Carl von Ossietzky Universität Oldenburg
Facult II – Computer Science, Business and Law
Department of Computer Science

Federal University of Uberlândia
Faculty of Computing
Posgraduate Program in Computer Science

Multi-formalism in Different Levels of Abstraction for Requirements Engineering and Architectural Design of Real-Time Embedded Systems

From the faculty of Computer Science, Business and Law at the Carl von Ossietzky University Oldenburg and Federal University of Uberlândia to obtain the degree and title of

Doctor in Natural Sciences (Dr. rer. nat.)

Accepted dissertation from

M.Sc. Fabíola Gonçalves Coelho Ribeiro

born on 25.09.1985 in Catalão, Goiás (Brazil)

Evaluators:
Prof. Dr. Achim Rettberg
Prof. Dr. Martin Fränzle
Prof. Dr. Michel Soares dos Santos
Prof. Dr. Carlos E. Pereira

Day of Disputation: 03.09.2019

I dedicate this research to Luzia Gonçalves, my mother, and to Maria Fernanda, my daughter.

Acknowledgements

The initial desire for accomplishing this research grew from a family dream. A dream that was not mine, but one that came to sculpture and be the sculptor of the person I would become. In all certainty, looking back over my expectations some four years ago, I never imagined that study and qualification would fuse into my persona in such a dignifying manner. The student, professor, researcher and the very human being of four years ago are no longer the same.

The contributions presented in this thesis are the result of four years of dedication, resilience and love for research. Indeed, it also comes from the help, support, comprehension, dedication and fidelity of the innumerous individuals involved. To my almighty Lord, thank you for giving me health and strength to continue and for supporting me throughout.

I was fortunate to have the privilege of counting on the contribution of three supervisors who supported me at different and specific moments over my research, while delivering excellent tuition in our meetings and lectures; they also provided life based examples. Therefore, I would like to thank in advance the Professors Dr. Achim Rettberg, Dr. Carlos Pereira and Dr. Michel Soares for their immeasurable and constant support. My personal development, without doubt, was greatly incremented through such support and experiences. I profoundly hope that someday, I too may pass onto my students a little of what you were to me.

Prof. Achim Rettberg, you received me in your research group gave me support advised and guided me through my double PhD. Prof. Achim, you are an example of a skilled, qualified and caring person that provides personalized attention to students. Thank you so much for your comprehension, contribution to my research topic, the help provided in achieving the attained results, and especially to your immense dedication while making the final corrections to my thesis. I would like also to express gratitude for the several developmental and learning capacity opportunities that exceed those of a doctoral student's curriculum.

Prof. Carlos Pereira, I still remember well our first personal contact. You have shown me that despite the numerous commitments and responsibilities, you have always been sensitive to the demands of your students. Professor Carlos, you have an incredible subtlety and intelligence in each assessment, evaluation and supervision. Furthermore, it is admirable the manner in which you share your knowledge and experiences. All of these factors were primordial to my personal and professional development. Therefore, I sincerely wish to thank you for all the support and many life enhancing lessons I received.

Prof. Michel Soares, we have worked together since my master studies, from that time until now nine years have passed! Michel, what you have done and still do for your students is much more that provide technical and professional supervision: you have established a teaching based on examples. I will never be able to compensate the time you spent with me. We can recount several weekends working together, tight deadlines and many papers/thesis analyses. Moreover, there were, I recall "tellings-off" and even in situations where I was agitated and felt anxious, you would always tell me: "calm down, everything will be all right!" Thank you Michel for your belief in my ability and always guiding me on to the better path.

I wish also to thank my Family. Each and every member, close or distant, has helped me in distinct ways: encouraged, cared, prayed, wept and laughed with me. I will always be thankful. You cannot imagine how much you are important to me! I would like to give special thanks to Geovanna Gonçalves and Maria Rita Bernardes, my dear sisters, to my dear cousins Gleyce Kelle Bernardes and Américo Gonçalves, to my aunts Amélia Gonçalves, Nívea Rodrigues and Juvenília Gonçalves and to my uncles João Batista Gonçalves and Carlos Gonçalves.

I would like to thank, with all my heart, my friends Dayse de Oliveira, Kênia Santos, Clénia Rios, and Renata de Oliveira. Thank you for always being here for me, for rejoicing in my achievements and encouraging me to do better every day.

Marcos Bhering, you have been "walking by me" for so much longer! You have been benevolent, friendly, kind and supportive at all times over the completion of my thesis. Thank you so much for your comforting words and care, for the many hours dedicated to improving the text and specially, for the love and affection with which you always helped me.

My thanks also go to IF-Goiano for having allowed for my exclusive dedication to Ph.D. research. I am grateful to the institution, my colleagues and the management team for their support and encouragement.

Thanks to CAPES for the financial support during my sandwich Ph.D. studies in Germany.

I would like to thank the Carl von Ossietzky University Oldenburg and the University of Applied Sciences Hamm/Lippstadt for the support provided that aided in the accomplishment of my Ph.D. studies.

Considering the period I lived in Germany, all the enriching experiences I have had and the lessons learned, I would like to express my gratitude to André Faria, Charles Steinmetz, Cíntia Faria, Edenilda Brachtendorf, Filiz Polat, Renata Lutkehaus and Vincent Marnier. Thank you for dedicating your time, care, friendship and words of encouragement and support over the points of this journey.

Finally, I want to thank my mother, Luzia de Fátima Gonçalves, and my daughter, Maria Fernanda Gonçalves. You are the ones who deserve all the honors and are to whom I dedicate all the results and jubilation that come from this research. I know how hard this period was for you both: many absences, relinquishment, days without dialogue and several "rainy days". Thanks for the true friendship, your support and unconditional love. My dearest of friends, you have lived and supported my dream. I could never have done all this without you, I love you infinitely.

"Porque não há nada mais belo que o amor."
(Marcos Jungmann Bhering)

Resumo

Os Sistemas de Tempo Real e Embarcados (STRE) têm se tornado cada vez mais onipresentes nas atividades humanas. O grau de confiabilidade e de corretude com que estes sistemas são desenvolvidos têm um decisivo impacto em sua futura operação. Sendo assim, o sucesso no desenvolvimento destes sistemas está relacionado não apenas com a sua correta execução computacional, mas também com a confiabilidade em que as restrições de tempo real e embarcadas são atendidas. A engenharia de requisitos e o projeto arquitetural constituem importantes atividades do desenvolvimento dos STRE, pois lidam com domínios complexos e diversificados como software, hardware, mecânico, eletrônico e o ambiente físico. Dentro do domínio dos STRE, a engenharia de requisitos e o projeto arquitetural devem atender aos requisitos funcionais e embarcados e aos respectivos requisitos não funcionais. Além disto, o design destes sistemas deve considerar as propriedades de custo, qualidade, confiabilidade e segurança. Este estudo relaciona-se ao desenvolvimento e análise de uma metodologia que abrange diferentes fases de projeto de STRE. A presente pesquisa propõe distintas estratégias para analisar restrições temporais possibilitando avaliá-las em diferentes níveis de abstração, tais como em modelos iniciais de requisitos e a partir da avaliação empírica de anotações dos modelos arquiteturais. Inicialmente, construtores, estereótipos e enumerações do *profile* MARTE são rastreadas para descrever requisitos não-funcionais dos STRE. Com base na semântica e sintaxe do *profile* MARTE, no uso combinado da SysML, do formalismo do *Timed Automata* e das diretrizes propostas na SPES, a metodologia MARTeSys^{ReqD} foi desenvolvida e apresentada nesta tese. A metodologia MARTeSys^{ReqD} emprega conceitos de *Model-Driven Systems Engineering* e propõe orientações para o design de STRE baseando-se em *viewpoints*, níveis de granularidade, anotações e verificações de importantes características de tempo real. Além disto, MARTeSys^{ReqD} define novas estratégias para formalizar as atividades de modelagem arquitetural, para medir a complexidade do *design* dos STRE e para validar restrições temporais desde os modelos iniciais do *viewpoint* de Requisitos. A metodologia desenvolvida é validada de maneira quantitativa e qualitativa de modo a

atestar suas contribuições e expressividade para o desenvolvimento dos SRTE.

Palavras-chave: Sistemas de Tempo Real e Embarcados, Engenharia de Requisitos, Arquitetura de Sistemas, SysML, MARTE, SPES, Restrições não-funcionais, Complexidade do Design.

Abstract

Real-time embedded systems (RTES) are increasingly omnipresent in human activities. The reliability and accuracy with which these systems are developed have a predominant impact when dealing with system operation. Thus, the success in the development of these systems relates not only to the accurate computational execution itself, but also how reliably real-time embedded constraints are developed. Requirements Engineering (RE) and architectural design of RTES are challenging activities, since they deal with complex and diversified domains such as software, hardware, mechanical, electronics, electrical and the physical environment. Requirements specification and architectural design, associated with the RTES domain, must attend to embedded and functional system requirements and their related non-functional concerns, while considering cost, quality, reliability and safety properties. This thesis relates to the development and analysis of a methodology that covers different phases of RTES design. Different strategies for analyzing timing constraints are proposed in such a way that they are evaluated at different abstraction levels, such as early analysis of requirement models and empirical evaluation of architectural models assumptions. Initially, constructors, stereotypes and enumerations of the MARTE profile are traced and linked to specific and non-functional concerns of the RTES domain. Through the collected data and the combined use of the SysML profile, Timed Automata and SPES guidelines the MARTeSys^{ReqD} methodology is proposed. The proposed methodology employs Model-Driven Systems Engineering approaches and presents distinctive guidelines to design RTES based on viewpoints, refinements, granularity levels, annotation and verification of real-time embedded concerns. Furthermore, the proposed methodology provides new strategies to formally describe architectural design decisions, to measure the design complexity of RTES, as well as to validate timing constraints from the early model requirement viewpoint. The methodology proposed in this study is both quantitatively and qualitatively validated, which aims at demonstrating the expressiveness and contributions made toward RTES development.

Keywords: Real-Time Embedded System, Requirement Engineering, System Architecture, SysML, MARTE, SPES, Non-Functional Constraints, Design Complexity.

Abstrakt

Eingebettete Echtzeitsysteme (RTES) sind bei menschlichen Aktivitäten zunehmend allgegenwärtig. Die Zuverlässigkeit und Genauigkeit, mit der diese Systeme entwickelt werden, haben einen überwiegenden Einfluss auf den Systembetrieb. Daher hängt der Erfolg bei der Entwicklung dieser Systeme nicht nur von der genauen Ausführung selbst ab, sondern auch davon, wie zuverlässig die Anforderungen an die Echtzeit mit entwickelt wurden. Anforderungs-Engineering (RE) und Architekturentwürfe von RTES sind herausfordernde Aktivitäten, da sie sich mit komplexen und diversifizierten Bereichen wie Software, Hardware, Mechanik, Elektronik, Elektrik und der physischen Umgebung befassen. Anforderungsspezifikation und Architekturentwürfe, die mit der RTES-Domäne verbunden sind, müssen die eingebetteten und funktionalen Anforderungen des Systems berücksichtigen und die damit verbundenen nicht funktionalen Anforderungen unter Berücksichtigung von Kosten, Qualität, Zuverlässigkeit und Sicherheitseigenschaften. Diese Arbeit befasst sich mit der Entwicklung und Analyse einer Methodik, die verschiedene Phasen der RTES-Entwürfe abdeckt. Verschiedene Strategien zur Analyse von Zeitbeschränkungen werden so vorgeschlagen, dass sie auf verschiedenen Abstraktionsebenen bewertet werden, wie beispielsweise die frühzeitige Analyse von Anforderungsmodellen und die empirische Bewertung von Annahmen zu Architekturmodellen. Zunächst werden Konstruktoren, Stereotypen und Aufzählungen des MARTE-Profils verfolgt und mit spezifischen und nicht funktionalen Anforderungen der RTES-Domäne verknüpft. Durch die gesammelten Daten und die kombinierte Verwendung des SysML-Profils, der Richtlinien für zeitgesteuerte Automaten und SPES wird die MARTeSys^{ReqD}-Methodik vorgeschlagen. Die vorgeschlagene Methodik verwendet modellgetriebene Systemtechnik-Ansätze und enthält eindeutige Richtlinien für den Entwurf von RTES, die auf Gesichtspunkten, Verfeinerungen, Granularitäts-Ebenen, Anmerkungen und Verifizierungen eingebetteter Echtzeitanforderungen basieren. Darüber hinaus bietet die vorgeschlagene Methodik neue Strategien zur formalen Beschreibung von Architekturentwurfsentscheidungen, zur Messung der Entwurfskomplexität von RTES sowie

zur Validierung von Zeitbeschränkungen unter dem Gesichtspunkt der frühen Modellanforderungen. Die in dieser Studie vorgeschlagene Methodik ist sowohl quantitativ als auch qualitativ validiert, um die Aussagefähigkeit und die Beiträge zur RTES-Entwicklung zu demonstrieren.

Schlüsselwörter: Eingebettetes Echtzeitsystem, Anforderungs-Engineering, Systemarchitektur, SysML, MARTE, SPES, nicht funktionale Anforderungen, Entwurfskomplexität.

List of Figures

List of Tables

Acronyms List

AMF Architecture Modeling Framework

AOSD Aspect-Oriented Software Development

AMoDE-RT Aspect-oriented Model Driven Engineering for Real-Time systems

ACS Automotive Control Systems

ADC Automotive Doors Control

BCM Body Control Module

BDD Blocks Definition Diagram

CPU Central Process Unit

DRM Detailed Resouce Modeling

DSMLs Domain-Specific Modelling Languages

DERAF Distributed Embedded Real-time Aspects Framework

DERCS Distributed Embedded Real-time Compact Specification

ECU Electronic Control Unit

GenERTiCA Generation of Embedded Real Time Code based on Aspects

GRM Generic Resource Modeling

GCM Generic Component Model

GQAM Generic Quantitative Analysis Modeling

HRM Hardware Resource Modeling

IBD Internal Block diagram

IL Interaction Level

IS Interface Size

IPS Industrial Packing System

MARTE Modeling and Analysis of Real-Time Embedded Systems

MDSE Model-Driven Systems Engineering

NFP Non-Functional Properties Modeling

NL Natural Language

NL-TA Natural Language to Timed Automata

OO Objected-Oriented

OAC Operation Argument Complexity

PC Product Complexity

PWM Pulse-width Modulation

STRE Sistemas de Tempo Real e Embarcados

RTS Real-Time Systems

RTES Real-Time Embedded Systems

RE Requirements Engineering

RCM Rubus Component Model

RTMS Road Traffic Management Systems

SRM Software Resource Modeling

SysML Systems Modeling Language

SPES Software Platform Embedded Systems

SoCs Systems-on-Chip

TA Timed Automata

TIS Turn Indicator System

UML Unified Modeling Language

UML-RT Unified Modeling Language - Real Time

Contents

I hereby certify that I have obtained all legal permissions from the owner(s) of each third-party copyrighted matter included in my thesis, and that their permissions allow availability such as being deposited in public digital libraries.

Fabíola Gonçalves Coelho Ribeiro

Chapter 1

Introduction

Software systems have been increasingly present in human activities, and many of these have a high level of complexity and automation. These systems are composed of critical, non-functional, temporal, embedded and frequently real-time requirements [3], [4]. The development of these systems is a complex activity, since qualitative and quantitative aspects, such as efficiency, reliability, safety and real-time behavior must be considered [5], [6].

Real-Time Systems (RTS) can be defined as "a computer system whose correctness depends not only on the output, but also the time at which the output is produced" [7], [8]. The term "real-time" is used for systems that react to external inputs given strict time requirements. These systems must analyze in a correct manner, as well as answer their external stimuli in a finite and pre-defined period [9]. RTS are complex systems that involve multiple perspectives of analysis, domain application and increasingly depend on the interaction between various disciplines, such as Mechanics, Electronics and Software Engineering [10], [11], [12]. Moreover, these systems are frequently developed for embedding in physical devices and, consequently, are named as Real-Time Embedded Systems (RTES) [8].

Embedded software holds a substantial relationship with one or more computers/processors, which have an imperative functionality in the system [13]. This kind of system faces several constraints such as non-functional and real-time requirements, resource limitations and hardware dependencies [3]. As of this point, in this thesis, non-functional requirements with a real-time (as detailed on Table 1) specification are simply called "constraint". Generally, RTS "include many embedded and safety-critical systems that are subcomponents of a larger complex system operating in a safety-critical environment" [14]. The class of RTES is composed of electronic, mechanical, electrical, sometimes hydraulic components in which they are encapsulated. Besides, these systems must coordinate in a temporal manner their software, hardware and mechanical elements [15].

The design, development, implementation and maintenance of RTES has always been

considered difficult and challenging [16], [17], [8], [18], [19]. RTES must accomplish their functional requirements, also meet cost, quality, reliability and safety requirements [20]. In this sense, Software Engineering proposes a variety of activities, processes, methods and tools that assist in the analysis, description, development and maintenance of RTES [10], [13], [21].

A variety of standards, formalisms, languages and approaches have been proposed in recent years to assist developers in managing software development activities and help them to deal with the inherent complexity of RTES [22], [23], [13], [24]. Model-Driven Systems Engineering (MDSE) approaches can contribute and facilitate RTES development, since it encompasses design strategies in order to develop, evolve, verify, formalize, configure and maintain embedded, real-time and distributed software [25], [26]. MDSE approaches are based on models that direct system development, perform refinements of different abstraction levels and provide several types of stakeholder interaction [27]. Thus, MDSE approaches must be able to support specification and design of embedded hardware and software components, which includes functional and non-functional requirements, as well as to ensure the correct translation of specifications into executable embedded systems [6].

In RTES development, Software Engineering activities are characterized as being difficult and complex, since since they must address functional and non-functional properties in a correct and consistent manner, while performing analysis and description of distinctive domains [28], [29]. Thus, such activities require, increasingly, high levels of knowledge in order to specify and formalize RTES concerns along architectural design [30]. RE defines a set of well-established practices that are able to describe and detail the properties of an RTES [3]. Requirements viewpoint [18] of RTES aims at ensuring that the complete set of needs, requirements and restrictions of a system will be captured and, posteriorly, transformed into a valid set of requirements for all activities of the software life cycle [31], [26], [21], [32]. Furthermore, the Requirements viewpoint provides the basis for further Functional, Logical and Technical viewpoints of architectural system design [33], [18].

Among the other software development activities, Architectural Design, mapped here in the Functional, Logical and Technical viewpoints, is defined as the set of activities that involves analyses, refinements and development of system requirements at different abstraction levels. The architectural design represents a fundamental activity during the development of the RTES [34], [35], [36], [20], [18]. Architectural design artefacts aims to describe software structure (detailing its components), to capture initial decisions of the project, to present the behavioral overview of the system, to specify system technical artifacts and, also, to contribute to the general reliability of the system [37], [29]. It also allows one to describe software architectural characteristics in which associations/communications between components and connectors are made explicit [38], [34], [35].

Architectural design and the artefacts that are derived from this process are vulnerable

to erroneous choices in the early stages of RTES development [24]. Such choices may impact negatively on the development and deployment phases of RTES [38], [39], [37], [36]. Therefore, it can be inferred that, for most RTES projects, it is of great importance to understand, analyze, specify and validate the artefacts which arise from the design since it can minimize complexity in its description and correct development [40], [39].

The development of RTES must specify structure and behavior of software, along with its physical and logical resources, its infrastructure and temporal constraints [10], [41], [42], [26], [43]. Therefore, using a single modelling language/method may not be sufficiently suitable to cover all the multidisciplinary aspects that describes the RTES domain [44], [45], [46], [47]. The combination of modelling techniques, languages and methodologies can contribute to the full RTES description [48], [49], [5].

In a general sense, RTES engineers tend to use different approaches (formal, graphical models or object-oriented) for modelling, analysis and specification of RTES [50], [51], [47]. Furthermore, the specification language must be robust enough to represent the system requirements at different abstraction levels in order to avoid omissions in the specification and enable elicitation of physical, logical and temporal requirements, as well as other aspects that describe these systems [20]. Thus, adoption of approaches based on perspectives, multiple granularity levels and refinements, together with formal and semi-formal languages, contributes to the system design, development of their components and complex relationships [52], [11], [44], [53], [54].

Several Domain-Specific Modelling Languages (DSMLs) [45] have been proposed in the form of extensions to the Unified Modeling Language (UML) metamodel and are named as profiles [55], [56], [2], [57], [1]. UML is designed to be customizable and extensible and it has been applied to several domains [44], [58], [59]. The UML metamodel includes many semantic variables and provides special constructors in the language for refinement. Such constructs, stereotypes, tagged values, and tags are used to define DSMLs based on UML [60]. As these capture domain-specific concepts, profiles are typically used together with other specific stereotypes of the same domain. Profiles are usually based on only one subset of the UML metamodel (as opposed to the complete metamodel), resulting in simpler and more compact DSMLs, commonly contributing to representation of requirements that are intrinsic to RTES [12].

In this thesis, UML, MARTE and SysML profiles, SPES methodology and Timed Automata language are combined in order to provide an overall methodology to specify and design RTES. The combined adoption of these languages/profiles encompasses functional and non-functional concerns in model elements along the architectural viewpoint design of RTES. The proposed methodology aims at defining a general and appropriate architectural viewpoint definition to design RTES. Thus, it adopts viewpoints and refinements of architectural viewpoints with focus on RTES design. In the Requirements, Functional, Logical and Technical viewpoints, these refinements consider specific diagrams and non-

functional annotations from the early steps of design, while fortifying the description of important aspects concerning these systems.

The Software Platform Embedded Systems (SPES) methodology provides a framework toward RTES development, while providing guidelines to software and system design. The adoption of SPES viewpoints and their perspectives contributes to minimizing the complexity of RTES development, as it performs separation of non-functional concerns and functional system services [33]. The proposed decomposition of system services can favor one's understanding and development of such services [61], [62]. Modeling and Analysis of Real-Time Embedded Systems (MARTE) extend the UML with constructors to analyze, modelling and design RTES [2]. Systems Modeling Language (SysML) is also a UML extension that supports system modelling allowing either high or low-level description of RTES [1]. MARTE constructors can be annotated directly in SysML models without any extension as MARTE is a UML profile [59]. Therefore, SysML and MARTE can be combined in MDSE approaches to express RTES properties.

In this thesis, MARTE profile and SPES methodology are employed in modelling strategies, from SysML and Timed Automata (TA) [63], in order to specify, design and validate the requirements, services and systems components. The combination of these strategies culminates in the development of the MARTeSys^{ReqD} methodology. The MARTeSys^{ReqD} methodology depicts how to design structured and dynamic models of RTES, while highlighting hardware and software requirements of these systems. The proposed methodology, which incorporates described models and guidelines to design RTES considering distinctive and complementary perspectives, is able to represent RTES characteristics such as functional and non-functional features.

The MARTeSys^{ReqD} methodology also proposes a formal definition of RTES design activities and viewpoints. The proposed formalization allows for different measurements regarding complexity of system design, from the initial design activities, as well as the definition of a formalized manner in order to analyze RTES complexity without interference of user external knowledge. In addition, it presents two novel strategies for managing RTES timing constraints: (1º) it traces RTES constraints along the architectural design to the system realization so, that it can be formally analyzed, (2º) it provides a formal grammar to transform timing specifications, written in natural language, to timed-automata. This last aspect allows for the verification of timing constraints such as period and deadlines from requirements specification documents in an unambiguous manner.

The MARTeSys^{ReqD} methodology has been applied in different case studies such as Road Traffic Management Systems (RTMS), Automotive Control Systems (ACS), Automotive Doors Control (ADC), Industrial Packing System (IPS) and Turn Indicator System (TIS)). In this thesis, the full example of MARTeSys^{ReqD} methodology is applied in particular to the TIS in order to show the use of the proposed methodology, guidelines for the design of complex system features, and to direct RTES development.

1.1 Motivation

The development of RTES must cover different development phases, such as requirements specification and architectural design [64]. Other phases, such as source code development, testing and integration, are direct consequences of modelling activities [13]. RTES operate in domains in which UML extensions can be used to provide greater expressiveness, such as mutual exclusion mechanisms, temporal features, concurrency, specification of deadlines, among others [59], [65], [47]. Moreover, formal and semi-formal languages, combined with MDSE principles, can contribute to RTES specification and design [66].

In this thesis, the author adopts MDSE concepts to support development of RTES considering analysis, modelling, evaluation and validation of timing critical constraints. The thesis scope can be subdivided into the following motivations:

1. **Development of strategies to cover the design of RTES**. This means, specifically, that the thesis concentrates on formal and semi-formal methods to model RTES along the requirements and design activities.

2. **Enabling of annotation of RTES concerns at different abstraction levels**. This thesis culminates in the MARTeSys^{ReqD} methodology and it presents a wide analysis and guidelines to represent real-time and embedded concerns into architectural models.

3. **Analysis, verification and validation of timing concerns from multiple architectural models**. MARTeSys^{ReqD} methodology presents guidelines to annotate models describing non-functional concerns of RTES design. Furthermore, it provides a strategy to analysis, validate and verify specific timing concerns along of RTES development.

RTES are complex, composed by functional and non-functional properties and their development must accomplish critical concerns [67]. However, this thesis is restricted to the analysis, validation and verification of designed timing concerns based on three reasons.

First, time is an essential variable to RTES [68]. Representing time is a crucial activity in RTES modelling and the mapping of temporal characteristics is of fundamental importance to RTES design activities [16], [69]. Thus, the development of RTES with a high correctness level and reliability depends on the correct specification and design of system functionalities [64], which includes analysis and specification of their time constraints [11], [15]. Therefore, design of these systems requires a complete and effective support in order to define and express temporal parameters that are related to the application under development [49].

Secondly, it is important to trace and validate timing constraints from the early design steps of RTES development. The correct description and evaluation of these complex data along the RTES design are directly related to their reliability, safety and quality [22]. It is possible to create formal and semi-formal strategies to provide guarantees regarding the time bounds of the architectural models. Here, these constraints are attached in the models, at different abstraction levels, and can be verified and validated by model-checking and simulation strategies.

Finally, the investigation of MDSE strategies to RTES design can benefit their development [70]. The RTES projects involve multiple domain-specific languages to cover different characteristics of these systems [47]. Given the heterogeneous and complex nature of these systems, and the several different categories of stakeholders with different concerns and interests, it is necessary to use multiple languages [71], [12]. These languages should have compatible models in order to represent the various aspects of a system [44]. It is expected that the chosen design strategies for the MARTeSys^{ReqD} methodology provide expressiveness of RTES design, while expressing the models of Requirements, Functional, Logical and Technical viewpoints.

This thesis has been motivated by the need to manage the complexity of RTES development, while focusing on the design of critical properties and their analyses. This is performed by the development of specific criteria for specification, modelling and the architectural design of these systems.

1.2 Research Methodology

As emphasized in [72], a method is a set of organizing principles around which empirical data is collected and analyzed. A variety of methods can be applied to any research problem and, oftentimes, it becomes necessary to use a combination of methods to fully understand the problem and to conduct relevant and non-refutable issues in relation to the research criteria. Figure 1 depicts the Activity diagram followed by the adopted research methodology. This research method encloses the overall research methods and techniques to develop and evaluate the proposed study of this thesis.

Research methods are applied in the context of a scientific research study. Therefore, they relate to goals, objectives, bibliographic reviews and validations [73], [74]. Research strategies, in Software Engineering, are applied in a specific context/strategy of investigation and seek to provide new knowledge that is expressed in the form of theoretical and practical results [75]. In the top part of Figure 1, the research methods that provide the main insights to the proposed study are presented. Regarding the data collection technique, this thesis applies mixed approaches and methods, with greater detail provided in [72] and [76]. As it can be noticed in Figure 1, the framework to develop MARTeSys^{ReqD} is based on data collection techniques, from field research (for example case studies), in-

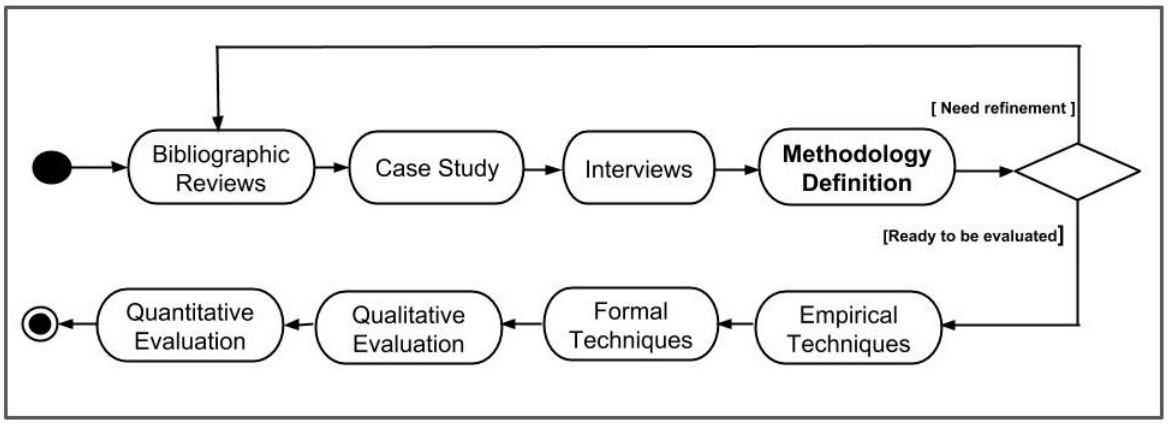

Figure 1 – Research Methodology Applied in the MARTeSys^{ReqD} Methodology.

terviews and bibliographic reviews. These techniques help to define the scope, guidelines and design decisions of the proposed methodology.

