional requirements. Like the other requirements, this requirement includes the recording information in a database.

4.2.4 Logical Solution Representations

When we have the system technical requirements, the standard EIA-632 proposes to build a logical solution representation which follows the system technical requirements. This logical solution is an analysis of the system technical requirements where these requirements, expressed in natural language, are transformed to a logical representation (functional flows, behavioral responses, state and mode transitions, time lines, data flows, etc.) which can be verified. The standard suggests analysis such as Functional analysis, object-oriented analysis, structural analysis and information engineering analysis.

Each requirement expressed in the system technical requirements is represented in a element or concept of the logical solution.

The analysis can conduit to other requirements which are called derived requirements by the standard.

The logical solution has to be recorded as well as the derived requirements.

4.2.5 Physical Solution Representations

Having logical solution representations of the system, the designer proceeds to build a physical prototype based on the logical solution, the derived technical requirements and the unassigned system technical requirements. In this step are decided the partition software/hardware, the physical interfaces, the technology, and ohters concerns to the development of the physical solution.

In these requirements, there are generated different physical solution alternatives and it is chosen the best solution which is among the constraints expressed in the other stakeholders' requirements.

Finally, the result of this requirement is recorded in a database.

4.2.6 Specified Requirements

Chosen a physical solution which performs the logical solution and follows the constraints expressed in the other stakeholders requirements and derived technical requirements, the designer has to specify the requirements of the system; it includes: functional and performance requirements, physical features, and test requirements.

4.2.7 Building block

The EIA-632 standard defines that the system is represented by a conceptual structure called "building block". A building block consists of End Products (product to be developed or ready to use) and Enabling Products (products which allow the development, the test, the production, etc of the end product).

Each End Product can be decomposed in two or more subsystems, if an End Product cannot be reused from another project or bought from another supplier. In the figure 12 is depicted this concept. The figure shows the descomposition of a system and how a subsystem is a system at the same time contained End Products and Enabling Products.

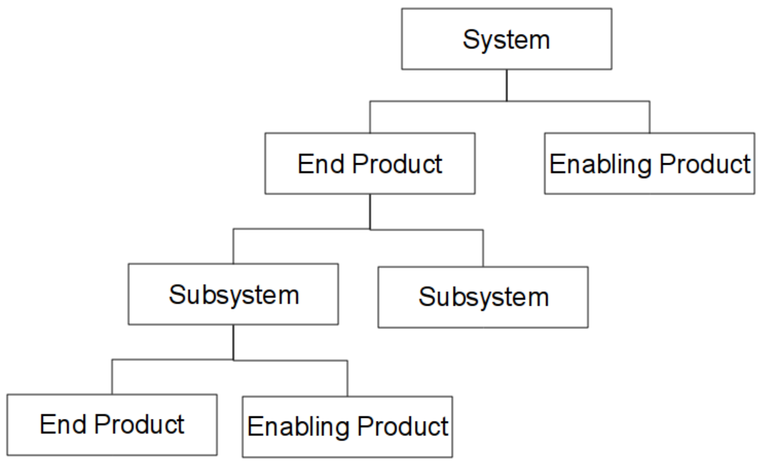


Figure 12: Building block concept.

4.3 System Modeling Language (SysML)

System modeling language or SysML is a graphical language used to support the specification, analysis, design, verification and validation of systems that includes hardware, software, data, personnel, procedures and facilities [OMG08a]. It is developed by the Object Management Group (OMG) with the help of the International Council on System Engineering (INCOSE).

SysML is a UML profile, which is the utilization of some UML diagrams, the expansion of others in order to create a new language, in this case a language oriented to System engineering. In the figure 13, we show what the relationship between UML and SysML is.

Like UML, SysML has two main groups: behavioral diagrams and structural diagrams.

4.3.1 Behavioral Diagrams

The behavioral diagrams describe or define the functionality of the system or subsystem, its interaction between users, and extern system, also intern systems or subsystems. In these diagrams we find:

- *Use Case Diagram:* This diagram describes the main services offered by the system to its environment. The environment can be a user or another system.

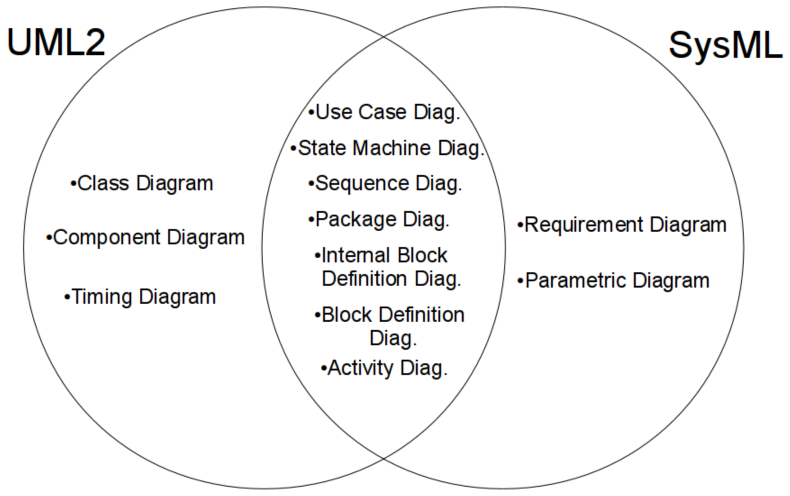


Figure 13: Relationship between SysML and UML.

- *Sequence Diagram:* This diagram defines the scenario of the use case and describes the different iteration between the objects which are part of the system, and also with the environment. In this diagram, the designer deduces the services that the system and subsystem have to use or implement.
- *Activity Diagram:* This diagram is used to describe the flow of actions which are executed in the system or subsystems. Since UML 2.0, activity diagram behavior is similar to a Petri Net.
- *State Machine Diagram:* This diagram describes the states and state transitions of a structure.

4.3.2 Requirements Diagram

This diagram is used to represent the stakeholder requirements expressed in natural language. This language allows having traceability from the requirements to other diagrams elements such as use cases diagram and block definition diagram.

4.3.3 Structural Diagrams

These diagrams define the static definition of the system, such as the blocks (system), composed blocks (sub-systems), services offered by blocks, ports, etc. In these

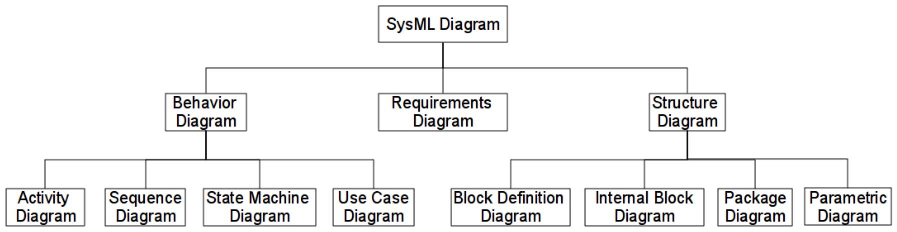


Figure 14: SysML diagram groups.

diagrams we find:

- *Package Diagram:* This diagram is used to organize the different diagrams and diagram elements in a project. We can use it to classify some diagrams according to its type, origin and feature.
- *Block definition Diagram:* This diagram defines parts of the structure of a system and its subsystems. The designer can define the hierarchy in the system structure and split its design in different level of abstractions.
- *Internal Block Diagram:* This diagram captures the internal structure of the system or subsystems and its relationships. The relationship is defined in terms of services, flows and ports.
- *Parametric Diagram:* This diagram defines the relationship between the constraints identified on the system. This relationship allows to calculate values which are not part of the specification using third tools.

In the Figure 14, we show the groups and their elements in SysML. We can find more information in [Wei08], [FMS08] and on the standard [OMG08a].

4.4 Methodology

Nowadays, the complexity of the systems has risen. The traditional methodologies in system design, and especially in the design of complex system such as embedded systems for airplanes and automobiles are more difficult to make at market time and the technology progress increases each time faster.

The community in the design of complex system has begun to change the abstraction level in the design and a new conception of “system level” is created. We propose a methodology which follows this principle of “System level” from a point of view “object”. We base our methodology in a previous work described in [EOPP05] and we make some extension and complete some gaps in the methodology.

In this section, we will explain the methodology steps that we developed, which language base is SysML and whose process follows the EIA-632 standard.

4.4.1 Requirement Analysis

The fist step in this methodology is to get the needs and wishes from the stakeholders, these are called “Requirements”. Stakeholders are the people involved in the system, they are users, operators, maintainers, clients, etc. Following the EIA-632, these requirements are split in two: Stakeholder requirements and System technical requirements. Inside Stakeholder requirements we have: Aquirer Requirements and Other Stakeholder Requirements. We use the same concept defined in EIA-632 to express the requirements, using SysML as description language. We use the package diagram and the requirement diagram to express the system requirements.

Our Requirement Analysis is presented in a package diagram called “Requirements”, and inside of this, we have two packages called: “Stakeholder Requirements” and “System Technical Requirements”. The package “Stakeholder Requirements” is split in two packages: “Acquirer Requirements” and “Other Stakeholder requirements”. The information in each package is the same information described in the section 3.2, the first one is the description of the stakeholders needs, and the other one is the constraints which limit the acquirer requirements.

Inside the package “Acquirer Requirements”, we create a package for each stakeholder who participates in the development of the system. Each stakeholder has his participation, his own contribution to the project and all its ideas taken into account to the project development. In [INC00], there is a complete guide for the person who captures the requirements of a system. For each “stakeholder package”, we create different packages following a standard classification, this helps to guide the designer on what he has to ask to the stakeholder and to cover a big part of the

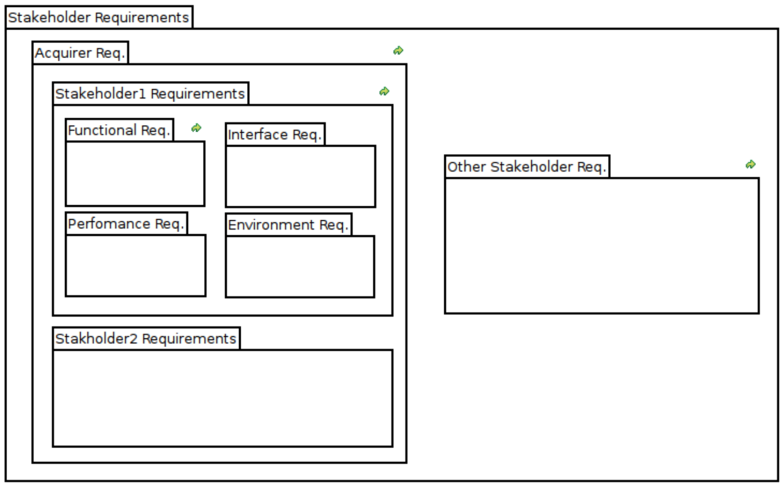


Figure 15: Stakeholder Requirements Package Structure.

requirement space in the building of the system. These packages are: Functional requirements, Performance requirements, Interface requirements and Environment requirements. If the designer considers that there are more packages, he is free to add them.

For each classification package, the designer creates the requirements presented by the stakeholder according to his own classification. It is important to note that the requirements are expressed in natural language and they may be derived from texts, presentations, drawings, etc. which are given by the stakeholders. The subject of the requirement description has to be the stakeholder, other stakeholder or systems which interact with the system in developing. The reason of this notation is to reduce the restriction that the designer and also the stakeholders can have when the requirements' subject is the system [Mai08]. In the figure 15, we show how the distribution of the package is.

In the package "Other Stakeholder Requirements", the designer can add packages for each constraint topic, e.g. standards, environment laws and rules, and inside of each package its own classification.

It is possible to make different iterations in the analysis of the stakeholder requirements in order to have all the information possible concerning the system. Also, it is useful to use relations such as "contain to" when the requirements have to be

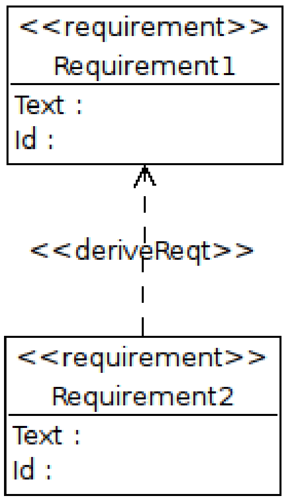


Figure 16: Connection “Derive Requirement”.

split in order to detail more the stakeholder whishes.

When the Stakeholder Requirements are finished, the following step is to derivate, from the stakeholders requirements, the system technical requirements. These requirements are functions and constraints that the system has to fulfill. Now, the subject in each requirement has to be the system, and the description of these requirements is more exact, more detailed and atomic, that means, only with one action in the description of the technical requirement.

The system technical requirements are also represented by a package, where inside of this, the designer can build a classification, as well as he did with the stakeholder requirements, they are: Functional, performance, environment, technology, interface requirements. In side each classification package, we create the technical requirements derived from the stakeholder requirements, specifying its relationship, using “deriveReqt” connection. In the figure 16, it is shown how this connection can be done. In this figure “Requirement2” is derived from “Requirment1”.

A good way to control a large number of requirements, it is to use the table format offered by SysML.

4.4.2 Functional and Architectural Analysis

Once the System technical requirements are done, we begin the construction of the logical solution representation of our system, following the Solution Definition Process of EIA-632 standard. We split this solution in two representation groups: Behavioral and structural.

In the behavioral representation, we describe the dynamic part of the system using a method by steps, which starts in a high level description, and finish in a detailed functional description using the different diagrams offered by SysML. The structural description is the static part of the system. In this group is proposed a logical architecture, where is grouped the functionality of the system in subsystems. Also in this group, we can see the relationship between the system, its subsystems, and its environment, and which are the services the system and subsystems offer. These services help to show what the functionality of each subsystem is and they help to re-use of the components or objects in other projects.

The behavioral description is split in two stages: Definition of Use Cases and scenarios using the sequence diagram. The structural description is divided in three levels: Hierarchical composition, Relationship between components, and Constraints.

4.4.2.1 ***Behavioral Description***

This description is contained in a package diagram called Behavioral definition, which is at the same level of the Requirement package.

- *Use Cases:* The use cases express the possible operation scenarios of the system. A scenario is composed by different actions where the system and the actors have to perform in order to complete a task or functionality. It is derived from the system technical requirements, specifically from the functional requirements, where it is described the behavior of the system. In this stage, features, as well as performance and technology, are not taken into account for simplicity of the design and the main idea is to be clear in the interpretation of the functionality described in natural language in the following step. In the use cases, the actors are defined and their relationship with the system.

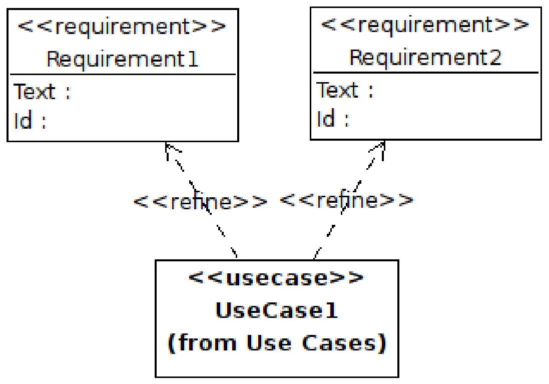


Figure 17: Relationship between requirements and use case.

- *Sequence Diagram:* The scenarios are described by sequence diagrams, which show in more detail the iterations among actors, system and subsystems. We create different level of description in order to help identify the actions which can do between the entities and distribute the task between subsystems. We will explain in more detail these levels in "Abstraction Levels".

Once the use cases are defined, we make the relationship with the requirements defined in the Requirement Analysis. Each functional system technical requirement has to be linked with a use case in order to give the relationship between the first two steps. If we find that some functional requirement is not represented in the defined use cases, we realize our functional analysis is not completed. In the figure 17, we show how it is possible these relationships. We can see the UseCase1 refines the Requirement1 and Requirement2, that means the functionality expressed in the use case and its scenario is according to the requirements and it is the implementation start point.

4.4.2.2 ***Structural Description***

The structural description is represented by a package diagram with the same name and it is at the same level of requirements package and the behavioral description package. Inside this package, we make a partition between components and constraints, where each of them has their own package. Components package contains the definition of the architecture of the system in logic level, which means, this architecture is only for the logical solution representation, and it can change when the physical solution representation is created. On the other hand, constraints package contains the constraints that the system has to achieve.

Inside the components package, there are two packages: "components composition" and "Ports and Services Definition". The components composition package contains a Block Definition Diagram (BDD), which is used to define the entities generated in the sequence diagram. The entity is represented by a block where its methods (internal functions which are called by services), its properties and its constraints are related. Also in this package, we define the system hierarchy and we can see the abstraction levels of the system which are represented by subsystems. This representation is according to the concept "building block" proposed in the EIA-632 standard where a system can be decomposed in subsystems and so on. Inside of the block which is decomposed in blocks, we can create an internal vision using the Internal Block Diagram (IBD). This diagram allows defining the services which are offered or requested by the block parts. Also it defines the relationship among the parts of the block. These services are derived from the actions defined in the sequence diagram.

In order to use the services inside the IBD, it is necessary to define them formally. Inside of the Ports and Services Definition package, we define in a BDD the ports which request and offer services. We can have different services offered and requested by two blocks and these services are encapsulated in an interface. In the figure 18, we show how this definition is possible. We have two ports, "Port1" and "Port 2", which will be used by the blocks in a BDD inside of Components Composition package. The relationship between the two ports is an interface called "interface1". This interfaces contains two services, "service1()" and "service2()". "Port1" has a realization relationship with "interface1", so the block which uses "Port 1", implements the services of the interface. "Port2" has a usage relationship, which means the block which uses this port, uses the services inside the "interface1".

In the figure 19, we depict the use of the ports defined in Ports and Services Definition package.

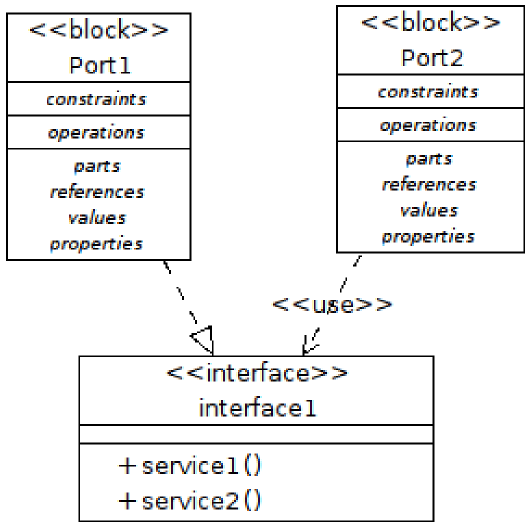


Figure 18: Services and ports definition.

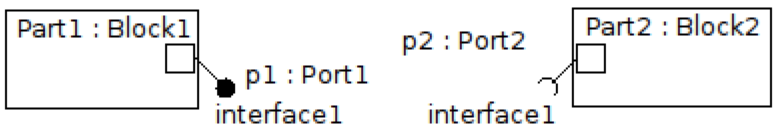


Figure 19: Ports use.

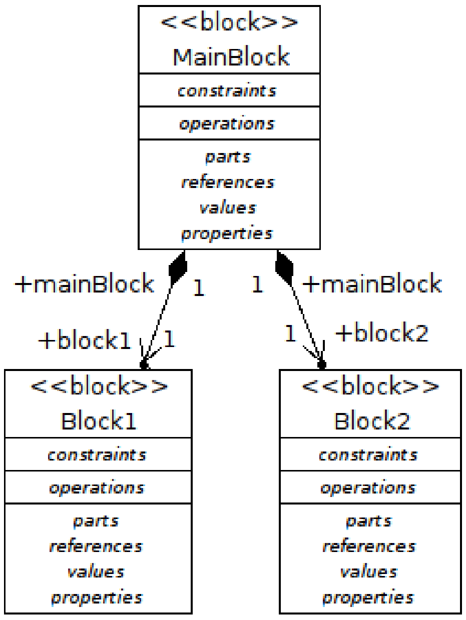


Figure 20: BDD of a block composition.

We can see two parts, “Part1” and “Part2” which are instances of “Block1” and “Block2”, blocks which are defined in the BDD of Components Composition package. These blocks are subsystems of a bigger block called “MainBlock”, and its hierarchy is showed in the figure 20. “Part1” uses “Port1” and it has “interface1” as interface. The symbol lollipop means “Part1” implements “interface1” and it offers the services contained in the interface. “Part2” uses “Port2” and it has “interface2” as interface. The symbol “half circle” indicates “Part2” uses or requests the services offered by “interface1”. In order to show the relationship between “Port1” and “Port2”, we can connect them by a connector.

Finally, the constraints package has a BDD where are expressed the constraints of the systems. Like the BDD used in the Components Composition package, each constraint is represented by a block called “Constraint block”. These blocks can be split in sub-blocks where the meaning of this partition is to calculate the value of the block by the calculus result of their sub-blocks. The relationship between sub-blocks inside a block is represented by a Parametric Diagram where each constraint has ports called “parameters”, which are related with the ports of their internal

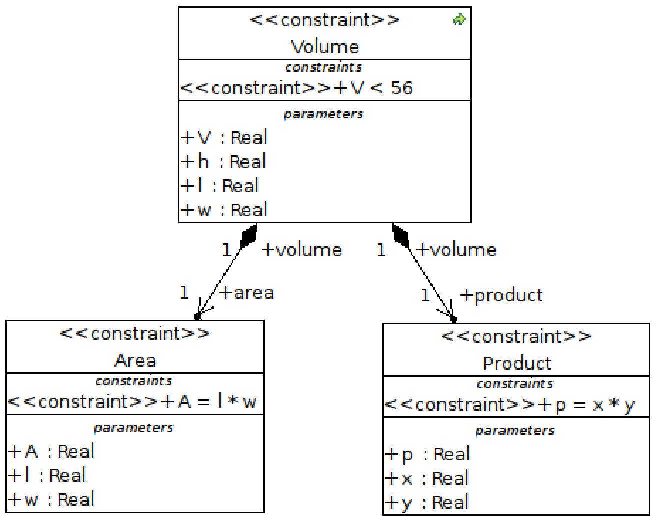


Figure 21: Constraints blocks Example.

constraints. These internal constraints have also relationships and these are possible by their parameters. In the figure 21, we show an example of a constraint.

We have a main constraint which is that the volume of a box has to be less than 56. The information we have from the box is the high, length and width. In order to calculate the volume, we need the area which is calculated by the product of its length and width. This product is represented by a new constraint block which does the operation to calculate the area. When we have the area, we have to do the product between the area and the high to obtain the box volume. Thus, we use another constraint box to represent the operation "product" and we make the relationship among this block, the area block and the volume block. In the figure 22, we show the parameter diagram of this example, where we can see the relationships between the parameters of the blocks.

These constraints can be attached to the block which represents a system or subsystem in the BDD of Components Composition Package. In the figure 23, we show this link.

The SysML objective is to connect the parametric diagram to third tools in order to resolve the system constraints, when the parameters change. In [PBF$^+$07a] and [PBF$^+$07b], we can find work on this topic.

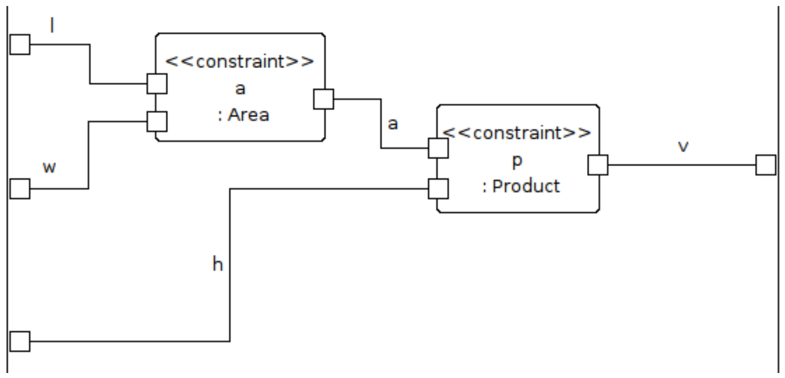


Figure 22: Parametric diagram Example.

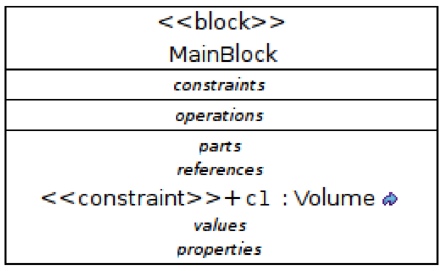


Figure 23: Attach constraints to a block.

Many of the properties or parameters have a type joined. In system engineering there are many types in order to describe a property, such as time, frequency, length, speed, and others. These properties in general have a value and a unit or maybe only a value. The units can be from different measure systems such as the International System of Units. We have to define these types in order to use in the model.

We create a new package dedicated to define data types and this package is in the same level of Requirement package, Structural package and Behavioral package. Inside this package we create a new package containing the data types definition from a specific measure system. A measure system package has three elements: a dimension, a unit and a value type. The dimension is a physic quantity that we want to measure, e.g. length, temperature, etc. The dimension has a unit, which is an specific quantity of a dimension; in length for example has meter, millimeter, centimeter, etc. Once the dimension and its unit are defined, we can assign a value type, which is the quantity of the dimension. We base this work on the Apendix C of [OMG08a]. In the figure 24, we illustrate a data type definition example which is "meter" from the International System of Units. First, we define the dimension for "meter" which is "length". Later, we represent "meter" as a unit. Finally we assign "real" as the quantity of this unit.

If the package Structural definition and Behavioral definition want to use the datatypes defined, it is necessary create a connection among them. This connection is called "import".

In the figure 25, we depict the package distribution of this step.

When the structural definition is finished, we have to connect the requirements to the blocks in order to complete the traceability of our methodological steps. Every requirement has to be represented in a block, it includes functional, performance, interface and environment requirements. This trace gives us an idea that our logical solution representation satisfies the requirements generated in the first step of our methodology. In the figure 26, we depict this relationship between requirements and blocks. We can note the link is made in the same system technical requirement package, such as the use cases and it uses the relationship "satisfy". With this final step, each analysis result is reflected in these links (Requirements Use Cases Blocks)

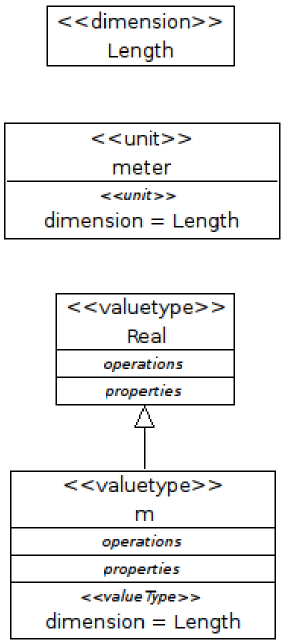


Figure 24: "Meter" unit definition.

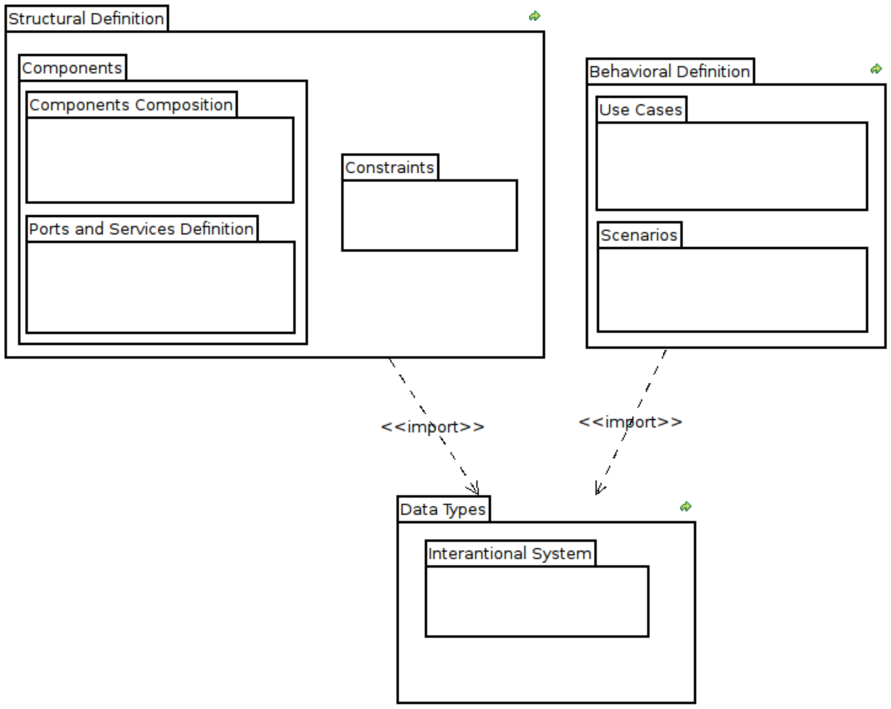


Figure 25: Functional and Architectural Analysis Packages distribution.

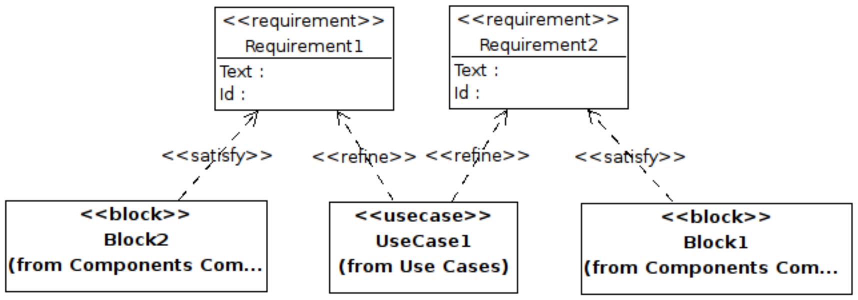


Figure 26: Relationship between requirements, use case and blocks.

and we can see the relationship between the elements gotten in the methodology steps.

4.4.2.3 *Abstraction Levels*

We have explained what the sub-steps are in our methodology, now we show how the evolution in this process is. We saw that the stages in this process are:

1. Define Use cases.
2. Define Scenarios in Sequence Diagrams.
3. Define the relationship between use cases and system technical requirements.
4. Define Structural composition in a BDD based on entities defined in the sequence diagram.
5. Define the ports, interfaces and services in a BDD based on actions defined in the sequence diagram.
6. Define the interaction between subsystems using IBD based on actions defined in the sequence diagrams, the subsystems defined in the BDD (step 3) and the ports, interfaces and services defined in the BDD (step 4).
7. Define the datatypes using a BDD.
8. Define the constraints using a BDD.

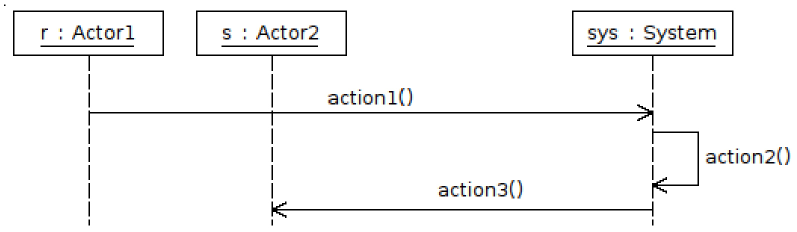


Figure 27: Scenario in level 0.

9. Define the relationship between constraints in order to generate other information to take into account in the design of the system using Parametric Diagrams.

10. Define the relationship between blocks and system technical requirements.

We apply these steps in different levels of abstraction in order to start from a high abstraction level to descend to a more detailed level, it helps to consider first the system as a big system and later we can distribute the work to different components which are the subsystems. We define some abstraction level in order to begin the system design. These levels can be more; we just have to follow the idea of the different levels described here.

- *Level 0:* The first level is the level 0, where is defined the interaction between the actors identified in the use cases and the system. This is like a context diagram, but more complete, because we can see the sequence of the actions between the environment (actors) and the system. In the figure 27, we show how is a scenario in level 0.

  Later, we define the structure representation, the ports, the services and the interfaces. In this description, we build the interaction between the system and its environment and we can see, in a structural level, what services provide and request the system to the environment.

  The constraints are also defined, and all of them are assigned to the block which represents the system. In this level, we can do a first constrain decomposition.

  In the figure 28, we depict the structural definition of the system in this first level. “Port1” implements “action1()” which is in the “interface1”. “Port2”

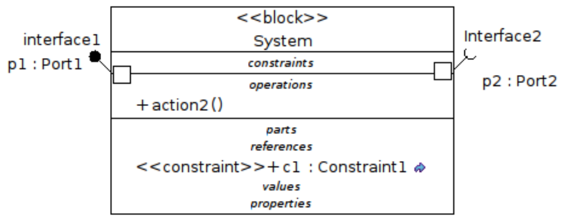


Figure 28: System structure representation in level 0.

requests the "action3()", which is in the "interface2". "action2" is an operation of the system and "c1" is a constraints defined in the constraints packages.

- *Level 1-N:* We make the first partition of entities, in terms of the EIA-632 standard, in subsystems. These partitions are derived from the action deduced in the level 0. We note that some actions can be grouped in an entity and they are tied to the action domain, functionality and others. The partition criterion is given to the designer and he will find the best way to do a logic functional partition. It is important to advice that this first partition has to be the most general possible; this cannot include any details such as sensors, keyboard, etc.

  Once the partition is made, the actions are distributed in the resulting entities on the sequence diagram, and the structural representation is redrawing, as well as ports, interfaces and services. Some constraints probably descends to the subsystems, this shows the constraints specialization that we can create when the system is split in subsystems.

  In the figure 29 and 30, we show a partition example from our level 0 example. In the sequence diagram, we see the system is partitioned in two subsystems, "Subsystem1" and "Subsystem2". We can note the actions have been distributed in the subsystems. "Subsystem1" implements "action1()" and "Subsystem2" calls "action3()". "Action2" is divided in two actions, "action21()" and "action22()". The criterion to do the partition is a designer decision and it is the result of his analysis. The partition consequence is the interaction

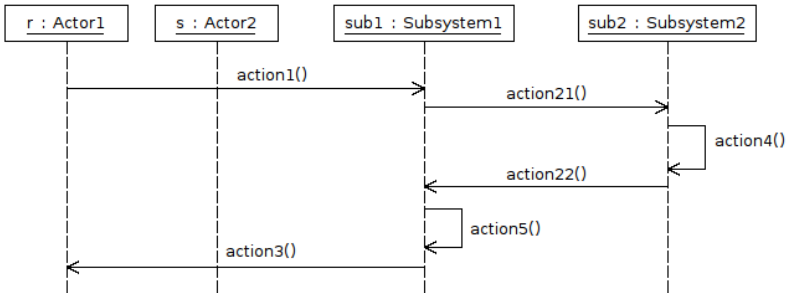
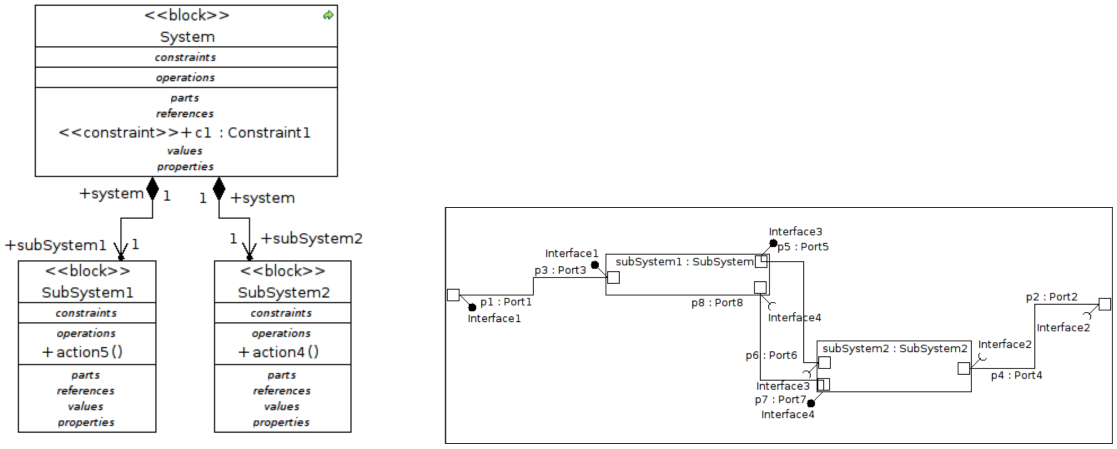


Figure 29: Scenario in the level 1.



(a) (b)

Figure 30: System structure representation in level 1.

between "Subsystem1" and "Subsystem2" to implement completely the functionality defined in "action2()". New actions ("action4" and "action5") can be created in order to complete the functionality of each subsystem.

This scenario is again represented in the structure representation and we see the hierarchy that the system has between the system and the subsystem. We interpreted the BDD is a system which contains the subsystems "Subsystem1" and "Subsystem2". In this representation we can show the constraints, in this case, it continues in the high level of the system, because both subsystems have to fulfill this constraint. When a constraints is only related to a subsystem, it has to descend to this subsystem. The IBD shows the relationship between the subsystems and the services offered and demanded by them. Also, we can see the relationship between the system and their subsystems, when connections are done from the subsystems ports and the systems ports. We can appreciate which subsystem implements actions requested by the environment and which subsystem requests actions to the environment.

At this point, we can verify that our SysML model is according to the stakeholder requirements. We propose to transform this representation to HiLeS formalism, where the sequence diagram is translated to Petri Nets and the structural representation to structural and functional blocks. The logical behavior is completed into HiLeS level and a virtual prototype is generated by HiLeS Designer. We can simulate the behavior of the system and we verify that the behavior of the model follows the requirements. We can also verify the logical behavior formally, analyzing the Petri net using a Petri net Analyzer such as TINA. In order to do this analysis, we have to make a transformation between HiLeS and TINA. This transformation is made by HiLeS Designer and it consists in generating the TINA's Petri net format and executing the Tina's analyzer engine to get the analysis results.

We think the validation stage follows the MDA concept. In the figure 31, we show how the MDA conception is applied. First, we transform our system model described in SysML to a model in HiLeS formalism. We need a transformation engine which converts the SysML concepts to HiLeS concepts. In this case, BDD, IBD and Sequence diagram are the PIM in a MDA conception, and Structural blocks, functional blocks and Petri Nets are the PM. When the mapping between SysML and HiLeS is done, we get the PSM, which is the HiLeS model of our system. From the HiLeS model, we can validate our system in two ways, one way is to transform the HiLeS model (PIM) in a VHDL-AMS model called virtual prototype (PSM) and the other way is to extract the Petri Net from the HiLeS model, to generate the Petri Net description in TINA format. The virtual prototype simulation gives us a behavioral validation of our system design and the Tina's Petri net format can be analyzed by TINA and it gives us possible errors in our logical model which are

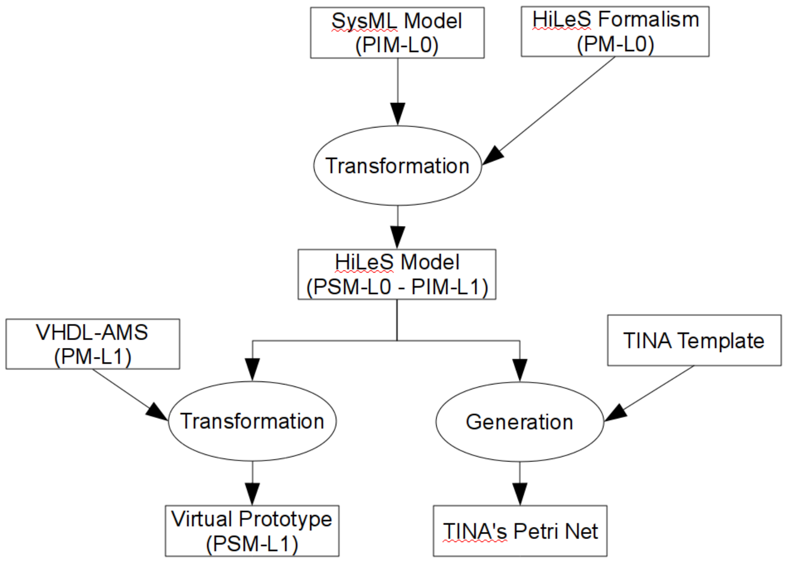


Figure 31: Methodology validation using MDA concepts.

not taken into account in the development of the model.

Once our model is validated, we can search physical components whose behavior is according to the virtual prototype and the services that the subsystems offer. If we find physical components which fulfill the virtual prototype behavior and the services defined in the logical representation solution, we can stop and replace the logical description solution for the virtual prototype which simulates the behavior of the physical component. If we do not find any physical component, we can begin a new level, decomposing the subsystems without solution and we repeat the steps.

The physical component selection is the same approach given in the Platform Based design method, which follows the MDA approach. We have the functionality model (Virtual Prototype) as PIM and the physical component model as PM. We match services, constraints and behavior to choose the best mapping between them, in MDA words, we transform a logical solution model (virtual prototype) to a physical solution model (physical prototype). In the figure 32, we depict this transformation.

We can descend to the levels necessaries until we get a developed subsystem

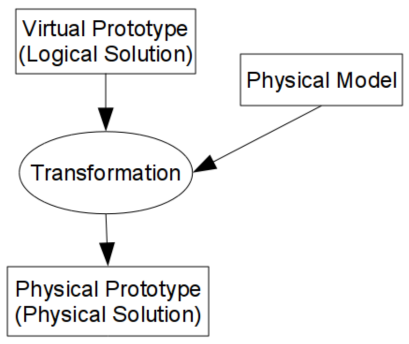


Figure 32: Logical solution to physical solution mapping in the proposed methodology using MDA concepts.

or develop a degree level where we have to build and design everything. This idea comes from the EIA- 632 standard, in its system design process.