Results of a research study can be expressed as procedures or techniques, as qualitative, descriptive, empirical or analytic models, also as tools and notations, as specific solutions, as evaluations/comparison, or, yet still, as reports with observations or rules. In the bottom part of Figure 1, the methods to perform the thesis evaluations are shown. Empirical Techniques, Formal Techniques, Qualitative and Quantitative Evaluations are considered to check the expressiveness, significance and adequacy level of the research results to RTES development. In this thesis, simulation, testing, measurements (in specific case studies) and formal validations of the design artifacts are considered in order to analyze the proposed methodology.

Qualitative and Quantitative measurements perform significant evaluation of the proposed strategy. The qualitative approach, used in this research, aims to construct a theoretical framework that emerges from the analysis of collected data during research and that explains the research results in a coherent manner [77]. The **Quantitative Evaluation** focus on assumptions evaluation and aims to measure their reliability when faced with design decisions. The quantitative research is based on the measurement of objective variables, on the comparison of results and/or on the intensive use of statistical techniques. As stated previously, the numerical values from the simulation activities aims to check if there is consistency between the architectural constrained models and the dynamics results.

Qualitative and quantitative techniques are applied in order to evaluate the results of this thesis regarding the artefacts generated from the application of the proposed methodology in an industrial case study. The case study technique stands out among the main tools for exploratory studies, and in this thesis, it will be used to support the research proposal, to understand the scope of the problem and to answer the research questions. Case studies are appropriate for various research objectives and they can be

used in contexts of description (with the objective of answering questions of type "what", "where" and "how"), of explaning questions (with the objective of clarifying questions of the type "why"), of predictions (it includes state predictions of the short-term/long-term, behaviors, events) and of control process (it attempts to influence attitudes, cognitions and behaviors for a particular case) [78], [79], [80].

An analysis of usability and expressiveness, of the proposed methodology, will be performed regarding an industrial case study. For this, a team that is constituted of professionals, researchers and students who work and do research on RTES, are invited to evaluate the proposed methodology. In this way, as detailed in Chapter 7, interviews and questionnaires are applied to analyze the MARTeSys^{ReqD} methodology and its contributions to requirements specification and architectural design of RTES.

Each subject receives in advance a manual and basic guidelines about the MARTeSys^{ReqD} methodology. Posteriorly, the experts analyze the artefacts coming from the application of the proposed methodology, in an industrial case study. Then, interviews and questionnaires are performed. Thus, the experts may qualify and quantify their experiences. The proposed questionnaires are based on known standards of the RE process and Architectural Design such as ISO/IEC/IEEE 29148:2011 [81], ISO/IEC/IEEE 42010:2011 [37] and ISO/IEC/IEEE 29148:2011 [81]. Finally, different qualitative analyses are performed, by the author, from the collected data regarding the answers from the expert.

Regarding the data collection technique, as noted in Figure 1, the background to develop MARTeSys^{ReqD} is based on data collection techniques through research in the field, specification techniques and RTES design. Therefore, the research proposed in this thesis uses mixed approaches and methods, which is given greater detail in [72], such as bibliographic analyses, literature review and case studies.

1.2.1 Goals and Scope Delimitation

Different approaches to the Requirements Specification process have been proposed. Many of these approaches focus on the specification, analysis, elicitation and requirements management processes of RTES. In [82], [83] and [84] extensions to the SysML Requirements diagram were presented in order to allow for definition of a modelling methodology with relevant criteria for documentation, requirements analysis, classification, traceability and design of systems at different abstraction levels.

The author's publications, which were mentioned previously, had their focus on extensions of attributes and relationships of the SysML Requirements diagram. These extensions were applied fundamentally to activities of requirements specifications. Furthermore, it did not focus on RTES constraints annotation and traceability between artefacts from different development stages. Following the studies of [82], [83], [84], [85],

and with the goal of portraying the gaps not covered in these works or in findings from the literature, this section provides details of the principle and specific objectives.

The main objective of this thesis is to provide a methodology for RTES design that covers functional and non-functional properties, while allowing timing analyses at different abstraction levels. With this in mind, it is initially intended to use the MARTE profile in combination with SysML diagrams and SPES guidelines in order to develop a methodology that can be applied to the structural and dynamic design of RTES. Furthermore, this methodology provides a manner to represent the characteristics that are inherent to RTES at separate abstraction levels. The overall design phases proposed in this thesis have been formalized to allow its application for different RTES examples. Hence, it is estimated to significantly promote the specification, description, validation and verification of RTES concerns for physical and logical system components. The specific objectives of this thesis are:

- ❏ To develop a methodology, based on viewpoint refinements and different abstraction levels, to the Requirements Specification and Architectural Design of RTES;
- ❏ To trace stereotypes and constructors of MARTE to express specific RTES concerns;
- ❏ To apply VSL to standardize RTES descriptions within design artefacts;
- ❏ To propose, in the developed methodology, a strategy to trace timing constraints between artefacts coming from the different design viewpoints and along the RTES development;
- ❏ To qualitatively evaluate the combined adoption of the SysML diagrams and MARTE stereotypes, together with SPES, to describe Requirements, Functional, Logical and Technical Viewpoints;
- ❏ To propose a formalization of the proposed methodology, through the specification of the algorithms, and its design activities;
- ❏ To perform complexity analyses related to the adopting of the proposed methodology;
- ❏ To provide a strategy to predict the global system complexity when it adopts the proposed methodology;
- ❏ To create a grammar to transform timing constraints descriptions, written in natural language, from Requirements viewpoints models, to formal syntax representation;
- ❏ To formally validate timing constraints from the early design phases and over the architectural design; To formally validate timing constraints since early design phases and over architectural design;

- ❑ To apply the proposed methodology for specification and design of RTES in order to qualitatively and quantitatively validate the proposed methodology.

The MARTE profile provides a wide and robust framework of definitions, constructors and formalisms to define, describe and analyze RTES [59]. However, this profile still lacks the dissemination and consolidation of its model [86], as well as applications that present, qualitatively and quantitatively, its suitability, expressiveness and usability in modelling/specification processes [87], [88], [89], [90], [91].

In light of these aforementioned challenges, this thesis aims to show how MARTE stereotypes and constructors can be effectively applied to detail essential concerns of RTES. The MARTeSys^{ReqD} methodology also provides the trace between RTES concerns and specific constructors from MARTE (see Chapter 4). In addition, this thesis provides several guidelines to combine UML profiles to design architectural models of RTES. Finally, this thesis defines how to link the MARTE stereotypes in different RTES views, at each viewpoint, of the system. It contributes to meaningful refinements of the system design and their RTES concerns, to favor the reproducibility of design decisions and to develop architectural design. Thus, non-functional concerns can be analyzed, as of the early design phases, through the development of RTES, until the system realization.

Considering the prior discussion, one can narrow the scope of this thesis as follows: (**1**°) To provide a global reasoning for challenges and different components that may be evolved in the development of RTES. It directs the author to propose a framework with concepts and guidelines to design the intrinsic properties to these systems while they may contribute to decreasing the complexity in their design and development. (**2**°) To highlight how MARTE and SysML profiles, SPES methodology and TA can be combined in Requirements, Functional, Logical and Technical viewpoints of the proposed methodology to handle, during early design descriptions, several functional and non-functional properties of RTES. (**3**°) To formalize design activities to provide a standard manner to adopt a MDSE approach. It allows the application of the methodology, its guidelines and concepts for different developers and in distinctive domains. Moreover, the proposed formalization contributes to quantitative measurements and a new complexity analysis of design activities. (**4**°) To perform the management of RTES constraints from distinctive perspectives in order to favor to their seamless description and analysis. It means that this thesis aims to annotate, design, trace and realize formal and simulation analysis of these constraints in order to accomplish their meaning.

1.2.2 Assumptions and Research Questions

This work is based on the assumption that "The combined use of MARTE stereotypes, structural and behavioral diagrams of SysML together with TA formalism contribute to RTES design, favoring the analysis and comprehension of its inherent properties and,

therefore, its development". To validate this assumption, the following research questions are proposed:

Q1: What is an expressive manner to combine SysML diagrams with stereotyped markings of the MARTE profile, handling non-functional constraints, in order to perform requirements specification and architectural design of RTES?

This question is answered in Chapter 4 through the definition of MARTeSys^{ReqD} methodology. Together with the definition of the proposed methodology, it provides a group of well-formed strategies for high and low level modelling, viewpoints refinement, documentation and specification of requirements and architectural artefacts. Furthermore, Chapter 4 presents theoretical foundations regarding timing constraints annotations through the MARTE profile packages. The analyses carried out in this research question provide a methodology able to handle timing constraints from Requirements viewpoint artefacts, by transformation rules and TA models, and from Model and Unit Design artefacts.

Q2: How to formalize the design activities within an RTES methodology?

The answer to this research question is provided in Chapter 5. This formalization is performed in pseudo-code and shows formal instructions for applying the MARTeSys^{ReqD} methodology in order to develop RTES. In general, this question is important for establishing a manner in which the proposed methodology can be adopted, while supporting the development of system architectural models. It also provides quantitative criteria to measure and standardize a MDSE methodology. Based on this, distinctive measurements can be performed in order to quantitatively analyze the complexity, regarding to time and number of components, of the proposed methodology.

Q3: How to effectively apply the proposed methodology to the Requirements Specification and Architectural design?

The third research question is discussed in Chapter 6, which depicts the application of the MARTeSys^{ReqD} methodology, its design activities and specified viewpoints in an industrial case study. In addition, Chapter 6 shows a concrete specification and design of RTES constraints along with Requirements, Functional, Logical and Technical viewpoints.

Q4: How to provide traceability patterns between artefacts of the Requirements Specification and Architectural design?

Chapter 7 performs the evaluation of the proposed methodology while detailing a concrete strategy to trace RTEs constraints along architectural viewpoints. The

proposed strategy traces constraints, from different abstraction levels, in architectural models in order to follow their evolution and refinements. Moreover, it performs automatic translations of annotated constraints, from Model and Unit Design model, to implemented code. The main goal of this topic is to propose a strategy to comprehend, express and trace RTES constraints along of requirements specifications and architectural design artefacts. Moreover, it may facilitate the processes of development and management of changes in these systems.

1.3 Contributions of the Thesis

The research study proposed in this thesis intends to contribute, through the spread of its results to the academic community, knowledge and the adoption of semantics and syntax of the MARTE profile and of its real-time embedded constructors in the RTES development processes. Guidelines to RTES requirements specification and architectural design are proposed. Further, it provides in the same methodology ways for integrating software and system design.

The proposed methodology is named as MARTeSys^{ReqD} and combines MDSE approaches, MARTE profile, SysML diagrams, SPES guidelines, TA concepts into a single methodology. The MARTeSys^{ReqD} methodology contributes to manage the complexity of RTES development, while supporting their viewpoints refinement, granularity levels definition, annotation and verification of real-time embedded constraints.

Another important contribution of this thesis is to formally analyze requirements using Requirements viewpoint models. The MARTeSys^{ReqD} methodology produces different formal rules to represent timing constraints from natural language to formal specification. Therefore, it defines a strategy to formally analyze and verify timing constraints while proving the correctness of these constraints. A formal grammar is developed in order to generate TA models from the specification in natural language. This can contribute toward minimizing the ambiguity of requirements views and specifications. Hence, it allows the application of model checking strategies at the early design levels.

The MARTeSys^{ReqD} is a well-described methodology for RTES development. It provides a group of formalized guidelines to be applied into the design phases and, it offers a complete set of constructors and combined diagrams to structure RTES artefacts. This strategy provides a manner to trace the system constraints from different views and viewpoints, as well as to trace back these constraints, from a higher to a lower abstraction level. Therefore, it may contribute to the analysis, modelling and comprehension of the interdependencies between design elements.

Software and system components are treated as integrated components. Software in RTES performs specific functions within the system. In this way, it is possible to link these domains, into a single methodology, to favor their design. MARTeSys^{ReqD} performs

it by adopting specific and integrated diagrams to model these domains and tracing artefacts and their constrained values along software and system architectural models. Thus, the proposed methodology addresses software models and system qualities such as real-time constraints and also provides combined strategies to verify and validate these constraints.

1.4 Organization of the Thesis

This thesis is structured through 8 chapters, and organized as follows:

Chapter 2, **Theoretical Foundation**, contemplates RTES theoretical concepts, describes MDSE and the specific needs of RTES design. Moreover, this chapter performs an introduction of the UML profiles adopted in this thesis and their contribution to RTES requirements specification and architectural design.

Chapter 3, entitled **State of the Art Analysis**, provides an analysis of correlated scientific studies from the last five years regarding the scope of this thesis. This chapter aims to analyze, and contextualize the research performed and the contributions of this thesis. In addition, Chapter 3 provides the literature review, regarding the MDSE to RTES development and the formalization of design steps, and it is based on the author's papers [92] and [93].

Chapter 4, entitled **The MARTeSys^{ReqD} Methodology**, presents a methodology to specify, design and architect RTES. This chapter provides a complete explanation of the four viewpoints of MARTeSys^{ReqD} methodology, its refinements and adequate SysML diagrams, along with a formal grammar to design requirements and their timing constraints through TA. The contributions of this Chapter are mainly presented in [94], [95], [96] and [97].

Chapter 5, entitled **The Formalization of MARTeSys^{ReqD}**, follows the contributions of this thesis and defines a strategy in order to formalize all the design decisions for the MARTeSys^{ReqD} methodology. The formalization proposed in this Chapter is based on papers [98] and [99].

Chapter 6, entitled **Adoption of the MARTeSys^{ReqD} Methodology**, MARTeSys^{ReqD} depicts an application of the proposed methodology in a case study. The chosen case studies are not necessarily defined to combine with the MARTeSys^{ReqD} methodology specification or to fit exactly with it. It means that the designed methodology is independent of any application domain. Thus, their design processes can be adopted to develop RTES. This Chapter is based on the author's papers [100] and [101].

Chapter 7, entitled **Evaluation of the MARTeSys^{ReqD} Methodology**, presents a qualitative evaluation of the MARTeSys^{ReqD} methodology regarding the review and assessment of experts of the RTES domain. Moreover, this chapter provides the trace and analysis of timing constraints from architectural to the implementation models. Thus,

it develops a strategy to simulate these constraints and verify design assumptions, from dynamic executions, while quantifying these. This Chapter is based on papers [102], [103] and [104].

Chapter 8, **Conclusion**, summarizes the main contributions of this thesis and highlights the scientific resulting publications. This chapter also lists future extensions that may be considered regarding the scope of this thesis.

Chapter 2

Theoretical Foundation

This chapter provides an overview of the theoretical concepts that are related to and deliver the necessary background to the proposed study. It aims at describing the RTES and their characteristics, while introducing concepts, methodologies and principles that contribute to RTES development.

2.1 Real-time Embedded Systems and their Properties

The term RTES is usually related to those systems with strict non-functional constraints [53]. These constraints are related to timing, security and/or performance concerns and these can be imposed on the overall system services/functions or their subsystems [105].

The class of RTS can be characterized by time-aware, reactive (time-triggered or event-triggered) and/or concurrent controllers. It means they must readily react to each environmental stimuli, while also operating in parallel [13]. Therefore, in RTS design, the acceptance of a system requirement must consider timing satisfaction, in a concurrent environment, in which asynchronous events can be triggered at any time [11], [14], [106].

RTS can be classified in three primary classes [16], [22]: "*Hard*" real-time systems, "*Soft*" real-time systems and "*Firm*" real-time systems. *Soft* real-time systems are those in which the response time is important, but they still operate well enough even if some temporal restrictions are, occasionally, lost. Thus, missing of deadlines may diminish system performance, but it will not result in significant losses due to failure to meet response time constraints [14]. As examples of *Soft* real-time systems, multimedia applications, interactive or process control systems are put forward as examples [22].

Hard real-time systems are classified by their strict observance and attendance to constrained deadlines [7]. Therefore, failures to meet timing constraints can lead to intolerable degradation of system and, in some cases, result in catastrophic losses and other

undesirable consequences. *Hard* real-time systems are all those of which it can be affirmed that "although functionally correct, the results produced after a certain predefined deadline are incorrect and, thus, useless" [15]. According to [14], *Hard* real-time systems must always react when requested, contrary to *Soft* real-time systems, which eventually, may not fulfill necessary time needs. Aircraft, nuclear power plants and traffic controls, among others, can be cited as examples of hard RTS [22], [17], [69].

According to [13], *Firm* real-time systems can behave as *Hard* or *Soft* RTS. As in a *Soft* real-time system, small delays do not lead to total failure of the system, however, huge delays can result in global and dangerous failures. As examples of these systems, ATM and online transaction controls are noted.

In general, embedded systems are characterized as computational systems, with a specific purpose, able to control and support the operation of technical systems (also named as embedded systems). These systems are composed of mechanical components for which specific and pre-defined tasks are controlled through an encapsulated system. As well as RTS, embedded systems must operate in accordance with their temporal constraints and multi-task activities respecting different scheduling, synchronization and resource management policies [15].

An RTS is named RTES if developed to be embedded within some larger system [53]. In the RTES domain, software is the logical component, it is usually embedded, and it is developed under performance, security and reliability criteria. RTES covers a wide range of systems such as automobile, industrial telecommunications, air traffic, rail and traffic controllers [107]. Other examples are medical, nuclear and process control systems [10], [108]. RTES encompass different embedded and real-time properties/requirements and, as RTS, the temporal behavior of their physical and logical subsystems is as important as their functional behavior [52]. These systems define, in addition to their functional properties, control of several peripheral components, of their constraints, communication interfaces and temporal and non-functional requirements [109].

According to the presented definitions, one can state that the predictability of temporal behavior is a very important property of RTES [15]. An RTES must describe and prove, at design conception, that in all cases all requests will be served within predefined within predefined [110]. In this case, the temporal behavior of the system and subsystem (*Hard*, *Soft* and *Firm*) must be fully represented in their development.

The development of RTES consider a complete and effective support of their functional and non-functional concerns. As presented in standard ISO/IEC/IEEE 42010:2011 [37], a concern can relate to the needs, objectives, goals, restrictions of requirements specification, design constraints, system attributes, architectural decisions, quality features and other questions which can influence the functional and non-functional behavior of the system. An analysis of these concerns is important to clarify their design and description along the RTES development. The following sections provide an overview regarding

concerns found with RTES and their design along architectural descriptions.

2.2 Real-Time Embedded Constraints in Architectural Models

High quality in RTES development depends on its correct requirements specification, which includes analyses and representation of its different constraints [16]. In addition, it is necessary to provide appropriate support for embedded devices when representing their characteristics, communication interfaces (often distributed), energy management, operation protocols and offered services. Therefore, RTES requirements specification demands a complete and effective support of RTES constraints in order to define, express and validate non-functional and timing constraints, which are related to the application under development [68].

RTES constraints describe a particular real-time behavior/condition of the external system or its interaction [111]. Moreover, these constraints can restrict reactive controllers of RTES actions/events. In the last case, these constraints can detail response time, from the system to the environment or to any other component, as for example, execution time and system deadlines.

Development activities of RTES must consider different constraints/concerns of these systems as, for example, desired performance and reliability under different viewpoints [31], [2], [26], [21], [32]. RTES must deal with correct interaction between embedded components, along with critical and complex real-time components. According to many authors [112], [113], [8], [51], non-functional requirements of an RTES, and their concerns, may be contained within different categories. Table 14 depicts non-functional concerns and their subtypes.

The consideration given to these concerns/constraints and their correct description and design contributes to a trustworthy definition of the system under development. Understanding these concerns and how they can be detailed can favor activities for the specification, analysis and architectural design of RTES.

2.3 Model-Driven Systems Engineering

Model-Driven System Engineering (MDSE) based on models, transformations, such as Model-to-Text and Model-to-Model, and modelling tools to support the management of systems [114]. MDSE intends to increase the level of abstraction, through the use of models and DSMLs, in system development activities [115].

MDSE activities, such as specification, modelling architectural design and system evolution, start at the beginning of the software development process and continue through-

Non-Functional Requirement	Non-Functional Requirement Subtypes
Reliability	Accuracy, Maturity, Fault Tolerance, Recoverability, Compliance, Threshold.
Time	**Timing**: deadline, period, cost, release time, activation latency, start time, end time, event duration, clock, instants. **Precision**: jitter, tolerated delay, freshness, resolution, drift.
Performance	Accuracy, Response Time, Operation Capacity, Throughput, Recovery Time.
Safety	Tolerance to Failures, Risk Level, Risk Prevention, Access Levels, Redundancy, Integrity, Privacy, Minimum Metric Access Level, Maximum Metric Access Level.
Distribution	Tasks Allocation, Hosts Definition, Communication, Concurrency, Distributed Log, Parallelism, Synchronous Process.
Interoperability	Bus Structure, Inter-Process Communication, Communication Protocols.
Embedded	Layout of Devices (area, size, amount of hardware), Input Interruption Control, Outputs Interruption Control, Energy consumption, Communication Channels, Memory Size, Memory Organization, Power Consumption, Heat Control, Bandwidth.
Resource Utilization	Deadlock, Policies of Access, Resource Features.
Security	Subsystem Integration, Privacy, Control Access, Distributed Log, Integrity, Privacy.
Concurrency	Correctness, Performance, Parallelism, Robustness, Control Access, Synchronization.
Flexibility	Transportability, Interchangeability, Expansion, Capability, Contraction, Effectiveness.
Maintainability	Modifiability, Evolvability, Modularity, Mean Time to Repair, Maximum Time to Repair, Mean Time between Maintenance Actions, Maintenance Time, Maintenance Staff Hours, Skill Levels of Staff, Frequency of maintenance, Complexity Level, Evolvability, Repairability.
Usability	Capabilities of Usage, Measures of Performance, Measure of Safety, Measures of Reliability, Measures of Availability.
Inter-process Communication	Effectiveness, Policies of Access, Policies of Control.
Deadlock	Policies of Access, Correctness of Outputs, Deadline Satisfaction.

Table 1 – Real-Time Embedded Concerns.

out the software life-cycle. MDSE principles favor the RTES, since these allow it "to cope with the complexity of software development by raising the abstraction level and introducing more automation into the process" [116]. Adoption of MDSE in RTES development can contribute to reducing risks in the design, toward control costs and chronogram achievements from the early phases of the software lifecycle [117]. Moreover, MDSE may increase the level of productivity during software development, while also improving the impact analysis of changes to the requirement processes and design [65].

MDSE is practiced widely [115] and it has been adopted by distinct areas of RTES industries such as automotive, banking, printing and so on. As described in [27], the adoption of MDSE principles allows for a concise software documentation and a structured software architecture. These principles can influence productivity gain due to automatic code generation support [65]. However, even though MDA is conceptually simple to

understand, it is complex to implement into RTES development [118]. The successful adoption of MDSE may relate to the knowledge of DSMLs, through the compression on model-centric approaches and adoption of coherent tools [119].

The MDSE approach differs from traditional approaches of software development, which often follow a pure or extensions of Waterfall model [120]. "Traditional models are treated as reference artefacts for subsequent design and coding whereas in MDSE, models are treated as code" [118]. MDSE approaches can be employed under different perspectives and interests in RTES design [70]. As stated in [121], approaches that consider MDSE guidelines should provide three primary goals: portability, interoperability and reusability through the architectural separation of concerns.

2.4 Specification and Design of Real-Time Embedded Systems

RTES design is not a trivial process [13], [114]. Development of RTES must consider the system specifications and the interactions between software, hardware and mechanical components. Software plays an important role in RTES conception, as it has become more intensive and complex [122].

Modelling is fundamental for designing RTES since it allows for a detailed application structure, behavior, and requirements within particular domains [34], [123]. Several approaches have been proposed over recent years to the goal of RTES development. Basically, these approaches are classified as graphical representations, textual representations or the combination of both [39], [124], [120]. RTES development embraces different views and representations of the system, and it can apply formal and/or semi-formal methods to realize the requirements specification and the architectural design.

There is a consensus that these approaches should consider the system design, from requirements and along architectural description, and provide correct and consistent artifacts to direct the implementation of the system [18]. In accordance with [125], [126], [18], regardless of the approach or methodology, RTES modelling must incorporate the following practices:

- ❏ To design the system features, along with Requirements specification and architectural design, under dynamic and structural views;
- ❏ To provide relationships, such as refinements, of models at different abstraction levels;
- ❏ To refine and map concepts in different views;

- ❏ To adopt the principle of separation of concerns regarding each view and its purposes;
- ❏ To use consistent design methods and techniques throughout the specification, for example, top-down decomposition, structured approaches, formal approaches and/or Object-Oriented approaches;
- ❏ To use consistent abstraction levels within models and in conformity between refinement levels of the models;
- ❏ To model and check non-functional concerns, as well as real-time properties, across different granularity levels;
- ❏ To omit hardware and software attributions in early requirements specifications;
- ❏ To represent the RTES context and the interactions between them.

UML and domain-specific modelling languages have been applied to RTES development [127], [128], [129].These languages allows for the customization of specific domain needs and are refined through distinctive diagrams, constructors and stereotypes. UML and its profiles are employed in the proposed study and due to their importance; they are detailed in the sections that follow. Furthermore, the proposed methodology combines concepts of TA formalism to prior requirements analyses. Therefore, brief introduction to TA is also described next.

2.5 Characterization of UML

UML [124] is a standardized general-purpose modeling language for visualizing, specifying, constructing, and documenting software artifacts [130]. The word *unified* indicates that the language can be used in software systems as well as in a large number of software development domains, including business processes and software products [131]. UML provides a manner to design different views of one system through its diagrams. These views can be described by different UML diagrams such as Use Case, Classes, Objects, Sequence, Communication, States and so on [57].

UML, despite the existence of several models for modeling RTES, cannot completely deal with specification of this software by itself. There is a lack of semantic and syntactic models to properly specify, design and validate non-functional properties/constraints of RTES [132], [133], [134]. According to [135], UML represents the system structure and its behavior in several abstraction levels. However, time is rarely part of behavioral modelling, since it has essentially undetermined duration. Either in dynamic evaluation mechanisms or in structural aspects of the project, time is not modelled.

UML was designed to be customizable, as a potential family of languages [57]. Its definition includes several semantic variables and provides special constructors, in the language, for refinement. Such constructors, stereotypes, labeled values and tags are used to define DSMLs based on UML. This fact contributes to a conception of UML profiles, which contains a set of related stereotypes. Modern RTES have certain characteristics that demand new approaches for their specification, design and implementation. The modelling of such systems requires a set of heterogeneous notations, which reflect: continuous time, concurrency, control flow, embedded requirements and discrete and/or reactive events. In this context, several proposals for dealing with UML problems regarding real-time software modelling were created. Several profiles that extend UML and add elements that model requirements of time, system and non-functional properties were created. The profiles SPT [55], SysML [1], QoS [56] and MARTE [2] can be mentioned.

SysML and MARTE profiles, adopted in this thesis, are detailed throughout next sections in order to elucidate its fundamental concepts. The proposed study combines the UML diagrams, such the Class diagram, and UML profiles in order to develop a methodology to requirements specification and architectural design of RTES. Therefore, in the following sections, some of the adopted diagrams/profiles are summarized.

2.6 Characterization of Timed Automata

A Timed Automata is composed of a finite automaton-based structure extended with a finite set of clocks [63]. The behavior of the system is defined by the overall automata locations, the clock values and the values of discrete variables [136]. The group of automata clocks increases in the same temporal quantitative order [137]. These clocks can be used to represent timing constraints, and they are subject to reset and comparison operations. According to [136], a TA is defined by a tuple (L, l_0, C, A, E, I), where:

- L is a set of locations,
- $l_0 \in L$ is the initial location,
- C is the set of clocks,
- $B(C)$ is the set of conjunctions over simple conditions of the form $x \diamond c$ or $x - y \diamond c$, where x, $y \in C$, $c \in \mathbb{N}$ and $\diamond \in <, \leq, >, \geq$,
- A is a set of actions, co-actions and the internal τ-action,
- $E \subseteq L\ X\ A\ X\ B(C)\ X\ 2^C\ X\ L$ is a set of edges, called transitions, between locations with an action, a guard and a set of clocks to be reset,
- $I : L \rightarrow B(C)$ assigns invariants to locations.

Constraints on the clock variables, that is, guards on the edges, allow for the restricting of automaton behavior [137]. Non-negative $\mathbb{N}$, $\mathbb{Q}$ or $\mathbb{R}$ numbers can represent the clock domain. A transition represented by an edge may be taken if the different clocks values satisfy the guard labeled on the edge and/or the invariant is satisfied. Invariants can be applied to define constraints of the automata locations. These invariants restrict the possible values of the clocks for being in the state, which can then enforce a transition to be taken. Details concerning the semantics of the labelled transition system and the semantics of a network of TA can be found in [63], [137].