In the figure 33, we depict how a solution of the system can be made. We can see Requirements analysis follows the Requeriments 14, 15 and 16 from the EIA-632. This requirements are represented by package and requirements diagrams in SysML. Once we finished the requirement analysis, we pass to the Functional and Architectural Analysis. This step follows the Requirement 17 from the EIA-632.

We represent the system behavior by Use cases diagram and sequence diagram in SysML. We connect the use case to the system technical requirements. We use the information getten in the sequence diagram and we build the hierarchical composition of the system in a BDD and its relationship between the subsystems, defining the ports, services and interfaces. We transform the constraints from the system technical requirements to constraints blocks in a BDD and we build the relationship between constraints block using the parameter diagram. Finally, we connect the blocks to the system technical requirements.

When the logical solution representation in SysML is finished, we see if it is possible to build the virtual prototype in HiLeS in order to verify the behavior of the system and search a physical system which fulfills the defined model in SysML and the virtual prototype. If the development of the virtual prototype is difficult, we perform other iteration in the functional and architectural analysis dividing the

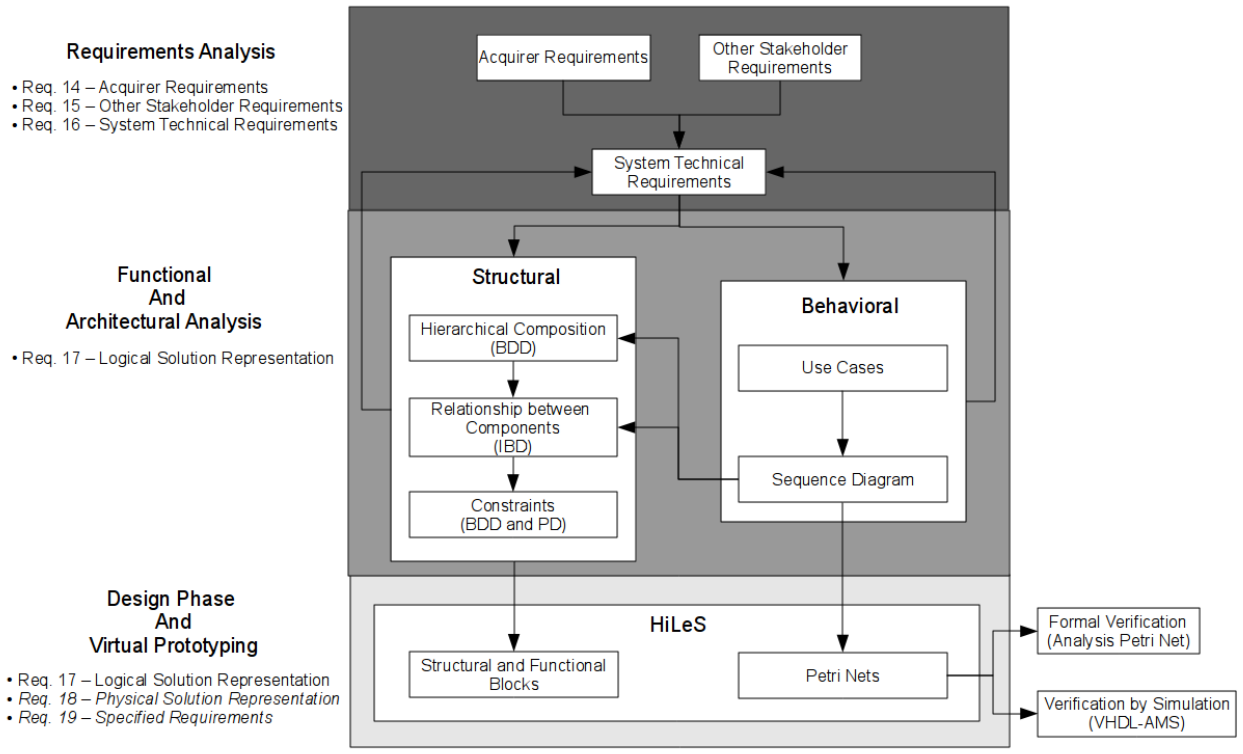

Requirements Analysis
• Req. 14 – Acquirer Requirements
• Req. 15 – Other Stakeholder Requirements
• Req. 16 – System Technical Requirements
Acquirer Requirements
Other Stakeholder Requirements
System Technical Requirements
Functional And Architectural Analysis
• Req. 17 – Logical Solution Representation
Structural
Hierarchical Composition (BDD)
Relationship between Components (IBD)
Constraints (BDD and PD)
Behavioral
Use Cases
Sequence Diagram
Design Phase And Virtual Prototyping
• Req. 17 – Logical Solution Representation
• Req. 18 – Physical Solution Representation
• Req. 19 – Specified Requirements
HiLeS
Structural and Functional Blocks
Petri Nets
Formal Verification (Analysis Petri Net)
Verification by Simulation (VHDL-AMS)

Figure 33: Methodology summary.

system in subsystems. We observe if it is possible to develop a virtual prototype of our subsystems, if it is not possible, we continue with other iteration until we have the posibility to create a virtual prototype of our subsystem.

When we have a virtual prototype, we search physical subsystem whose features are similar to our SysML HiLeS model. If we do not fine one, we split our subsystem and repeat the process again until we find the physical component which solves the functionality and the constraints described in the logical solution representation (SysML model and HiLeS model).

4.5 Conclusion

In this chapter, we define the concept methodology, we describe its elements and its relationship. We summarize the EIA-632 standard, we focus on the System Design subclause and the definition of building block which are used in our methodology. We explain what is SysML and its diagrams. We define our methodology and its relationship with EIA-632 and SysML. We also define two MDA transformations: One is the SysML to HiLeS transformation in order to validate the logical solution representation and the other is HiLeS model to Physical component model in order to get a physical solution representation from the logical solution.

In the next chapter, we will give examples applying our methodology. The first example is a Skating Manager system which was used to develop our methodology. The second is an example is of our methodology in an example system given by Airbus.

CHAPTER V

Examples

A methodology is not completely clear without examples. In this chapter, we present examples where the methodology is applied. We use Topcased tool to develop the logical solution representation in SysML. The first example was used to elaborate the methodology that we propose in this work, it is the design of a system to manage a skate race. The second example is an exercise made from real requirements of the industry, in this case, the requirements to build a logical solution of the System Display Selector of the airplane A380 produced by Airbus.

5.1 Skating Manager System

In this section, we develop an example to design a system which manages and takes the skaters' time in a race. We use it to apply our methodology, to illustrate how it works in its different steps and to detail how the SysML diagrams are used. The example begins in the stakeholder requirements, which is a text written by us simulating the race organizers. Afterwards, we explain the methodology applied to the example, finalizing in a logical solution representation in SysML.

5.1.1 Stakeholder Requirements

- *Skating Race Description:* The skating race is organized in a closed-circuit. The start line is the same finish line, and during the race, it is called lap-line. The objective of the competence is to skate the longest distance in a defined period of time. The figure 34 depicts the track. At the beginning of the race, the competitors are located at the start line. The race starts when the pistol

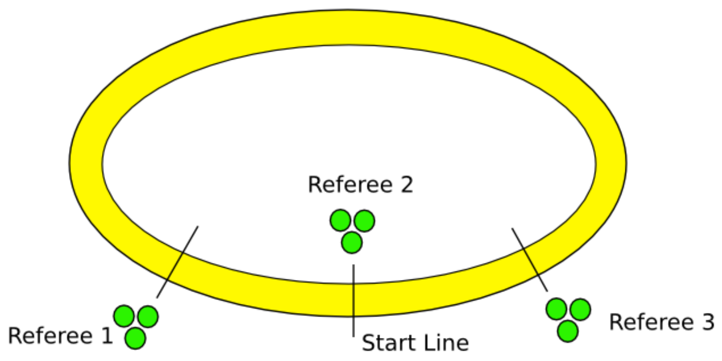


Figure 34: The skating course.

is fired by the referee and at the same time the referee starts the chronometer which manages the race time. The competitors leaves the start line when they hear the shot and they skate following the track. Each time that a skater passes the lap-line, the referee takes its number, time, and lap. The referee always checks the chronometer to control the race time. When the referee sees that the chronometer achieves the race time, he rings the bell, and the skaters skate to the finish line and then they stop. The skater, who has crossed first the finish line and has made the highest number of laps, is the winner. The winner is announced by the referee.

- *Present Timekeeping System:* In order to reduce errors taking skater numbers and time, there are three referee-teams, after, over and before the lap-line. Each referee-team is located near a line (referee team-line) in order to take the time and the number of each skater who passes for this line. The race time starts when the pistol is fired. Each referee-team, at the same time starts its chronometers manually when they hear the starting shot. Each time that a skater passes over the referee team-line, its time and its number is taken. When the race time is over, every skater has to pass the finish line, and it is the last lap and time registered of each competitor.

- *Problem:* The main problem with the present timekeeping system is the time accurateness and to take skater's number. Each time this race is carried out, there are always problems, as it is shown in the figure 35. This figure represents

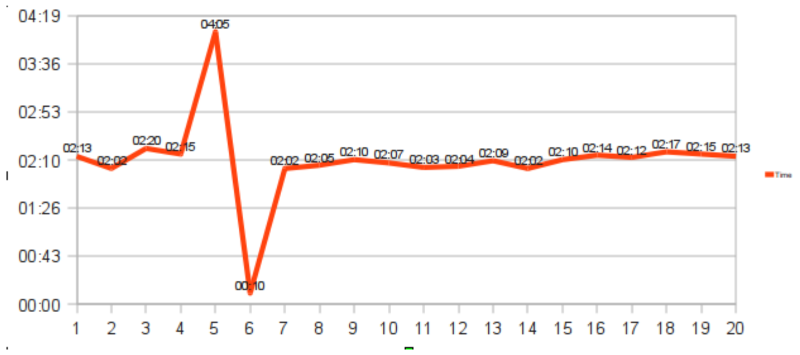


Figure 35: A skater times.

the time of one skater taken by a judges-team. It is possible to see that in the lap 5, the skater spends 4.05 minutes, a lot of time if it is compared to its regular time that is between 2:20 and 2.02 minutes. Also, in the lap 6, the time is shorter compared to the other times. A possible reason is the referees did not see this skater pass in the lap 5 and they realized it in the lap 6, then they tried to correct the mistake, passing him twice in a short period of time.

Other problem is when two skaters are in the first positions and they pass the finish line at the same time with a very little difference between each other. However, with the present system, it is difficult to know who passes first.

After the race is finished, the judges deliberate to decide who the winner is. They take some time in order to compare results and to take a final decision.

- *Wished System:* The race's organizers want to reduce the inaccuracy taking the time and the skater number. Also, they want to take both of them automatically. Additionally, they want to have a single point to take the time and the skater number which will be located in the start line. The system has to work in any environment conditions.

- *Some Stakeholders Point of View:*

  - *Referee point of view:*

    * Referee wants to have the skater's list (name and number).

    - ∗ Referee wants to start the race firing the pistol. At the same time, Referee actives the chronometer.
    - ∗ Referee wants to take the time and number of the skater when he/she crosses the lap-line.
    - ∗ Referee wants to register the time and number of the skater each time he/she crosses the lap-line.
    - ∗ Referee wants to calculate the skater's time by lap subtracting the last lap time to the present time lap.
    - ∗ Referee wants to count the laps of each skater when she/he crosses the lap-line.
    - ∗ When the race time is over, the referee wants to alert the skater activating an alarm.
    - ∗ Referee wants to announce the winner based on the number of laps done by the skaters. The skater, who does the greatest number of laps, is the winner.
    - ∗ Referee wants to improve the accuracy when the skater's time is taken. Also, to improve the taken skater's number error.
    - ∗ Referee wants to register the race info in a similar way to make the skater's list.
  - – *Skater point of view:*
    - ∗ Skater starts the race when he/she hears the pistol is firing.
    - ∗ Skater follows the track and crosses the lap-line.
    - ∗ Skater accelerates when he/she hears the alarm which indicates that the time race is over.
    - ∗ Skater stops when he/she crosses the finish-line.

5.1.2 Applying the proposed methodology

First of all, we create a project SysML in Topcased where the main diagram is a Block Definition Diagram (BDD). In this diagram is drawn the four packages

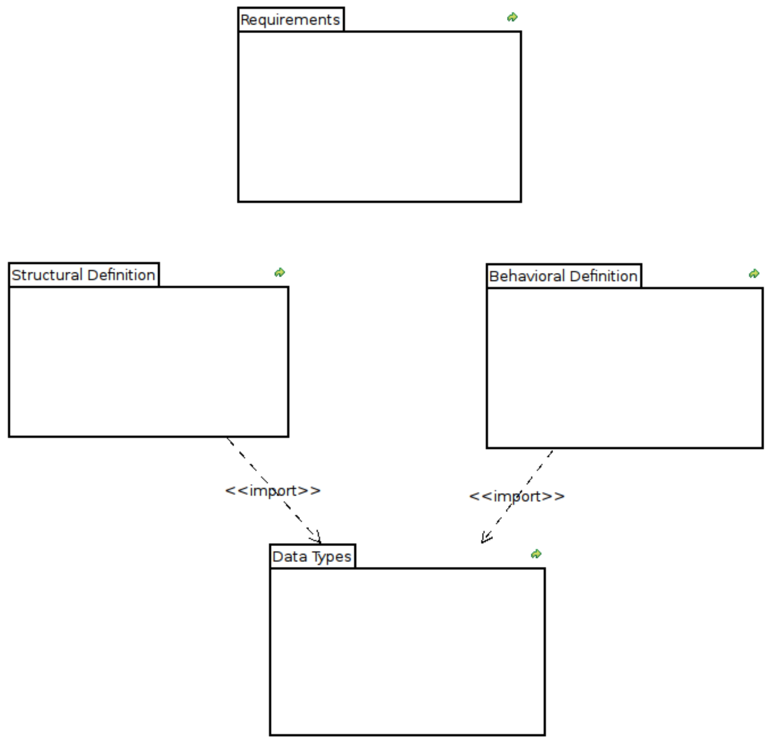


Figure 36: Package distribution in the SysML project.

describe in the chapter 4: Requirements, Structural definition, Behavioral Definition and Data Types. In the figure 36, we show the BDD.

The BDD we call it "SkaterDetectorL0", because we are developing the level 0 of the example.

5.1.2.1 ***Requirement Analysis***

Inside of the Requirement Package, we create the same distribution described in chapter 4. Inside Acquirer Requirement package, we create two packages: Referee Requirements and Skater Requirements. These packages correspond to two of the stakeholders involved in the developing of the system. In the Referee Requirements, we identify four classification packages: Functional, Performance, Interface and Environment package.

Inside of the Functional Requirement Package, we draw requirement blocks for

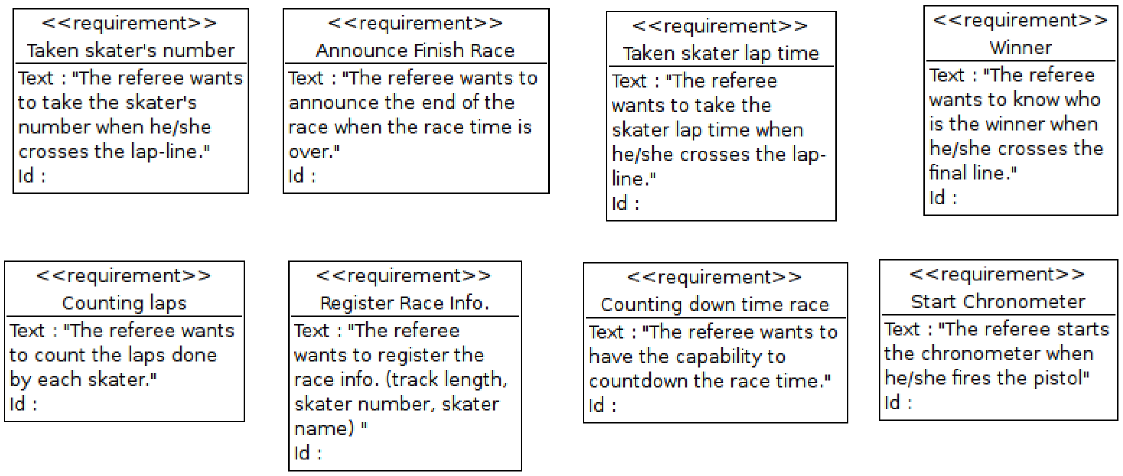

<<requirement>>
Taken skater's number
Text : "The referee wants to take the skater's number when he/she crosses the lap-line."
Id :
<<requirement>>
Announce Finish Race
Text : "The referee wants to announce the end of the race when the race time is over."
Id :
<<requirement>>
Taken skater lap time
Text : "The referee wants to take the skater lap time when he/she crosses the lap-line."
Id :
<<requirement>>
Winner
Text : "The referee wants to know who is the winner when he/she crosses the final line."
Id :
<<requirement>>
Counting laps
Text : "The referee wants to count the laps done by each skater."
Id :
<<requirement>>
Register Race Info.
Text : "The referee wants to register the race info. (track length, skater number, skater name) "
Id :
<<requirement>>
Counting down time race
Text : "The referee wants to have the capability to countdown the race time."
Id :
<<requirement>>
Start Chronometer
Text : "The referee starts the chronometer when he/she fires the pistol"
Id :

Figure 37: Referee Functional Requirements.

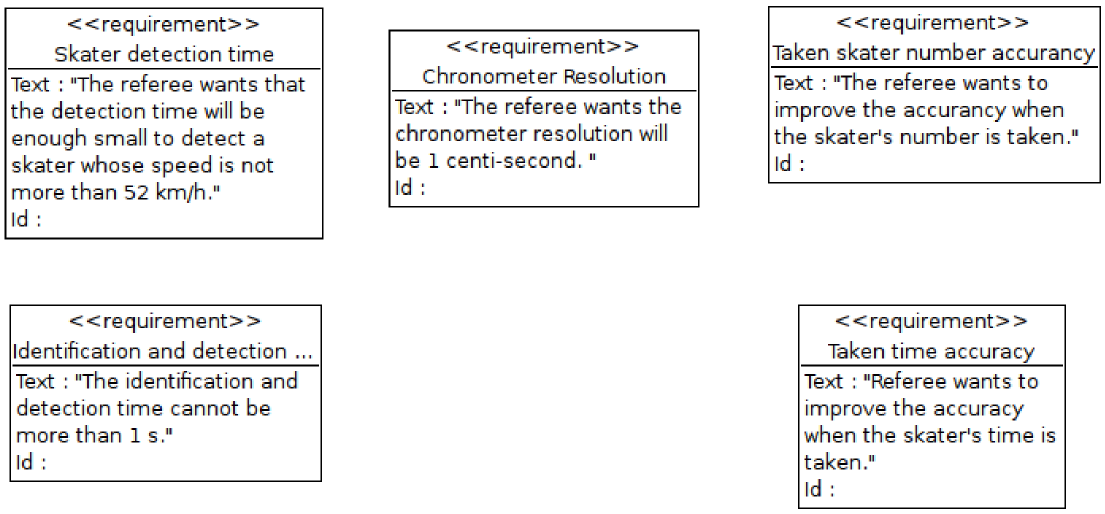
each referee requirement. In the figure 37, we show them. We can see the requirements are atomics, it means that there is a need or wish for each requirement. This does not say we cannot create composed requirements. When the complexity of the requirements demands progressive decomposition of the stakeholder requirements, we can create a block requirement with a composed requirement and later to make the partition of this block in atomic requirements (See [FMS08] and [dSV07]).

It is important to note the requirements are expressed as needs or wishes of the referee and the word “system” is not involved in the description of the requirement. This way to express the requirement is important in order not to limit the solution.

In the Referee Performance package (figure 38), we create the requirement blocks according to the classification. Speed, accuracy and time are some examples of features which belong of this group.

Inside the Referee Interface Package, we find three requirements, one related to cross the line, other to use a keyboard to insert the information and finally which element is used to start the race. All of them have different features, one is the interface skater-system when he/she passes the line from the referee point of view, the others are the interfaces between the referee and the system in order to add information or to start the race. In the figure 39, we show these requirements.

Finally, in the Referee Environment Package, we add two block requirements:



<<requirement>>
Skater detection time
Text : "The referee wants that the detection time will be enough small to detect a skater whose speed is not more than 52 km/h."
Id :

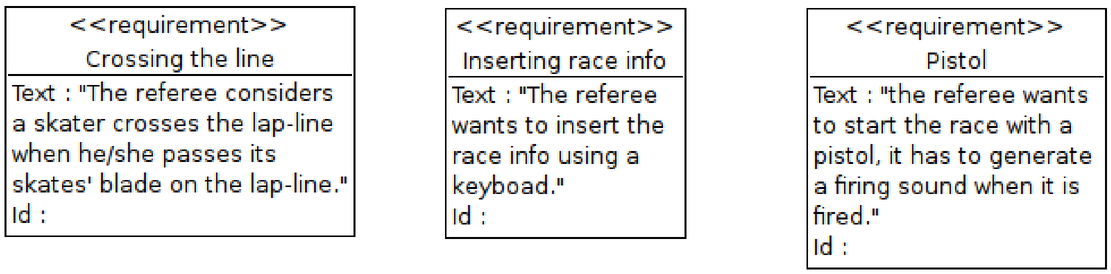
<<requirement>>
Chronometer Resolution
Text : "The referee wants the chronometer resolution will be 1 centi-second. "
Id :

<<requirement>>
Taken skater number accurancy
Text : "The referee wants to improve the accurancy when the skater's number is taken."
Id :

<<requirement>>
Identification and detection ...
Text : "The identification and detection time cannot be more than 1 s."
Id :

<<requirement>>
Taken time accuracy
Text : "Referee wants to improve the accuracy when the skater's time is taken."
Id :

Figure 38: Referee Performance Requirements.



<<requirement>>
Crossing the line
Text : "The referee considers a skater crosses the lap-line when he/she passes its skates' blade on the lap-line."
Id :

<<requirement>>
Inserting race info
Text : "The referee wants to insert the race info using a keyboad."
Id :

<<requirement>>
Pistol
Text : "the referee wants to start the race with a pistol, it has to generate a firing sound when it is fired."
Id :

Figure 39: Referee Interface Requirements.

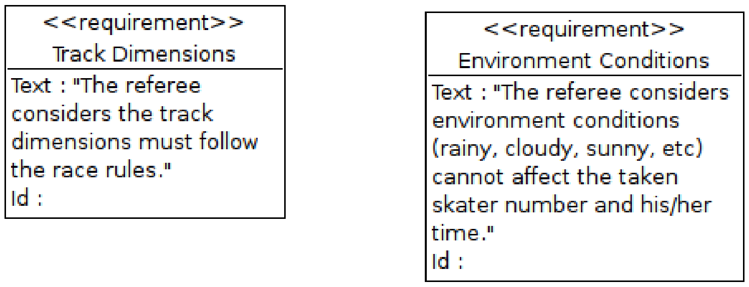

<<requirement>>
Track Dimensions
Text : "The referee considers the track dimensions must follow the race rules."
Id :
<<requirement>>
Environment Conditions
Text : "The referee considers environment conditions (rainy, cloudy, sunny, etc) cannot affect the taken skater number and his/her time."
Id :

Figure 40: Referee Environment Requirements.

Track dimensions and Environment conditions. The first requirement is valuable information to do the dimension of the system. This will probably be a constraint in the system logical solution. The second requirement is a typical constraint in the developing of a system; it can be more specific according to the system. In the figure 40, we show these requirements.

Once the Referee Requirements are finished, we develop the Skater requirements. In the skater requirement package, we only identify functional requirements and they are depicted in the figure 41.

When the Acquirer Requirements are finished, we begin to find the constraints of these requirements or requirements which enrich their description. They are the Other Stakeholder Requirements. For our example, we only find one and it is the race rules. This is information that it is not given by the referee or skater, even more it constrains or complete some requirements defined before. In the figure 42, we show the race rules defined in the example.

Once the stakeholders are finished, we start the analysis process of the requirements in order to get the system technical requirements. We take one requirement by one and we derivate system requirements which are more accurate and more clear than the stakeholder requirements. In our example, most of them have accuracy, so few requirements were modified. The subject in the requirement description changes because now we are creating the system requirements, therefore the system is the subject of these requirements. In the figure 43, we can see the connection between the stakeholder requirements and the derived requirement. We also appreciate the change of subject in the description of the requirement.

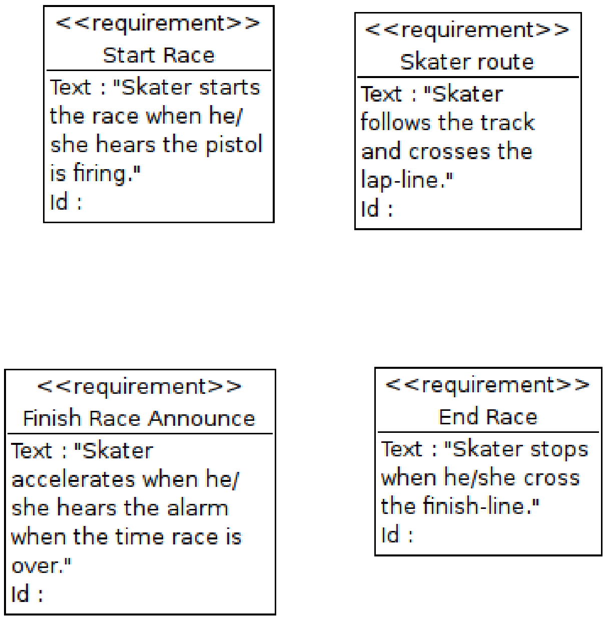

<<requirement>>
Start Race
Text : "Skater starts the race when he/ she hears the pistol is firing."
Id :
<<requirement>>
Skater route
Text : "Skater follows the track and crosses the lap-line."
Id :
<<requirement>>
Finish Race Announce
Text : "Skater accelerates when he/ she hears the alarm when the time race is over."
Id :
<<requirement>>
End Race
Text : "Skater stops when he/she cross the finish-line."
Id :

Figure 41: Skater Functional Requirement.

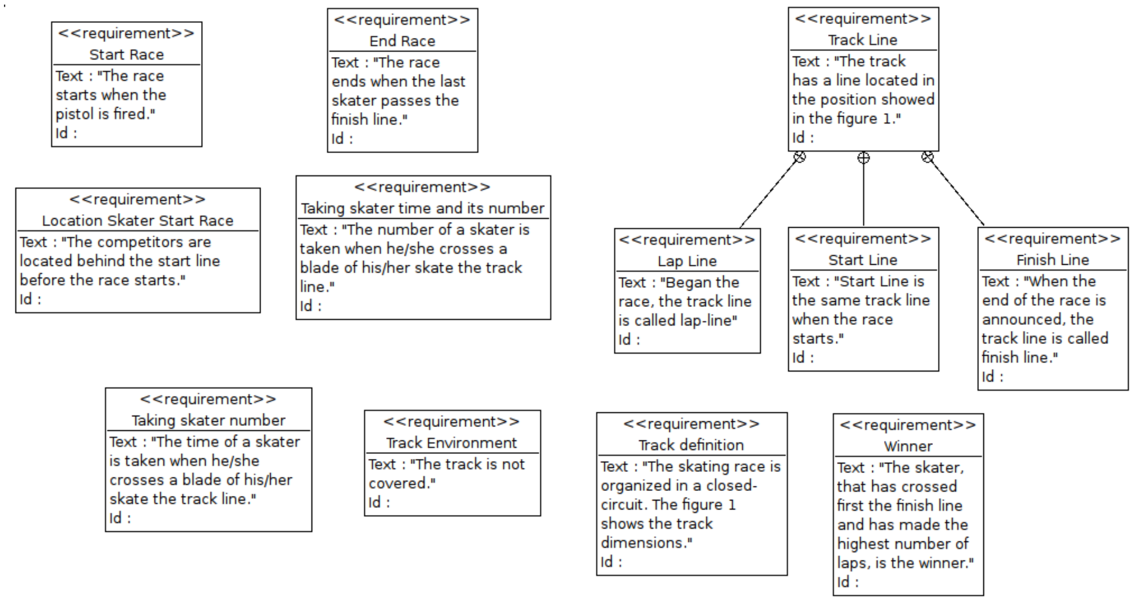

<<requirement>>
Start Race
Text : "The race starts when the pistol is fired."
Id :
<<requirement>>
End Race
Text : "The race ends when the last skater passes the finish line."
Id :
<<requirement>>
Track Line
Text : "The track has a line located in the position showed in the figure 1."
Id :
<<requirement>>
Location Skater Start Race
Text : "The competitors are located behind the start line before the race starts."
Id :
<<requirement>>
Taking skater time and its number
Text : "The number of a skater is taken when he/she crosses a blade of his/her skate the track line."
Id :
<<requirement>>
Lap Line
Text : "Began the race, the track line is called lap-line"
Id :
<<requirement>>
Start Line
Text : "Start Line is the same track line when the race starts."
Id :
<<requirement>>
Finish Line
Text : "When the end of the race is announced, the track line is called finish line."
Id :
<<requirement>>
Taking skater number
Text : "The time of a skater is taken when he/she crosses a blade of his/her skate the track line."
Id :
<<requirement>>
Track Environment
Text : "The track is not covered."
Id :
<<requirement>>
Track definition
Text : "The skating race is organized in a closed-circuit. The figure 1 shows the track dimensions."
Id :
<<requirement>>
Winner
Text : "The skater, that has crossed first the finish line and has made the highest number of laps, is the winner."
Id :

Figure 42: Race Rules requirements.

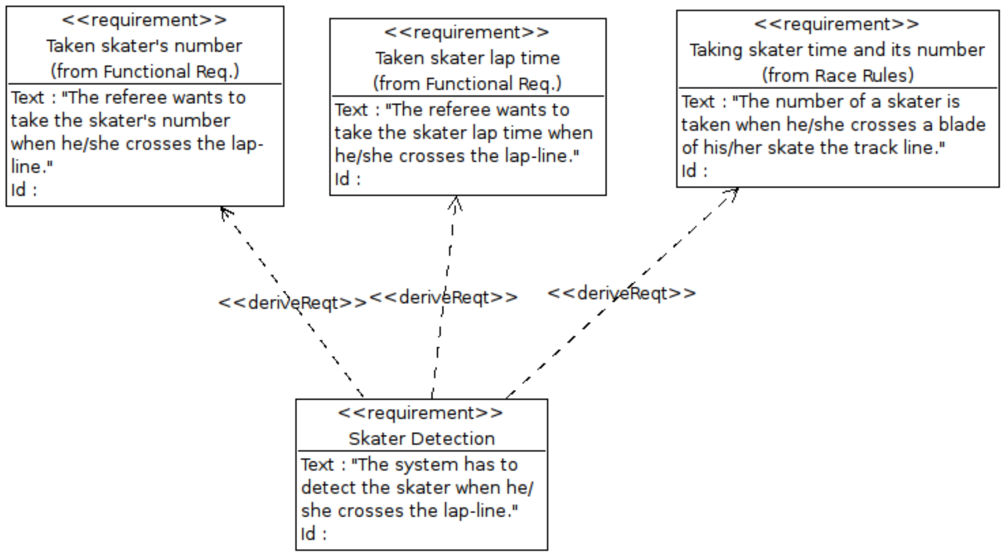

<<requirement>>
Taken skater's number
(from Functional Req.)
Text : "The referee wants to take the skater's number when he/she crosses the lap-line."
Id :
<<requirement>>
Taken skater lap time
(from Functional Req.)
Text : "The referee wants to take the skater lap time when he/she crosses the lap-line."
Id :
<<requirement>>
Taking skater time and its number
(from Race Rules)
Text : "The number of a skater is taken when he/she crosses a blade of his/her skate the track line."
Id :
<<deriveReqt>>
<<deriveReqt>>
<<deriveReqt>>
<<requirement>>
Skater Detection
Text : "The system has to detect the skater when he/she crosses the lap-line."
Id :

Figure 43: Derived System Technical Requirement from Stakeholders requirements.

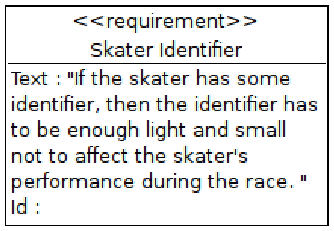

<<requirement>>
Skater Identifier
Text : "If the skater has some identifier, then the identifier has to be enough light and small not to affect the skater's performance during the race. "
Id :

Figure 44: Technology requirement.

In the System Technical package, there are also classification packages where we unify the stakeholders requirements and we relate them with the other stakeholder requirements in order to get unique, atomic and classified requirements. In this package, we cannot find any composed requirement and any measure without quantity; it has to be very exact. The classification package is the same package described in Acquire package, we only add a new package called Technology Requirements and it defines the technology constrains related to the development of the system. In the figure 44, we show an example of a technology requirement.

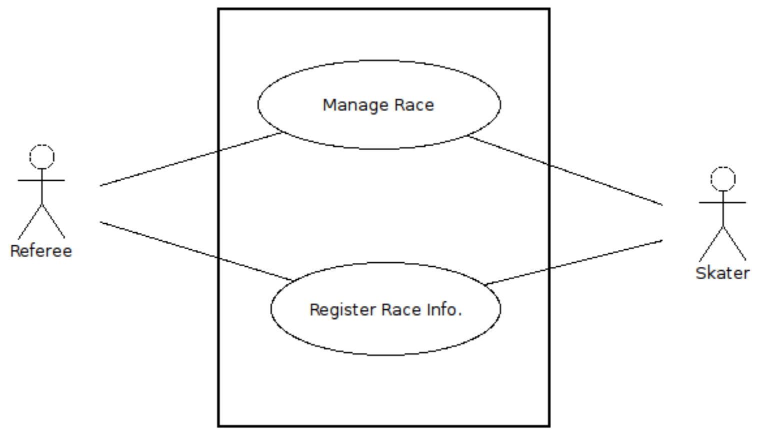


Figure 45: Skate Manager System use case diagram.

5.1.2.2 ***Functional and Architectural analysis***

Finished the requirement analysis, we pass to the functional and architectural analysis where our first task is to develop the use cases of our system. In the figure 45, we depict the use case diagram that we obtained.

We identify two main functionalities or use cases of our system, "manage race" and "register race info.". In "manage race", we include: take the time and the skater, count the laps given by each skater, start and finish the race, announce the winner and give the race results. We see these functionalities are different, although they are connected to the same scenario which is the management of the race. On the other hand, Register Race Info is a functionality which represents other scenario different to the management of the race. In this use case, we register the information of the skater, the course, the referee and so on.

When the use cases are identified and defined, we create the actors which interact with the system. They have been identified in the requirements analysis in the creation of the stakeholder requirement packages. Once the actors are drawn, we proceed to link the actors and the use cases. For example, we see in the figure that the actor "Referee" is related to two use cases defined before, that means "Referee" participate in the development of the functionality of both use cases.

Once defined the use cases, we continue with the scenarios definition using the

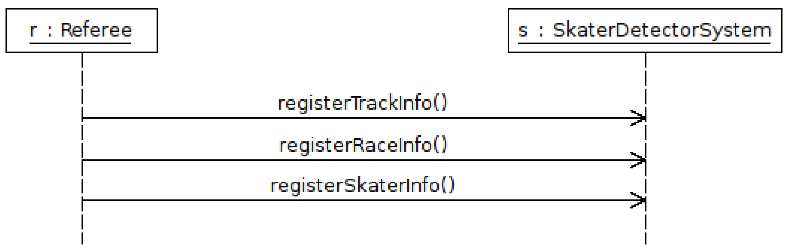


Figure 46: Register Race Info scenario in the Level 0.

sequence diagram. In the figure 46, we show the scenario in Level 0 of "Register Race Info.". In this scenario we identify three actions which are: registerTrackInfo, registerRaceInfo and registerSkaterInfo. These actions have a sequence that the system has to obey.

In the figure 47, we show the scenario identified as "manage race" in the use cases. The scenario describes the race since it starts until it finishes. We can see in more detail the interaction between the actors and the system. Also, we can see the sequence of the actions created by the designer based on the requirements. These actions are defined in high level, when we go down in the abstraction level, the actions can change.

When we have the use cases and their scenarios described in sequence diagram, we make the connection between the use cases and the requirements. In the figure 48, we show how the use cases are connected to the system technical requirements.

Once the connections are made, we pass to the structural representation. We define first a system as a block in a BDD. Later, we define the ports, interfaces and services deduced in the sequence diagram which are showed in the figure 49. We can see the actions defined in the sequence diagram and how they are distributed in the interfaces. Each interface has a relationship with ports. This actions partition gives some idea of how the first partition entities in the sequence diagram will be. We also define two flow specification, these flows represent exits of the system which gives information to the environment and it decides if it does or it does not take into account this information.

Next step is to define the constraints into the Constraints package. First, we define the structure constraints derived from the system technical requirements and

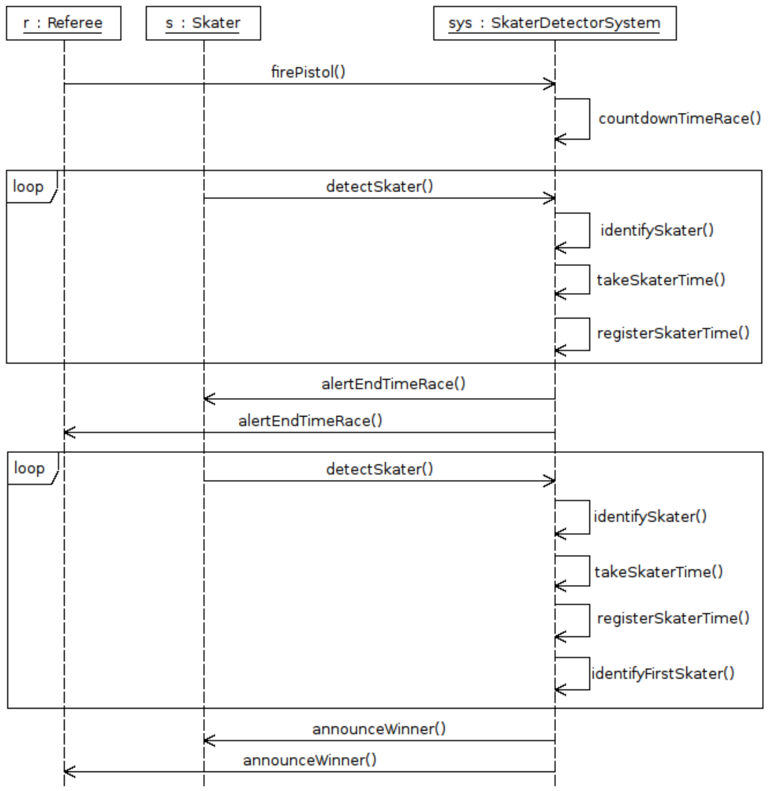


Figure 47: Manage Race scenario in the level 0.

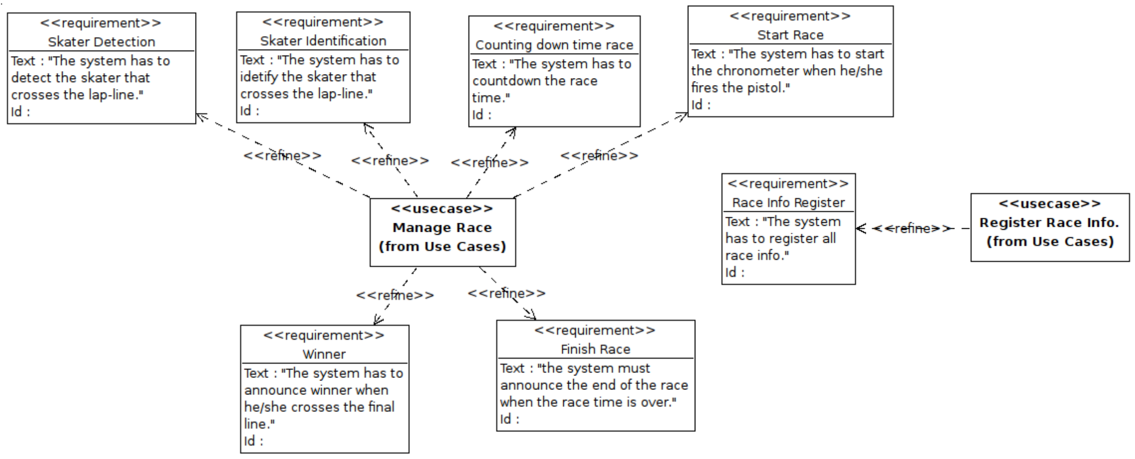

<<requirement>>
Skater Detection
Text : "The system has to detect the skater that crosses the lap-line."
Id :
<<requirement>>
Skater Identification
Text : "The system has to idetify the skater that crosses the lap-line."
Id :
<<requirement>>
Counting down time race
Text : "The system has to countdown the race time."
Id :
<<requirement>>
Start Race
Text : "The system has to start the chronometer when he/she fires the pistol."
Id :
<<refine>>
<<refine>>
<<refine>>
<<refine>>
<<usecase>>
Manage Race
(from Use Cases)
<<requirement>>
Race Info Register
Text : "The system has to register all race info."
Id :
<<refine>>
<<usecase>>
Register Race Info.
(from Use Cases)
<<refine>>
<<refine>>
<<requirement>>
Winner
Text : "The system has to announce winner when he/she crosses the final line."
Id :
<<requirement>>
Finish Race
Text : "the system must announce the end of the race when the race time is over."
Id :

Figure 48: Connection between requirements and use cases.