TA models have been used to model and analyze temporal system behavior and non-functional properties of RTES [137]. TA formalism allows to verify important aspects of RTES, such as qualitative and qualitative requirements. Therefore, liveness, deadlocks and non-determinism properties can be qualitatively analyzed, as these allow for verification of quantitatively periodicity, deadlines and timing delays of the system [63]. The UPPAAL tool [138], which will be given detail later, can be adopted to automatically verify the mentioned properties.

2.7 Characterization of SysML

SysML profile supports the specification, analysis, design, verification, and validation of complex systems [1]. SysML is a UML profile and reuses some of its constructions. The basic difference is that SysML was proposed to support systems engineering, which means that some specific constructions from UML, not necessarily for system modelling, were avoided. SysML is effective "in specifying requirements, structure, behavior, allocations, and constraints on system properties to support engineering analysis" [1].

As depicted in Figure 2, SysML inherits some of the diagrams of the UML metamodel and proposes additional diagrams. Some diagrams were inherited and new concepts were added for modelling.

In general, the following aspects can be cited as general contributions of the SysML when compared to UML:

- **Architecture Organization**: These include modelling concepts for organizing the system architecture descriptions as defined in ISO/IEC/IEEE 42010:2011 [37]. Among these, logic, visualization and viewpoints concepts are the most important.

- **Blocks and Flow**: Blocks diagram and SysML Internal Block diagram allow for the specifying of system decomposition and its interconnections. It provides a wide back- ground to represent system properties, along with the means to design the interaction of blocks and parts and how the items flow across connectors.

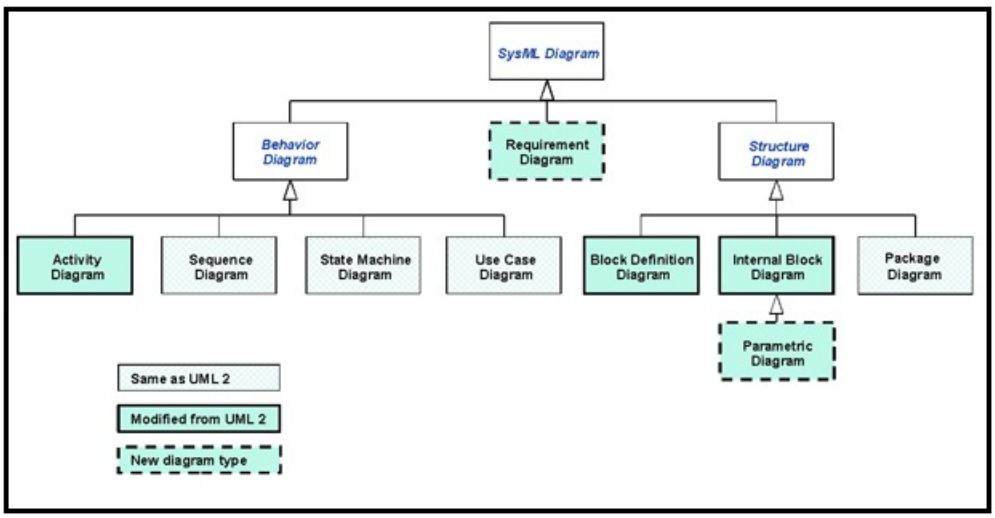

Figure 2 – Diagrams of SysML Profile, adapted of [1].

- **Behavior**: Although the majority of the SysML behavior constructions are similar to UML (interactions, state machines, use case and activities), SysML refines some of these for continuous systems modelling and for description of probabilities in the Activity diagram.
- **Requirements Modelling**: SysML Requirements diagram provide support for modeling individual requirements and their relationships. This diagram provides graphical, tabular or tree structure format for requirements specification. It increases spectrum of understanding and expressiveness of real-time system requirements.
- **Parametric**: The Parametric diagram is a restricted form of Internal Block diagram that shows only the use of constraint blocks along with the properties constrained within a context [1]. It allows for analytical and mathematical analysis of the system in design models.

SysML diagrams are used in the proposed methodology for RTES requirements specification and architectural design. The following subsections provide a brief explanation of SysML diagrams adopted in this thesis. This explanation describes the Use Case diagram, Requirements diagram, Blocks diagram, Block Definition diagram and Activity diagram in order to clarify their main concepts and adoption into RTES specification and design.

2.7.1 Use Case Diagram

The Use Case diagram shows scenarios of the system and their interaction/usage by the system actors. This diagram represents a system view based on its functionalities, capabilities and associated communications [1].

Actors, use cases and associations are the basic components of the Use Case diagram. Figure 29 provides an example of this diagram. Actors represent the system stakeholders and these can be related to users, systems or other environmental entities. Use cases allow for describing the system functions and show relevant system artifacts. These artifacts can be refined, along with requirements specification and architectural design, by other diagrams such as Activities diagrams, Sequence diagrams and so on.

There are four types of interactions or communication between the actors and use cases. They are named as communication, include, extend and generalization. These represent possible relationships that occur between the actors and system use cases, in order to provide the expected system behavior. Further details on relationships of the Use Case diagram can be found in [1].

2.7.2 SysML Requirements Diagram

SysML provides a new diagram, named as Requirements diagram, to model requirements and their relationships with other elements of the model. In SysML specification [1], a requirement is defined according to Figure 3.

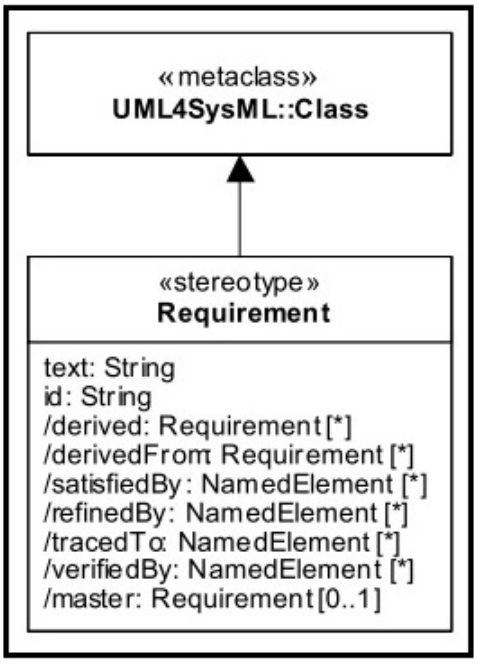

Figure 3 – SysML Requirements Model, adapted of [1].

A requirement is represented by the $\ll$ *Requirement* $\gg$ class stereotype, that is, a requirement is a stereotyped class. This mean that a requirement has the semantic of a class, which is extended by a specific semantic and by the properties/attributes of the stereotype.

SysML Requirements diagram allows for different ways of representing the possible relationships between requirements. The SysML Requirements diagram has seven types of

relationships. These relationships are named *DeriveReqt*, *Hierarchy*, *Verify*, *Refine*, *Trace*, *Copy* and *Satisfy* [1]. The focus of these relationships is to describe how requirements relate to one another and the relationships between different elements of the model, such as Use Cases diagrams. These are of great importance for the requirements specification, as besides describing contracts between software requirements and model elements; these also promote the management of changes, evolution and the traceability management of the specified requirements.

2.7.3 SysML Block Definition Diagram

Blocks represent a modular description of the system units. Each block can define a set of characteristics of a system component or of other elements of interest. Block design can include both structural and behavioral characteristics, such as properties and operations for representing the system state and its expected behavior [1].

SysML Blocks Definition Diagram (BDD) allows for the representing of properties and operations of one component and their relationships, such as associations, generalizations and dependencies. Figure 4 represents the SysML Blocks diagram model, which extends the UML Class diagram, with elements and restrictions, and expands its capabilities and connectors by reusable restriction forms and several properties to their connectors [1].

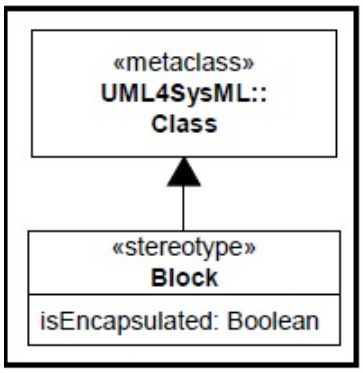

Figure 4 – SysML Blocks Diagram Model, adapted of [1].

As depicted in Figure 4, a block has an *isEncapsulated* attribute of Boolean type and, if it is true, the block is treated as a black box. A typed "part" component can be only connected by its ports or directly by its exterior limit. If it is false or if a value is not present, then connections may be established for elements of its external structure through the extremities of connectors [139]. A block can include properties for specifying values, parts and references to other blocks. Ports are a special class of properties used to specify permissible types of interactions between blocks.

2.7.4 SysML Internal Block Diagram

The Internal Block diagram (IBD) is an extension of the UML Composite Structure diagram [1]. The Internal Block diagram aims at modeling the internal structure of a block concerning properties and the connectors between properties. An Internal Block diagram represents a block and details, additional elements by adoption of shapes, notes and comments. Thus, a block, from a Block diagram, is refined and detailed by the mentioned properties and connectors.

IBD is composed of four general categories of properties: parts, references, value properties and constraints properties. The IBD **parts** show the decomposition of one block; the IDB **references** provide a manner to model services or information changes of one internal block; the IDB **value properties** are used to define the system property types by values and units to measure these; the IDB **constraints properties** can be added in constraint block compartment and support the description of different system properties in the IBD [140], [1].

Figure 34, Chapter 6, depicts one example of an Internal Block diagram. This diagram is composed of parts and connectors. The proposed models can describe the internal components of the block and its interconnections, through connectors, ports and item flows. Moreover, this diagram can represent the type of interface between the internal parts. An internal block can design either a required or provided interface.

2.7.5 SysML Activity Diagram

The SysML Activity diagram specifies the control and coordination of the system actions. Thus, input and output flow, sequential and conditional actions can be used to model the system behavior. Figure 37, Chapter 6, provides one example of the SysML Activity diagram. The Activity diagram is composed of the following elements: initial and final node, activity, flow, fork, join, condition, decision, merge, partition or swimlanes, sub-activity, final flow and notes. For further and detailed definitions see [1].

An activity is defined as an action or one step of a process. Therefore, it designs how actions are performed as, for example, calculation, data manipulation, information search, data sending/receiving. A transition represents the end of one activity and indicates which activity is next or which process ends.

SysML extends the UML Activity diagram and provides new extensions to: disable actions that are executing, model discrete and continuous flows, detail probability values of edges and output parameters. Moreover, the Activity diagram adds new semantic rules to model composition association between activities [1]. Furthermore, it is possible to specify, by annotations, timing constraints on the properties of the inputs, outputs, and other system properties.

2.8 Software Platform Embedded Systems

SPES [18] provides a framework for software development of RTES. SPES describes fundamental guidelines for requirements specification and architectural design and is based on two concepts: viewpoints and different granularity level. These viewpoints organize and control how the system views should be described. A system view shows individual concerns of system stakeholders and can detail, for example, user requirements, functional components, architectural solutions, subsystems and so on.

SPES does not define the languages, tools or techniques that should be employed, but it focuses on providing the main foundations for the engineering processes of RTES [18]. Thus, SPES depicts a manner for decomposing the system into fine-grained granularity subsystems. Both concepts generate structured and manageable design artifacts in RTES development.

SPES methodology consists of a group of guidelines for RTES design, divided into four categories/system viewpoints. These viewpoints are named **Requirements viewpoint**, **Functional viewpoint**, **Logical viewpoint** and **Technical viewpoint** [18]. By adoption of SPES methodology, software designers are able to translate requirements specification into physical and technical models. Designed models describe concepts for hardware and subsystem development processes [141]. Figure 5 depicts the viewpoints that are related to this thesis studies, as well as their interactions.

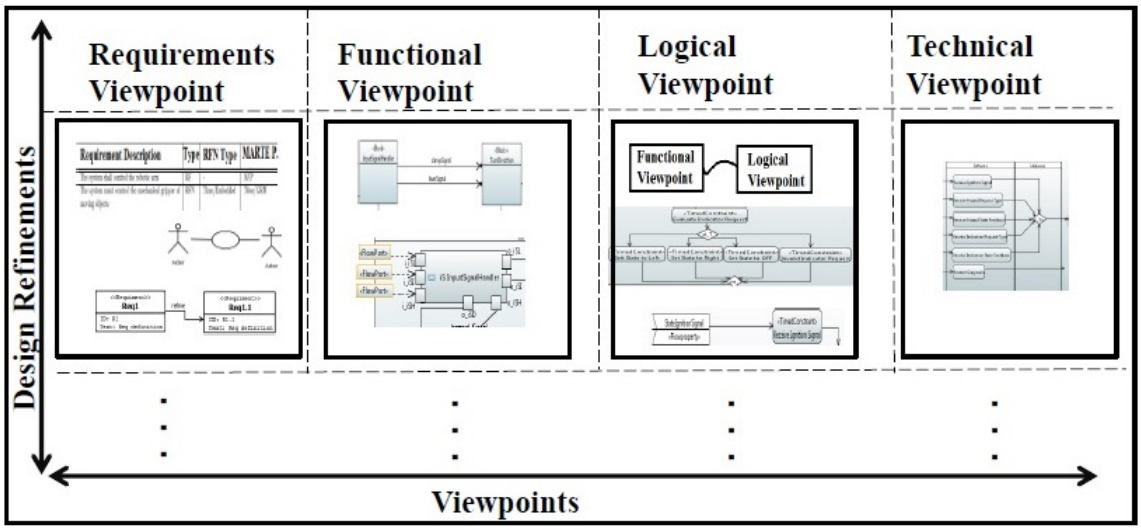

Figure 5 – Viewpoints and Granularly Views of SPES Methodology.

Requirements viewpoint is composed of the activities related to system RE. This viewpoint explores the requirements that are related to system context and it covers the requirement analysis and elicitation until the requirements validation and verification [126]. In this stage, four different types of models can be developed in order to describe distinct classes of requirements.

- **Context model** consists of the overall documents able to describe the system context. These documents should provide the system foundation and the elementary features for the requirements elicitation.

- **Goal model** shows the stakeholders goals that are related to the system under development. In this stage, the analyses on the communication between users and system services is performed in order to figure out more concrete requirements.

- **Scenario model** is responsible for representing interaction of the system under development and its context.

- **Solution-oriented model** is able to describe technical system requirements. This specification should be precise and it provides the foundations for system development.

Functional viewpoint [33] aims at representing the system requirements as correlated and structured components. This viewpoint provides different views of system functions, details of system services and the user functions interaction. Moreover, it presents how user functions are linked to their respective fine-granular system functions. The proposed refinements of Functional viewpoint can depict structural and behavioral perspectives of RTES.

Logical viewpoint describes the decomposition of the system functions, from the Functional viewpoint, into an architecture of logical components. The components design does not focus on domain features, physical or technological constraints. This viewpoint provides a group of refinements to detail what the system must perform in terms of services. Furthermore, logical views show the user functions for each logical component and behavioral models are defined as an output [142].

In the **Technical viewpoint**, a concrete technical solution can be designed and it highlights the technical system elements, their relationships and attributes [18]. It can be highlighted as contributions of the technical viewpoint, the study of interactions between software and hardware, the details of hardware that is coupled into system (as an example, ECUs, memory, communication channels and peripheral devices), the refinement of the logical subsystems and so on. It is important to mention that these viewpoints are not strictly top-down. For example, some activities that which are considered subsequent to the Logical viewpoint may require modification, updating or removal of an artefact from the Functional viewpoint. This should be performed interactively and before starting the Technical viewpoint design.

2.9 Characterization of the MARTE Profile

MARTE is a UML profile for modelling and analyzing RTES that provides a wide support for specification, design, verification and validation of complex systems [2]. Large parts of the MARTE concepts are refined for both modelling and analyzing concerns [44]. MARTE profile has, as advantages, the possibility of offering a common way for hardware and software modelling, improvement of communication between developers, standardization and interoperability between tools developers and the supply of constructors that allows for creation of models, which may describe quantitative, temporal and restrictive aspects of systems [59], [143].

MARTE metamodel extends UML [57] and adds capabilities for model-driven development of RTES. Figure 6 shows the general architecture of MARTE profile and its sub-packages. MARTE profile is structured into three packages named **MARTE Foundations Package**, which has as its objective to define fundamental concepts for RTES, **MARTE Design Model**, which provides the necessary support to make a detailed specification of a RTES design, and the **MARTE Analysis Model** that intends to support accurate and trustworthy evaluations using formal quantitative analyses based on mathematical models. The MARTE Analysis Model also presents several domains in which the analysis is performed, based on several software behaviors, such as performance, schedulability, energy consumption, memory, reliability, availability and security [2].

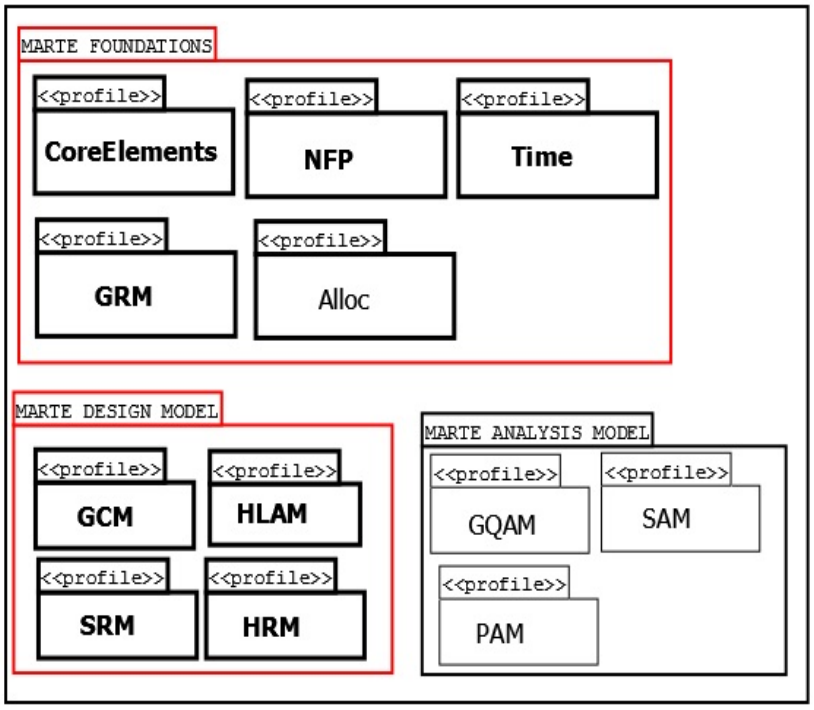

Figure 6 – MARTE Profile Architecture, adapted from [2].

Figure 6 highlights (red lines) the MARTE Foundations and MARTE Design Mo-

dels, as their internal packages are adopted in this thesis. Packages *Core*, *NPF*, *Time*, *GRM*, *GCM*, *SRM* and *HRM* are considered in the proposed methodology and provide meaningful semantics to model non-functional concepts in architectural models.

The constructors, stereotypes and constraints of these packages are classified and detailed in the following sections. Furthermore, a brief semantic description and application contexts/suggestions of each of these packages are presented as follows.

2.9.1 MARTE Foundations Model

MARTE Foundations package details the basic concepts for the use and comprehension of the other elements of the MARTE profile [2]. It presents, through specification of its subpackages, concepts that are useful for providing details of further components that are used for building the other MARTE packages. Packages Core Elements, Non-Functional Properties, Temporal Aspects and Generic Resource Modelling compose the MARTE Foundations Model and these are briefly elucidated in this section.

Core Elements Package

Core Elements package presents a number of important concepts on the elements of MARTE specification. These concepts provide background for the remaining packages of this specification. This package is subdivided into two packages named as Foundations and Causality packages.

Foundations package allows for representing design-time classifiers, such as the classes and the runtime instance elements that are created from these classifiers. It provides a consistent group of modelling elements and describes a wide metamodel to cover non-functional aspects necessary for RTES specification, design and analyses. Causality package describes the basic elements that are necessary for behavioral modeling and their run-time semantics. Furthermore, this package shows a high-level view of run-time semantics for modelling elements.

Non-Functional Properties Package

Non-Functional Properties Modeling (NFP) package provides a wide background for RTS specification and non-functional properties description, such as memory and energy consumption. It also shows how NFPs can be linked to the model elements, providing the capacity to encapsulate rich annotations inside non-functional models.

NFP provides the capacity to describe several types of values that are related to physical quantities, such as time, mass and energy. These values are used to describe the architecture, the behavior and the properties of computing system through the modelling elements.

Analysis and comprehension of NFP domain model and its constructors/stereotypes are fundamental for the proposed research. This model allows for representing and annotating NFP concerns in model elements, along with the definition of different types of relationships between modelled elements.

Temporal Aspects Package

Time package describes a general framework for timing modeling. Besides, it details structures, concepts and mechanisms that are appropriate when specifying, design and representing temporal aspects of RTES. Thus, it allows for the describing of several characteristics of these systems, such as delays, duration, clock time, chronometrical and logic time models.

When modelling RTES, time cannot be considered an external model, since time and behavior are frequently coupled. Development phases as, for example, modeling, design, implementation, performance and schedulability analysis can describe needs to take care of timing constraints. Thus, the Time package identifies concepts that relate to time and behavior, enriching the Causality package specification with explicit references to time.

This model also allows for the capturing of the influence of time on the behavior of objects, the behavior executions and the occurrence of events that may explicitly refer to clocks. In MARTE, a clock is used to measure, in the model, physical or logical time progress being elements able to access the time structure.

Generic Resource Modelling Package

Generic Resource Modeling (GRM) package provides stereotypes and marked values to represent resources as media, computing resources and storage resources of RTES. Moreover, this package includes resources that are necessary to model execution platforms at different abstraction and modelling levels.

GRM package, together with the Time modelling package, can be applied to specify time constraints and, when it is used with the NFP package, can be adopted to specify quality of services. This package is able to represent different characteristics of an execution platform. It includes constructors for the representation of software and hardware elements in a platform. Moreover, it provides constructors that are fundamental for refining elements of MARTE Design Model.

2.9.2 MARTE Design Model

MARTE Design Model performs refinements of Core Elements package in order to contribute for modelling of applications and to detail hardware and software platforms. The Generic Component Model (GCM) and Detailed Resource Modeling (DRM) packages

details semantic concepts of this package. Thus, in order to support its adoption, in this thesis, they are explained below.

Generic Component Model Package

The Generic Component Model (GCM) package describes important concepts for modelling artifacts in the context of RTES and of component-based approaches. This package has a dependency relationship with the Core Elements package and with each one of its subpackages. The domain description of GCM supports modelling of several communication criteria of embedded, real-time and/or distributed components allowing for the design, in the domain model, of many types of interactions for a structured component.

The GCM package provides constructors to model the properties of a communication port, the characteristics of the operation (reception/operation) of servers, the types of flows for an interface, signals sent to provide/request operations, storage policies of a port, invocation actions for a *FlowPort* or for a *ClientServerPort*.

Detailed Resource Modeling Package

The Detailed Resouce Modeling (DRM) package provides a detailed set of resources for modelling software and hardware platforms. It specializes many concepts that were defined, previously, by the GRM package. In order to facilitate the representation of the domain model this package is divided into two other subclauses/subpackages named Hardware Resource Modeling (HRM) and Software Resource Modeling (SRM).

SRM allows for the description of several characteristics of multitasking applications through sequential and multitasking approaches. Therefore, the SRM package is able to describe the software resources and services that are described in multi-task platforms (API) and to specify a set of modelling artefacts. HRM allows for the description of different views and details of a hardware element. The HRM package is made of two complementary visions: the logical, which classifies a hardware resource dependent on their functional properties, and the physical, which concentrates on the physical properties. Stereotypes introduced in this package are organized under a set of successive heritages from more generic to more specific stereotypes.

The Detailed Resource Modelling package, presented in this section, finalizes the semantic and syntactic contributions of the MARTE Design Model. The DRM package allows for the representing of characteristics of multitask platforms and provides guidelines to model software resources, concurrent execution contexts, memory management, synchronization and communication interactions. Furthermore, it contributes to specifying and designing characteristics of the hardware platforms, while describing constructors

for both physical and logical platforms, in the classifying of hardware entities and physical components of an application.

2.10 Contributions of this Chapter

RTES and their features are characterized here in order to detail the main non-functional concerns which are pertinent in this domain. This chapter provides an analysis of the specification and design of RTES and their inherent real-time embedded constraints. Moreover, MDSE, SPES and UML were described in order to contextualize their further adoption in the proposed study.

Another contribution of this chapter is the study of the semantic and syntax of the MARTE profile. This chapter provides a wide mapping of MARTE concepts which can support the modelling specification and design of RTES. The SysML profile and some of its diagrams are depicted throughout this chapter. The chapter ends with a mapping of important concepts of SysML and MARTE for the system specification and design.

Chapter 3

State of the Art Analysis

This chapter initially presents an analysis of the state of the art regarding the methodologies for the design of RTES, focusing on the studies related to Model-Driven Systems Engineering methodologies, approaches or techniques applied to requirements specification and/or architectural design.

3.1 Methodologies to Design Real-Time Embedded Systems

RTES development must follow strategies that guarantee specific and stringent non-functional requirements [144]. In the past, several methodologies were proposed for developing RTES, while considering their inherent functional and non-functional properties.

The research presented in [44] discusses the different strategies for combining MARTE and SysML profiles in a single modelling framework, which presents an initial debate regarding the expressiveness of the MARTE profile and its contributions toward designing RTES. Moreover, the authors detail inconsistencies in the combined use of multiple profiles in distinctive, complementary and several other perspectives. The study proposed in [44] allows for an in depth theoretical analysis of the alignment of MARTE and SysML profiles in architectural frameworks, in RE, in system level design and in quantitative analyses. The fundamental contribution of [44] is to provide guidelines regarding practices for combining MARTE and SysML for RTES design.

The *MeMVaTEx* methodology is presented in [145]. *MeMVaTEx* provides a framework to express and trace requirements in the automobile applications domain. For such, the authors integrate EAST-ADL2 [146], AUTOSAR [147], MARTE and SysML in order to define three modelling perspectives that are named as requirements model, solution model and V&V model. From the MARTE profile, the authors apply the concepts of the Time package for each proposed abstraction level. However, the MARTE profile has foundations for the Generic Quantitative Analysis Modeling (GQAM) which has not

been considered in this methodology. The authors do not explore the constructors for constraints specification or model verification. Besides, traceability criteria is presented only through the relationship between requirements that are defined in SysML without any adequation/enhancement.

The UMLsec profile is developed in [148] and presents a new proposal to combine MARTE and SysML profiles. UMLsec involves the specification and management of requirements. Requirements table and SysML Requirements diagram are employed in the description of requirements and their relationships. Stereotypes of the MARTE profile were used to model non-functional and temporal concepts. UMLsec profile is applied in a case study for modelling security requirements in telecommunication systems. However, despite the proposed case study in the scope of 3G telecommunication networks, it was not evidenced how traceability, power consumption and energy characteristics were represented through the MARTE profile.

GASPARD is a framework based on MARTE for the design of parallel embedded systems [149]. It provides a modelling support based on UML and MARTE, and prescribes a group of refinements activities at higher and lower granularity level. The proposed study performs code generation to verify and simulate performance constraints of massively parallel embedded systems. In a more specific manner, GASPARD focuses on design issues concerning Systems-on-Chip (SoCs), while considering the repetitive Model of Computation and MARTE profile. However, the GASPAR framework does not present the trace of non-functional requirements along with the architectural models and their performed refinements.

The work presented in [150] combines SysML and MARTE into one multi-views approach for modelling energy consumption requirements and their relationships with other functional/non-functional and structural/behavioral aspects. In this approach, each domain can be treated separately through different views, keeping the connection of elements of the model with other views. MARTE stereotypes are applied in five system views to specify power-consumption properties such as voltage values ($\ll Nfp\_Voltage \gg$ stereotype), discrete time ($\ll Clockstereotype \gg$) and frequency description ($\ll Nfp\_Frequencystereotype \gg$). The authors perform a group of transformations to analyze power consumption concerns. As stand by the authors, the proposed study lacks of strategies to trace the developed views in order to provide analyses and simulation of constrained models elements. Furthermore, the study proposed in [150] mainly presents structural models of the system.

The Aspect-oriented Model Driven Engineering for Real-Time systems (AMoDE-RT) approach, detailed in [41], combines UML, Platform-based design, RT-FRIDA approach and Aspect-Oriented Software Development (AOSD) to RTES design. It adopts different tools, code generation scripts and modelling concepts over to RTES development, from the initial design phases. Distributed Embedded Real-time Aspects Framework (DERAF)

was developed in order to support the design of functional and non-functional requirements at different development phases. This approach applies MARTE profile stereotypes to annotate non-functional concerns in structural and behavioral models. Thus, UML diagrams and MARTE stereotypes are used to show different system viewpoints and refinements until system realization. Furthermore, in [41] AOSD foundations provide separation of concerns and modular management of crosscutting requirements. Generation of Embedded Real Time Code based on Aspects (GenERTiCA) tool automatically generates code, from UML models, which consider mapping rules to create the correspondent code for elements of Distributed Embedded Real-time Compact Specification (DERCS). Qualitative evaluation is performed, through qualitative factor measurements, to check system attributes. This evaluation considers code generated from the AMoDE-RT approach against that produced by the OO application.

A design methodology based an existing code-centric real-time system, through MARTE constructors, is defined in [151]. It provides semi-automated support for the synchronous evolution of the code base and model. Initially, existent code is re-engineered in order to obtain the initial UML reverse model. Then, manual modelling is performed in the reverse model providing the refined reverse model. MARTE constructors enrich the non-functional data which could not be retrieved automatically by real-time experts. Finally, the methodology purposes distinctive analyses, such as schedulability, on the resultant models, while defining three refinements of the architectural models. The methodology proposed in [151] adopts MARTE constructors, specially, to annotate resources and schedulability constraints.

Another approach proposed in [152] develops an Actor-oriented modelling and design methodology for RTES. The role-oriented design approach is applied to model the system subsystems. Then, these subsystems are integrated into a whole model allowing further simulations and analysis of the design. Finally, it can automatically generate software code and hardware description code from debugger models. In [152], a group of design viewpoints for RTES is defined and it provides hardware and application models at different abstraction levels. However, the authors do not specify any empirical simulation strategies neither a non-functional properties analysis.

The MADES methodology, proposed in [153] as an extension of [154] and [87], combines a subset of diagrams and elements of SysML and MARTE profiles into a single methodology. The proposed methodology defines specific contributions for designing in the avionics and surveillance embedded systems industries. MADES provides different system viewpoints such as system requirements, Use Case scenarios, high-level specification and refined high-level specification. Considering these viewpoints, a group of extended diagrams as for example, the MADES Requirements diagram and the Blocks Functional Specification diagram, are developed. The MADES Requirements diagram integrates SysML requirements concepts to model system requirements. MADES Blocks

Functional Specification is an extension of the SysML Block diagram concepts to perform functional specification and refinements. In addition, MADES also applies MARTE constructors to provide details of the SysML Block components and other SysML/MARTE concepts to specify non-functional properties. In this case, the MARTE constructors are adopted in low granularity abstraction levels/refinements. Regarding code generation, MADES methodology performs model transformations as, for example, model-to-model and model-to-text transformations.

The research study presented in [88] is an extension of [155] and gives detail to the MDEReq methodology (*Model Driven Engineering for Requirement Management*). The proposed methodology is able to describe the processes of Requirements Engineering through the combination of UML Use Cases diagram, UML Class diagram, UML Sequence diagram and SysML Requirements diagram with MARTE profile constructors. In this thesis, the VSL language was employed to represent conditional temporal expressions related to elements of the domain. Traceability relations were described through trace relationship, of SysML Requirements diagram, and SysML matrixes.

The methodology COMPLEX UML/MARTE DSE is presented in [89]. This research combines concepts of Model Driven Engineering, Electronic System Level and design exploration technologies to create design solutions. UML is adopted to design abstract and graphical models. The MARTE profile concepts are applied along six design viewpoints and its constructors allow for the differentiation of functional and non-functional concerns. In this methodology, the user input is a system model following component-based modelling and separation of concerns. Simulation-based performance models are automatically generated from the input model. Thus, COMPLEX can provide executable system models allowing for functional validations and fast simulations.

In [156], the authors narrows down the application of MARTE into three distinct industrial problems: architectural modelling and large scale configuration of highly configurable integrated control systems (Submarine Production Systems), strength tests based modelling of intensive communication systems (Cisco Systems Conference System) and environment simulators-based modelling for the generation, on a large scale, of RTES for testing the WesternGEco Seismic Acquisition Marine System. The authors aim to report on the experiences for solving problems by the application of the MARTE profile. The research study, proposed in [156], provides a relevant contribution when systematizing a set of considerations/guides on how to apply MARTE in industrial contexts, thus helping to reduce the gap between modelling standards and market needs. Moreover, a detailed analysis of three domains is performed that describes which MARTE package can be used to distinguish different concepts of the domain. Differently to [156], MARTeSys^{ReqD} methodology shows how to combine the MARTE profile and MDSE approaches, into a single methodology, for RTES development.

In [157], a semantic and syntactic extension of UML/MARTE is proposed for design

space exploration in distributed embedded systems. This extension is named Network Profile and it defines a set of stereotypes, as an extension of the MARTE profile, to represent distributed embedded systems with UML metaclass elements. In the Network Profile, UML and MARTE profile are used to model the simulation oriented communication environment. The MARTE profile is extended and a precise semantic is presented to describe the specificities of system elements.

The Modeling Oriented Scheduling Analysis of Real-Time systems (MoSaRT) is a methodology for modeling and analyzing critical real-time systems [158]. As stated in [158], MoSaRT is an intermediate framework between real-time design languages and temporal analysis tools. Therefore, it aims at listing important guidelines regarding the most suitable temporal analyses to develop real-time systems. The MoSaRT methodology presents the Front-End framework, which is based on a design language, and the Back-End framework which relates to a meta-model of analysis repository. The authors formalize software operators and propose mathematical notations adopted in the Front-End framework models. This formal semantic substantiates the model elaboration along the hardware, software architecture, behavioral and functional viewpoints. Thus, the study proposed in [158] shows non-functional constraints analysis, focusing on schedulability tests, from the proposed set of rules.

An artefact-based approach for domain-independent Requirements Engineering (AMDiRE) is presented in [159]. The AMDiRE approach proposes the context specification, requirements specification and system specification viewpoints. AMDiRE suggests guidelines for the definition of project scope, context specification, description of the system vision, requirements specification and high-level description of architectural models. AMDiRE presents a group of principles for structural requirements engineering. Regarding the architectural design, AMDiRE mainly designs external interfaces and the system functions. Moreover, it provides a behavioral system view via state machines and performs empirical evaluations in the designed models.

The study proposed in [19], presents an Architecture Modeling Framework (AMF) semantical framework to design time on different development levels in the automotive industry, and is known as MULTIC Design Paradigm. MULTIC is based on viewpoints, a compositional semantic framework, timing specification, models of computations and converter channels. Contract-based Design describes system constraints by the adoption of assumptions-guarantee specification rules. It provides concepts to timing specifications through refinements of components behavior and their ports (from MATLAB/Simulink models). Therefore, it enables compositional schemes to detail components features/services and the guaranteed behavior of these components. In the MULTIC approach, SysML Requirements diagram and Block diagram are applied to design first models on a functional level; these models are refined by timing specifications. In this refinement, natural language patterns are used to express timing concerns. Thereafter, further refine-

ments of the functional level defines the models of computation and converter channels. To support the implementation phase SystemC and SystemC-AMS code were generated from the models. Thus, timing contracts are transformed manually into SystemC observers and the timing analysis is performed on executable models.

The AutoModel methodology allows for the specifying of system requirements in terms of its components, actions, goals and constraints [160]. This methodology is based on the UML-RT profile and provides structural and behavioral models from the specified requirements. Furthermore, it applies different tools to generate the system models, while adopting the UML-RT language to model the system and Papyrus-RT for code generation.All the system constraints are defined by Boolean logic as refinements of textual specifications. From this prior specification, the authors apply model transformation techniques to generate a structural model of the system in UML-RT. By adopting the AutoModel methodology, it is primordial to get a detailed and well-defined specification of the system requirements as of the initial design steps. This early definition can be a challenge in RTES development [120]. Moreover, the authors state that textual is a quite simple textual language. However, for complex and potentially large systems this type of specification can culminate into a more difficult requirement definition.

MoVES is a model-driven methodology for development of distributed vehicular RTES on single and multi-core platforms [161]. MoVES automatically provides all the Rubus Component Model (RCM) elements from the design models considering the initial EAST-ADL design level. Furthermore, RCM models are analyzed through model transformations. This approach considers six distinctive model transformations. The methodology starts with an automatic generation, through the FDA2RCM tool, of the software architecture and its timing properties, at the implementation level, for RCM models. Moreover, it passes by automatic generation of the RCM model, which represents the execution platform at the implementation level. The final step of MoVES is the refinement of the initial EAST-ADL models through timing analysis results. As stated in [161], MoVES presents a powerful methodology for the design of non-functional constraints of vehicular embedded systems. It is based on the interplay of domain-specific modelling languages, such as EAST-ADL and RCM.

It is quite remarkable the importance of MDSE approaches for well-defined strategies, in terms of integrated design and support for RTES development [162]. Considering the knowledge of the author of this thesis and based on the previous state of the art analysis, **the definition of a methodology**, with a consistent proposition of design activities [153] to design RTES constraints [19] [161], supporting empirical analysis [159], providing formal evaluations [158] and performing views and viewpoints refinements [126], [33], [142], [163] into one **single approach** is not in the scope of studies found in this literature review. Thus, Table 2 was created **in order to summarize the contributions and focus of research studies** detailed in the literature review. Moreover, Table 2 depicts

in each column the checking of these approaches facing the general and specific objectives of this thesis. The proposed analysis is not exhaustive, regarding the state of the art, as it does not aim to compete with other design approaches or evaluate them. Instead, it focuses on contributing toward minimizing the gaps and challenges that exist in RTES development.

The performed analysis, Table 2, from column 2 to column 8, is based exclusively on the research objectives, already detailed in Chapter 1, and their enclosed strategies for RTES design. This analysis contributes to forthcoming design strategies definition and to the final comparison of the state of the art against the final results of this thesis study. Thus, Table 2 covers, in its columns, the following criteria:

- ❑ **C1:** design of non-functional constraints
- ❑ **C2:** verification of non-functional constraints
- ❑ **C3:** definition of traceability strategies along the architectural design
- ❑ **C4:** description of different and complementary views/perspectives
- ❑ **C5:** description of viewpoints/abstraction levels
- ❑ **C6:** development of qualitative analysis of the proposed study
- ❑ **C7:** Is the research work a RTES methodology?
- ❑ **C8:** list of modelling languages.

As depicted in the literature review and complemented by the analysis on Table 2, multiple studies have addressed the RTES design. However, there is still a gap in the definition of strategies that contribute toward verifying RTES constraints (**C2**), to trace RTES constraints along of architectural design (**C3**) and to evaluate the design methodology and their contributions (**C6**). In addition, some approaches are developed in line with requirements engineering processes, others to architectural design and these do not define in their scope a methodology or a single approach in which requirements specification, architectural design and non-functional constraints are considered, as evaluated in (**C7**). Thus, it is expected that the proposed study can support RTES design while presenting significant results and contributions regarding the selected criteria.

Summarizing the Design Strategies of Literature Review								
Papers/Criteria	C1	C2	C3	C4	C5	C6	C7	C8
[44]	x		x				No	-
[145]	x	x	x	x	x		Yes	SysML, MARTE, EAST-ADL2
[148]	x			x	x		No	SysML, MARTE
[149]	x	x		x	x		No	UML, MARTE
[150]	x			x	x		No	SysML, MARTE
[151]	x	x		x	x		Yes	UML, MARTE
[152]	x			x	x		No	UML
[41]	x		x	x	x		No	UML, MARTE
[153]	x	x		x	x		Yes	SysML, MARTE
[88]			x	x	x		Yes[1]	UML, SysML, MARTE
[89]	x	x		x	x		Yes	UML, SysML, MARTE
[156]				x	x		No	UML, MARTE
[157]				x	x		No	UML, MARTE
[158]	x	x		x	x		Yes	MoSaRT language
[159]	x	x		x	x		No	UML and others[2] Concepts
[19]	x	x	x	x	x		No	SysML, MATLAB/Simulink, SystemC
[160]	x	x		x	x		Yes[3]	UML-RT[4], Boolean Logic Specification
[161]		x		x	x		Yes	EAST-ADL, RCM

Table 2 – Analysis of the State of the Art.

3.2 Formalization of Architectural Viewpoints

Architectural simplicity is an important factor to the success in the development of RTES [164]. However, to quantify the difficulty, cost and complexity of one systematic and creative activity is not sp easy [165]. In engineering, complexity commonly addresses the coupling rate of code components, and in software development, complexity is measured by estimating the program algorithm/code and its complexity [166]. Estimating the complexity of engineering design allows for a better definition and control of the development process [167]. It contributes with early analysis of the system evolution and

[1] MDEReq is Model Driven Engineering for Requirement Management methodology. The methodology relates to the Requirements Engineering process.

[2] Others means, on Table 2, the adoption of REM, REMsES and Bisa approaches and ARAMiS concepts in the proposed methodology.

[3] AutoModel methodology is a methodology specific to system Requirements Engineering process.

[4] Unified Modeling Language - Real Time (UML-RT)

eventually provides details about on a number of design artefacts, from the development cycle, and their risks.

Several research studies have been proposed to study the complexity that is related to the RTES design in order to evaluate the development process, architectural artefacts from design and/or to minimize the impact of a high complexity in the development of these systems. In [168], a proposal to evaluate quality of the design process is presented in order to provide quality measurements to design before the overall system implementation. A complexity measure is proposed by an analysis of inter-module and intra-module attributes. Overall analysis is performed in a unique model of the software design process and it aims to provide an estimation of the development error rate through a subjective assessment of design quality. In [168], measurements do not account for the external I/O in the functions definition.

The studies performed in [169] describe two sets of metrics to measure the complexity of systems development techniques and methods. The proposed metrics provide different guidelines for the designers regarding adoption of complexity metrics in their development process. Therefore, the study proposed in [169] performs a survey on metrics for the complexity of specification approaches to posteriorly contribute with the description of the modelling approach features and the analysis of the complexity of a set of modelling approaches. The complexity analysis is performed based on the measured complexity to learn the techniques and in the estimated complexity of the architectural models.

In [165], a new metric to measure functional complexity is proposed for assessing design complexity. The authors aim at evaluating the design and product complexity. The prior evaluation considers the measurement of complexity as a design attribute, and the latter provides the effort estimation in order to develop the system. Therefore, this research study considers the system functionalities to measure design complexity. These functionalities are based on user requirements, and the authors assume a direct relationship between functionality and resource consumption. According to [165], "more functionality that is required, the more complex a product is, and the more resources are required to design it". Thus, based on the tree of system sub-functions, and functional decompositions, the studies, proposed in [165], creates the Product Complexity (PC) metric. The PC is calculated by the number of functions, at level j, multiplied by the number of levels. The proposed metric counts the number of functions at each level and weights these to the number of the level. However, the complexity analysis can be biased to different sub-system decompositions due to the involved subjective activities. Moreover, the authors do not detail the relationship between product complexity and design effort.

Metrics for evaluating Objected-Oriented (OO) design complexity are presented in [170]. The authors explore empirically the validation of three existing OO design complexity metrics to assess their ability to predict maintenance time. An empirical study and

subjectively analysis of metrics: Interaction Level (IL), Interface Size (IS), and Operation Argument Complexity (OAC) was performed in the proposed studyis that there exists a relationship between the complexity of a system design and the maintenance time required to make changes.

In [166], a broader theoretical definition of complexity and existing complexity measurement methods is provided to create objective measurements properties and metrics for system architecture models. In order to estimate the complexity, it defines equality relationships among systems and models. Therefore, the authors assume that the complexity of the model correlates with the cost and difficulty of implementing the system. In order to perform such analysis, the research, proposed in [166], measures "the number of distinct types of things (objects, processes, states), number of processes affecting an object, number of objects being affected by a process, number of operands per process and performs the sum of the number of things of each distinct type".

The concept of design complexity is discussed in [171]. The authors address the design complexity by the intrinsic complexity of the multi-disciplinary structure. Here, the main emphasis is to figure out the main problems that can occur due to low interaction between the multidisciplinary designs and qualitatively measure the level of difficultly to manage it. According to [172], good software design practices should minimize coupling and maximizing cohesiveness. In [172], design patterns and role-based component design approaches are analyzed to present a theoretical study about how these design principles can help the designer in complexity management.

The study proposed in [173] presents a complexity analysis for embedded systems design considering qualitative measurements of design process from the user perspective. In this case, it performs an evaluation of design models using a cognitive complexity checking. The authors suggest a measure which describes the efforts to understand the artifacts of design from the viewpoint of clients based on their experiences, know-how and landscape. According to [173], the design complexity level is related to the relationship between the efforts of users to understand and adopt models versus the concepts that are used in models representation. As a result, a set of insights is provided to trace useful basic-level concepts for modeling embedded systems at the system level.

A systematic mapping study was published in [174] in order to review software metrics for measuring the understandability of architectural structures. The metrics consider measured artefacts, attributes, quality characteristics, and representation model used to define software metrics. Thus, the measured concepts come from the general properties such as size, complexity, coupling, cohesion and modularization of the architectural artefacts. The authors conclude that in general there is a gap in the architectural analysis metrics regarding the evaluation of design artifacts and due to this, it is necessary to create quality measurements of the high-level views of design. There is a lack of useful metrics that can be adopted by practitioners and appropriate analysis to estimate the

understandability level of a software system.

There is a lack of approaches to measure quality features of architectural models [174]. It is important to evaluate the level of satisfaction of architectural design, since in RTES development, this process manages overall system functions and their critical aspects [167]. The formalization of architectural design steps contributes to show how software design models are better mapped to code.

Based on the fact that during development of RTES, it is not always possible to wait until the final stages of design to estimate the overall system complexity [167], this thesis aims to quantify the complexity of RTES design, in order to show complexity cost measurements to adopt the developed methodology. Thus, formalization performed in this thesis provides a novel and formal manner to analyze the complexity of RTES design activities and the system cost function. Differently to other studies, here it is possible to perform early estimation of system complexity based on cost function to design evolution and the overall system complexity, taking in to account the number of system components.

The proposed measurements contribute with a quantitative and formal analysis, of this unpredictive design process, by means of following a specific methodology. Once the system design activities involves human activities and interactions, it could be hard to predict them. The term unpredictive is adopted here, since the system design process is a non-deterministic task. Therefore, final models, the workflow of the process and artifacts from this process are impacted by human expertise and knowledge. Performed analysis here contributes to a common and formalized manner to identify the design costs. Noted here, as advantages of the complexity measurements are the development of a cost function as inputs to estimate required resources, the non-subjective analysis of the design complexity activities and the computation of the number of design components, from the outset of architectural design activities.

3.3 Contributions

This chapter briefly presents different methodologies to requirements specification, architectural design and complexity analysis of RTES design. In addition, a comparison of the literature review against the proposed methodology can be performed in order to provide insights concerning contributions and innovations of this thesis.

Although different papers present several contributions in the RTES development there is still a lack of studies to define traceability criterion, distinctive and complementary architectural design of RTES viewpoints, consistent and early analyzes of non-functional requirements, the combination of graphical, textual and formal approaches into a single methodology for RTES design, while furnishing complexity analyzes of RTES design activities.

Chapter 4

The MARTeSys^{ReqD} Methodology

This chapter provides details of the MARTeSys^{ReqD} methodology and its main concepts. The general framework and the group of activities covered by the proposed methodology are defined and detailed here.

4.1 MARTeSys^{ReqD} Scope

The proposed methodology contributes to architectural design of RTES, while highlighting how real-time and embedded constraints can be modeled in different viewpoints. As noted in Figure 7, MARTeSys^{ReqD} adopts the Software Platform Embedded Systems (SPES) guidelines, SysML diagrams and MARTE stereotypes.

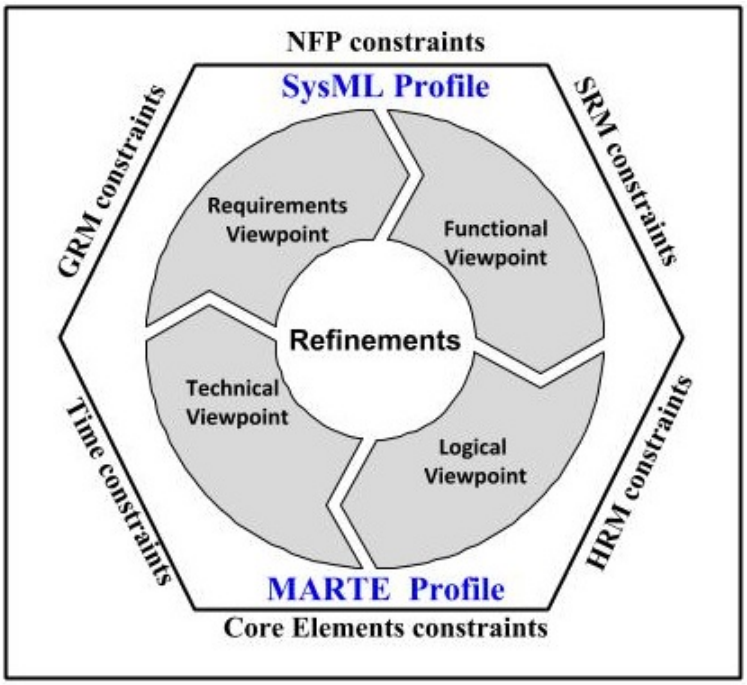

Figure 7 – Global View of the MARTeSys^{ReqD} Methodology.

MARTeSys^{ReqD} methodology is composed of four distinctive viewpoints. Thus, **Requirements**, **Functional**, **Logical** and **Technical viewpoints** provide the basis for arranging an architectural description of RTES focusing on description, modeling and evaluation of non-functional concerns. Moreover, the SysML profile and its diagrams are adopted in those viewpoints, within each refinement, to model behavioral and structural design of systems. The MARTE profile and its sub-packages provide the main foundations for engineering processes of RTES and for representing non-functional concerns of these systems. As emphasized in Chapter 2, the adoption of the UML profiles are justified by their adequacy level for the analysis, as well as the implementation, specification and architectural design of RTES.

The MARTeSys^{ReqD} methodology proposes an approach to RTES development and it is centered within the following topics:

- ❏ Adoption of well-recognized modelling languages. UML profiles, and their diagrams, have been extensively employed in this research, since they allow for the representation of the system under different abstraction levels.

- ❏ Stereotypes, annotations and enumerations of the MARTE profile are traced to model RTES concerns, as this profile was developed to contribute to design and analysis of RTES properties.

- ❏ SPES methodology provides a background to architectural design resolutions. It conducts a collection of suggestions for integration between software, systems, hardware, mechanical and physical parts in architectural design. Besides, SPES provides guidelines to RTES development, while considering essential strategies for the design these systems.

- ❏ Combination and Extension of SysML diagrams applying a standard nomenclature through the VSL formalism. The proposed combination provides a formal style to express constraints values, properties and stereotypes. VSL contributes with a standard and unambiguous annotation of constraints in model elements, while supporting the system implementation activities. In this thesis, VSL allows for annotation of the non-functional constraints regarding the Value Specification grammar.

- ❏ Adoption of tools for code generation, for constraints definition and for their representation, in a low granularity level, in the system implementation.

- ❏ Model checking: UPPAAL is used here to validate and verify models of Requirements viewpoints. In this thesis, the proposed studies create a strategy to describe, in TA formalism, timing constraints from requirements specification documents and analyze them since early design steps.

- ❑ Strategies for empirical evaluation of non-functional constraints. Therefore, it is possible to evaluate the MARTE annotations from the architectural models and check the consistency between the graphical models and the execution models.

Figure 8 depicts the overall flow of the MARTeSys^{ReqD} methodology. The MARTeSys^{ReqD} is a methodology based on MDSE concepts which provides a group of recommendations for RTES development. The main characteristics of the proposed methodology are highlighted as follow:

- ❑ ***C1***: A general methodology for RTES design based on refinements and viewpoints. The refinements present different viewpoints of the system. In each abstraction level, a SysML diagram or SysML table is extended with MARTE stereotypes. Thus, there is expected a syntactic and semantic improvement in the definition of architectural artefacts.

- ❑ ***C2***: MARTeSys^{ReqD} proposes a formalization to the RTES activities. In this formalization, a group of algorithms is defined to describe the general design decisions. This formalization is performed in standardized pseudo-code and defines the set of steps for adopting the proposed methodology in developing RTES.

- ❑ ***C3***: It provides a new strategy for analyzing the complexity of RTES design activities. It contributes to quantitative measurements and to describe an analysis of the design complexity, from the outset of the initial design activities.

- ❑ ***C4***: In RTES development, non-functional concerns, for example timing observations, must be concisely and correctly expressed in design artefacts. The thesis studies focuses on description and tracing of different non-functional concerns along the RTES development.

- ❑ ***C5***: Empirical and analytical techniques are applied to evaluate constraints attached to design models. These techniques involve respectively empirical validations and model checking strategies. Moreover, these strategies substantiate the description and analysis of constraints in two validation steps, respectively, from functional models and executable code.

- ❑ ***C6***: Description of an independent representation of physical and logical components in RTES architectural design. The proposed viewpoints and their refinements allow for the specification of structural and behavioral views of RTES. Therefore, it is possible to show how the design decisions can be separated from any architectural implementation in requirements specification and system design.

- ❑ ***C7***: It provides an strategy to represent and trace RTES constraints along the development process. It allows for the performing of early evaluations of critical

constraints before an initial implementation of the system. Moreover, it contributes to the definition, representation, refinement and validation of RTES concerns in an integrated way to the system services.

In general, a methodology is composed of one or more processes, methods and tools [175] [120] [81]. A methodology must organize these components into systematic activities in order to be applied in a well-defined context. MARTeSys^{ReqD} is a MDSE methodology that brings together modelling diagrams, SPES guidelines and stereotypes from the MARTE profile. The proposed methodology carries out activities that are related to specification, documentation, architectural design, validation and management of RTES requirements.

4.2 General Flow of the MARTeSys^{ReqD} Methodology

The MARTeSys^{ReqD} methodology provides a sequence of steps for RTES development. Figure 8 shows a general overview of the proposed methodology. In Figure 8, the orange arrows between phases as, for example, the four initial viewpoints, indicate that all them, at the different levels, can be (if necessary) iteratively updated. The blue arrows denote sequential transitions of different development phases. For example, from architectural viewpoints (Phase 1) to the formalization activities (Phase 2).

The left-down corner side, Figure 8, displays the main technologies that have been adopted in this methodology. The MARTeSys^{ReqD} methodology is grounded on the concepts, guidelines, libraries and definitions of the FreeRTOS, SysML profile, MARTE profile, SPES methodology, Arduino Mega and UPPAAL. Moreover, the number of employed methods are not restricted to those mentioned above, that is, they can be extended by other technologies regarding the needs of the study domain.

The first activities, labeled in Figure 8 as 1, aim at providing architectural descriptions for RTES in four design viewpoints. The Requirements, Functional, Logical and Technical viewpoints already defined in Chapter 2 provide the basis to direct the design activities of this thesis.

In the **Requirements viewpoint**, a group of activities for the analysis, specification, elicitation, definition, validation and verification of system requirements is proposed. Here, the expected output is the graphical, textual and/or tabular artefacts that represent the system context.

Functional viewpoint combines a set of user functions into a model that describes the system services/functionalities. Functional viewpoint transforms the integrated requirements models, from the Requirements viewpoint, into rational system specifications.

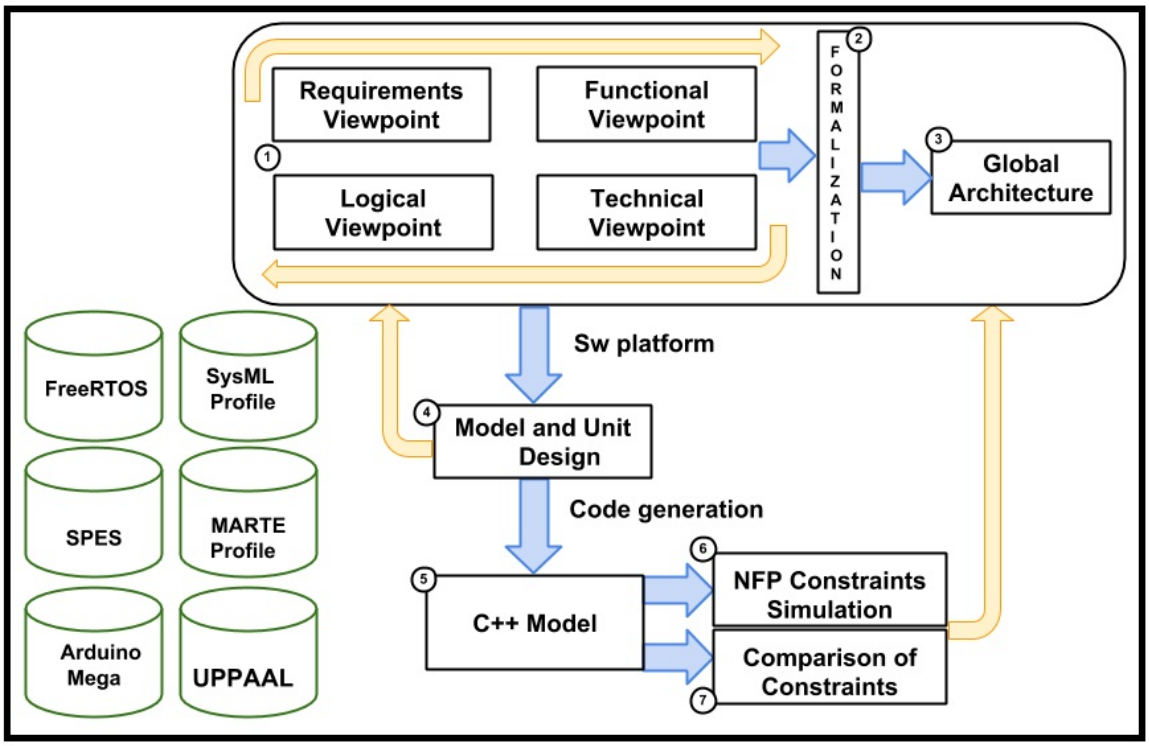

Figure 8 – General Flow of the MARTeSys^{ReqD} Methodology.