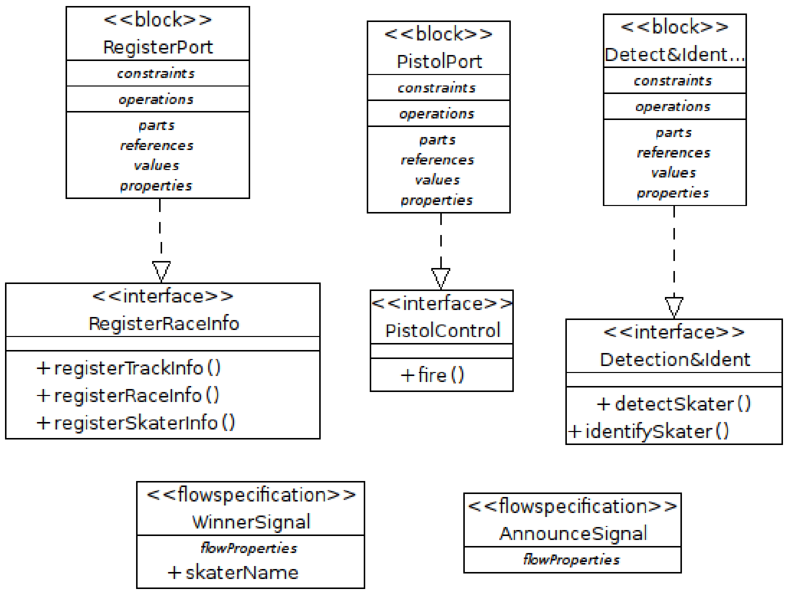


Figure 49: Ports, interfaces and services definition for Skater Manager System.

we draw the path to calculate the other constraints which can be deduced from the others. In this step, we use datatypes to define time, speed, length quantities in the constraints, so we have to define the Datatype Package first (Figure 50). In the figure 51, we illustrate in white blocks, the original constraints traduced from the system technical requirements and in gray blocks, the constraints deduced from the white blocks and the addition of more block in order to calculate the wished information. In our example, we want to know how to make the partition between the identification and detection time. The minimum detection time is given by the use of the skater speed. In order to get the time, we need to convert units from km/h to mm/cs. Later, we need to use the simple speed equation to get the time.

To calculate the minimum identification time, we use the constraint "Ctotal-Time" which defines the minimum time that the system has to respond between the detection time and identification time, constraint derived from the system technical requirements. When we have previous calculus of detection time, we can use it to calculate the identification time. In the figure 52, we show the relationship between the constraints since the Parameters diagram point of view.

When the ports services and constraints are defined, we add this information

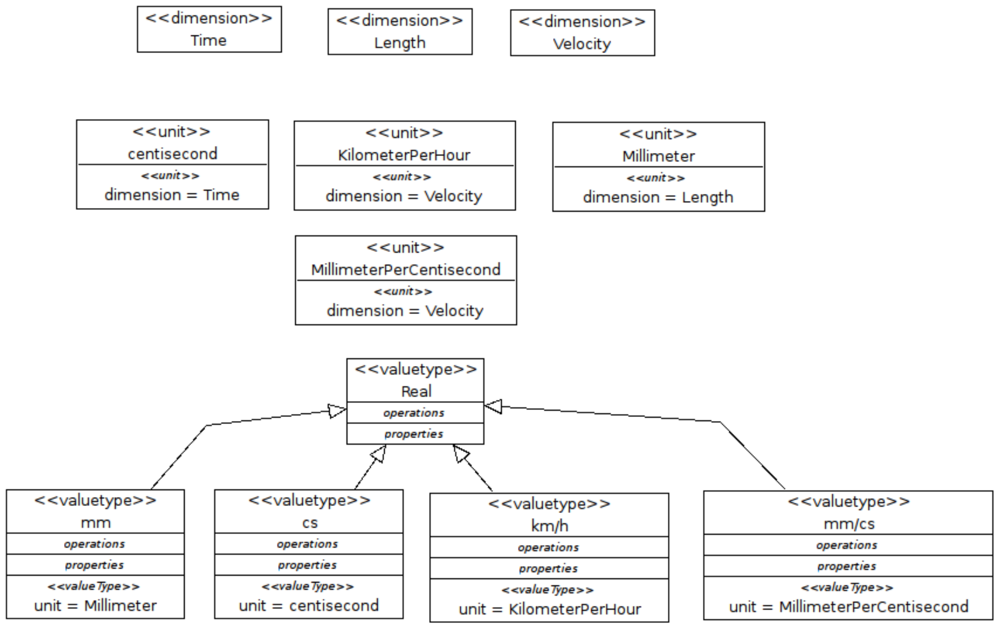

<<dimension>> Time
<<dimension>> Length
<<dimension>> Velocity
<<unit>> centisecond
<<unit>> dimension = Time
<<unit>> KilometerPerHour
<<unit>> dimension = Velocity
<<unit>> Millimeter
<<unit>> dimension = Length
<<unit>> MillimeterPerCentisecond
<<unit>> dimension = Velocity
<<valuetype>> Real
operations
properties
<<valuetype>> mm
<<valueType>> unit = Millimeter
<<valuetype>> cs
<<valueType>> unit = centisecond
<<valuetype>> km/h
<<valueType>> unit = KilometerPerHour
<<valuetype>> mm/cs
<<valueType>> unit = MillimeterPerCentisecond

Figure 50: Data type definition used in the constraints definition.

to the block diagram defined in the beginning of the analysis. The final structural representation is showed in the figure 53. In this diagram we can see the point of interaction with the environment (ports), the services which offers and demands and their constraints.

The final step is to connect the block with the requirements, but in this level the structural description is only one block, thus it is not interesting to relate one block to every system technical requirements. We make this connection in the next levels.

5.1.2.3 *First Iteration*

Once the first level is defined, we proceed to split the system in subsystems as the EIA-632 recommends. We make the partition in the sequence diagram using the previous orientation derived from the structural definition. We note we have one service to register the race info, so we must have an entity in charge of this service. We also has to register other kind of information such as race time, skater time and skater number, thus we can use the same entity to perform these tasks. Detection and identification are considered in one entity, following the structural definition

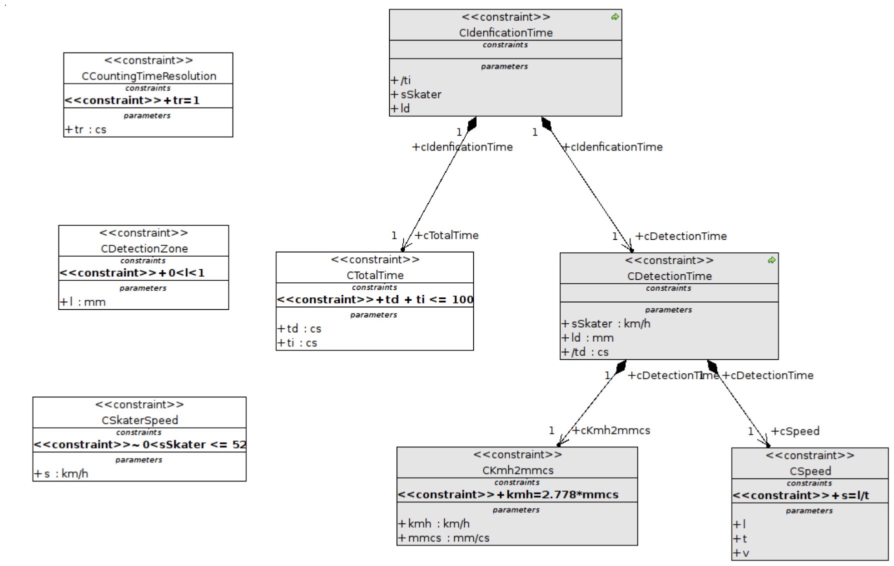


Figure 51: Skater Manager System Constraints defined in a BDD.

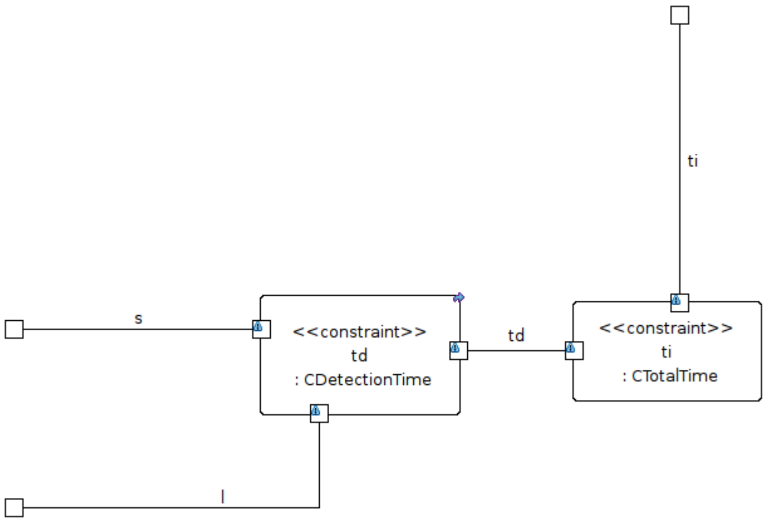


Figure 52: Parametric diagram for the Identification Time Constraint.

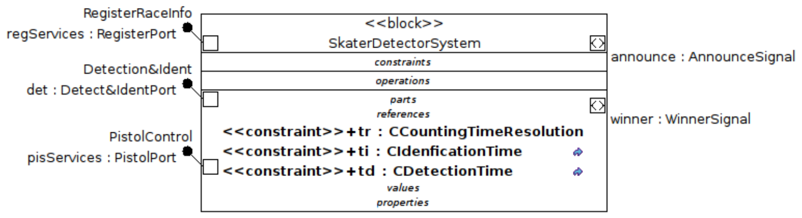


Figure 53: Block Definition Diagram in Level 0.

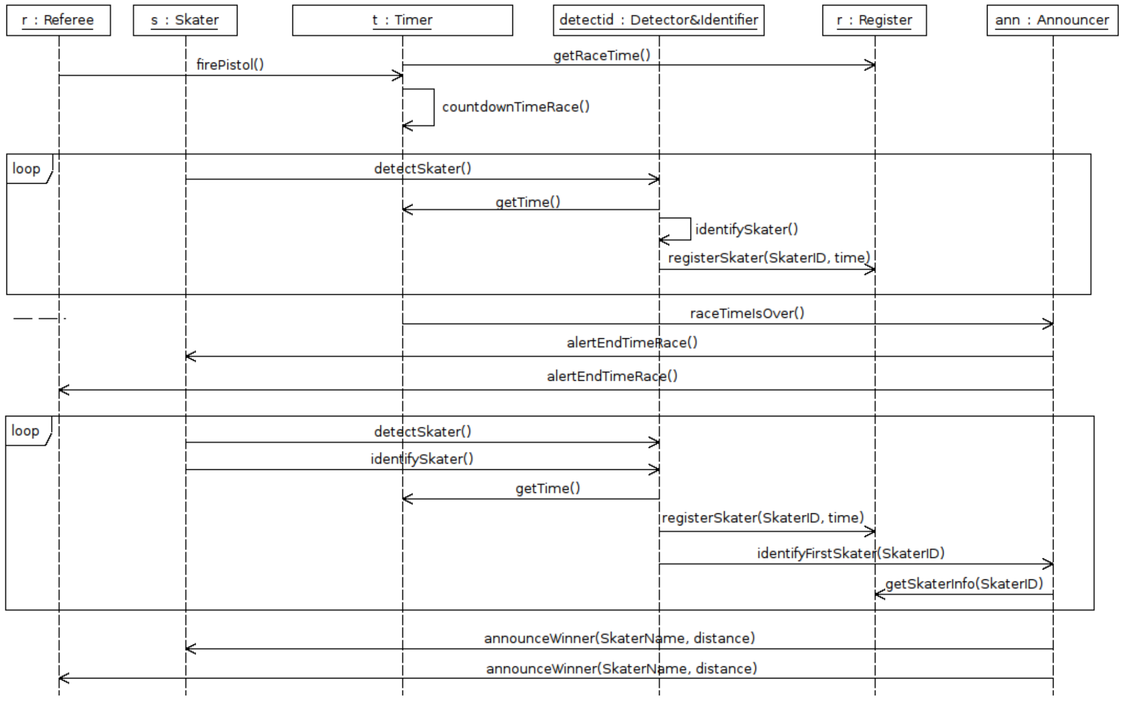


Figure 54: Manage Race scenario in the level 1.

guide where there is only one port to do both actions. In other level, they will be split. To give the finish race announce and to show the winner, we create the entity "announcer". Finally, we need an entity which manages the notion of time, so we create an entity called "timer" which takes the skater time and the race time. In the figure 54, we depict the sequence diagram in the level 1.

In the new scenario representation, we can see the new entities and the distribution of the system actions. We also see that "DetectorIdentifier" entity does the control to take the skater time, register it and start the race.

We repeat the structural definition based on the sequence diagram obtained and we get the description showed in the figure 56. We see the system composition and its constraints derived to its subsystems. For example, the "Detector&Identifier" block receives the constraints: "identification time" and "detection time".

How we have a sublevel in our structural definition, we can define the Internal representation of the system in the level 1. This representation is showed in the figure 57. In this diagram, we can see the subsystems, its services and which entities

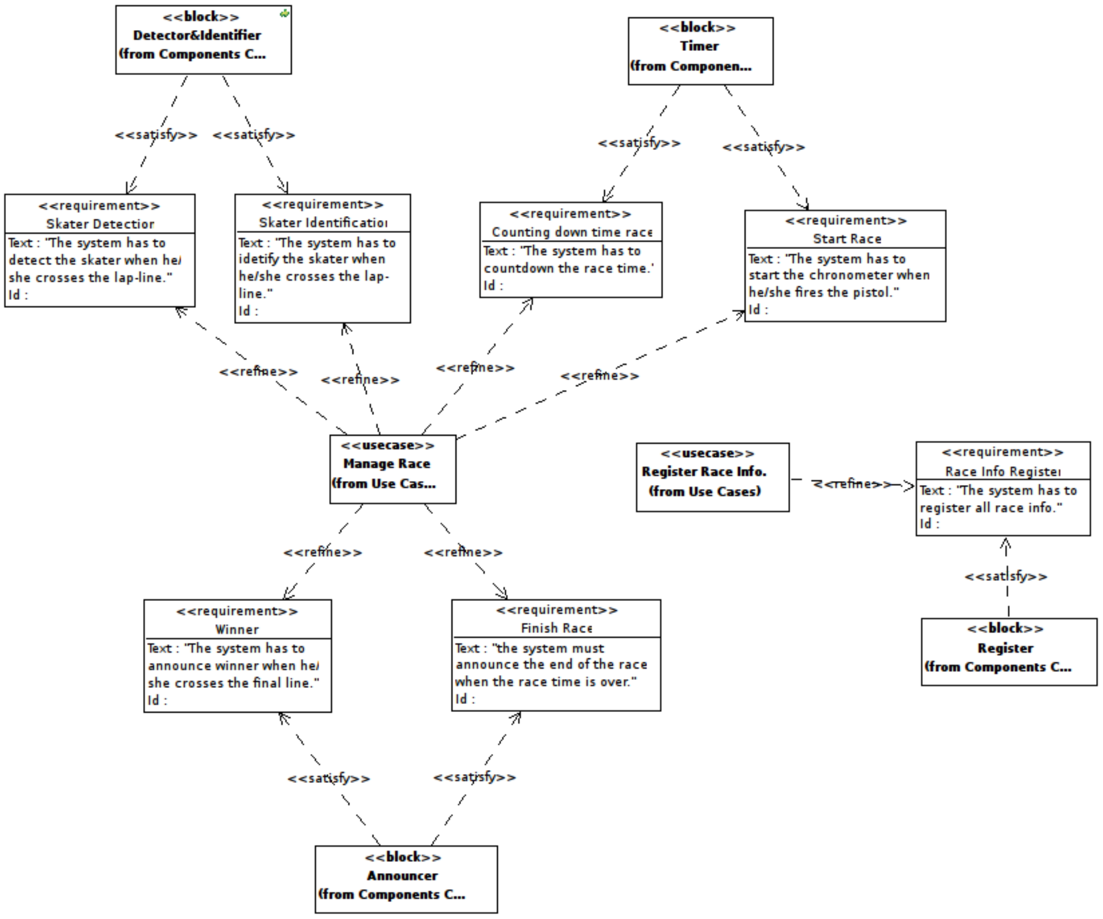


Figure 55: Relationship between Use Cases, Blocks and Requirements.

implement the services which interact with the environment. Obviously, we need to define the ports, interfaces and actions in the corresponding package following the scenario diagram.

Finally, we make the connections between the blocks and the requirements. We can see, in the figure 55, the same relationship between use cases and functional requirements, such as the figure 48, but in this case, the blocks which represent subsystems satisfy specific functional requirements. In our example, each requirement is connected at least to a block, functional or non-functional (constraint blocks), therefore our SysML model is according with the system technical requirements.

At this level, we can transform our SysML model to HiLeS in order to create an

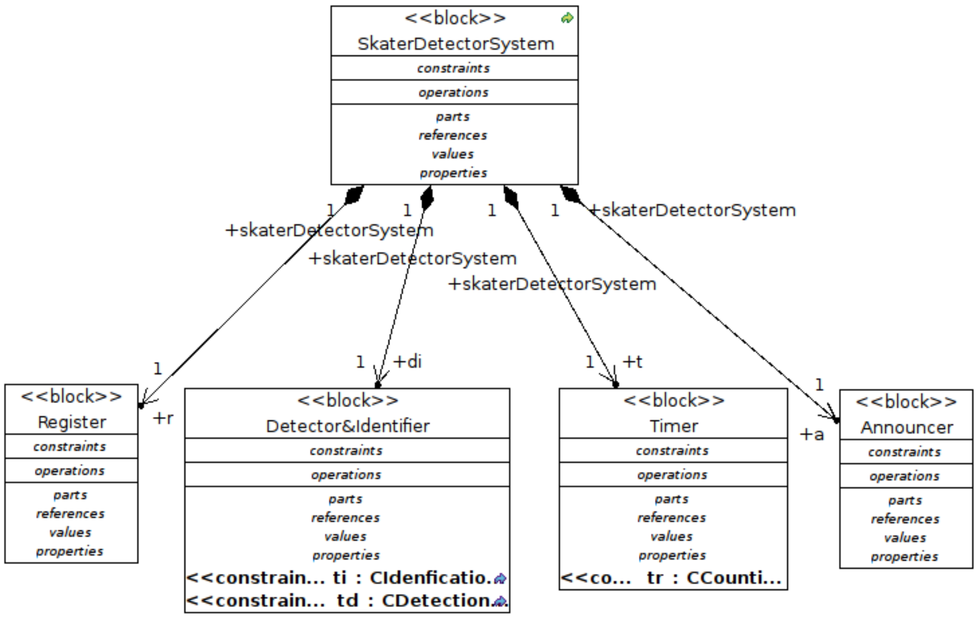


Figure 56: Block Diagram Definition in Level 1.

executable virtual prototype and start our searching in physical components.

Finished this process, we can descend other abstraction level and repeat the process. In the next levels, we can derivate specific requirements for each subsystem created, so we can detail subsystems requirements in each abstraction level and we can take each subsystem as a new system, with its own requirements, its own subsystems and so on.

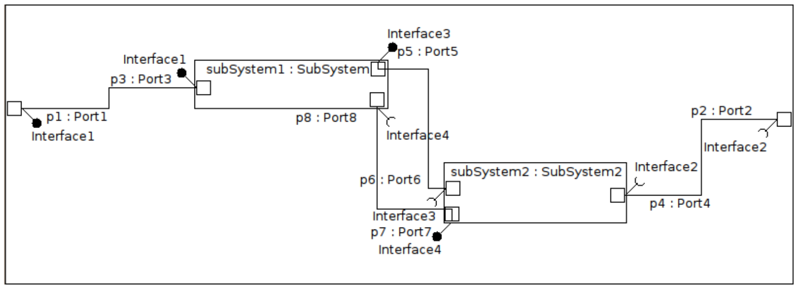


Figure 57: Internal System Representation in Level 1.

5.2 System Display Selector

During the development of our methodology, Airbus gave us an example to apply our methodology. It was the functional specification of a System Display selector which is a system used in airplanes produced by Airbus. We show the difficulty to apply our methodology which follows an object approach because the requirements are oriented to a functional solution. We apply directly HiLeS formalism; which follows a functional approach and we indicate the analysis that we can do.

5.2.1 Methodology applied by Airbus

The methodology used by Airbus to design a system is depicted in the figure 58. The first step is to capture the needs. This is a based text process where the needs are obtained by discussion with pilots and reusing previous designs. The second step is to transform the needs into specifications which have to be accurate, well defined, without misunderstanding. However, the specifications are not very clear and part of this misunderstandings is because their representation is text-based. The third step is to transform the specification to code in order to build a model executable. The suppliers cannot only achieve a model from the specification; they have to send engineers to Airbus in order to clear the specifications which are misunderstood or incomplete. When the model is made, the forth step is executed, which is the model validation. This is made in the Airbus laboratories by simulation and they see if the behavior of the system corresponds to their needs. If they find a problem in the model, they return to the specification and they review if the problem is in this step; if it is, they make the respective modifications and they send them to the suppliers to make the changes needed. Once the model validation is approved, the system is produced.

Airbus presented to us the difficulties that they find in this methodology and they are:

- Long cycle from specification to final validation.
- Different methods and tools along development cycle.

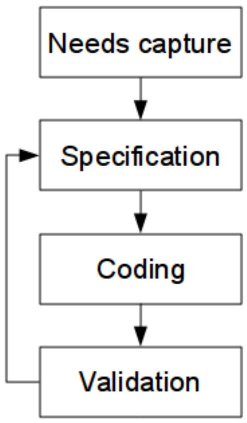


Figure 58: Airbus Methodology

- No pre-validation of specification before deliver it to the supplier.
- Maturity of the delivered specification.
- Interpretation by the supplier of the specification.

They proposed us to change its methodology replacing the textual based specification to a more adapted formalism, taking the textual specification and rewriting a new methodology to specify the system, improving early validation of the specification before deliver it to the supplier, and introducing novelties in SD selector specification to evaluate the impacts.

5.2.2 System Description

A System Display Selector (SD Selector) is a system which selects the correct page to be displayed according to their input signals, such as alerts, crew requests, advisories and flight phases.

It has four main processing units are:

- *ADV Processing:* This unit receives the advisory signals and selects the appropriated advisory image request. An advisory is a signal which indicates that certain parameter passes the normal range and it is not yet a warning or caution.

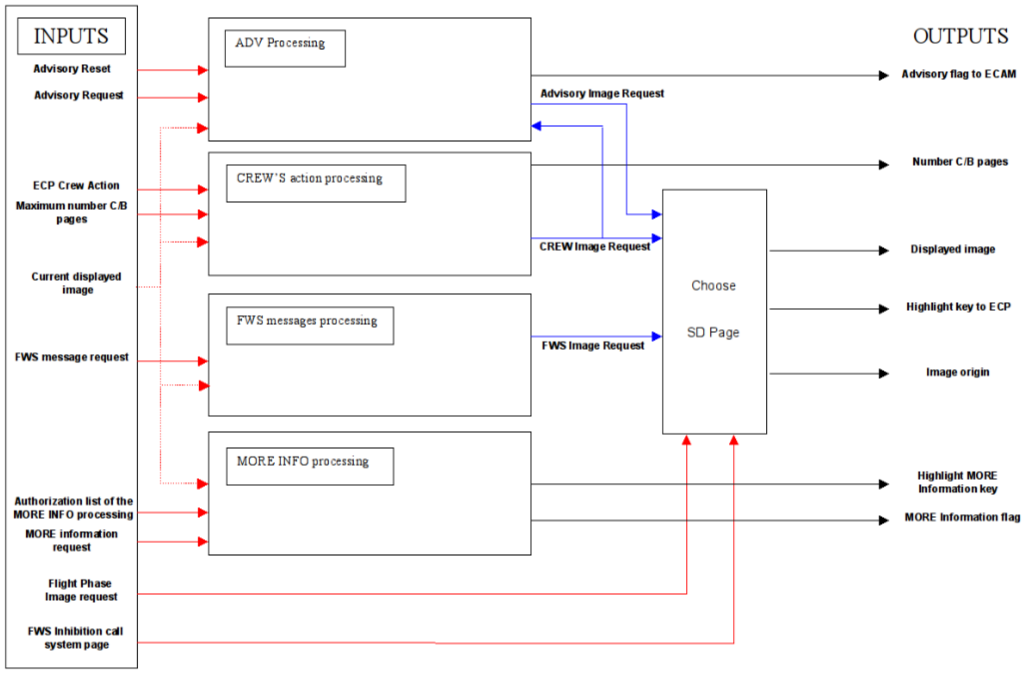


Figure 59: General architecture SD selector

- *CREW's action Processing:* This unit receives the signals from the ECAM Control Panel (ECP) and it selects the appropriated image request. ECP is a control panel used by the crew to see aircraft parameters displayed on the screen controlled by the SD Selector.

- *FWS messages Processing:* This unit receives the Flight Warning System (FWS) signals and it selects the appropriated image request.

- *Choose SD Page:* This unit receives the image request signal from the three units before and sends the image signal to be displayed and highlights the selected key on the ECP and gives the image request origin.

In the figure 59, we show the general architecture proposed in the SD selector specification for the A350 aircraft.

5.2.3 Proposed Methodology applied to Airbus Example

For this example, we choose the Crew's Action Processing to apply our methodology. The specification of this unit is more descriptive, it includes its subsystems and three levels of abstraction.

We apply our methodology beginning from the System Technical Requirements which are the textual requirements given by Airbus. We split the textual requirements in packages, according to the subsections of the document.

These packages are:

- Generalities
- Key Section
- Circuit Break Pages
- Computer Crew Image Request
- All Keys

Inside of each package, we create classification package, e.g. inside of the Generalities package, we have Functional Requirements and Performance Requirement packages. In the functional requirements we send the textual requirements from the Airbus documents to Requirements boxes in this package, later we split the requirements which have composed requirements. In the figure 60, we show the result of this analysis. We can see the requirement "Key Position and Status" is divided in two requirements because the requirement originally has two functionalities: Position and Status. The derived requirements have a unique single functionality which is "Key position" and "Key Status".

Once we have the system technical requirements well defined, we continue with the Functional and Architectural analysis. We begin with the use cases where we get the diagram showed in the figure 61.

We have an use case to manage the events produced in the ECP by the Crew and other use case which selects the request image according to the the Crew's action.

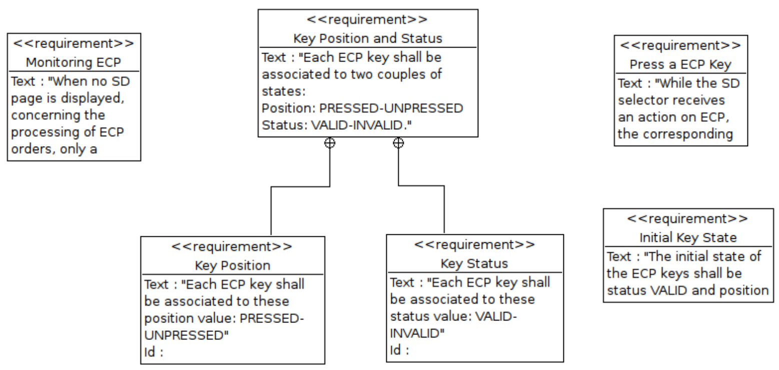

<<requirement>>
Monitoring ECP
Text : "When no SD page is displayed, concerning the processing of ECP orders, only a
<<requirement>>
Key Position and Status
Text : "Each ECP key shall be associated to two couples of states:
Position: PRESSED-UNPRESSED
Status: VALID-INVALID."
<<requirement>>
Press a ECP Key
Text : "While the SD selector receives an action on ECP, the corresponding
<<requirement>>
Key Position
Text : "Each ECP key shall be associated to these position value: PRESSED-UNPRESSED"
Id :
<<requirement>>
Key Status
Text : "Each ECP key shall be associated to these status value: VALID-INVALID"
Id :
<<requirement>>
Initial Key State
Text : "The initial state of the ECP keys shall be status VALID and position

Figure 60: Functional requirements of Generalities from Crew's Action Processing.

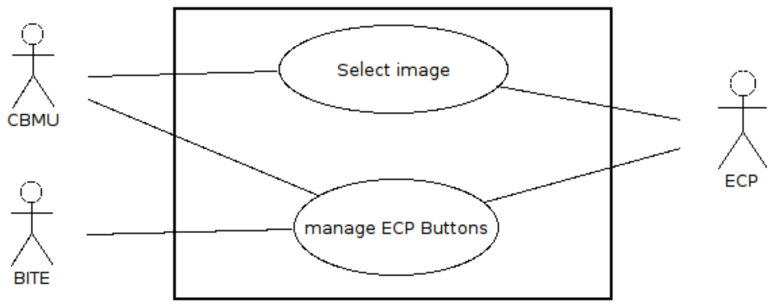


Figure 61: Crew's Action Processing Use cases diagram.

We identify also three actors: Circuit Breaker Monitoring Unit (CBMU), Built In Test Equipment (BITE) and ECP. CBMU is a system which gives the maximum number of CB pages, information which is used when the crew presses “C/B” button on the ECP. BITE is a system used in simulation environment.

When the use cases are defined, we develop the scenarios, even though we realize the “manage ECP Buttons” use case is split in different scenarios according to the button which is pressed. If we create a scenario for each crew selection, we will have difficulties to join the sequence diagrams and the verification of the functionality of the model will be made by parts, and consequently this will cause a hole in the verification process.

Another problem that we find is the difficulty to split the system in subsystems. The Airbus document defines an architecture in the Crew’s action processing, and each subsystem has one functionality, e.g. there is a subsystem which manages the ALL key actions when it is pressed. The functionality of this unit is to send a key request each specific time. We do not have more functions for this unit, so this approach is different from the object approach where an object encapsulates the functions. It is more approximated to a functional approach. We show in the figure 62 a possible sequence diagram which mixes the possible scenarios when the crew presses a button on the ECP.

5.2.4 Applying HiLeS Formalism

As the approximations of the Airbus specification are functional, we decide to transform them directly to HiLeS formalism. We create a fist abstraction level where we have a system, an environment and their interactions represented by signals. This level is similar to the level 0 of our methodology. The figure 63 depicts this level. We can see most of the interaction signals are events, because of the logical functionality of the system.

For the following levels, we are subsequent to the architecture given by the Airbus document, preserving the same hierarchy. We can see the level 1 and level 2 of the system, specifically Crew’s action processing unit in the Airbus document and HiLeS, figure 64 and 65 respectively. We can see the units are the same in both

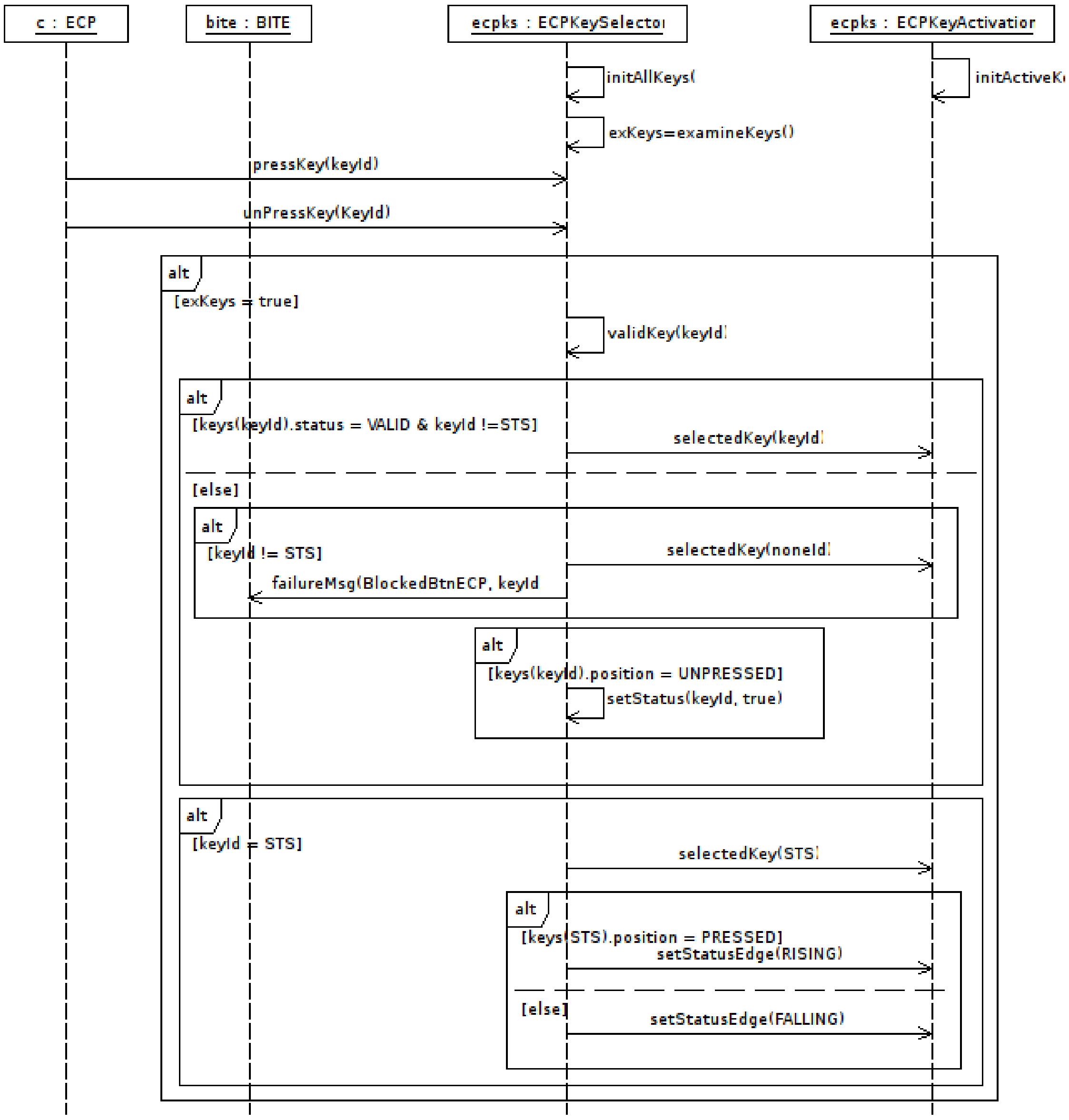


Figure 62: Part of a scenario in the Crew's action processing.

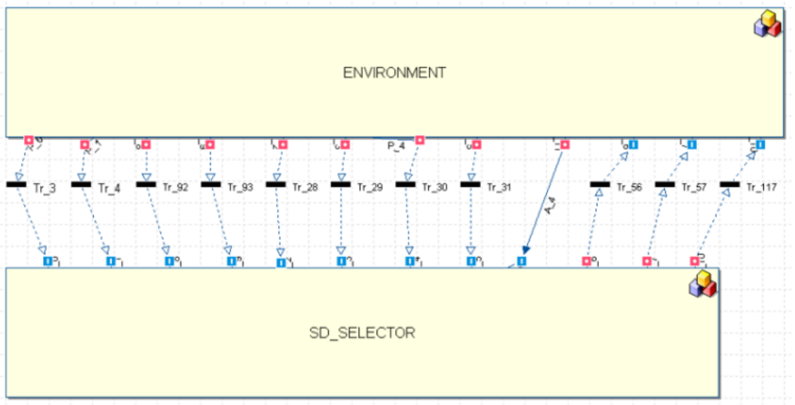


Figure 63: SD Selector Level 0 using HiLeS.

representations, just the interaction among them changes.

Having the structural base of our unit, we begin to develop the behavior of each unit in the level 2, that means, "Examine Key By Priority", "Compute Crew Image Request" and "Processing ALL Key".

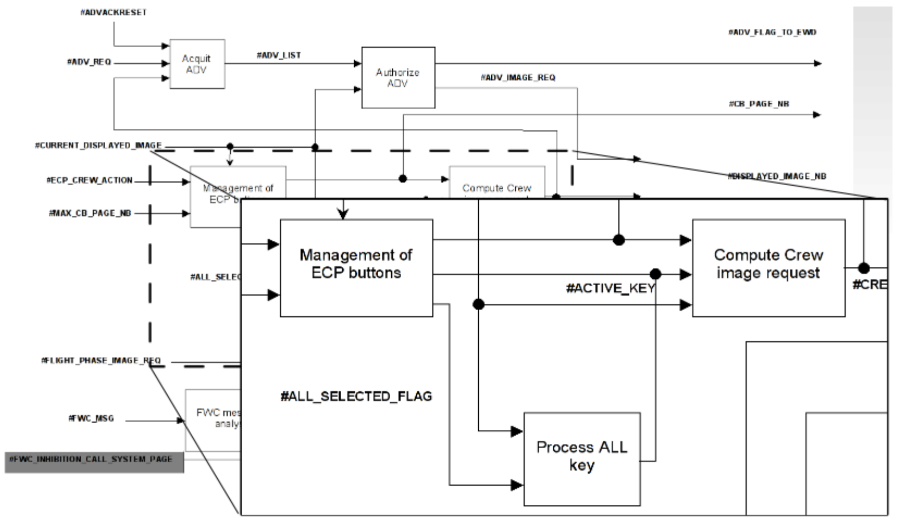
First, we take "Examine the table of manual requests" and we go down a level more to describe its behavior (Level 3). In the figure 66, we illustrate how we distribute the behavior in architectural units, and each of them follows the behavior described in the Airbus document requirements.

To show how it works we take the "Examine the table of the manual request". Inside of this unit, we have two main units: "Examine Key By Priority" and "Valid X Key". The first unit follows the requirement CDSA380_A1_SELSD_012-2 and A350XWB_DGWP_A1.11_SD_Selector_0020 where say:

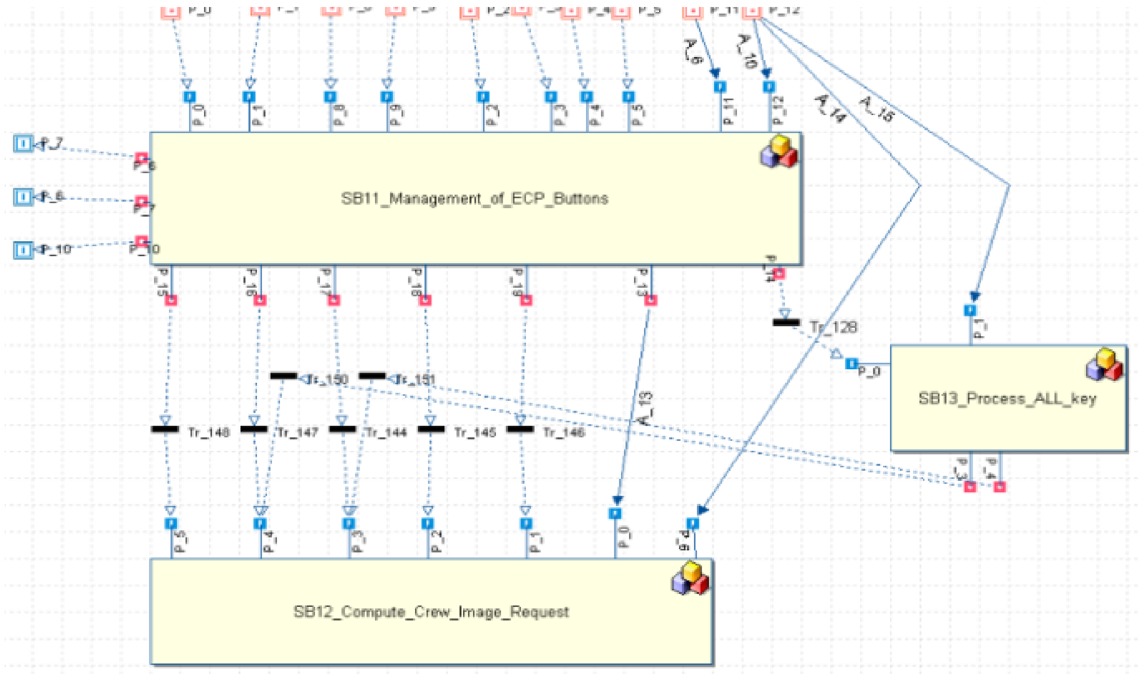
"ECP keys shall be examined in the following order: ENG, APU, BLEED, COND, PRESS, DOOR, EL/AC, EL/DC, FUEL, WHEEL, HYD, F/CTL, C/B, ALL, VID and STS."

Thus, this unit examines in order which key is pressed and it gives the result. If two keys are pressed at the same time, according to the key order, the first key pressed in the list is selected.