From this viewpoint, artefacts are generated to show the system behavior, the details of the system services and the user functions interaction.

Logical viewpoint aims at performing the analysis of system components and describing their logical solution. Here, the logical architecture is gradually refined into fined-grained logical elements in order to provide a structured design model of the system.

In the **Technical viewpoint**, software and hardware components, which are related to the system under development, are combined. Technical viewpoint artefacts are related to final system implementation and should ensure that the design is in accordance with logical viewpoint assumptions and with logical and physical system constraints. In this viewpoint, artefacts describe the final system implementation while preserving Logical viewpoint assumptions with logical and physical system constraints.

In Figure 8, the blue arrows represent the sequence flow for adopting the methodology steps. This flow is also correlated to the labels $1-7$. The orange arrows show interactive steps. It means that from a step, it is possible to return to the previous one and refine it. Phase 2 shows **Formalization** of the design activities. From each solution of architectural design different algorithms are created to formalize the design decisions of the MARTeSys^{ReqD} methodology. This formalization intends on collaborating toward the understanding and adoption of the MARTeSys^{ReqD} design decisions. Furthermore, it contributes to the complexity analysis of RTES development.

A Global Architecture is generated towards a RTES (Phase 3). The main objective

of this view is to describe the global system structure and its relationships. Within these models, hardware and software components are modeled and their external communication is also defined. The Global Architecture model is able to show the system design in a global perspective and to represent how each software/hardware component cooperates with other components. From these models, the **Model and Unit Design** artefacts are created (Phase 4).

The Model and Unit Design (MUD) phase provides models that describe system features at a low abstraction level. Models represent the system in a viewpoint that is closer to the implementation stage. It means that they should describe software details as well as how smaller software components cooperate at the implementation level. Moreover, these models show how software components interact and control physical and embedded elements. Phase 5 relates to the Automated Code Generation from MUD models. Finally, Phases 6 and 7 shows the system realization and final system evaluations activities of the MARTeSys^{ReqD} methodology.

In Phase 5, **C^{++} Model** is automatically generated by models transformation. Here, the Papyrus [176] tool has been adopted to design activities and to source code generation. It is important to highlight that the Papyrus tool performs a partial code generation and due to this, manual code development is necessary. However, an automatic code generation is performed while considering non-functional requirements and constraints of RTES. Furthermore, it provides a simple way to guide RTES design, their non-functional requirements development and maintenance activities regarding model constraints.

NFP Constraints Simulation adopts empirical/experimental techniques to evaluate the system services realization and the constraints that are imposed on these services (Phase 6). The main objective of this phase is to describe a common means for simulating timing, schedulability and precedence constraints of RTES, by considering the runtime behavior of the tasks. Moreover, **Comparison of Constraints** (Phase 7) allows one to check if the constraints, under the system services, are reached in the final development stage.

Phases 4 , 6 and 7 allow for interactive improvement of RTES architectural models. As it can be observed in Figure 8, from these phases there is a path to return and refine the architectural viewpoints. It is possible to connect information from validating results (Phases 6 and 7) with early design artefacts and improve them. This analysis shows that the RTES constraints can be represented in early design viewpoints, fulfilled at different abstraction levels and reached and simulated in the final development phase. Moreover, it proves that there is consistency between the architectural constrained models and the final model implementation regarding constraints estimation. Finally, it also indicates that constrained parameters, which are related with real-time schedule/precedence, can be satisfied and evaluated.

4.3 Requirements Specification and Architectural Viewpoints within the MARTeSys^{ReqD} Methodology

As previously presented, when defining the methodology scope, one must consider the set of logical activities to be followed in order to achieve a proposed goal, that is, the process definition. A process describes "what" must be performed without detailing "how" each activity has to be performed. Methods detail techniques that should be employed, within a methodology, in order to fulfill an activity/process. Therefore, affirmations can be made that a method describes "how" an activity will be realized. Tools and well-defined design strategies support the methodology definition, provide theoretical background to its application and favor its analysis and development activities.

Figure 9 shows a detailed view of the MARTeSys^{ReqD} methodology and the covered/specific activities for structuring an RTES architectural description, from requirements definition to system realization.

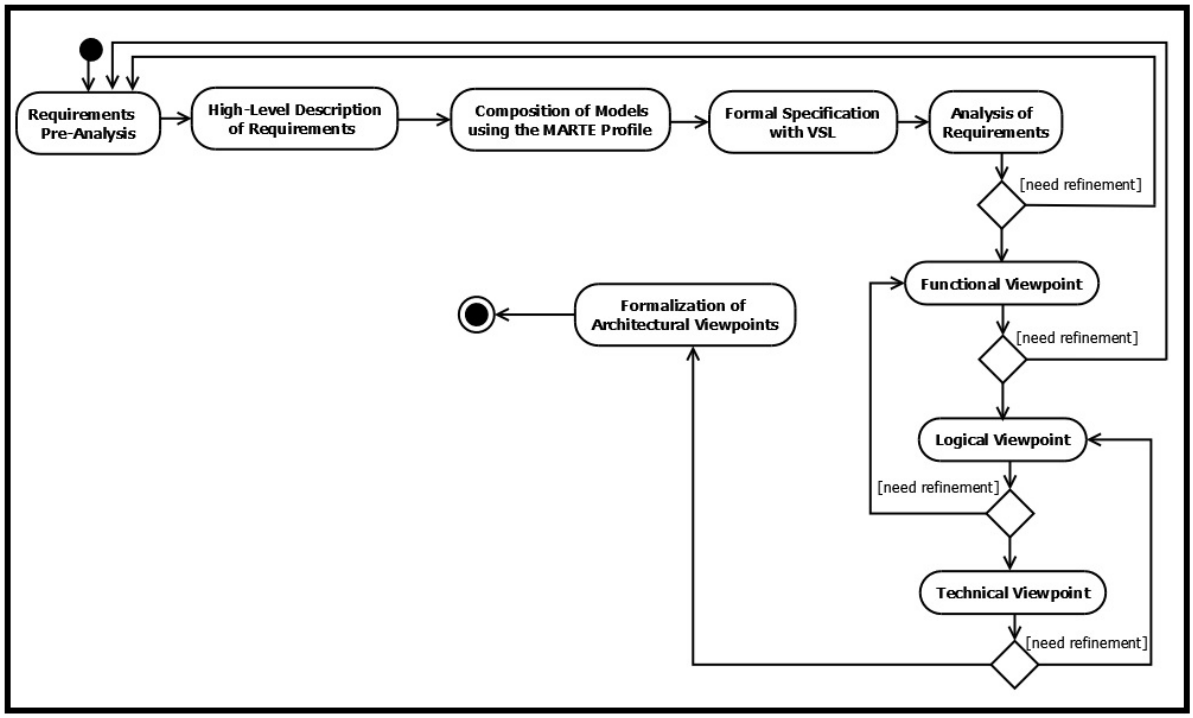

Figure 9 – The MARTeSys^{ReqD} Methodology.

Requirements viewpoint, in Figure 9, is placed on the upper level of the flowchart. On the other hand, the Functional, Logical and Technical viewpoints are grouped in the lower right-hand corner. This flowchart shows the four proposed viewpoints in a group of interactive views. Many refinements can be performed inside each viewpoint. The decision node represents the case where internal refinements, of one viewpoint, imply the update of theprevious ones. Formalization of Architectural Viewpoints is the last proposed activity and it is detailed in Chapter 5.

The following subsections provide detail of the internal activities of each MARTeSys^{ReqD} methodology viewpoint. These sections provide details for the specific activities of each viewpoint. Moreover, it gives details on how the design decisions can be made and shows how to incorporate RTES characteristics, such as functional, non-functional and embedded features in a methodology.

4.4 Requirements Viewpoint of the MARTeSys^{ReqD} Methodology

The Requirements viewpoint covers the definition and analysis of the overall system functions considering non-functional requirements and RTES constraints. As presented in Figure 10, it is possible to link this viewpoint with well-defined processes/stages of RE [177], [178], [120]. Requirements Analysis, Requirements Specification, Verification and Validation of Requirements and Requirements Management, from RE, are covered within the **Requirements viewpoint** definition. Figure 10 shows the correlation between internal activities of Requirements Viewpoint, of MARTeSys^{ReqD} methodology, and the RE process.

The proposed methodology is not exhaustive, as it does not fully cover RE activities. Specifically in the Requirements Verification, Requirements Validation and Requirements Management, it partially contributes with guidelines and design methods.

Requirements viewpoint, within the MARTeSys^{ReqD} methodology, is composed of Requirements Pre-Analysis, High-Level Description of Requirements, Composition of Models using the MARTE Profile, Formal Specification with VSL and Analysis of Requirements. The following sub-sections discuss these views in detail. As noted in Figure 10, Requirements Gathering Artefacts and Requirements Elicitation are not covered by the proposed methodology. **MARTeSys^{ReqD}** has, as input, an initial group of elicited requirements. It is expected that this requirements document describes the scenario, purpose and the global idea of the system. These requirements are posteriorly refined and detailed in the Requirements viewpoint. The refinements, or internal stages, of the Requirements viewpoint definition are detailed in the subsequent sections. Moreover, these sections present a comprehensive explanation of the chosen design strategies, SysML diagrams and MARTE packages that can contribute to RTES design.

4.4.1 Requirements Pre-Analysis

Requirement Pre-Analysis represents the first activity addressed in the MARTeSys^{ReqD} methodology. This activity, from the Requirements viewpoint, receives as input a group of previously elicited requirements. The requirements elicitation process performs requirements elicitation, domain analyses and description of user needs. Thus, techniques

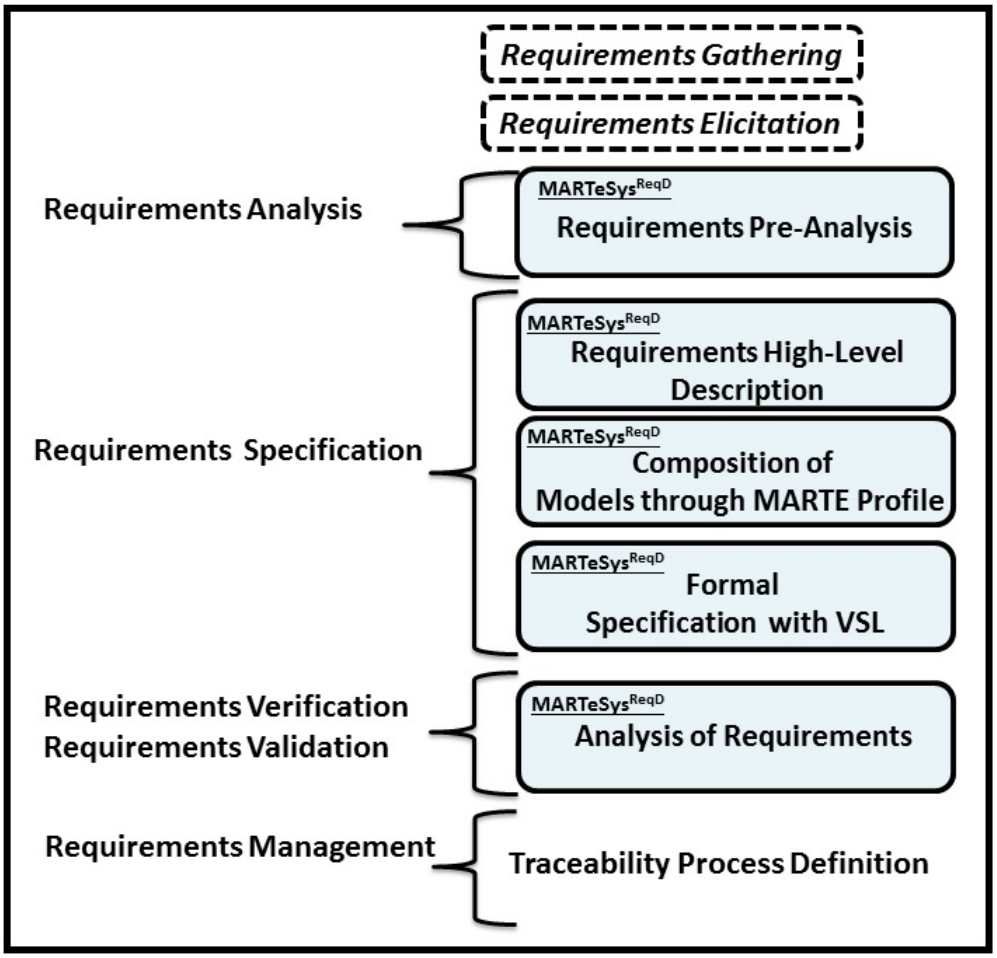

Figure 10 – Correlation between Requirements Engineering Process and Requirements Viewpoint of the MARTeSys^{ReqD} Methodology.

and guidelines of requirements elicitation should be applied in order to describe, as input at this step, the requirements document. This document contains an overview of the main system properties, as well as its scope and stakeholders.

Requirements Pre-Analysis analyzes each requirement in order (**1**°) to improve its description, (**2**°) to define functional or non-functional criterium and (**3**°) to relate this requirement to the MARTE profile. Here, a group of guidelines, which will be described later in the text, is adopted to write and refine the requirements specification. Therefore, previous elicited requirements must be analyzed in order to improve the understanding of the critical properties of the system. Table 3 shows a template that should be adopted as the first refinement of Requirements viewpoint.

The first column has a requirement identification (ID), that is a variable or acronym which names the requirement. The second column is filled with a textual description of requirements and depicts their definition. The third column performs requirements classification. An initial requirements categorization is considered here, i.e., each requirement is related with a different category, including functional, non-functional and

ID	Requirement Description	Type	RFN Type	MARTE Package
R1: ID of Requirement 1	Description of R1	Type of R1	Non-Functional Requirement of R1	Related MARTE Package
R2: ID of Requirement 2	Description of R2	Type of R2	Non-Functional Requirement of R2	Related MARTE Package
. . .	. . .	. . .	. . .	
RN: ID of Requirement N	Description of RN	Type of RN	Non-Functional Requirement of RN	Related MARTE Package

Table 3 – Framework for the Requirements Pre-Analysis.

domain-specific types. The third column allows for the identification of possible non-functional properties related to that requirement. This prior analysis is important to the initial activities of Requirements viewpoint, since it allows one to identify and derive the main embedded and non-functional functionalities that are required by a system. This analysis is performed according to RTES concerns, which are classified in Annex C.

Finally, the last column enables one to relate a requirement to the MARTE profile packages. It contributes to refinement of the Requirements viewpoint or to further viewpoints definition, while facilitating the trace of MARTE stereotypes/packages, which contributes to requirements design throughout RTES development.

In the proposed methodology, the activity of Requirement Pre-Analysis is an interactive process and provides artefacts to document the system. These artefacts are input to the next activity that deals with a high-Level Description of Requirements; also, these can be used in communications with stakeholders and to the development of test cases.

4.4.2 High-Level Description of Requirements

As shown in Figure 10, the activity of High-Level Description relates to the first activity of Requirements Specification process. It employs an extension of SysML Table to refine the RTES requirements. This viewpoint guides the classification of system requirements, in a tabular manner, considering cohesive criteria. From this classification, a group of categories is created by respecting requirements correlations or similarity. Thus, each category maintains requirements that are related to each other and can define a group of correlated system services. Table 4 shows the template to categorize requirements at the High-Level Description of Requirements activity.

The first column of Table 4 describes the category in which one or more requirements are allocated (placed). The second column receives the ID of each requirement, where $a1$, $a2 \in \mathbb{N}$, $a2 > a1 + 1$ and $a2 + 1 < N$. Finally, in the third column, the guidelines for naming a requirement, provided in [81], are adopted to refine the natural declaration of these requirements.

Related Context	ID	Requirement Declaration
Category Definition 1	Requirement R1	Description of Requirement 1
	Requirement R2	Description of Requirement 2
	. . .	. . .
	Requirement R(N-a1)	Description of Requirement R(N-a1)
Category Definition ...	Requirement R(N-a1+1)	Description of Requirement R(N-a1+1)
	. . Requirement R(N-a2)	. . Description of Requirement R(N-a2)
Category Definition M	Requirement R(N-a2+1)	Description of Requirement R(N-a2+1)
	. . .	. . .
	Requirement RN	Description of Requirement RN

Table 4 – Requirements Categorization Table - **Requirements viewpoint**.

On Table 4, a requirement is described in natural language and must use, in its declaration, the keywords *Shall*, *Must*, *Are Applicable*, *Responsible* and *Will* in order to express priority criteria to its future development. Moreover, the refinements performed here contribute to the requirements description and to minimizing ambiguities. The definition of each keyword is presented as follows:

- **Shall**: it describes a fundamental system requirement. Requirements named through shall are, in the majority of cases, fixed into a general development contract and they must be implemented and verified.
- **Must**: it defines non-functional requirements or constraints, such as performance and safety, which must be developed.
- **Should**: it describes non-obligatory requirements. In this type of declaration it may be acceptable to provide reasons to ignore the specified requirement.
- **Are applicable**: it permits the addition, by reference, of extra information and regulatory standards into a specification.
- **Responsible**: it defines imperative requirements for an architecture.
- **Will**: it describes information related to the operational or development environment in order to provide requirements capabilities.

From the proposed categorization a second refinement is performed. The SysML Use Case diagram allows for the graphical representation of system requirements and the generated artefacts, in this refinement, provide the link between the Requirements viewpoint and Functional viewpoint models. Figure 11 depicts a generic example of a possible refinement of the system categories by SysML Use Case diagram in the activity

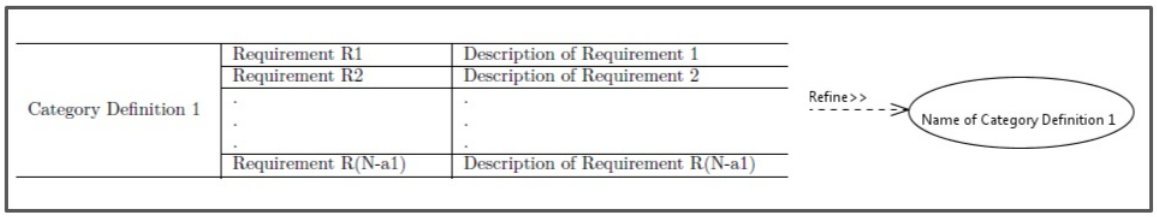

Figure 11 – Example of Requirements Refinement within the Activity of High-Level Description of Requirements.

of High-Level Description of Requirements. This level provides the definition of the major system functions, describes an overview of these requirements at a high level of abstraction and represents external entities that influence the scenario.

4.4.3 Composition of Models using the MARTE Profile

This activity strengthens previous requirements models with non-functional information. It allows for the combining of requirements specification models with annotations, stereotypes, constructors or enumerations of the MARTE profile. Annex C, presents another contribution of this thesis, along with providing details of the specific purposes of a stereotype. In Annex C, Table 14 traces the MARTE stereotypes able to map a specific RTES constraint (concern). The proposed mapping provides guidelines to non-functional annotations along the M viewpoints.

Adoption of annotations, stereotypes or enumerations in requirements models allows for early description and analysis of RTES concerns. Therefore, this activity refines the Requirements viewpoint models by modelling requirements in an atomic fashion, defining cooperation and relationships between these requirements and adding non-functional annotations. In order to perform this, the SysML Requirements diagram is applied to represent system requirements and to provide details of its main functions.

Initially, requirements and their relationships are graphically expressed. The SysML Requirements diagram does not consider, in its original metamodel, the ability to represent non-functional requirements. Thus, as a refinement of this model, the constituting process of SysML models and of the MARTE profile, when necessary, can enable the early representation of RTES concerns.

The proposed composition contributes to specifying and describing non-functional criteria, relating, for example, with non-functional properties, such as timing, accuracy, performance, embedded and distribution requirements, as well as the description of physical and logical system resources.

4.4.4 Formal Specification with VSL

VSL allows in a standard manner to declare annotations to be inserted into models. This activity describes another refinement of previous models of the Requirements viewpoint. Here, non-functional annotations are added to the models of the SysML Requirements diagram using the VSL formalism. Adoption of VSL allows one to specify, in a standard manner, different value specifications, timing expressions, constraint descriptions and stereotype features. Figure 12 describes the concrete syntax that can be employed to express value specifications in model elements.

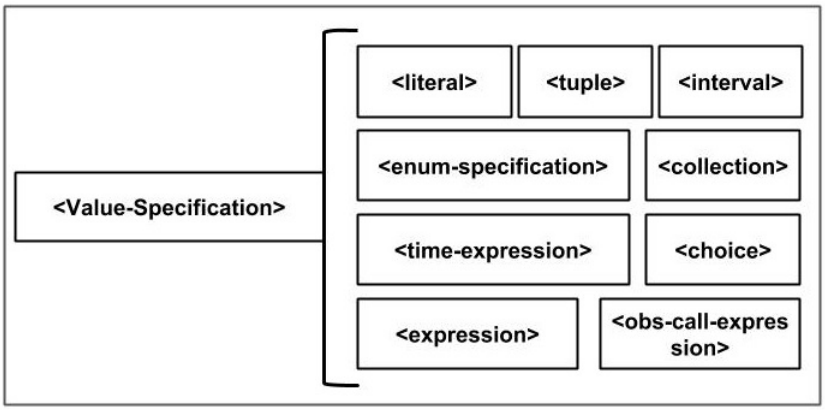

Figure 12 – Formalization of Concerns in Requirements Models by VSL.

The studies in this thesis adopt the Value Specification Language (VSL) to annotate and detail non-functional constraints in requirements and architectural design. Adoption of VSL to annotate constraints, abstractions, requirements specifications, which are most often described in natural language, contributes to their definition and to strengthens non-functional properties of RTES. The Annex B depicts the necessary background of VSL to express Value Specifications. VSL allows for the describing of literal values, mathematical expressions, collections and temporal value and others. Here, this formalism has been adopted in order to standardize expressions and specifications values in the model elements.

The refinements of requirements models and their constraints annotations, through VSL, contributes to diminishing ambiguity in specifications [88]. VSL can be used to express marked values and to define constraints for any model element that is associated to a value specification. Therefore, it possible to verify correctness and consistency of artefacts of the specification process, through automated techniques, from the initial stages of the RTES development. Thus, specification errors may be found earlier throughout the development cycle [2].

4.4.5 Requirements Analysis

Analysis of Requirements seeks to confirm, by examination, that requirements are well-formed and able to describe the requested system. MARTeSys^{ReqD} methodology defines a strategy to formally analyze and verify timing constraints of RTES requirements, at the outset of the Requirements viewpoint specifications.

The analysis of requirements, in the Requirements viewpoint, starts with a simple reading of the requirements document. Regarding the MARTeSys^{ReqD} methodology, this document depicts the system requirements, already specified in natural language (NL), in a tabular manner. Besides, it considers keywords to determine prioritization of requirements and timing concerns such as deadline, period, events according to Table 1. The proposed transformation considers three timing concerns: **period**, **deadline** and **event occurrences**. To perform the proposed analysis from already specified requirements, from the High-Level Description of Requirements (Section 4.4.2), must be manually checked in order to discover:

- ❑ **Period constraints** defines a precise and predefined time interval in which a task instance occurs (is activated) over time.
- ❑ **Deadline constraints** define a time in which a task must be finished.
- ❑ **Events occurrence** is a situation or stimuli that occurs during the system lifetime. RTES behavior and the scheduling of its tasks can be triggered by internal or external events. For example, a task can be triggered by clock interruptions that occur periodically or through an external user action, as in the pressing of a button.

MARTeSys^{ReqD} methodology considers the analysis of **period**, **deadline** and **event** constraints, from Requirements viewpoint specification, in order to check early timing constraints. The proposed evaluation is important, in this study, since it allows for the formal verification of important properties of RTES.

As presented in Chapter 2, TA is defined by a tuple (L, l_0, C, A, E, I). Thus, after the identification of timing constraints, the correlation (**C**) to Timed Automata formalism is performed as follows:

$$\textbf{C:} \left\{ \begin{array}{l} \textbf{Period} \equiv \text{clock (P)} \in C, \text{ guard} \in E, \text{ inv} \in I, \text{ Reset Operation} \in E \\ \textbf{Deadline} \equiv \text{clock (D)} \in C, \text{ guard} \in E, \text{ inv} \in I, \text{ Reset Operation} \in E \\ \textbf{Events} \equiv \text{channel} \in A, \text{ Action} \in A \end{array} \right\}$$

Typically, in RTES specification, period is defined as the distance between two instants in time or duration [179]. Deadline is defined as an instant in time in which the system must produce a result [179]. However, the proposed specification, in the Requirements Analysis, assumes that these time values are considered as clock specifications in

order to describe/map these into the TA formalism. Requirements labeled with periodic constraints must attend the following design guidelines in order to consistently create the correspondent TA:

1. Define the initial location l_0 of the automata.
2. Create for each solution path, where $l \in L$ location, an ending state, in accordance with Equation 2, as described below.
3. The must exist a loop from each ending location l_i to l_0, where $l_i \in$ set of ending locations of intermediary design paths. Thus, $\exists\ e \in E$ between these two locations.
4. The ending state must have an invariant such that $inv = i \leq$ period.
5. There must exist a guard function that labels e such that: guard(e) must check if clock (related to P) $\geq$ period
6. There must exist a guard function that labels e such that: guard(e) must reset the clock (related to P), which sets the expected period.

Based on the previous guidelines, it is possible to design TA while considering criteria of interest as, for example, timing constraints. This strategy formalizes RTES constraints from high-level assumptions and allows for their formal verification and validation, by specialized tools as, for example UPPAAL, in the initial design steps.

$$\forall r_w\ \exists\ l_t \in L \mid l_t\ describes\ the\ group\ of\ the\ states\ relate\ to\ r_w \tag{1}$$

$$\forall l_i\ \in\ l_{endPath}\ \exists\ e_j \in E \mid e_j \Rightarrow (l_i, l_0) \begin{cases} l_i\ has\ an\ invariant \Rightarrow inv\ \leq\ P \\ e_j\ has\ a\ guard\ function \Rightarrow clock(P)\ \geq\ |P| \\ e_j\ has\ a\ reset\ function\ where\ clock(P) = 0. \end{cases} \tag{2}$$

Equation 1 defines that each atomic requirement (r_w, where $r_w \in R$ and $w \leq |R|$ can be modeled by l_t locations, where $l_t \in L$. Thus, the "meaning" of one requirement is designed by locations (including initial and final locations) and necessary invariant, guard and reset components. In Equation 2, $l_{endPath} = \{l_0, ..., l_i\}$ is composed of the set of ending locations, where, $l_{endPath} \supset L$ and $1 \leq i \leq |L|$. The edge set is represented by $e_j \in E$, thus $1 \leq j \leq |E|$ and the periodic clock is labeled as P. As previously suggested, the path represented by the l_t locations must have a final state $l_i \in l_{endPath}$, where for the group $r_w\ l_i \cup l_t = L$. This shows that an automata path fulfills the meaning of r_w, in terms of periodic execution of a system requirement.

As presented in the correlations rules an event description, from the requirements specification, can be designed by a channel between two TA and a guard condition. A

channel denote action–co-action and it defines synchronization links between the main and secondary timed automata. Equation 3 shows the formalization of an event constraint.

$$\forall ev_i \in Events \begin{cases} \exists\, a_j \in A \\ \exists\, c_w \in Channel\ between\ (e_m, e_n) \in E. \end{cases} \tag{3}$$

In Equation 3, $Event \in A$ denotes the set of system events. In this equation, A represents the of actions, co-actions and the internal τ-actions of the automata. In this case, for each event $ev_i \exists\ a_j label$ for denoting an action–co-action. Thus, there is a label ($a_j label$) to stamp the channel synchronization. The *Channel* describes the synchronization path between two automata. This path describes a **consumed relationship** e_m and a **produced relationship** e_n. In the consumed relationship, the event ev_i triggers e_m and produces e_n. The produced relationship generates ev_i to trigger e_m.