"Valid X Key" is a unit which is reused for each key when they are pressed and selected by priority. This unit follows the requirements CDSA380_A1_SELSD_101-1,

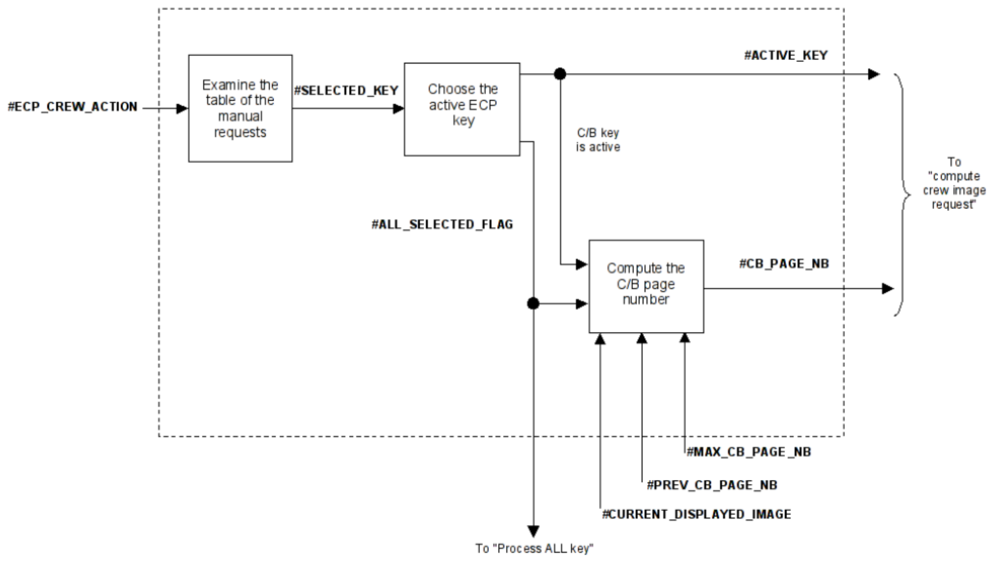


(a) Crews' Action Processing Architecture from Airbus Document.

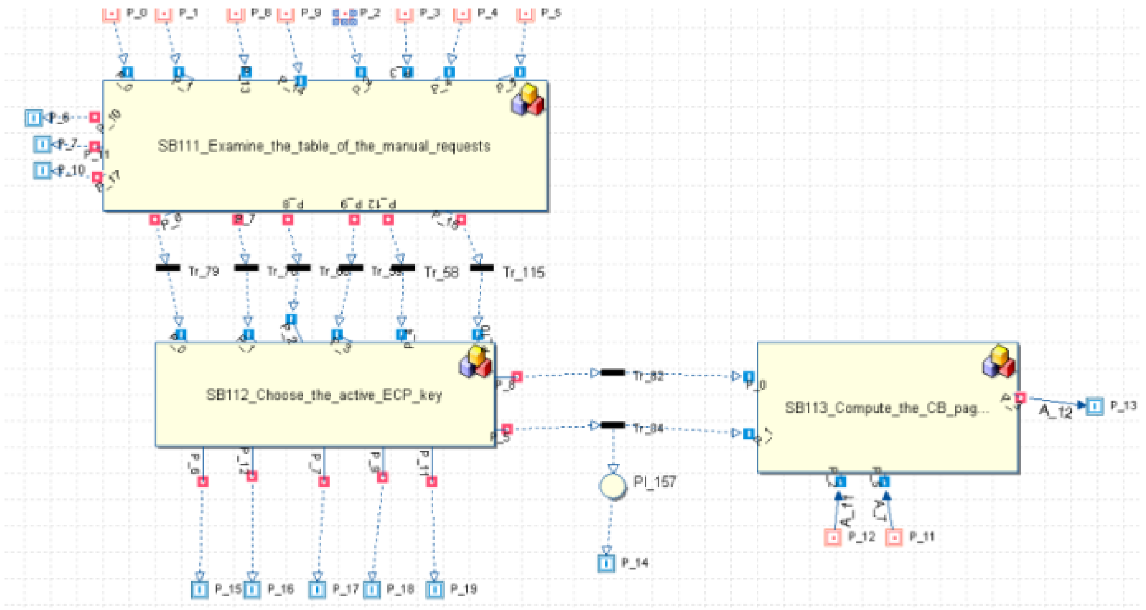


(b) Crews' Action Processing Architecture Level 1 using HiLeS.

Figure 64: SD Selector Level 1 Representation



(a) Management of ECP Buttons Architecture from Airbus Document.



(b) Management of ECP Buttons Architecture Level 1 using HiLeS.

Figure 65: SD Selector Level 2 Representation.

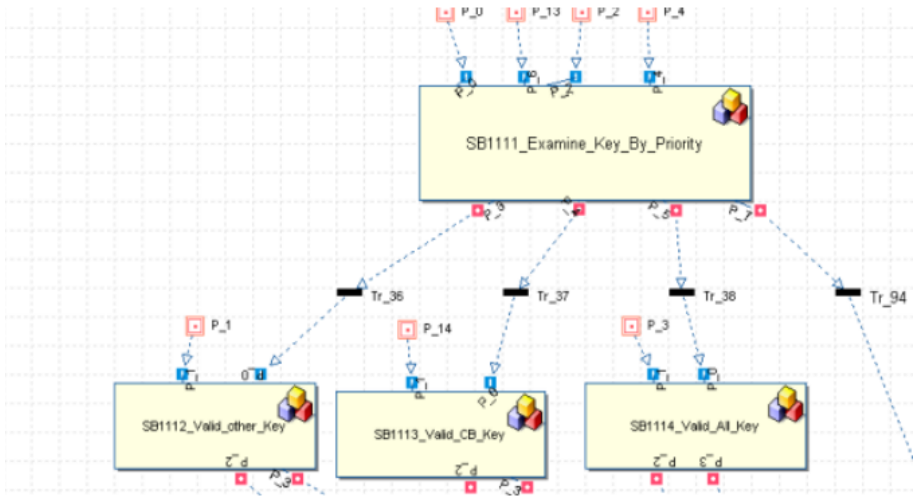


Figure 66: SD Selector Level 3 Representation.

CDSA380_A1_SELSD_071-1, CDSA380_A1_SELSD_014-5, CDSA380_A1_SELSD_102-3, CDSA380_A1_SELSD_015-5, CDSA380_A1_SELSD_018-4 and DSA380_A1_SELSD-019-3. Its behavior is described by Petri nets and it is depicted in the figure 67.

The behavioral description of "Valid X Key" is the following: Pl_66 is the position "Unpressed" and Pl_69 is the status "valid". In PL_64, there is a token if the Key X is pressed and selected. When that happens, the Timer T2 starts, and the unit wait a token in Pl_65 which corresponds to the action "unpressed". Thus, at the same time, the token in the Pl_66 is removed and a token is put in the Pl_67 which is the representation of the position "Pressed". If we have the key "Pressed" and "Valid", then the token in the Pl_67 and Pl_69 are absorbed by the transition Tr_65, and it put a token in the Pl_71 which correspond to the key is selected. The key is "valid" until the time in the timer is over (until 45 seg), then a token is send to Pl_72, which correspond to the error signal send it to the system BITE. At the same time, other token is sent to Pl_70 which corresponds to the status "Invalid". If the key is unpressed before the time in the timer is over. The key continue in status "valid". If the key is "invalid" and later it is unpressed, it becomes "valid" and wait to be pressed again.

Some requirements are described by algorithms, thus we directly translate them

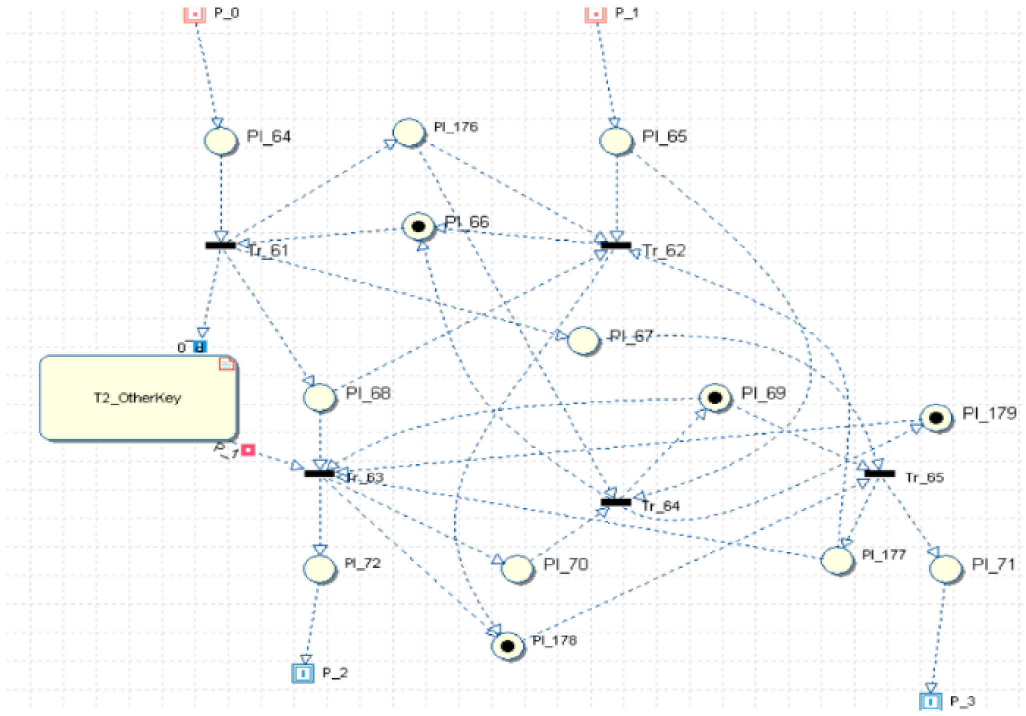
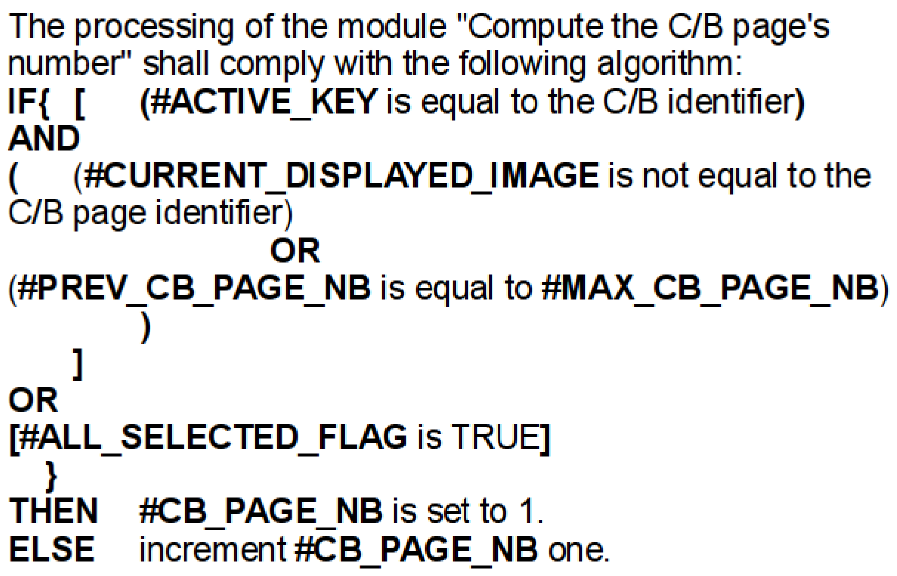


Figure 67: "Valid X Key" behavioral description using HiLeS.

to VHDL-AMS code in a functional block. An example is the requirement CDSA380-A1_SELSD_023-3 and, in the figure 68, we illustrate how its transformation is done in HiLeS. If we want to describe in more detail this behavior, we can transform the algorithm in a Petri Net representation in order to perform a formal verification analysis. We will also have to change the functional block by a structural block.

Returning to the Level 0, the environment is represented by the system which interacts with the SD selector system. In our example, we only need the systems which send signals to and receive from "Crews action processing" unit. In the figure 69, we show the environment levels in HiLeS. We only include the systems which send signals to the unit, and they are ECP and CBMU. Inside of ECP we have the blocks which represent the behavior of a key. We extract the actions of special keys such as "ALL" and "C/B", because they generate special behaviors in the system that is necessary to study separately in order to verify the behavior of our model with the requirements.

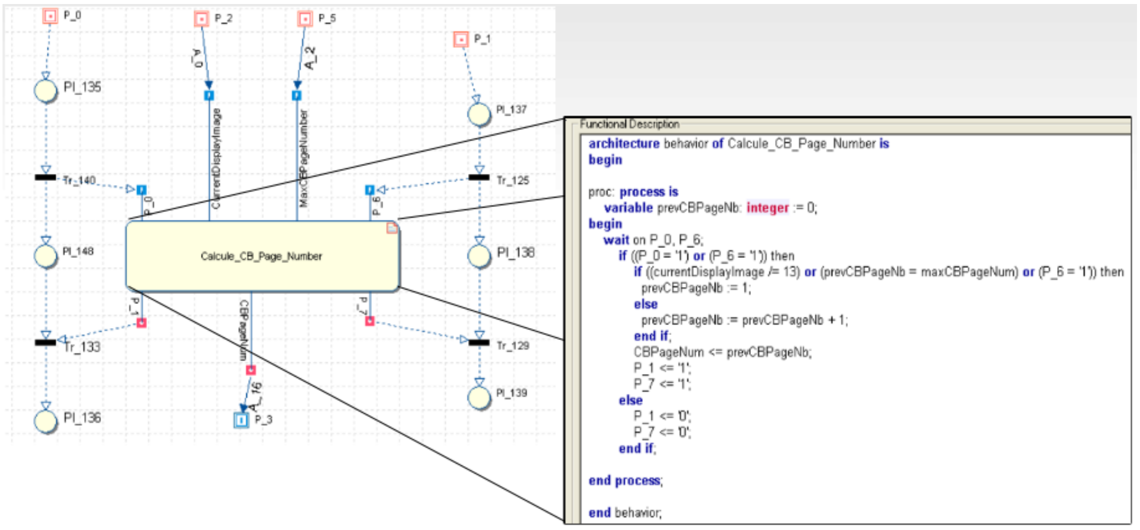
In the figure 70, the key functionality is presented. We can see the key behavior



The processing of the module "Compute the C/B page's number" shall comply with the following algorithm:

```
IF{  [     (#ACTIVE_KEY is equal to the C/B identifier)
AND
(     (#CURRENT_DISPLAYED_IMAGE is not equal to the
C/B page identifier)
                          OR
(#PREV_CB_PAGE_NB is equal to #MAX_CB_PAGE_NB)
             )
      ]
OR
[#ALL_SELECTED_FLAG is TRUE]
   }
THEN    #CB_PAGE_NB is set to 1.
ELSE    increment #CB_PAGE_NB one.
```

(a) Airbus Document Requirement.



```
architecture behavior of Calcule_CB_Page_Number is
begin

proc: process is
   variable prevCBPageNb: integer := 0;
begin
   wait on P_0, P_6;
      if ((P_0 = '1') or (P_6 = '1')) then
         if ((currentDisplayImage /= 13) or (prevCBPageNb = maxCBPageNum) or (P_6 = '1')) then
          prevCBPageNb := 1;
         else
          prevCBPageNb := prevCBPageNb + 1;
         end if;
         CBPageNum <= prevCBPageNb;
         P_1 <= '1';
         P_7 <= '1';
      else
         P_1 <= '0';
         P_7 <= '0';
      end if;

end process;

end behavior;
```

(b) Requirement HiLeS representation.

Figure 68: Requirement algorithm represented in HiLeS

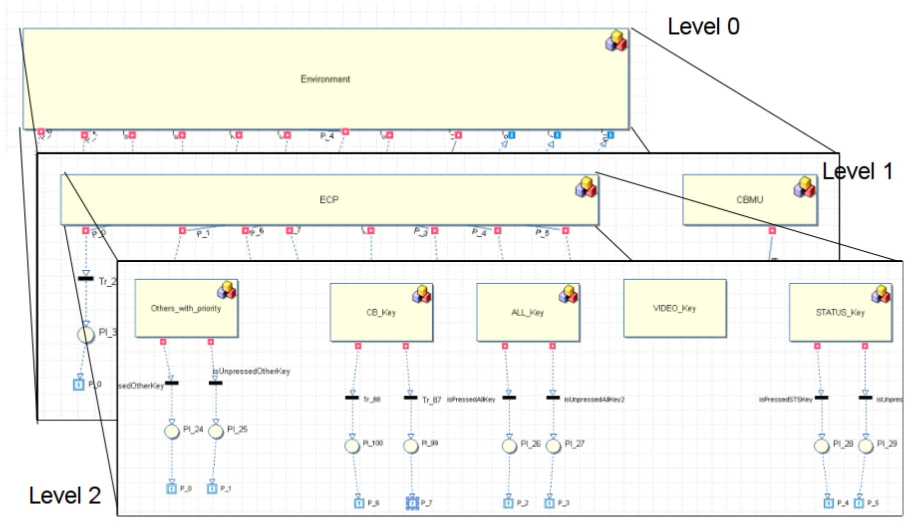


Figure 69: Environment levels using HiLeS.

is the key pressed and unpressed. It has a timer, equal to "Valid X key", to represent the delay between the action "press" and "unpress" a key. In order to create different scenarios, we can modify the configuration of these blocks.

5.2.5 SD Selector System-HiLeS Model Verification

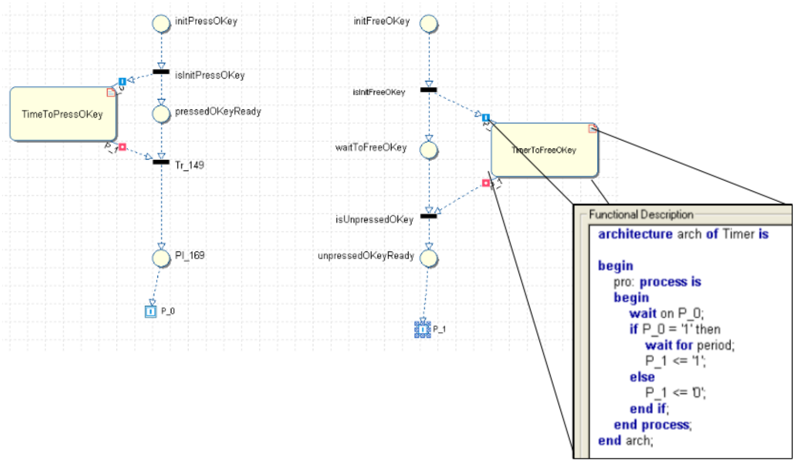
The idea is to show the HiLeS model can be verified in two ways: Formal analysis verification by Petri Net analysis and verification by simulation. The first verification is made using a Petri net analyzer called TINA [BRV04], the second one is using a tool called SystemVision.

5.2.5.1 *Formal Logic Analysis Verification*

We chose the "Valid X Key" unit to verify two requirements:

"**CDSA380_A1_SELSD_018-4:** #SELECTED_KEY shall be the identifier of the first PRESSED AND VALID key encountered in the examination defined in CDSA380_A1_SELSD_012.

For "Status key", #SELECTED_KEY shall be set to STATUS page ident on rising and falling edge For "Status key", the property #EDGE shall be associated to #SELECTED_KEY. #EDGE shall be set to rising or falling according to the



```
architecture arch of Timer is

begin
  pro: process is
  begin
    wait on P_0;
    if P_0 = '1' then
      wait for period;
      P_1 <= '1';
    else
      P_1 <= '0';
    end if;
  end process;
end arch;
```

Figure 70: Action Key representation using HiLeS.

Pressed (P)	Valid (V)	None (N)	Selected (S)	Req.
0	0	1	0	19
0	1	1	0	
1	0	1	0	
1	1	0	1	18
0	0	0	0	N/D
0	1	0	0	
1	0	0	0	

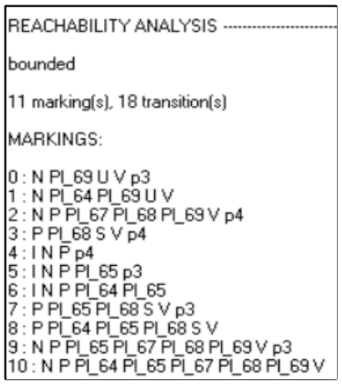
Table 5: Cases to evaluate the requirements 18 and 19 of the SD selector.

edge detected in "Status key", value."

"**CDSA380_A1_SELSD_019-3:** If there is not a (PRESSED AND VALID) key then #SELECTED_KEY shall be set to NONE."

We first build a table with the cases to evaluate the behavior of our unit and to identify which cases are tied to the requirements presented before. In the table 5, we show the table cases. We can observe the first three lines are related to the requirement 19, the fourth line with the requirement 18. We can also find other three lines which are not related with the requirements. These last lines are cases which cannot be possible in the behavior of our model and they are not taken into account in the requirements, therefore we can find a problem in the requirements.

Hence, we run an analysis of our model in TINA and the results are depicted in the table 5. The figure shows the possible markings for the model proposed in



```
REACHABILITY ANALYSIS ----------------------

bounded

11 marking(s), 18 transition(s)

MARKINGS:

0 : N Pl_69 U V p3
1 : N Pl_64 Pl_69 U V
2 : N P Pl_67 Pl_68 Pl_69 V p4
3 : P Pl_68 S V p4
4 : I N P p4
5 : I N P Pl_65 p3
6 : I N P Pl_64 Pl_65
7 : P Pl_65 Pl_68 S V p3
8 : P Pl_64 Pl_65 Pl_68 S V
9 : N P Pl_65 Pl_67 Pl_68 Pl_69 V p3
10 : N P Pl_64 Pl_65 Pl_67 Pl_68 Pl_69 V
```

Figure 71: Marking in our "Valid X key" unit.

the unit, and we can see in the marking 3, 7 and 8 that the places P, V and S have tokens, and we cannot find any other marking with S without P and V. This analysis corresponds to the requirement 18 and we can verify that our model is according to the requirement. Following this requirement, it is not possible to have token in S and P without have token in S. We can see in the figure 5 the lines 2, 9 and 10 which have other result, it is possible because the transition when P and V have tokens is not fired yet.

To verify the requirement 19, we realize if P or V has token, then the place N must have a token, also if P and V has not token. We can see this behavior in the lines 0, 1, 4, 5 and 6 of the figure 71. On the other hand, we appreciate in the same figure the lines 2, 9 and 10, where the same problem is presented before.

Finally, the other possible cases (the last three lines in the table), we cannot find them in the figure, so we can conclude our model is correct and it is valid according to the requirements including the non-defined case.

5.2.5.2 *Verification by simulation*

Once the formal logic verification is made, we continue the verification of the system behavior by simulation. We use this verification to test the possible scenarios in our system and to get expected results which are guided by the requirements.

We chose three scenarios to configure them in our HiLeS model and to get the

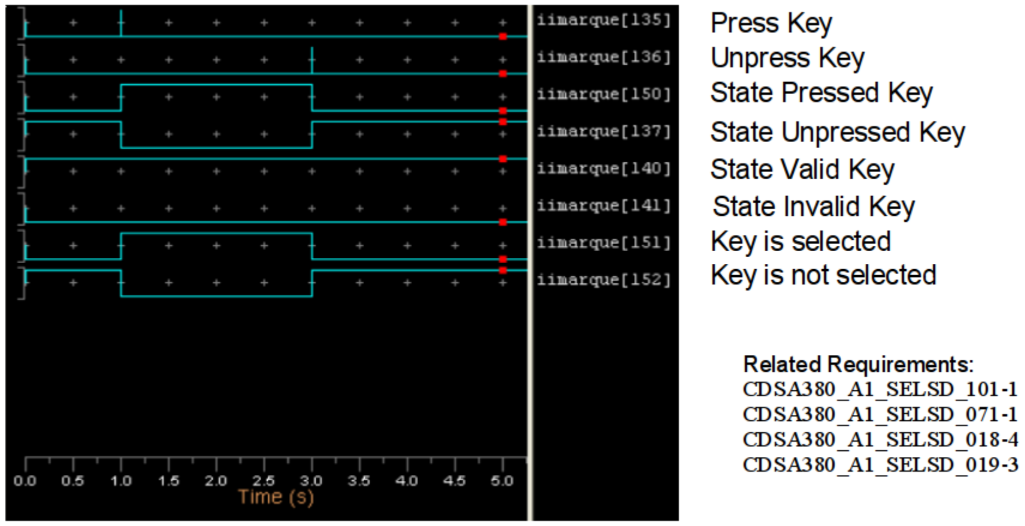


Figure 72: Simulation of "Valid X key" Scenario 1, press a key during 2 sec.

results using the VHDL-AMS transformation and simulate it in the tool SystemVision.

The first scenario is press a ECP key during 2 seconds. This scenario is according to the Requirements 18, 19, 71 and 101 from the Airbus document. We configure the environment in the HiLeS model, we generate the virtual prototype and we simulate it. The results can be appreciated in the figure 72. The first two signals in the figure corresponds to press and to release the key respectively. These are impulses because of the Petri net model behavior. Before the key is pressed, the key is "unpressed" and "valid" (signal 4 and 5 respectively). When the key is pressed, the key passes from the position "unpressed" to "pressed" and it continues in status valid. Once the key is in state "pressed" and "valid", the key becomes "selected" (signal 7). When the key is released, the position key changes from "pressed" to "unpressed", the status continues in "valid" because the timer do not arrive to 45 seconds, and the key is not selected, that means, it send a none signal. The results are according to the Airbus requirements.

The second scenario is to press a ECP key during 48 seconds. This scenario follows the Requirements 14 and 15 from the Airbus document. The HiLeS model is reconfigured for this scenario and the simulation is showed in the figure 73. We

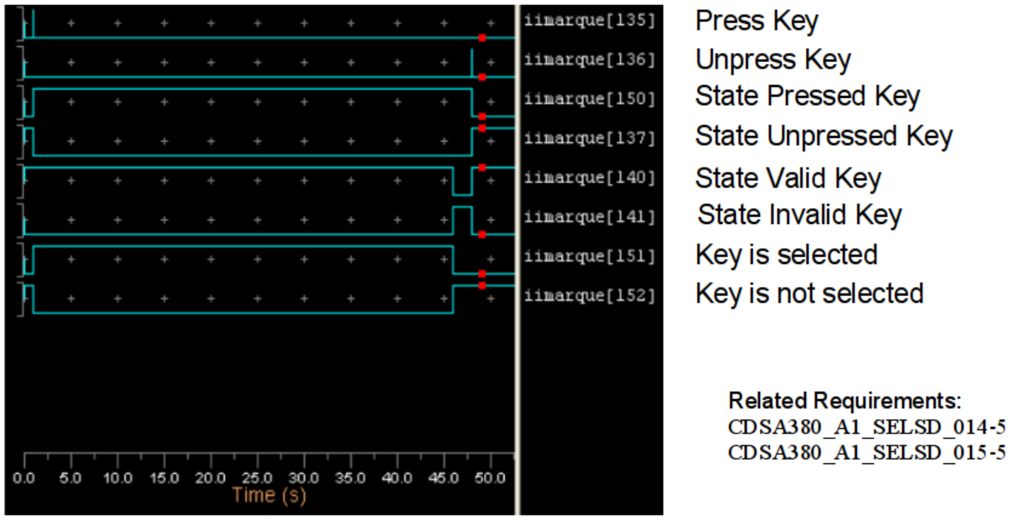


Figure 73: Simulation of "Valid X key" Scenario 2, press a key during 48 sec.

can observe after 45 seconds that the key is pressed, the key status changes from valid to invalid. Once the key is released, the status changes again to valid. Also we can see the key is selected during 45 seconds, and when the status changes, the key changes from selected to non-selected, and it continues at this level until a new pressing event arrives.

The final scenario is to press three times the C/B key each second. The maximum C/B page is equal to 2 and the image, which is displayed, is C/B. In the figure 74, we illustrate the simulation after the reconfiguration on the HiLeS model. The requirements related to this scenario are 23 and 104. We can see in the figure that before of the first pressed key, "C/B Page Number" is in 0. When C/B key is pressed, C/B changes to 1, which is the first page of C/B. Later, C/B key is pressed again and C/B page changes from 1 to 2. As 2 is the maximum C/B pages, when C/B key is pressed the third time, the C/B page is reset and it starts again in 1.

5.3 Conclusion

In this chapter, we present two examples in order to show the application of the proposed methodology explained in the chapter 4. The first example, the Skating Manager System, was used to develop the methodology and it helps to illustrate how

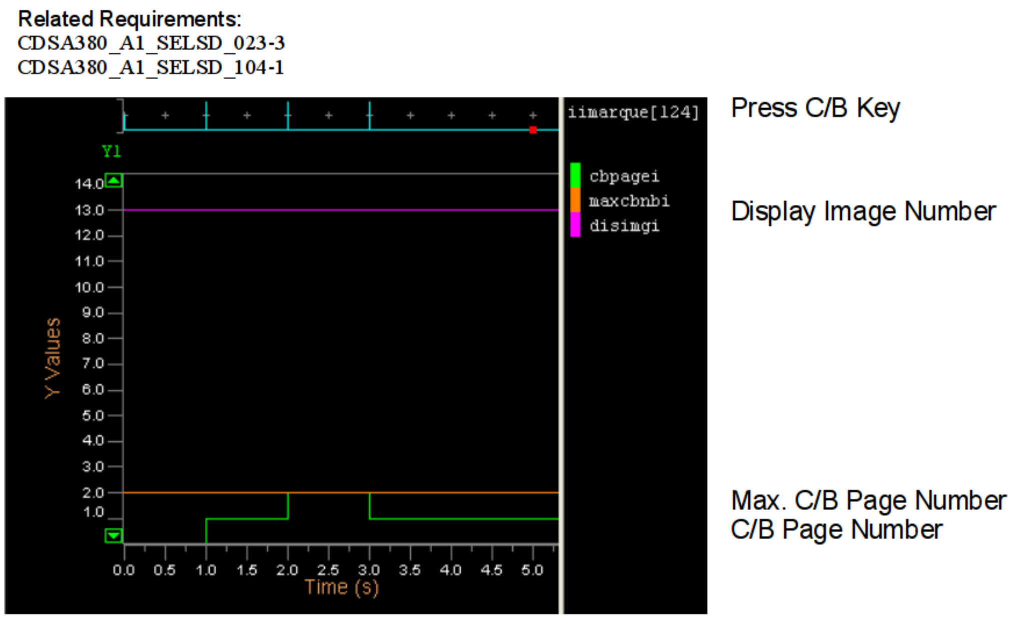


Figure 74: Simulation of Scenario 3, press three times C/B Key.

the methodology can be used. The second example, the System Display Selector, was an Airbus example used to apply the methodology starting from the system technical requirements. These requirements were deduced from a specification document given by Airbus. We realized in this example that the specification was built in a function approach, creating a difficulty to apply our methodology, which follows an object-oriented approach.

We show in the Airbus example the importance to verify in high level abstraction the logical solution which is according to the requirements given by the stakeholders. We illustrate the HiLeS properties in the verification process using the Petri net representation and the virtual prototype generated in VHDL-AMS.

In the next chapter, we will give the conclusions of our work and the perspectives that we can expect in the future.

CHAPTER VI

Conclusions and Future Work

In this work, we propose a new methodology to design complex systems. We use as basis the EIA-632 standard and as description language SysML. We define the conceptions of method, process, tools and methodologies to center our methodology inside a conception well defined. We explain different methods applied to system design, their relationships and the tools used to implement them. We classify the tools according to their features, developers and methods which can be implemented. We summarize the EIA-632, especially the System Design sub clause which is the basis of our methodology. We explain what SysML is and what it offers to system engineering. We present the different diagrams included in this model language. We define our methodology, its steps, its iterations, its SysML diagrams used and its relationship with EIA-632 standard. We show the relationship between our methodology and MDA. Finally, we develop two examples, the Skate race manager and the System Display Selector, where the methodology is illustrated and we show the difference between the functional approach and the object approach.

We show a clear tendency to rise the design level in embedded systems using the system level description, where there is not taken into account which part will be software and hardware. According to this tendency, the system is split in functionality and components. The result of the mapping between functionality and components is a system optimized following the system technical requirements. Also, we present another division between the model representation and verification (e.g. simulation and logical analysis), and, using our methodology, we describe how they are strongly connected to achieve a well-defined system design, reducing misunderstandings and verifying the design in each step of the system design process. SysML allows expressing the system in concepts easier to use for the systems engineers, e.g. the block concept used in the Block Definition Diagram. INCOSE, which is an organization to help to the development of the system engineering, is part of the development group of this language. On the other hand, UML, which is also a language used for systems development, has many diagrams, many ways to understand its elements and it does not follow an orientation to system engineering.

This work opens the path to add new SysML diagrams such as Activity diagram which is a more complete formalism than the sequence diagram to describe the behavior of the system or subsystems. Sequence diagram is used to describe the scenarios of the system, but it is limited to generalize the behavior of the system deduced from the scenario. We think they are complemented and the Activity diagram can be a next step in our methodology, after the sequence diagram, in the system behavior description.

Another work we propose for the future is the transformation of this SysML model to the HiLeS formalism. We note we have the structural description, behavioral description and the constraints that the system has to fulfill. However, these descriptions are not executables and this limits the verification process in a system design. We believe that HiLeS can be the execution environment for the SysML model and also a good platform to formally verify the logical behavior of our system using the Petri Net formalism included in HiLeS and to verify by simulation generating a virtual prototype in VHDL-AMS. Once we have the model in HiLeS, the details, which cannot be described in SysML, can be possible using VHDL-AMS.

Even though the HiLeS model transformation is developed in the present version of HiLeS Designer, we consider translating the VHDL-AMS transformation and the Petri net generation to an environment more suitable with the MDA concepts. We realize TOPCASED is a good option and even more when the SysML diagrams are implemented. The other tools are more closed, most of them do not follow the MDA concept, and among the tools which support MDA, some of them support SysML. MagicDraw group is developing a wonderful tool to support SysML, but its close code feature does not allow developing in its structure.

References

[16608] IEEE 1666. IEEE standard systemC language reference manual. IEEE, Mar 2008.

[AM06] A. Asaduzzaman and I. Mahgoub. Cache modeling and optimization for portable devices running MPEG-4 video decoder. *Multimedia Tools and Applications*, pages 239–256, February 2006.

[Bai07] B. Bailey. ESL-a methodology for handling complexity. `http://electronicsystemlevel.com`, Jan 2007.

[BCC$^+$00] L. Baker, P. Clemente, B. Cohen, B. Purves L. Permenter, and P. Salmon. Foundational concepts for model driven system design. *INCOSE Model Driven System Design Interest Group*, 2000.

[BDDM03] P. Boulet, J-L Dekeyser, C. Dumoulin, and P. Marquet. MDA for SoC embedded systems design, intensive signal processing experiment. *SIVOES-MDA workshop at UML*, October 2003.

[BRV04] B. Berthomieu, P.O. Ribet, and F. Vernadat. The tool TINA: Construction of abstract state spaces for petri nets and time petri nets. *International journal of production research*, 42(14):2741–2756, 2004.

[BWH$^+$03] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *Computer*, 36(4):45–52, 2003.

[Cal93] J.P. Calvez. *Real-Time Systems a Specification and Design Methodology*. John Wiley & Sons, 1993.

[CFDBSV$^+$05] L.P. Carloni, C. Pinello F. De Bernardinis, A.L. Sangiovanni-Vincentelli, , and M. Sgroi. Platform-based design for embedded systems. `www.cs.columbia.edu/~luca/research`, 2005.

[CL08] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event System*, section 4, pages 223–268. Springer, New York, New York, second edition, 2008.

[Clo08] R. Cloutier. Model driven architecture for systems engineering. *INCOSE International Workshop*, 2008.

[Dom02] R. Domer. SpecC language reference manual. `www.ics.uci.edu/~specc/reference`, Dic 2002.

[DPSV06] D. Densmore, R. Passerone, and A. Sangiovanni-Vincentelli. A platform-based taxonomy for ESL design. *IEEE Des. Test*, 23:359–374, 2006.

[dSV07] M. dos Santos and J. Vrancken. Requirements specification and modeling through SysML. *IEEE International Conference on Systems, Man and Cybernetics*, pages 1735–1740, 2007.

[EIA99] EIA. EIA-632 processes for engineering a system. *Electronic Industries Alliance*, 1999.

[EOPP05] P. Esteban, A. Ouardani, M. Paludetto, and J.C. Pascal. A component based approach for system design and virtual prototyping. *European Concurrent Engineering Conference*, pages 85–90, 2005.

[Est08] M. Estefan. Survey of model-based systems engineering (mbse) methodologies. `http://www.omgsysml.org/`, May 2008.

[FMS08] S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML*. Morgan Kaufmann Publishers, 2008.

[For09] Sysml Forum. SysML tools. `http://www.sysmlforum.com/tools.htm`, 2009.

[fSIS09] Institute for Software Integrated Systems. Model integrated computing. `http://www.isis.vanderbilt.edu/research/MIC`, 2009.

[FV09] P. Farail and F. Vernadat. Toolkit in OPen-source for Critical Applications and SystEms Development. `http://www.topcased.org/`, 2009.

[Gaj07] D.D. Gajski. New strategies for system-level design. *DDECS*, page 15, April 2007.

[Gra09] Mentor Graphics. System vision. `www.mentor.com/SystemVision`, 2009.

[Ham05] J.C. Hamon. *Mthodes et outils de la conception amont pour les systmes et les microsystmes.* PhD dissertation, Institut National Polytechnique, Ecole doctorale de Gnie Electrique, Electronique, Tl-communications, February 2005.

[HB03] J-L. Houberdon and J-P. Babau. MDA for embedded systems dedicated to process control. *SIVOES-MDA workshop at UML*, October 2003.

[HKL$^+$07] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y. Joo. PeaCE: A hardware-software codesign environment for multimedia embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 12(3):1–25, 2007.

[HS07] T.A. Henzinger and J. Sifakis. The discipline of embedded systems design. *Computer*, 40(10):32–40, 2007.

[IBM09a] IBM. Ibm rational tau. `http://www.telelogic.com/products/tau/index.cfm`, 2009.

[IBM09b] IBM. The telelogic rhapsody family from IBM. `http://www.telelogic.com/products/rhapsody/`, 2009.

[INC00] INCOSE. System engineering handbook. *Interantional Council on Systems Engineering*, 2000.

[Inc03] Mirabilis Design Inc. VisualSim datasheet. `http://www.mirabilisdesign.com/Pages/Product/mdi_products.htm`, 2003.

[Inc08] Synopsys Inc. System studio datasheet. `http://www.synopsys.com/TOOLS/SLD/DIGITALSIGNALPROCESSING/Pages/SystemStudio.aspx`, 2008.

[Inc09a] No Magic Inc. Cameo datahub. `http://www.cameosuite.com/products/requirements/cameodatahub`, 2009.

[Inc09b] No Magic Inc. Paramagic plugin. `http://www.magicdraw.com/main.php?ts=navig&cmd_show=1&menu=paramagic`, 2009.

[Jim00] F. Jimenez. *Spcification et Conception de Micro-systmes Bass sur des Circuits Asynchrones.* thse doctorat, INSA and Universidad de los Andes, November 2000.

[LdOFV07] M.V. Linhares, R.S. de Oliveira, J.-M. Farines, and F. Vernadat. Introducing the modeling and verification process in SysML. *ETFA*, pages 344–351, September 2007.

[Mai08] N. Maiden. User requirements and systems requirements. *IEEE Softw.*, 25(2):90–91, 2008.

[Mar96] J.N. Martin. *Systems Engineering Guidebook: A Process for Developing Systems and Products.* CRC Press, Boca Raton, Florida, 1996.

[MM05] G. Martin and W. Mller. *UML for SoC design.* Springer, Dordrecht, 2005.

[Mon07] M. Montoreano. Transaction level modeling using OSCI TLM 2.0. `http://www.systemc.org/downloads/standards`, May 2007.

[MT00] N. Medvidovic and R.N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, 26(1):70–93, 2000.

[MT07] Inc. MLDesign Technologies. MLDesigner documentation. `http://www.mldesigner.com/mldesigner/documents.html`, 2007.

[OMG03] OMG. Mda guide v1.0.1. `http://www.omg.org/docs/omg`, 2003.

[OMG08a] OMG. OMG systems modeling language (OMG SysML). *Object Management Group*, 2008.

[OMG08b] OMG. A UML profile for MARTE: Modeling and analysis of real-time embedded systems. *Object Management Group*, 2008.

[PBF$^+$07a] R.S. Peak, R.M. Burkhart, S.A. Friedenthal, M.W. Wilson, M. Bajaj, and I. Kim. Simulation-based design using SysMLpart 1: A parametrics primer. *INCOSE Intl. Symposium*, 2007.

[PBF$^+$07b] R.S. Peak, R.M. Burkhart, S.A. Friedenthal, M.W. Wilson, M. Bajaj, and I. Kim. Simulation-based design using SysMLpart 2: Celebrating diversity by example. *INCOSE Intl. Symposium*, 2007.

[Pim05] A. D. Pimentel. The artemis workbench for system-level performance evaluation of embedded systems. *International Journal of Embedded Systems*, 1(7), 2005.

[PL01] A. Bakshiand V. K. Prasanna and A. Ledeczi. MILAN: A model based integrated simulation framework for design of embedded systems. In *LCTES '01: Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems*, pages 82–93, New York, NY, USA, 2001. ACM.

[Pri09] Primordion. Xholon. http://www.primordion.com/Xholon/index.html, 2009.

[Sin09] Sinelabore. SinelaboreRT. http://www.sinelabore.com/, 2009.

[STF$^+$03] G. Schorcht, I. Troxel, K. Farhangian, P. Unger, D. Zinn, C.K. Mick, A. George, and H. Salzwedel. System-level simulation modeling with MLDesigner. *MASCOTS*, pages 207–212, October 2003.

[SV07] A. Sangiovanni-Vincentelli. Quo vadis, SLD? reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[Top09] Topcased. Topcased. http://www.topcased.org, 2009.

[Wei08] T. Weilkiens. *Systems Engineering with SysML/UML Modeling, Analysis, Design.* Morgan Kaufmann Publishers, mars 2008.