To express deadline constraints, from natural language descriptions to TA formalism, it is necessary to design clocks, invariants and Reset Operations, under a set of $L \cup E$, to symbolize the expected requirement behavior. Moreover, it assumes that deadline defines a specific clock named as D. Equation 4 denotes how to design an automata path respecting deadline constraints from requirement specifications in NL.

$$\forall l_i \in L^{'} \begin{cases} 1^{st}: l_i = \text{inv} \leq \text{D, guard} \geq D \text{ and Reset } D \\ 2^{nd}: L^{'} = \sum_i^n t(l_i) |\ inv_i \leq t(l_i)\ and\ \forall inv_i \exists\ guard \geq t(l_i). \end{cases} \tag{4}$$

In Equation 4, $L^{'}$ maintains the set of locations constrained by deadline, where $1 \leq i \leq |L^{'}|$. Adoption of an invariant (inv) forces the automata to exit of a current state. The value of inv is equal to the period where "an action is into one state". Thus, it is possible to state that $inv_i = c_i$, where c_i is the computation time of l_i location with inv_i. As noted from Equation 4, there are two mutually exclusive design paths. The first rule describes the case where one state is sufficient enough to design the correspondent TA and, in this case, $C = D$ and $i = 1$. The second case, shows that there exist l_i locations, with $i > 1$ and even that the clock is equal to D, the value of the inv of each state may vary. Equation 5 denotes the timing function t; this function is responsible for setting the invariant value of a place:

$$\begin{aligned} t(l_1) &= ds_1 \\ t(l_2) &= ds_2 \\ ... &= ... \\ t(l_i) &= d_i - \sum_i^{n-2} t(l_i) \end{aligned} \tag{5}$$

Thus, the function $t(l_i)$ describes the time portion of l_i, where $d = ds_i, ds_2 + ... + d_i$.

Annex B describes the formal grammar, proposed in the MARTeSys^{ReqD} methodology, in order to transform requirements specification, written in natural language, to Timed Automata. This formalism specifies formal rules which must be applied to generate Natural Language to Timed Automata (NL-TA) transformation in the Analysis of Requirements activity.

4.4.6 Application of the Requirements Viewpoint

Chapter 6 presents, in Section 6.2.1, artefacts from the activities of Requirements Pre-Analysis, High-Level Description of Requirements, Composition of Models using the MARTE Profile, Formal Specification with VSL and Analysis of Requirements. MARTeSys^{ReqD} Methodology is applied in an industrial case study related to automotive systems.

The activity of **Requirements Pre-Analysis** is detailed in Section 6.2.1.1. As detailed on Table 3, this activity provides as outputs a tabulated and classified group of requirements by their description, type, non-functional features and MARTE packages. Figure 13 shows, in part, an artefact from the Requirements Pre-Analysis focusing on the turn and hazard features of the Turn Indicator (see Chapter 6, Section 6.1.2). In sequence, slices of the proposed models are presented in order to clarify and contextualize the MARTeSys^{ReqD} Methodology application.

Requirement Description	Type	NFR	MARTE Packages
The turn indicator system has to identify when the indicator lever is pressed in order to control the car direction changes.	F	-	
The turn indicator functions have an overall response time of 540 ms.	NF	Timing	NFP/Time
The turn indicator system can indicate anomalies and hazardous situations of the vehicle. Hazards occurrences activate hazard lights and can turn a sonorous warning to the driver.	F	-	

Figure 13 – Example of Application: Requirements Pre-Analysis.

Section 6.2.1.2 gives details of the artifacts of **High-Level Description of Requirements**. This activity receives as input the artifacts from the requirement analysis, refines and categorizes them. Figure 14 depicts a slice of the high-level description activity for the Turn Indicator system.

Section 6.2.1.3 refines the Requirements viewpoint, in the activity of **Composition of Models using the MARTE Profile**, through the SysML Requirement diagram. Here, the system requirements are designed in graphical view and, as expressed in Figure 15, and MARTE stereotypes RTES allow one to annotate concerns/non-functional requirements.

Stereotyped constraints of the Composition of Models activity can also be refined. The activity of **Formal Specification of Requirements with MARTE and VSL**, of the

Context	ID	Requirement Declaration
TI	R9	**The turn indicator shall recognize the different lever events in order to provide the right turn direction.**
	R10	The feature turn indicator **shall** visually indicate the direction changes of the vehicle when the indicator lever is pressed in a hard/soft manner
	R11	The turn indicator feature **must** process itself with a maximal response time of 540 ms.
HI	R21	**The hazard feature shall activate the hazard lights.**
	R22	The hazard feature **shall** activate sonorous warning to the driver.
	R24	The hazard feature **must** process itself with a maximal response time of 540 ms - response time of the turn indicator feature.

Figure 14 – Example of Application: High-Level Description of Requirements.

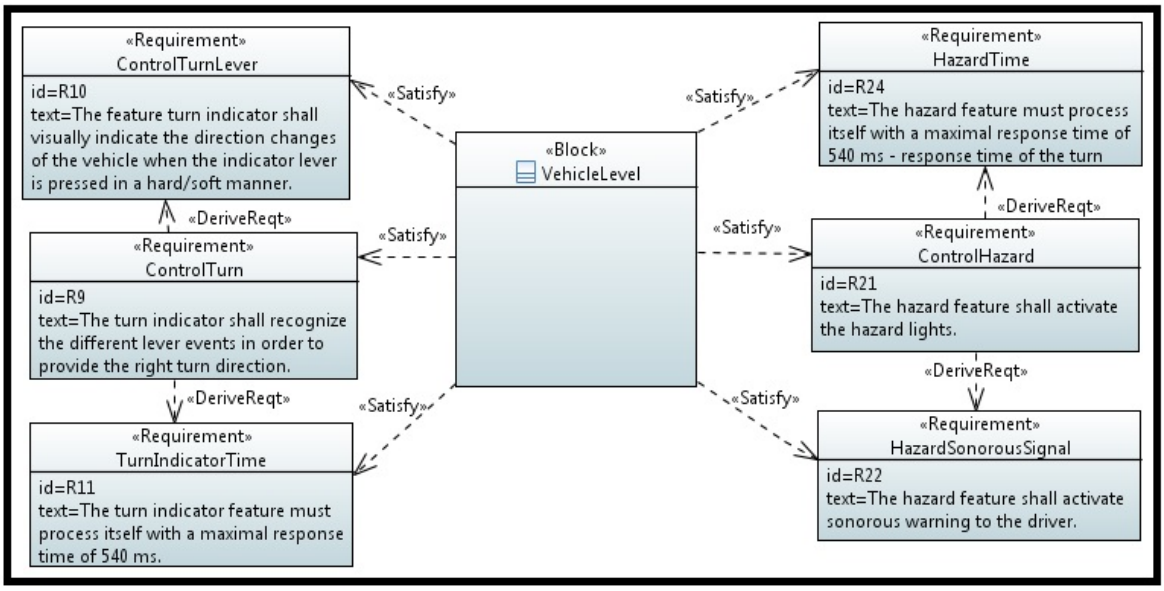

Figure 15 – Example of Application: Composition of Models using the MARTE Profile.

Requirements viewpoint, allow for the annotating of non-functional RTES requirements and properties by a concrete syntax. Figure 16 describes an example of timing constraint annotation through MARTE stereotype and VLS syntax.

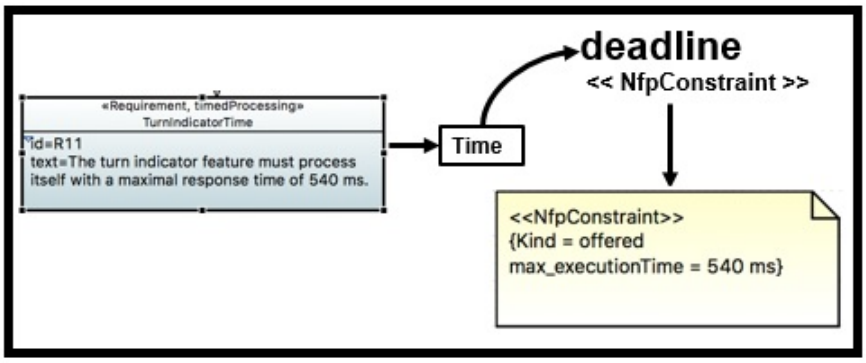

Figure 16 – Example of Application: Stereotyped annotations by VSL Formalism.

The **Analysis of Requirements** applies a formal grammar (see ANNEX B.1) to design RTES constraints from requirements specification. Section 7.2 shows, in the proposed case study, how to transform and verify timing, safety, deadlock and reachability constraints, from early artefacts of the Requirements viewpoint, to the Automata models.

Figure 17 depicts, in a generic manner how deadline and periodic constraints can be realized by an automata, from the requirements specification. It provides a group of guidelines to design and verify timing constraints from early design phases.

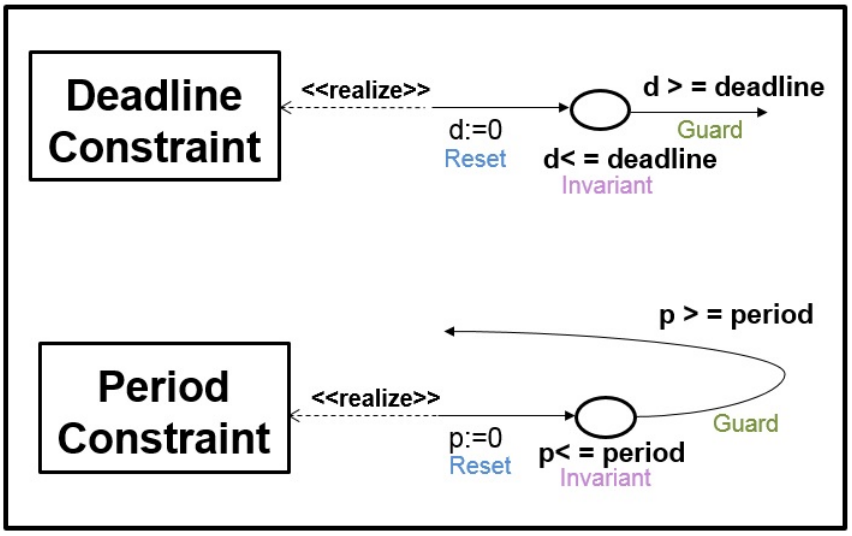

Figure 17 – Example of Application: Analysis of Requirements.

4.4.7 Summary of Requirements viewpoint

RTES have specific needs that should be considered in their description, specification, modelling and design. Requirements viewpoint of MARTeSys^{ReqD} methodology defines a group of activities for RE in order to contemplate the specificities of RTES. Table 5 summarizes the main modelling concepts adopted in each refinement of the Requirements viewpoint.

Requirements viewpoint	Design Strategy
Requirements Pre-Analysis	Extension of SysML Table
High-Level Description of Requirements	Extension of SysML Table, Use Case Diagram
Composition of Models using the MARTE Profile	SysML Requirement diagram
Formal Specification with VSL	Enrichment of Requirement diagram with VSL
Analysis of Requirements	Timed Automata

Table 5 – Modelling Concepts and Strategies adopted in the Refinements of the Requirements Viewpoint.

4.5 Functional Viewpoint of the MARTeSys^{ReqD} Methodology

Functional viewpoint specifies user needs and system requirements, already specified in the Requirements viewpoint, in rational and consistent models. Thus, the input of this viewpoint are artefacts from the Requirements viewpoints, and it provides as output the system services components. Regarding MARTeSys^{ReqD} methodology, this viewpoint employs different SysML diagrams in order to describe structure and general organization of the system and its logical decomposition.

Functional viewpoint has three refinements of system functions and early descriptions of timing concerns. Here, the first models focus on the detailing of system components and their internal relationships. The second refinement aims at describing and detailing their internal and external system functions. Finally, the last refinement of Functional viewpoint aims to add non-functional annotations into functional components. Table 6 shows the main diagrams and concepts adopted in this viewpoint.

Functional viewpoint	Design Strategy
First Refinement	SysML Block Diagram
Second Refinement	SysML Internal Block Diagram
Third Refinement	Time Package of the MARTE Profile

Table 6 – The Concepts adopted in the Refinements of the Functional viewpoint.

In this activity of architectural design, it is important to map and transform models from the last refinement of the Requirements viewpoint to the first refinement of the Functional viewpoint models. This mapping provides the link between the artefacts of both viewpoints and clarifies the relationship between functional and logical blocks. As presented on Table 6, SysML Block diagram is adopted to design the system components in the first refinement.

Thus, it is necessary to map artefacts, already modelled from the Requirements viewpoint, to a SysML Block diagram, which corresponds to the first view of the Functional viewpoint. Therefore, scenarios presented in Use Case diagram and on the Categorization Table (Table 4), from the activity of High-Level Description of Requirements, are

employed to delimit the main system services. Each category of service will be refined by a structural block. This refinement is responsible for showing software and data components and, also, a subdivision of architectural subsystems and their relationships.

SysML Internal Block performs the second refinement of the Functional viewpoint. This viewpoint models the system components, their internal characteristics, as well as their communication interfaces. In the last refinement of the Functional viewpoint, architectural models will be detailed with timing constraints information. *CoreElements*, *NFP*, *Time*, *GRM*, *HLAM*, *GCM*, *HRM* and *SRM* packages can have their stereotypes adopted here since they can contribute with logical/physical constructors and with concepts to quantify characteristics of elements.

Model elements of Functional viewpoint are refined in the Logical viewpoint architecture. In a general sense, the Logical viewpoint should show, the logical components, which are able to define the user's functions. This viewpoint is described in the next Section.

4.5.1 Application of the Functional Viewpoint

Chapter 6 presents, in Section 6.2.2, the architectural design models of the Turn Indicator regarding to the Functional viewpoint. As previously mentioned, this viewpoint applies the Block Definition diagram and Internal Block diagram, together with MARTE stereotypes, in distinctive system views. Figure 18 shows some examples of the Requirements viewpoint refined in distinctive models of the Functional viewpoint.

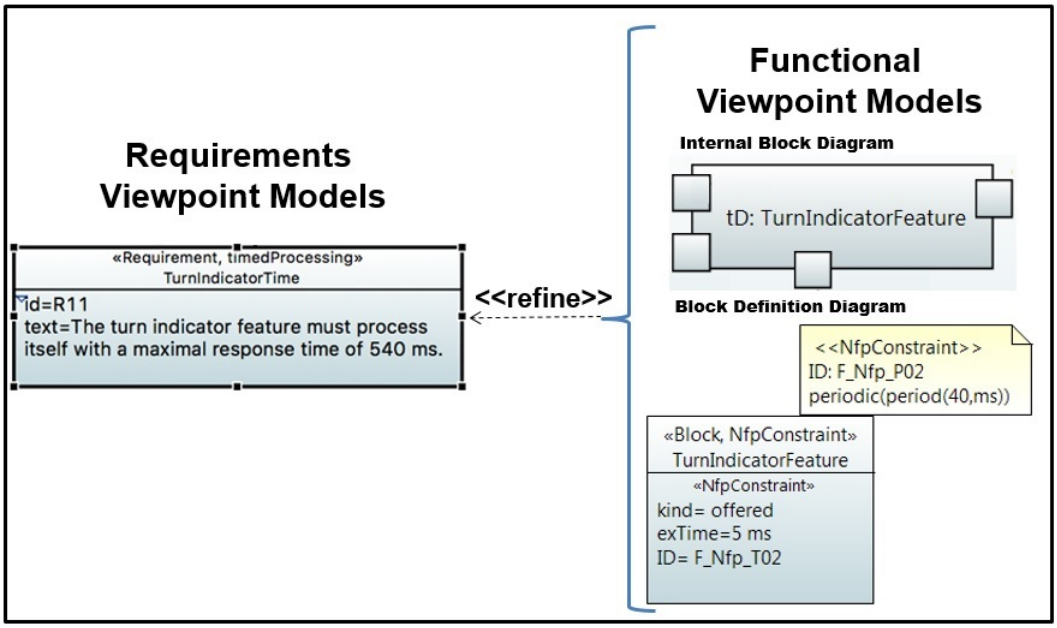

Figure 18 – Example of Application: Functional Viewpoint.

A complete view of the Functional viewpoint is presented in Section 6.2.2.

4.6 Logical Viewpoint of the MARTeSys^{ReqD} Methodology

The Logical viewpoint describes the decomposition of the system functions, from the Functional viewpoint, into an architecture of logical components. These components define a logical solution, which is independent of the domain and of any technological constraints. In this viewpoint, the logical architecture is gradually refined into a high-granularity level of logical components. Thus, the Logical viewpoint provides the "logical structure and the distribution of responsibilities functionality of a system" [142]. The views of the Logical viewpoint, and their respective models, must provide, as output, a group of logical components that are responsible for a set of functions.

In accordance with [18], the correlation between an artefact of Functional viewpoint (user function) and those from Logical viewpoint (logical component) is not clear. However, the MARTeSys^{ReqD} methodology provides a strategy, based on relationships between artefacts, to map these viewpoints. Mapping between the Functional viewpoint and the Logical viewpoint is a mandatory activity. Figure 19 graphically depicts a proposal to transform and correlate models of Functional and Logical viewpoints. This mapping describes the first correlation of functional and logical components. The proposed mapping is performed using the SysML Block diagram, and which is further refined through other diagrams in the Logical viewpoint.

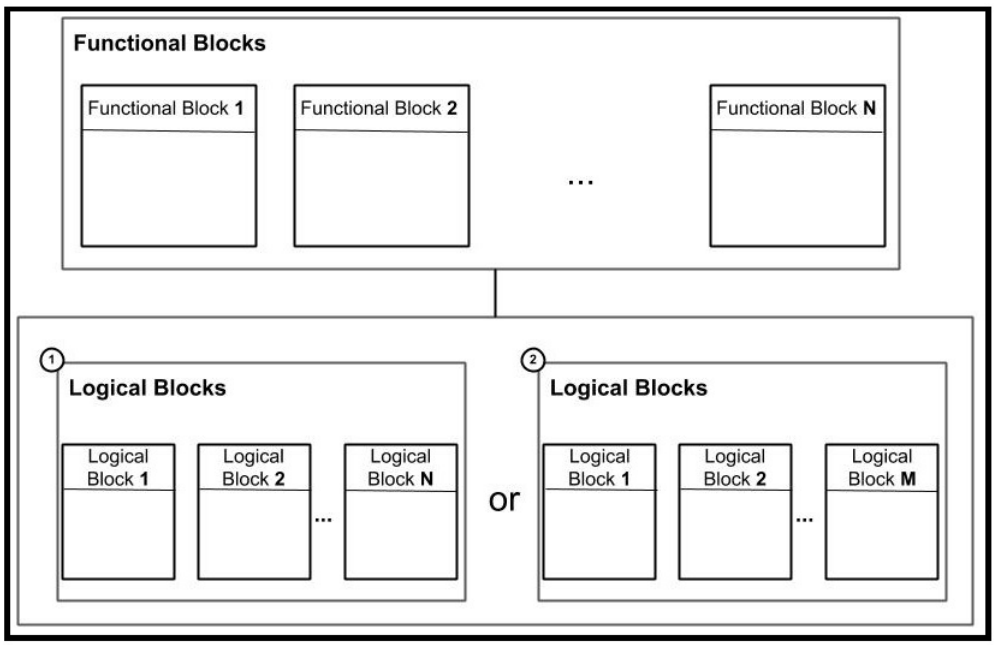

Figure 19 – Mapping between the Functional viewpoint and Logical viewpoint.

As noted from Figure 19, there is a relationship (N:M) which traces user functions and logical functions, where N represents the number of functional blocks and M the

number of logical blocks. Functional blocks can generate N logical Blocks, when $N = M$, see label 1 in Figure 19, or M logical Blocks, with $M < N$, see label 2 in Figure 19. In the latter case, for each user function, there is a group of logical components, from the Logical viewpoint, which is able to realize this functional artefact. However, it is possible to put together in one logical block one or more functional block(s), when $M < N$, when these functional blocks may be logically structured into one single logical component.

In a general sense, this viewpoint is composed by two refinements and all the models of the Logical viewpoint are designed with SysML Activity diagrams. Moreover, this diagram is adopted here, as it allows for the description of the system under development and the logical components, from the Functional viewpoint, from a behavioral perspective. In this architectural level, it is important to design models that are able to represent the functional system components as a rational group of logical components.

The first refinement of the Logical viewpoint aims at defining the internal behavior for each logical block. Here, the Activity diagram is employed using its original definition and the internal signals, fork, merge, decision nodes and atomic activities of the system are modeled. In order to enrich logical activities and the input/output signals of these activities the MARTE profile stereotypes were adopted in the second and third refinement of the Logical viewpoint models.

In the second refinement, each activity was constrained with a specific timing stereotype of the MARTE profile. These stereotypes allow for enriching the diagram with individually pre-fixed deadlines and the description of periodic behaviors. From these annotations, based on specific features of the application domain, it is desirable that the artefacts from the Logical viewpoint may be susceptible to different types of analysis, such as checking non-functional constraints.

The MARTE profile can also be adopted into the elements of the Logical viewpoint to detail the type of interaction from/to a logical component. In the third refinement, other MARTE constructors, especially from HRM and SRM packages, are adopted to identify the communication interfaces between the logical blocks. In this abstraction level, low granularity information about one activity or its interfaces can be highlighted. It allows for the description of physical and digital input/output and the detainment of different non-functional criteria, such as concurrency, distribution and performance. Artefacts from this viewpoint are further refined into Technical viewpoint models.

4.6.1 Application of the Logical Viewpoint

Chapter 6 presents, in Section 6.2.3, the architectural models of the Turn Indicator designed in the Logical viewpoint. Before modelling logical views, it is necessary to understand how the Functional viewpoint is refined by logical decisions.

Figure 20 shows one example of mapping between the mentioned viewpoints. The

mapping shows functional and logical blocks of the TIS. There exist an $n : m$ relationship that maps user functions and logical functions [18]. In this sense, for each user function there is a group of logical components (from the logical viewpoint), which is able to refine/realize this functional artefact.

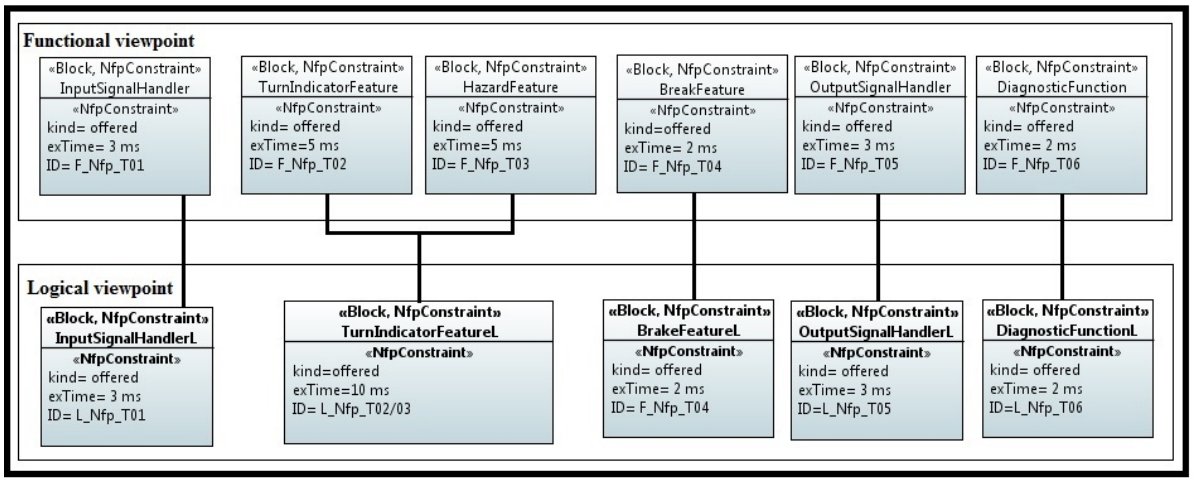

Figure 20 – Mapping between Functional and Logical viewpoints.

The Logical viewpoint should describe the system under development and the logical components, refined from the Functional viewpoint, in a behavioral perspective. Figure 20 allows one show graphically the design decisions between these two viewpoints. Moreover, for larger numbers of functional blocks, it is important to show how functional blocks are refined, since it rationally presents their compositions and refinements. The refinements of each logical block (see Figure 20) are performed through Activity diagrams. Section 6.2.3 shows the internal behavior of these logical blocks.

4.7 Technical Viewpoint of the MARTeSys^{ReqD} Methodology

In the Technical viewpoint, the software and the hardware components are combined [18]. Technical models allow for the defining of diverse physical and deployed viewpoint perspectives in accordance with previous design decisions. Therefore, overall models of this viewpoint have to represent the resources that are able to produce the logical components from the Logical viewpoint.

Contained within this viewpoint, the design decisions should represent the main interactions between software and hardware and the necessary refinement of the logical subsystems. Moreover, design decisions should show how hardware components as, for example, ECUs, memory, communication channels and peripheral devices are coupled into the system.

The artefacts from the Logical viewpoint, mainly models from the Activity diagram, are refined here using swimlanes. The main objective of this refinement is the decomposition and partition of the system into logical and physical components, which interact to satisfy the general system requirements. In this methodology, in order to represent a technical perspective of the system, a swimlane can represent either a software component or a hardware/physical component. Thus, peripheral devices, for example, can define a single swimlane and it allows for partitioning actions based on the involved participants. Therefore, a group of activities can be grouped into the same swimlane if these activities relate to the software or hardware component.

From this system partitioning, another refinement is performed to detail the main types of signals that interact with a hardware/software component. It describes the design decisions regarding digital and analog communications for one component. Furthermore, this strategy allows one to highlight the physical devices that are part of the RTES. In both cases mentioned, MARTE stereotypes can be adopted to identify the system interfaces (analog or digital signals) and to describe physical resources that can be used by the application. Moreover, the MARTE constructors can be used in new refinements of timing/non-functional properties over technical models and the different architectural decisions for the system development.

Artefacts from the Technical viewpoint presents a refined and closer view of system realization. Thus, annotated constraints may contribute to guide developers toward designing global models of system architecture. Furthermore, these low-level annotations can also direct the definition of MUD models and create the code implementation.

4.7.1 Application of the Technical Viewpoint

Chapter 6 presents, in Section 6.2.4, the architectural models of the Turn Indicator regarding to the Technical viewpoint. Figures 40, 41 and 42 show the technical design decisions and TIS models.

4.8 From Architectural Viewpoints to Global System Architecture

The Global System Architecture depicts a structure of system components and their relationships. The model of the Global System Architecture comes from the final models of the Technical viewpoint. Figure 21, Phase 4, shows a software component, named *SW_TurnHazardIndicator*, modeled using the SysML Block diagram. Within this model, hardware and software components are separately modeled and their external communication are also defined. Definition of the system view is important to show the system design in a complete perspective, as it defines how each software/hardware component

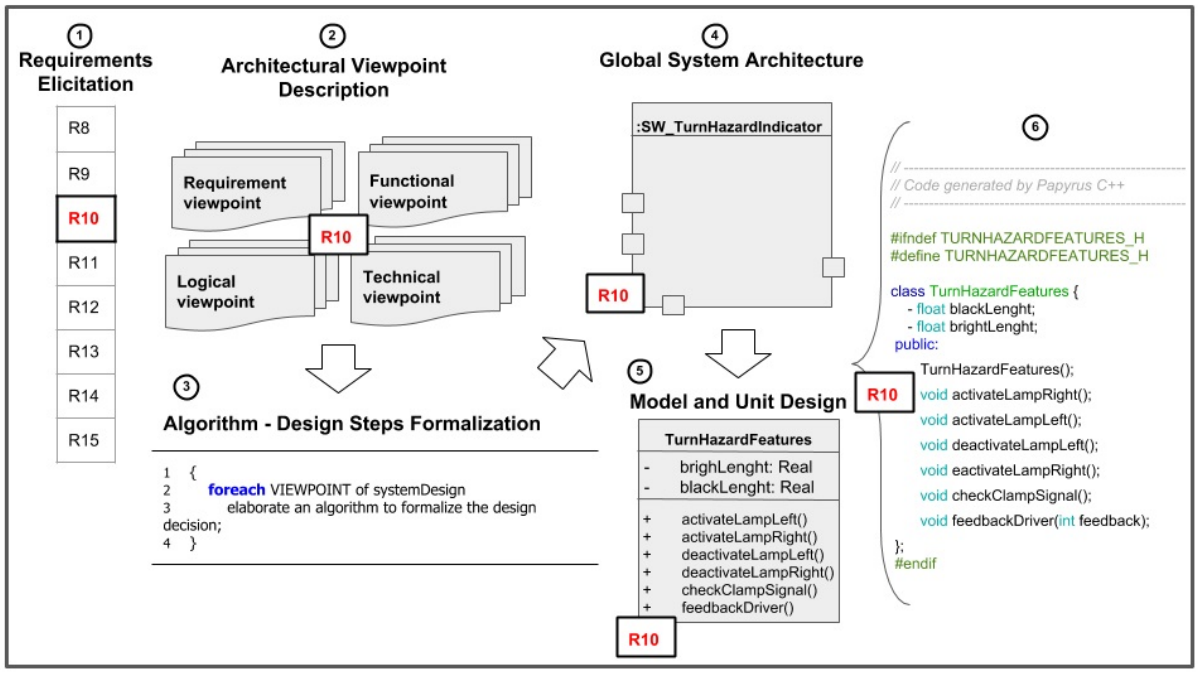

Figure 21 – General Framework of the MARTeSys^{ReqD} Methodology.

cooperates with each other. Chapter 6 provides a complete example of the Global System Architecture.

Figure 21 shows a global view of the MARTeSys^{ReqD} methodology from the Architectural viewpoints (Phase 2) to MUD view (Phase 5). MUD takes the software components already defined in Global System Architecture (Phase 4) and refines these into viewpoints closer to implementation models. Here, Class has been designed with focus on system features. These system features were already described in the Requirements viewpoint. Moreover, RTES constraints/stereotypes were also refined and kept, in order to guarantee the non-functional properties of system services. Through the Class diagrams, and their fixed constraints, Automated Code Generation (Phase 6) and final system evaluations are performed.

Formalization of design steps (Phase 3) and Automated Code Generation (Phase 6) are the respective subjects of the next chapters. As portrayed in Figure 21, the requirement named as **R10** is traced in all the design phases. MARTeSys^{ReqD} methodology presents a strategy to trace non-functional concerns along with the architectural viewpoints. Thus, the following section describes how RTES concerns can be annotated and traced in design models.

4.9 An Strategy to Trace Real-Time Embedded Systems Constraints in Architectural Viewpoints

Traceability of RTES constraints is performed in this thesis by the adoption of enumerated and standard labels. A strategy is developed to trace the constraints along the system viewpoints. A specific constraint has different nomenclature in its ID at distinctive abstraction levels. However, the integer number, which is part of the ID does not change in each viewpoint.

In all the architectural models each constraints of the MARTE profile is labeled as a specific "ID". Figure 22 shows the trace of non-functional constraints at different viewpoints. For example, a periodic constraint, of the Logical viewpoint, has the following L_Nfp_T"x" identification where:

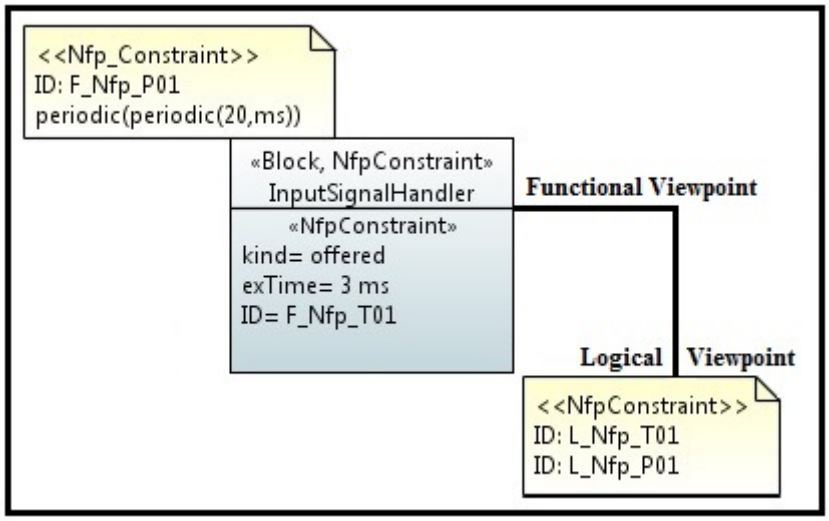

Figure 22 – Trace of $\ll NfpConstraint \gg$ from Functional Viewpoint to Logical Viewpoint.

- ❏ The first parameter identifies the viewpoint. For example, **L** corresponds to the Logical viewpoint.
- ❏ The second parameter relates to the type of the constraint. In the example, **Nfp** recognizes a non-functional constraint.
- ❏ The third parameter defines an immutable number to identify the constraint. The variable "**x**" is an integer variable. This number represents a value that traces and distinguishes a constraint in different views and viewpoints.
  - In the case where the constraint is labelled with $\ll NpfConstraint \gg$ stereotype, the label **P** or **F** is also applied to the third part of the ID. These characters describe if $\ll NpfConstraint \gg$ relates to a periodic time (**P**) or

timing duration (**T**). For example, **T** sets this constraint as the timing duration value (for example, deadline);

Specification, verification and maintenance of RTES constraints over their development is a challenge. Methodologies to support the analysis and verification of these constraints have a great impact on their development. MARTeSys^{ReqD} methodology applies the proposed labels to trace non-functional and embedded constraints at different viewpoints. This strategy guides the designers to annotate non-functional information, from the initial specification models to the system implementation models, in a standard manner.

An example of adoption of the proposed methodology is performed in Chapter 6 and it contributes to clarifying the design decisions, while illustrating the proposed methodology. Further, Chapter 6 provides an example of the traceability criteria from graphical models to implementation code.

4.10 Contributions

RTES design must consider the different non-functional concerns, the cooperation criteria between several system components and respect possible real-time communications. This chapter introduces the MARTeSys^{ReqD} methodology and its guidelines to RTES design. It presents MARTeSys^{ReqD} contributions regarding phase 2 of Figure 21.

The MARTeSys^{ReqD} methodology presents in detail a modelling methodology for RTES applications. The proposed methodology employs the SysML and MARTE profiles as the predominant languages and the SPES guidelines to support the viewpoints definition and their refinements. Thus, software designers and developers can consistently and unambiguously capture, analyze and specify software components and, while generating different views concerning the proposed system.

This chapter details the some contributions of this thesis already detailed in [94], [95] and [96]. Therefore, it defines distinctive strategies for modelling RTES constraints, which are imposed on different views, and describe how to annotate and refine these constraints throughout the architectural definition. Moreover, this thesis contributes with strategies to describe non-functional requirements, in order to analyze and make these these comprehensible to the stakeholders involved, since these can check necessary information by looking at different architectural viewpoints. In addition, as presented in [97], a manageable and consistent representation of these requirements contribute to their trace along the architectural design.

Chapter 5

Formalization of MARTeSys^{ReqD}

This chapter complements the overall descriptions and foundations of the MARTeSys^{ReqD} methodology. It provides a formalization of the proposed methodology, through algorithm specifications for design activities, and defines a group of guidelines to be adopted into the RTES domain.

5.1 Formalization of the Design Decision of Architectural Viewpoints

MARTeSys^{ReqD} proposes a methodology for the designing of RTES considering different abstraction levels of architectural design and description, analysis and evaluation of non-functional concerns. A group of algorithms is developed in order to describe the general design decisions of the MARTeSys^{ReqD} methodology. These algorithms adopt formal rules to employ MARTeSys^{ReqD} into RTES development.

The algorithmic formalism provides a consistent and restricted guide for the design of RTES over different viewpoints. while considering the description of non-functional properties. Figure 23 summarizes the proposed formalization and its contributions.

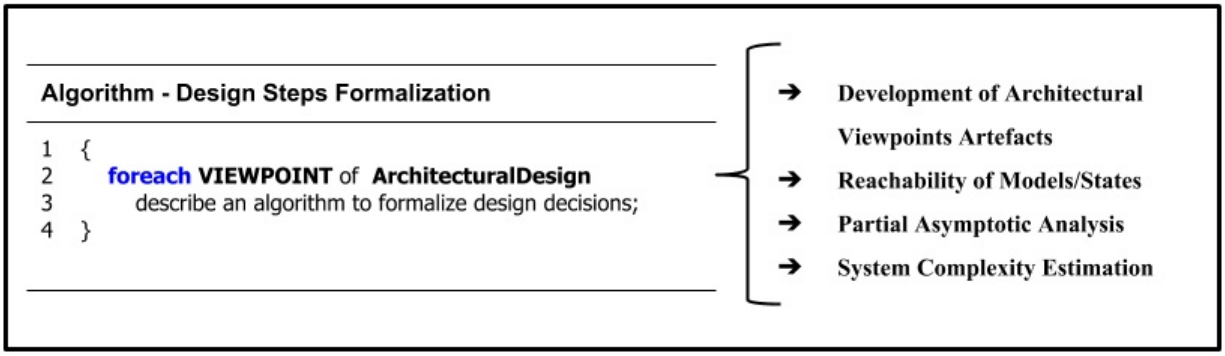

Figure 23 – Contributions of the Proposed Formalization.

As depicted in Figure 23, formalization of the design decisions intends to contribute to: Reachability of Models, Partial Asymptotic Analysis, and System Complexity Estimation. Pseudo-code is adopted to write all activities of the MARTeSys^{ReqD} methodology. It provides an unambiguous manner to follow a group of tasks/activities that must be performed for each specific viewpoint. Therefore, this strategy plays a role in the **Development of Architectural Viewpoints Artefacts** and this is reached by following the formalized design decisions. The formal instructions are mainly represented by sequential, parallel, conditional or repetitive actions. These algorithms aim at providinf a consistent set of guidelines to design RTES regarding the proposed design decisions. This strategy presents a systematic manner to apply, understand and adopt the methodology in order to develop the system artefacts over architectural viewpoints.

Reachability of Models and its states can be achieved through analyzing the adoption of formal algorithms. This formalization allows one to reach and design the global system models, which are first presented in the Requirements viewpoint, and then modeled further and refined in the Functional, Logical and Technical viewpoints. From this, , it is expected that out of the formalized design decisions a full system specification can be achieved. The models/state reachability may provide design artifacts from the Requirements viewpoints towards the Technical Viewpoints models.

Different algorithms are proposed to formalize design strategies to the RTES design of the MARTeSys^{ReqD} methodology. Within these algorithms, complexity analysis from Big O notation [180] is considered in order to express an effort measurement of the design activities. Therefore, a **Partial Asymptotic Analysis** is performed in order to provide an examination of design algorithms at each viewpoint. However, results of a complexity analysis cannot estimate human interactions. As a matter of fact, only a "partial" analysis is considered. Major details regarding user perspective, influence of human interactions in the software development and engineering can be found in [173], [181], [182].

Finally, the global **System Complexity Estimation** aims at providing overall estimations to the system development. Based on the proposed measurements, it is possible to provide a global cost function when using the MARTeSys^{ReqD} methodology, in RTES development, considering the worst-case scenario to the number of design artefacts.

The proposed formalism is helpful to RTES design, as it standardizes its architectural descriptions and provides a formal manner to measure the activities of design. This strategy can contribute with cost prediction measurements of system development (as analyzed in Section 5.3.1), and to analysis and estimations of the maximal number of artefacts of one viewpoint (as analyzed in Section 5.3.2). The algorithms proposed for the Requirements, Functional, Logical and Technical viewpoint provide a formal background for applying to MARTeSys^{ReqD} methodology in order to be manually followed and applied by engineers in RTES design and development.

5.2 Algorithms to Describe the Architectural Viewpoint

A set of elements is defined in order to allow formalization of architectural design components. The input sets for the architectural formalization are:

- R = set of system requirements; where $r_i \in R$.
- C = set of categories; where $c_i \in C$.
- Sc = set of system scenarios; where $sc_i \in Sc$.
- $FuncBlock$ = set of functional blocks; where $funcBlock_i \in FuncBlock$.
- $directEdges$ = set of directed edges; where $intDirectEdges_i \subset directEdges$ and $extDirectEdges_i \subset directEdges$.
- $mBDD$ = matrix of functional blocks ($FuncBlock$), from BDD, where each block $mBDD_{i,j}$ has an input/output edge with the $intdirectEdges_i$.
- $mIBD$ = matrix of internal blocks, from IBD, which models/refines the proposed BDD. Thus, $|mBDD| = |mIBD|$, where each block $mIBD_{i,j}$ has an input/output relationship with the $directEdges$. This relationship can be an internal edge association or a flow port association.
- S = set of signals ($intDirectEdges$ (internal signals) $\cup$ $extDirectEdges$ (external signals)) from the Functional viewpoint.
- $LogBlock$ = set of logical blocks from the Logical viewpoint mapping; where $logBlock_i \in LogBlock$.
- $activityList$ = set of activities of the Activity diagram from the Logical viewpoint; where $ac_i \in activityList$.
- $internalSigList$ = set of internal associations/signals of the Activity diagram from the Logical viewpoint.
- $externalSignal$ = set of external associations/signals of the Activity diagram from the Logical viewpoint.
- $constDetailMatrix$ = it is a bi-dimensional matrix to store one activity, from the Logical viewpoint, and its respective MARTE profile constraints.
- $ListActivities\_IN\_Software$ = set of activities of the Technical viewpoint. This is a container to store activities related to software components.

- $ListActivities\_IN\_Hardware$ = set of activities of the Technical viewpoint. This is a container to store activities related to hardware components.

This formalism allows a concrete manner to reference individual elements of the Requirements, Functional, Logical and Technical viewpoints. Therefore, it enables to formally model a graphical component of SysML/MARTE, as well as representing this component in the proposed algorithms.

5.2.1 Formalization of Requirements Viewpoint Decisions

Requirements viewpoint is composed of the activities related to RE system. As presented in Chapter 4, the MARTeSys^{ReqD} methodology proposes five global activities to describe, specify and analyze requirements of RTES. These activities are named as Requirements Pre-Analysis, High-Level Description of Requirements, Composition of Models using the MARTE Profile, Formal Specification with VSL and Analysis of Requirements. The artefacts from the Requirements viewpoint are structured at different abstraction levels and each level has its own formal description. This description provides a high level of details, that is, for each design decision there is a standard instruction for its adoption.

Initially the system requirements, which were already specified, are refined in the Requirements Pre-Analysis activity. In this case, each $r_i \in R$ must be checked in order to improve its description and provide details related to its non-functional concerns. Algorithm 5.1 describes a formalization to the Requirements Pre-Analysis stage.

Algorithm 5.1 Pre-Analysis - Algorithm to refine the Requirements Specification

```
1   {
2   1 numberOfRequirements = |R|;
3   1 ReqTable = matrix[numberOfRequirements][4];
4   n   for (j= 0; j < numberOfRequirements;  j++){
5   n     ReqTable[j][0] = rewritten r[j] in accordance with priority key-words
6   -   }
7   n   for ( j= 0; i < numberofRequirements; j++) {
8   n     if (ReqTable[j][1] is a Functional Requirement) {
9   A≤n    ReqTable[j][1] = Functional;
10  -    }else{
11  n-A   ReqTable[j][1] = Non-functional;
12  -      }
13
14  n    switch (RFN-Type){
15       case Reliability: ReqTable[j][2]= Receive one of reliability subtypes;
16         case Timing: ReqTable[j][2]= Receive one of subtypes of the timing (time) features;
17       case Precision: ReqTable[j][2]= Receive one of subtypes of the precision (time)
             features;;
18         case Performance: ReqTable[j][2]= Receive one of performance subtypes;
19         case Safety: ReqTable[j][2]= Receive one of safety subtypes;
20         case Distribution: ReqTable[j][2]= Receive one of distribution subtypes;
21         case Interoperability: ReqTable[j][2]= Receive one of interoperability subtypes;
```

```
22      case Embedded: ReqTable[j][2]= Receive one of embedded subtypes;
23        case Resource Utilization: ReqTable[j][2]= Receive one of resource utilization
              subtypes;
24        case Security: ReqTable[j][2]=  Receive one of security subtypes;
25        case Concurrency: ReqTable[j][2]= Receive one of concurrency subtypes;
26        case Flexibility: ReqTable[j][2]=  Receive one of flexibility subtypes;
27        case Maintainability: ReqTable[j][2]=  Receive one of maintainability subtypes;
28        case Usability: ReqTable[j][2]=  Receive one of usability subtypes;
29        case Deadlock: ReqTable[j][2]= Receive one of deadlock subtypes;
30        case default: ReqTable[j][2]= undefined;
31       }
32  n    switch (Package_Type){
33      case CoreElements: ReqTable[j][3]= CoreElements;
34        case NFP: ReqTable[j][3]= NFP;
35      case Time: ReqTable[j][3] =  Time;
36        case GRM: ReqTable[j][3] =  GRM;
37        case GCM: ReqTable[j][3] =  GCM;
38        case HLAM: ReqTable[j][3] =  HLAM;
39      case HRM: ReqTable[j][3] =  HRM;
40        case SRM: ReqTable[j][3] =  SRM;
41       }
42     }
43  }
```

The first group of instructions for Algorithm 5.1 guides refinements of the requirements specification document. Requirements already written in natural language are analyzed to check possible ambiguities. These requirements can also be refined by standard and meaningful keywords to improve their declaration. These keywords are explained in Chapter 4 along with the *High-Level Description of Requirements* view. Moreover, one notes that the three last columns of *ReqTable* are respectively related with the **requirements type** (classified as functional/non-functional), the **specific non-functional type** (it follows the guidelines of Chapter 3) and with the **MARTE package of each** r_i. This refinement contributes to standardize the design decisions of the requirements specification. In addition, it provides the background to future viewpoints definition.

As elucidated in Chapter 4, in addition of the Requirements Pre-Analysis activity (formalized in Algorithm 5.1), the Requirements viewpoint has four other refinements. Annex B, Section B.1, details the proposed formalization to the High-Level Description of Requirements (Algorithms B.1, B.2 and B.3), Composition of Models using the MARTE Profile and Formal Specification with VSL (Algorithm B.4) and Analysis of Requirements (Proposed Grammar to the NL-TA Transformation).

5.2.2 Formalization of Functional Viewpoint Decisions

Functional viewpoint is responsible for refining the system requirements, in order to describe structural and general organization of the system and its logical decomposition. In this viewpoint, the system behavior, the details of the system services and the user functions interaction are explained. Furthermore, the Functional viewpoint aims to

provide a rational representation of system services based on structural components.

The MARTeSys^{ReqD} methodology proposes three refinements to the Functional viewpoint. The first refinement is represented by Algorithms 5.2 and 5.3, Algorithm B.5 describes the second refinement and, finally, Algorithm B.6 depicts the third refinement. Algorithm B.6 details how to annotate MARTE constraints in functional models. These algorithms provide standard guidelines to model/describe a system function at a specific hierarchy level of abstraction. Therefore, the proposed algorithms are able to provide the background and the overall steps to setup the Functional viewpoint.

On the first level of the Functional viewpoint (first refinement) a group of rational decisions in the design is performed. These design decisions are important in this activity, for the architectural viewpoint, as these are supposed to transform the models of Requirements viewpoint into models of the Functional viewpoint. However, before starting out with formalization of design decisions, one should consider rationalizing the mapping between the Requirements viewpoint (last model) and the Functional viewpoint (first model). Therefore, it applies the scenarios from the categorization process (Algorithm B.1) and the Use Case diagram (Algorithm B.2), formalized in the activity of High-Level Description of Requirements, to provide the main system services.

The following three steps show formalization for viewpoints mapping and the performed design decisions:

- ❏ **Step 1**: From the Requirements viewpoint, models for an overall view of the system services, in Functional viewpoint, are is created. It is possible to map, in a 1x1 relation, each use case component to blocks of services. Initially, the number of scenarios must be known in order to define the number of structural blocks ($numberFuncBloc$). This number is represented by the set of system scenarios (Sc).
- ❏ **Step 2**: Algorithm 5.2 presents the proposed formalism to trace the system scenarios (sc_i) to their respective functional blocks ($funcBlock_i$).

Algorithm 5.2 First Refinement - Algorithm to create the system functional blocks from the scenarios

```
1  {
2  1   numberOfScenarios = |Sc|;
3  1   if (numberOfScenarios > 0){
4  1   funcBlock[0] = functional input block;
5  s-2 for (i=1; i< numberOfScenarios; i++){
6  s-2     funcBlock[i] = sc[i];
7  -       }
8  1    funcBlock[numberOfScenarios] =  functional output block;
9  -    }
10
11 }
```

Algorithm 5.2 defines three important steps to guide the design of the Block Diagram from the scenarios of the Requirements viewpoint. Line 3 shows the creation of a predefined input block, lines $4-6$ create one functional block for each system scenario and line 7 produces a predefined output block. The two blocks defined in lines 3 and 7 must be created to represent the I/O interface. These blocks represent the possible interactions or communication of the system.

❑ **Step 3**: Algorithm 5.2 provides the structural components of the Block Definition diagram. However, in this first refinement, it is necessary to consider the relationships between functional blocks. Therefore, links between these blocks should be modeled. Algorithm 5.3 formalizes relationships between blocks, and it shows how exchanged signals can be represented in a high level of abstraction.

Algorithm 5.3 First Refinement - Algorithm to create the block associations of the Block Definition diagram

```
1   {
2   1       internalS = |intDirectEdges|;
3   1       numberFuncBloc = |FuncBlock|;
4   1       mBDD = matrix [numberFuncBloc][2];
5   1       intSig[internalS] = vetor with internal signals;
6   n_i     for (i=0; i <= internalS; i++){
7   n_i*n_b   for (j= 0; j <=numberFuncBloc; j++){
8   n_i*n_b     if (intSig[i] is output of funcBlock[j]){
9   A≤n_i*n_b    mBDD[i][0] = funcBlock[j];
10  A≤n_i*n_b     break;
11  –           }
12  –          }
13  n_i*n_b   for (j= 0; j <=numberFuncBloc; j++) {
14  n_i*n_b     if (intSig[i] is input of funcBlock[j]){
15  B≤n_i*n_b    mBDD[i][1] = f[j];
16  B≤n_i*n_b    break;
17  –           }
18  –         }
19  n_i       if ((mBDD[i][0] <> empty) and (mBDD[i][1] <> empty)) {
20  C≤n_i       create an association from mBDD[i][0] to mBDD[i][1];
21  –           }
22  –       }
23  }
```

Algorithm 5.3 checks correlated blocks and also shows how to create their internal relationships. Two blocks are combined into a single block, if they have an association between each other. The $mBDD$ matrix stores, in the first column, the output block for a signal $intDirecEdges_i$ and, in the second column, the input block for a signal $intDirectEdges_i$. Vector $intSig[i]$ contains the internal directed edges of Functional blocks. Then, the main task of this algorithm is to associate the components of $mBDD$ matrix to the $intDirecEdges_i$ elements. This formalization shows the design of the functional blocks through MARTeSys^{ReqD}.

In Algorithm 5.3, for each directed edge or internal signal there is only one specific label/representation. It means that if an $intDirecEdges_i$ is input or output of a block it can not be related with another block. The instruction "Break", lines 10 and 16, indicates that a specific internal signal relates exclusively to one internal port or flow port. In addition, each internal signal is interactively checked in order to find the output and input blocks that are linked to $intDirecEdges_i$. Thus, lines $7-11$ formalize the discovery of the output blocks that are related to an internal signal. Lines $13-18$ show the input block to an $intDirecEdges_i$. Finally, lines $19-21$ represent the internal communications between the functional blocks. Annex B, Section B.2, clarifies the second and third refinements of the Functional viewpoint.

5.2.3 Formalization of Logical Viewpoint Decisions

The Logical viewpoint aims at decomposing into a high granularity architecture the blocks of logical components in order to detail the system functions of the logical components. Algorithm B.7, presented in Annex B, Section B.3, maps Functional and Logical viewpoints. This formalization aim to show how a functional block should be replicated without modification, in the Logical viewpoint. The first models of the Logical viewpoint are designed based on the artifacts that result from this mapping.

The SysML Activity Diagram is applied to the three refinements of this viewpoint. Although the Activity diagram has been widely adopted, in this viewpoint, it is important to highlight that different views, with high and low granularity information, are described here. Moreover, it allows for annotations of timing and communication criteria to the system activities and their interactions. Therefore, the MARTE profile stereotypes have been adopted in the second and third refinement of the Logical viewpoint models, in order to describe non-functional properties in the behavioral models.

Algorithm 5.4 formalizes the first refinement of the Logical viewpoint. The proposed refinements adopts the SysML Activity diagram at different abstraction levels in order to provide a behavioral perspective for each logical block. Algorithm 5.4 guides the proposal of a group of atomic activities to represent input of the logical block, output logical block and features logical blocks. The last block is related to the main system functions/services.

Algorithm 5.4 Logical viewpoint - This algorithm contains the steps to the design of the Logical viewpoint models

```
1 {
2  1 externalSigList = list of external signals;
3  1 activityList = list of activities;
4  1 internalSigList = list of internal signals;
5  1 numberLogBloc = |LogBlock|;
6  1 actI = list of elements from the Logical Input Block;
7  1 actO = list of elements from the Logical Output Block;
```

```
1 actGen =  list of elements from Feature Blocks;
nb for (i = 0; i < numberLogBlocks; i ++) {
nb  if ( logBloc[i] == 'input Logical Block'){
A≤nb  actI = LogicalViewpoint_Input;
–    }else{
nb−A    if (logBloc[i] == 'output Logical Block'){
B≤nb−A    actO = LogicalViewpoint_Output;
–              }else
C − nb−A  actGen = LogicalViewpoint_FeatureBlocks;
–     }
– }
}
```

As it can be observed in Algorithm 5.4, there is a "call" to other algorithms from the Logical viewpoint. The Annex B, Section B.4, explains the Algorithm B.8 (input logical blocks), Algorithm B.9(output logical blocks) and Algorithm B.10 (feature blocks) which formalizes the steps to design the Logical viewpoint.

5.2.4 Formalization of the Technical Viewpoint Decisions

Artefacts of the Technical viewpoint are related to the final system implementation and these should ensure that the design decisions are in accordance with logical viewpoint assumptions and with logical and physical system constraints. The Technical viewpoint refines the logical viewpoint models and splits their elements into hardware and software components. Besides, this viewpoint illustrates interactions between software and hardware components.

This viewpoint refines previous Activity diagrams of the input logical block, features logical blocks and the output logical block. Formalization of technical models aims to describe, at a high level of granularity, how the software and the hardware components should be developed. Algorithm 5.5 presents formalization of design decisions of the Technical viewpoint.

Algorithm 5.5 Technical Viewpoint - Algorithm to define the Technical viewpoint of the system

```
{
na for (i = 0; i < |activityList|; i++){
na     if (activity MUST be performed in a Software component){
A≤na    store activity in ListActivities_IN_Software;
–      }else{
na−A   store activity in ListActivities_IN_Hardware;
–      }
– }
A  for (i = 0; i < |ListActivities_IN_Software|; i++){
A      create (restore) the associations of this activity;
A      if(activity needs an internalSignal){
B≤A    create (restore) the internalSignal of this activity;
–      }
A      if(activity needs an externalSignal){
B≤A    create (restore) the externalSignal of this activity;
–      }
```

```
17  –   }
18  n_a−A for(i=0; i < |ListActivities_IN_Hardware|; i++){
19  n_a−A     create (restore) the associations of this activity;
20  n_a−A     if(activity needs an internalSignal){
21  C≤n_a−A    create (restore) the internalSignal of this activity;
22  –          }
23  n_a−A     if(activity needs an externalSignal){
24  C≤n_a−A    create (restore) the externalSignal of this activity;
25  –          }
26  –      }
27  }
```

The main objective of Algorithm 5.5 is to fulfill the list of activities that are performed by software and by hardware. Thus, *ListActivities_IN_Software* is a container to store the functions related to software components. On the other hand, *ListActivities_IN_Hardware* describes a container to store the functions pertaining to hardware components.

Lines $1-16$, of Algorithm 5.5, formalize the design steps to subdivide a group of activities, which comes from the Logical viewpoint, into software and hardware groups. Moreover, the clustered activities must keep their internal and external signals. These signals are already defined in *externalSigList* and in *internalSigList* structures. It is important to mention that associations between these software/hardware activities and the signals were defined in the Logical viewpoint. Therefore, lines $8-16$ formalize how to restore the signals of the activities performed in the software. In addition, lines $17-25$ formalize how to restore the signals of the activities performed in hardware.

The second refinement of the Technical viewpoint applies MARTE profile stereotypes to refine the external signals of one hardware/software component. Thus, concepts of SRM and HRM packages are applied to refine digital and analog inputs/signals of the system. Since the logical reasoning to annotate these stereotypes is similar to Algorithm B.4 the formalization of this refinement is not presented herein.

Technical viewpoint models are inputs to describe Global Architectural Model and MUD. The Global Architectural Model provides a general view of the software components of the system. In this case, each swimlane with *ListActivities_IN_Software* represents one atomic software component/block. This global model presents the software components of the Technical models in a system view and it highlights their interfaces. Furthermore, the implementation view of the software components can be explored and detailed in MUD. Both models are important to go from architectural design to system realization. The case study proposed in Chapter 6 provides an example of Global Architectural Model and MUD, in the context of the MARTeSys^{ReqD} methodology.

5.3 A Strategy to Analyze the Architectural Viewpoints in RTES Development

The MARTeSys^{ReqD} methodology shows a strategy for the designing of architectural viewpoints and formalize these at different degrees of granularity. From the proposed formalization, several algorithms were defined to standardize the design decisions, while allowing for cost evaluation and complexity analyses of design steps/decisions of the MARTeSys^{ReqD} methodology. In this section, a partial complexity analysis of the design steps for RTES development is presented and, then, a global measurement of the system development complexity is also defined.

5.3.1 Partial Asymptotic Analysis of the MARTeSys^{ReqD} Design Decisions

Section 5.2 presented one specific formalism for each refinement of the architectural viewpoints. Pseudocode and conventional algorithms are applied to write the design decisions of the MARTeSys^{ReqD} methodology in a formal manner. In addition to the design formalization, all algorithms describe, on their left side, the related complexity of each design step. Table 7 presents the cost function, in the worst-case of design decisions, for all the proposed algorithms. Some of these Algorithms are placed in Annex B.

Viewpoint	Algorithm	Cost Functions	Complexity
Requirements	Algorithm 5.1	n+n+n+n+A+n-A+16n+n+7n+2 = 28n	$\mathbf{O}(n)$
	Algorithm B.1	n+n+n*m+n*m+2A+n+2B+2C+2= 3n+2n*m+2	$\mathbf{O}(n * m)$
	Algorithm B.2	m+m+A+1= 2m+A+1	$\mathbf{O}(m)$
	Algorithm B.3	n+n+n+n*n+n*n+A= $2n^2$+3n+A	$\mathbf{O}(n^2)$
	Algorithm B.4	n+n+A+n+A= 3n+2A	$\mathbf{O}(n)$
Functional	Algorithm 5.2	s-2+s-2+1+1= 2s-2	$\mathbf{O}(s)$
	Algorithm 5.3	$4n_i$*n_b+$2n_i$+2A+2B+C+4	$\mathbf{O}(n_i$*$n_b)$
	Algorithm B.5	$4(n_s * n_{ib})$+$4n_s$+2A+2B+3C+2D+2E+4	$\mathbf{O}(n_s$*$n_{ib})$
	Algorithm B.6	$3n_b$+A+B+1	$\mathbf{O}(n_b)$
	Algorithm B.7	$2n_b$*(n_b-1)+$5n_b$+$2n_l$+2A+2B+2C+2D+E+4	$\mathbf{O}(n_b^2)$
Logical	Algorithm 5.4	$2n_b$-A+C+7	$\mathbf{O}(n_b)$.
	Algorithm B.8	$13n_e$-3A+2B-3D+2E	$\mathbf{O}(n_e)$
	Algorithm B.9	$7n_e$+$6n_a$+3A	$\mathbf{O}(n_i$+$n_a)$
	Algorithm B.10	$6n_i$+$6n_a$+2B-3C+2E+F	$\mathbf{O}(n_i$+$n_a)$
Technical	Algorithm 5.5	$7n_a$-2A+2B+2C	$\mathbf{O}(n_a)$

Table 7 – Algorithms and their Cost Functions

.

The variables shown on Table 7 represent the following artifacts: n is the number of requirements; m is the number of categories; s is the number of scenarios; n_s is the number of signals in the system; n_i is the number of internal signals; n_e is the number

of external system signals; n_b is the number of functional blocks; n_{ib} is the number of internal blocks; n_l is the number of logical blocks; n_a is the number of logical activities.

Algorithms B.1 and B.3 (from the Requirements viewpoint), Algorithms 5.3 and B.5, Algorithm B.7 (from the mapping between Functional viewpoint to Logical viewpoint), Algorithm B.9 (from the Logical viewpoint) and Algorithm 5.5 (from the Technical viewpoint) have the highest cost function of each viewpoint. Evaluation and analyses of their complexities allow for a better understanding of the effort to apply the MARTeSys^{ReqD} methodology in the RTES development.

The formalism proposed for the architectural design at different abstraction levels considers a group of specific design steps to create a RTES model. MARTeSys^{ReqD} does not measure human related activities, especially those that involve creativity and human reasoning, in the proposed complexity analysis. This process is related to the know-how of engineers, personal human decisions, and the effective understanding of the system domain and system services. Due to all of the above-mentioned reasons, it is not possible to present the overall complexity analysis to the architectural viewpoint design described in this research. Herein, it is considered a partial asymptotic analysis.

Within these algorithms, the cost function aims to measure the effort needed to "execute" each set of instructions. Equation 6 shows the time complexity function $f(n)$, which defines the time measurement that is required for a function $g(n)$. In other words, $f(n)$ is a function that represents the costs to execute a problem with its size n. In this case, the complexity grows linearly as input size increases.

$$g(n) = O(f(n)) \tag{6}$$

A function $g(n)$ is $O(f(n))$ if there are two positive constants c and m in which $g(n) \leq cf(n)$ for all $n \geq m$. The equation $T(n) = O(f(n))$ represents the complexity time $O(f(n))$ for the $g(n)$ considering n as input.

In the **Requirements viewpoint**, Algorithms B.1 and B.3 have, respectively, a cost function of $O(n * m)$ and $O(n^2)$. Within these algorithms, n represents the number of system requirements and m the number of categories in which the requirements should be classified. By analyzing the possible complexity cost of the design decisions, of Algorithm B.1, there are three possibilities for the cost function:

$$n * m \begin{cases} n < m & \rightarrow O(n * m) \\ n = m & \rightarrow O(n^2) \\ n > m & \rightarrow x | x < O(n^2) \end{cases}$$

Considering worst-case for the design activities in Algorithm B.1, it is expected that each specified requirement of the Requirements viewpoint defines a single category. In this case, $n = m$ and the complexity function **has a quadratic order**. It can be assumed that no optimized design decisions have been performed once it is expected,

from the Requirements viewpoint, the specification of a cohesive and correlated group of requirements.

Considering the scenario where $n < m$, the complexity function has a **polynomial order**. From a design perspective, if $n < m$ it ispossible that one atomic requirement is creating more than one category. However, the fact that the number of requirements is smaller than the number of categories is also inappropriate togood design decisions. Each requirement should be atomic, establish no dependents or be contained in different categories (therefore, this case is not considered).

Finally, when $n > m$, the complexity function is $x \leq O(n^2)$. In this case, the number of requirements is larger than the number of categories. Otherwise, all of the other designs will be affected and several blocks/components will be created. The complexity function is strictly smaller than $O(n^2)$. It means that the design steps, in Algorithm B.1, are polynomial and these are feasible, since the requirements systems are classified in less categories. In the case of small n, it is possible to reduce the design complexity.

Algorithm B.3 has the highest complexity for the Requirements viewpoint. In this case, design efforts relate directly to the number of requirements n. It is assumed that each requirement is checked with $n-1$ other requirements in order to model relationships between these. In the worst-case of the design decision, Algorithm B.3 is $O(n^2)$. This means that the number of relationships is larger than the number of requirements and each requirement has a relationship between them. However, due to the previous refinements of this viewpoint and considering the human expertise, it is assumed that this activity can be performed in an intelligible manner. This statement comes from the fact that different types of relationships from the adopted SysML diagram are known and the proposed algorithm describes the main design steps to model this refinement.

In the **Functional viewpoint**, Algorithms 5.3 and B.5 have higher complexity. Their complexities are O(n_i*n_b) and O(n_s*n_{ib}). Here, variable n_i describes the number of internal signals, variable n_b relates with the number of functional blocks, variable n_s shows the number of global system signals and variable n_{ib} corresponds to the number of internal blocks.

The complexity analysis of the Functional viewpoint considers the Algorithm B.5, as it has the highest complexity for this viewpoint. In this case, $n_i \subset n_s$ and, as explained in Section 5.2.2, the functional blocks generate the internal blocks thus $n_{ib} = n_b$. One can therefore consider two cases for the analysis of O(n_s*n_{ib}):

$$n_s * n_{ib} \begin{cases} n_s = n_{ib} & \rightarrow O(n_s^2) \\ n_s > n_{ib} & \rightarrow O(n_s^2) \end{cases}$$

If $n_s = n_{ib}$, then the out-degree and in-degree associations (item flow) are equal. It means that the blocks can have only one association between each other. However, this assumption is not realistic, since the Internal Block diagram allows for several item flows

between each block. In the IDD, the number of associations and the effort to design them relates to the interaction specificities of each RTES. In the case where $n_s > n_{ib}$, there are more associations than the number of internal blocks. Therefore, it is necessary to consider the worst-case for the design activities, in order that these remain consistent with the other viewpoint analyses. Here, the assumption is reached that one internal block has, at least, one association with all the other blocks. From this assumption, it is possible to measure the complexity effort of a complete graph: $n_s * (n_s - 1)/2$. Then, the Functional viewpoint has a $O(n_s^2)$ complexity.

Algorithm B.3 formalizes the design decisions to map the Functional and Logical viewpoints and it has an O($n_b{}^2$) complexity. The analyses presented reflect the main idea of the design step, as each functional block must be evaluated/compared against the other functional blocks. This analysis aims at refining and mapping functional blocks into logical blocks through a F X L relationship, where $L \leq F$.

Logical viewpoint algorithms present a linear complexity order and Algorithm 5.4 has the highest complexity for this viewpoint. The complexity of Algorithm 5.4 is O(n_i + n_a) and this value describes the linear sum of internal signals and logical activities. Regarding the **Technical viewpoint**, Algorithm 5.5 has complexity $T(n)$ = O(7n_a-2A+2B+2C). In this case, it should consider $T(n)$ = O(n_a), since n_a represents the number of activities from the activity list. In this context, A describes the number of activities which are related to a software component, where $n_a - A$ shows the activities that are related to a hardware component.

As analyzed in Algorithms of the Logical and Technical viewpoint (see Table 7), the complexity function has a linear order. Here, the design cost is linearly proportional to the input set. Algorithm B.8, for example, assumes that the maximal number of activities (of n_a) for each external signal is equal to n_e.

The complexity analyses performed in this thesis is not a common asymptotic analysis. It derives from the fact that the proposed formalization cannot consider human activities. These activities are strongly correlated with the creativity and knowledge level of engineers in the application of engineering processes. However, even this partial asymptotic analysis is helpful to RTES design, as it provides an initial cost and efforts estimation in the adoption of the MARTeSys^{ReqD} methodology. Through considering the worst-case scenario for this kind of complexity analysis means that the worst design decisions for system design were considered, from the outset of the first requirements engineering activity. As an example, it can be stated that each system requirement (explained in Section 5.2.1) creates a single category (see Algorithm B.1) and from this unusual design decision, the next viewpoints of the architectural design can be affected. As another example, one can possible to consider that each requirement category is composed of one single and atomic requirement. This non-usual design decision impacts on the design of functional blocks, as each category setups an atomic functional block.

Consequently, each functional block generates one logical block, and so on. Thus, when the worst-case scenario of design decisions occurs, as of the Requirements viewpoint, it reflects across the overall architectural viewpoints.

It is understoo and expected that human interaction will to improve the design decisions, as humans are able to think and use their expertise. Then, even in the worst-case to adopt the proposed methodology, for architectural viewpoints, the complexity is strictly smaller $x|x < O(n^2)$, in the Requirements viewpoint, and $O(n_s^2)$ in the Functional viewpoint. The author concludes that with the human adoption of the formalized architectural steps the execution of the worst-case can be avoided.

5.3.2 System Complexity Prediction of Architectural Viewpoint Design

Analysis of system complexity aims at providing a cost prediction for every viewpoint of the RTES design. This analysis seeks to measure how difficult it is to specify, design and reach the system services from the Requirements viewpoint, passing through the Functional and the Logical viewpoints, to the Technical viewpoint.

Figure 24 shows the functions $r(n)$, $f(n)$, $l(n)$ and $t(n)$, which are related to the architectural viewpoints defined in this research. These functions express through their values the maximum cost function for the number of design artefacts, in the worst-analysis case for each architectural viewpoint.

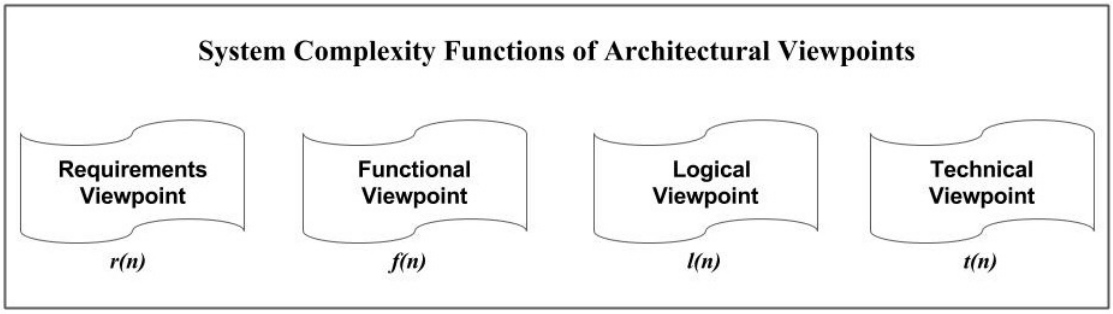

Figure 24 – Viewpoints of Real-Time and Embedded Design and their Complexity Function.

Requirements viewpoint is dealing with the global systems requirements description. Function 7 formalizes the maximum number of elements for this viewpoint.

$$r(n) = \sum_{i=1}^{n} r_i \tag{7}$$

Function 7 describes that $r(n)$ is a set of requirements where $n = |R|$ and $i \in \mathbb{N}$. From the set of system requirements $r(n)$, the requirements categories are defined. Thereby, the overall requirements should be clustered in m categories. Definition of these categories

is pertinent, in the proposed methodology, since they influence in the definition of the Use Case and of the forthcoming Functional viewpoint. Function 8 defines formally a category of requirements and Function 9 the group of system scenarios. Function 9 is adopted to perform the cost measurements of subsequent viewpoints.

$$c(m) = \{r_i | \forall r_i \in R : \exists c_j \text{ where } r_i \text{ is cohesive with } c_j\} \tag{8}$$

In the worst-case, every requirement r_i is a unique and an atomic category (c_j), where $i, j \in \mathbb{N}$ and $|C| = |R|$. Consequently, the number of scenarios must also be $|Sc| = |R|$, since the scenarios, in this methodology, are derived from the system categories.

$$sc(m) = \{sc_i | \forall c_i \in C : \exists sc_j \in SC\} \tag{9}$$

The Functional viewpoint provides the functional blocks of the system. In this design, for the worst-case analysis, the number of blocks is equal to the number of requirements. Moreover, in this viewpoint, it is necessary to model the possible relationships/associations between these blocks. Function 10 shows that each functional block $funcBlock_i$ is (refines) an element c_i of C. In this function, n_b corresponds to the number of functional blocks and $i, j \in \mathbb{N}$. In addition, this function measures the number of internal signals (ports) and the number of external signals (flowports). In the worst-case scenario, these communication ports are designed by the channel of a block $funcBlock_j$ with all other blocks. Moreover, this analysis assumes that all blocks are communicating with each other. Then, it is abstracted that a complete graph with n_b vertices can describe the associations of the block. In this case, each vertex is a functional block.

As previously introduced, the edges between a functional block are channels of communication. It is assumed that n_i edges ($n_i > 0$) can be internally modeled in a channel (ch). These internal edges define input/output signals (associations) of the system (Block Definition diagram).

$$f(n) = \sum_{j=1}^{C} c_j + ch_j + eS = \sum_{j=1}^{C} c_j + n_b(n_b - 1)/2 + eS \tag{10}$$

Function 10 presumes the worst-case scenario related to the number of functional components. In this analysis, c_j defines the number of blocks which are, in the worst-case, equal to the number of categories, where $0 < j \leq |C|$. Keeping in mind Functions 7 and 8, in the worst-case, $|C|$ = maximum number of requirements. Here, each r_i defines one atomic category. Therefore, $|C| = |R|$.

Moreover, the maximum number of associations or internal signals is measured by the complete graph function. The amount of associations between two blocks cannot be predicted, since it relates to human design decisions. Consideration has been given to the possibility that functional blocks are fully connected with each other. Therefore, there

is at least one association, known here as internal signals, between each block. On the other hand, the internal signals (n_i) between each block are abstracted by the channel ch, which defines an abstraction of multiple connections. Finally, eS formalizes the external communications of functional blocks.

Function 11 formalizes the maximal number of artefacts to the Logical viewpoint, where $n_l = |LogBlock|$ and $i \in \mathbb{N}$. These artefacts represent the logical blocks of the system. It is important to highlight that the internal design of a logical block is not measured in Function 11. In this Logical viewpoint description, the internal design of each artefact (logical block) is performed by humans, which means that behavioral system design, for each logical block, and its interactions and sub-activities are not considered in Function 11. The proposed function shows that the Logical viewpoint is composed of the sum of each logical block l_i.

$$l(n) = \sum_{i=1}^{n_l} l_i \tag{11}$$

The Technical viewpoint refines artefacts from the Logical viewpoint. In the MARTeSys^{ReqD}, each logical block is divided into S swimlanes. Besides, activities modeled by engineers are split into groups that represent the swimlanes. Function 12 formalizes the set of technical blocks of the Technical viewpoint in order to count its artefacts. In this function, s_i describes each hardware or software block of the Technical viewpoint. In this case, they are represented in different s_i lanes, where $i \in \mathbb{N}$. It means that each technical block is composed of a number $S \leq a$ swimlanes. Moreover, the Function 12 shows that for each l_i element there is one s_j swimlane, where these elements must be linked.

$$|T_B| = \{l_i | \forall l_i \in n_l : \exists s_j \in S \text{ where } 0 < j \leq a\} \tag{12}$$

Finally, Function 13 represents the $t(n)$ cost function for the Technical viewpoint.

$$t(n) = \sum_{i=1}^{n_l} \sum_{j=1}^{S} s_{j,i} = \sum_{i=1}^{n_l} T_{B_i} \tag{13}$$

In Function 13, for each n_l logical block a set of s_i swimlanes can be created in the Technical viewpoint in order to represent different hardware and software components where the activities can be allocated. Variables i and $j \in \mathbb{N}$ describe the number of logical blocks and the possible number of swimlanes.

Finally, from the composition/union of measurements performed in the viewpoint, it is possible to propose a global function to system complexity prediction. Therefore, Function 14 defines a global function to measure the overall amount of design artefacts based on the system requirements, functional and structural blocks of the system, logical system components and hardware and software components. This number provides a

formal prediction of the maximal number of artefacts that are involved in architectural description of RTES when it considers adoption of the MARTeSys^{ReqD} methodology.

$$G(n) = r(n) + f(n) + l(n) + t(n) \tag{14}$$

Approaches to measure the complexity of design activities and to predict the system complexity can contribute to minimize the difficultly level of RTES development. This section shows a strategy for measuring the possible number of design artefacts from the Requirements, Functional, Logical and Technical viewpoints. The analysis of the proposed methodology are based on the worst-case scenarios to the maximal number of viewpoint components. It aims to numerically predict the difficulty level and the impact of the MARTeSys^{ReqD} methodology adoption to RTES development. Moreover, the skills and expertise level of the developers and engineers plays an important role. However, the proposed notation for measuring the complexity level of one design approach can substantiate its application, while providing complexity analysis/predictions for its adoption. These algorithms detail the primitive and specific modelling activities for each viewpoint. Therefore, the cost analysis is supposed to describe the effort to adopt and apply each design instruction in a computational manner.

5.4 Contributions of the Proposed Formalization

The previous chapter presented a specific methodology for RTES development based on design viewpoints. The subsequent chapters how the proposed methodology works and its different viewpoint refinements. In this chapter, the focus is to formalize the proposed methodology, its architectural design and its refinements decisions. Some of the contributions explained in this chapter are given greater detail [98] and [99]. A new and formal description for RTES design was presented considering a group of algorithms. This strategy adopts pseudo-code to guide the design decisions and to formalize the proposed methodology. The setting of algorithms was explained and it collaborates with the global view of the design steps in a systematic manner.

As a general overview, one highlights a proposal for formalizing the RTES design activities (viewpoints), a measurement of the system design complexity from the initial design activities, as well as the definition of a formalized manner for analyzing the complexity of RTES development without interference of user external knowledge. The proposed analysis is mainly based on the proposed design framework/decisions of the MARTeSys^{ReqD}.

Another important contribution is the early estimation of system complexity regarding the developed artefacts. During development of RTES, it is not always possible to wait until the final phases of design to estimate the overall system complexity. However,

from these algorithms a cost function for design is proposed and it considers the efforts involved in creating the the overall system models. This analysis can contribute to RTES development once it performs an impact analysis of their algorithms. Finally, this chapter also provides a strategy to measure the global system complexity, as of the initial design steps through to the modelling of technical artefacts. A general equation for system complexity is defined in this thesis to estimate the measure for the expected number of design components/artefacts.

Chapter 6

Application of the MARTeSys^{ReqD} Methodology

This Chapter describes the application of the MARTeSys^{ReqD} methodology in an industrial case study in order to specify and design functions, services and constraints of the system under various and intelligible descriptions.

6.1 An Overview of Automotive Control Systems

ACS are composed of sensors, actuators, components, controllers, supervisory control and distributed data, which use electrical, electronic, mechanical and computer resources [29]. The heterogeneous nature of ACS makes their conception, design and development particularly difficult [4].

According to the authors of paper [183], there are five different major domains, which are described below,that need to be explored in the development of automotive systems. As depicted in Figure 25, these modules are named as power train domain, chassis domain, body domain, HMI domain and telematics domain.

The **Power Train Domain** deals with systems of vehicle motorization, including the engine, data transmission between brakes, accelerator and gear, speeding up, among others. The **Chassis Domain** is composed of systems that aim at controlling the interaction of the vehicle with the road (wheel, suspension). Chassis subsystems ensure the comfort of drivers and passengers, as well as their safety.

The **Body Control Module (BCM)** is composed of entities that offer support to activities, comfort and safety related to users. In this domain, the embedded systems are mainly related to door controllers, seat controllers, airbag controllers, windshield wiper controllers, air conditioning controllers, window controllers, light controllers and so on.

The **HMI Domain** consists of equipment that allows for information exchange between electronic systems and the driver. In this domain, the embedded systems represent

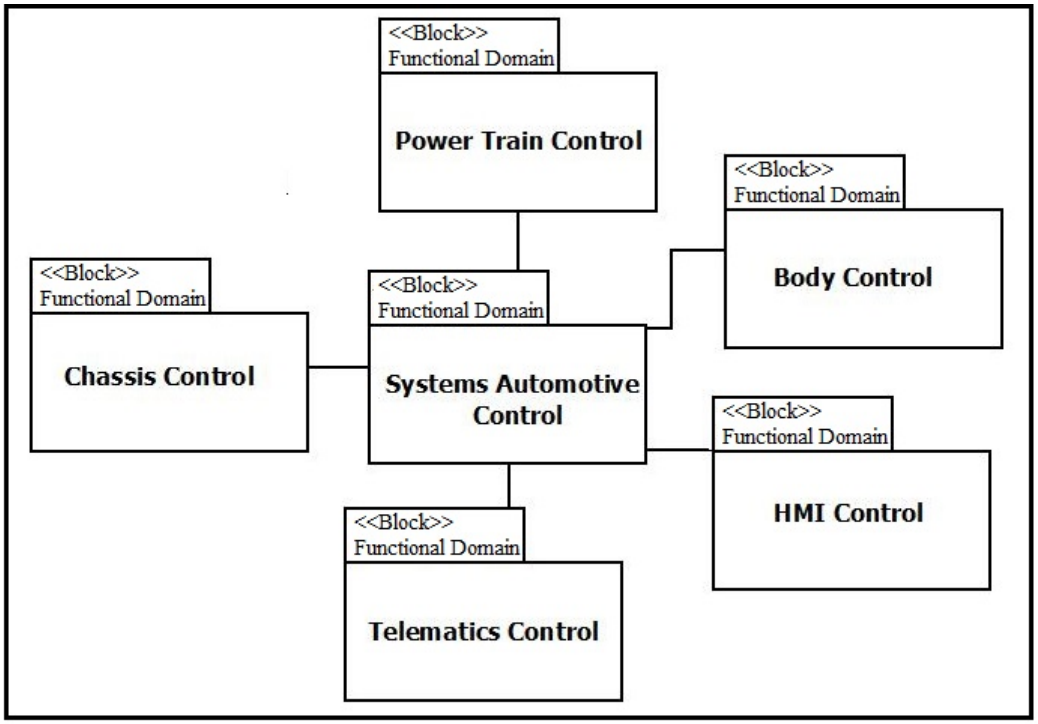

Figure 25 – Domains in Automotive System Development.

information concerning the status of the car as, for example, vehicle speed, oil level, status of a door, status of lights, status of multimedia devices (e.g., current frequency for a radio device), or the result of a request (e.g., visualization of a map provided by a navigation system). The **Telematics Domain** is related to components that allow for information exchange between the vehicle and the external environment such as radio, navigation system and Internet access.

In order to exemplify adoption of the MARTeSys^{ReqD} methodology, this thesis depicts the analysis of functional and non-functional system services, which are related to the Body Control Module.

6.1.1 Body Control Module

BCM [183] contains the embedded functions of a vehicle that are not directly related to its dynamics, but strongly relates to essential components, which provides greater comfort and safety to drivers. Nowadays, different functions coupled in vehicles, such as windshields, lights, windows, seat controls and mirrors are controlled by software-based systems. Figure 26 shows graphically the main components of a Body Control Module.

The **Door Control** corresponds to locking and unlocking control systems, according to internal users requests, to the control signs (sensors and actuators). This system can also provide automatic responses after vehicle inactivity. The **Windshield Wiper** represents management systems for back and front wipers. The **Control Seat** systems

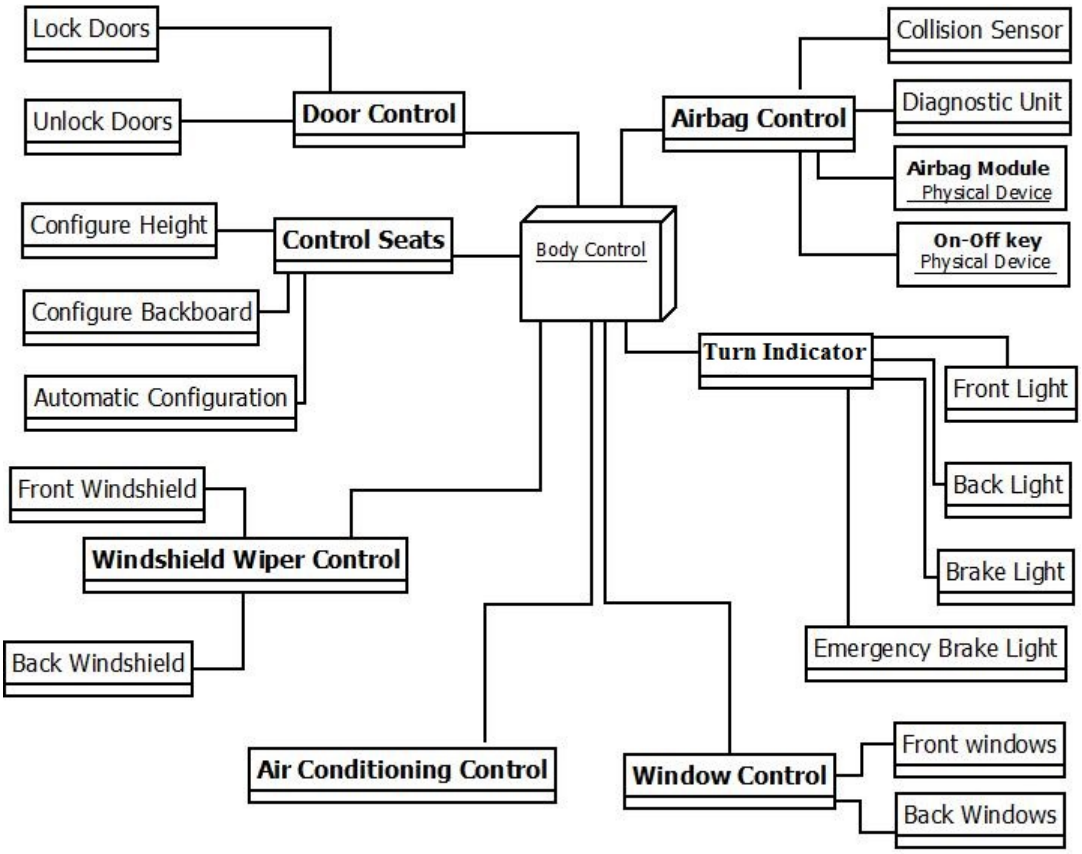

Figure 26 – Structure of the Body Control Module.

describes the embedded systems responsible for configuring height, backboard and other automatic functions related to seat controls.

The **Air Conditioning Control** system refers to functionalities able to manage several temperature conditions of the vehicle and to provide different states according to the environment conditions or the user preferences. The **Window Control** commands the window lifters of front and back doors. **Airbag Control** is composed by subsystems to diagnose collisions, through internal sensors, which can occur in the drivers and passengers compartments. The airbag control has external sensors and physical devices, which allows for manual activation/deactivation. This module is also composed of the diagnostic module, which is able to evaluate the operation of the airbag system, when the vehicle is turned on.

The **Turn Indicator Control** presents the functional modules that operate front lights, rear lights, brake lights and emergency/hazard lights. It represents different subfunctions to flash the system lights and to control their parallel and priority requirements. The case study, described in this Chapter, is related to the TIS.

6.1.2 A Motivating Case: The Turn Indicator System

The MARTeSys^{ReqD} methodology has been applied to design the functions of TIS under different abstraction levels. It aims at presenting the requirements specification and architectural design activities with focus on functional system services and non-functional constraints. This study provides strategies for model checking early in terms of the early design of requirements, as well as to provide empirical simulation of RTES constraints.

Figure 27 depicts, through a graph representation, the main components of the TSI and its perspectives. It highlights, by blue color, the Turn Indicator module and its link with BCM. The Electronic Control Unit (ECU), depicted in Figure 27, is responsible for controlling all subsystems and manages BCM functions.

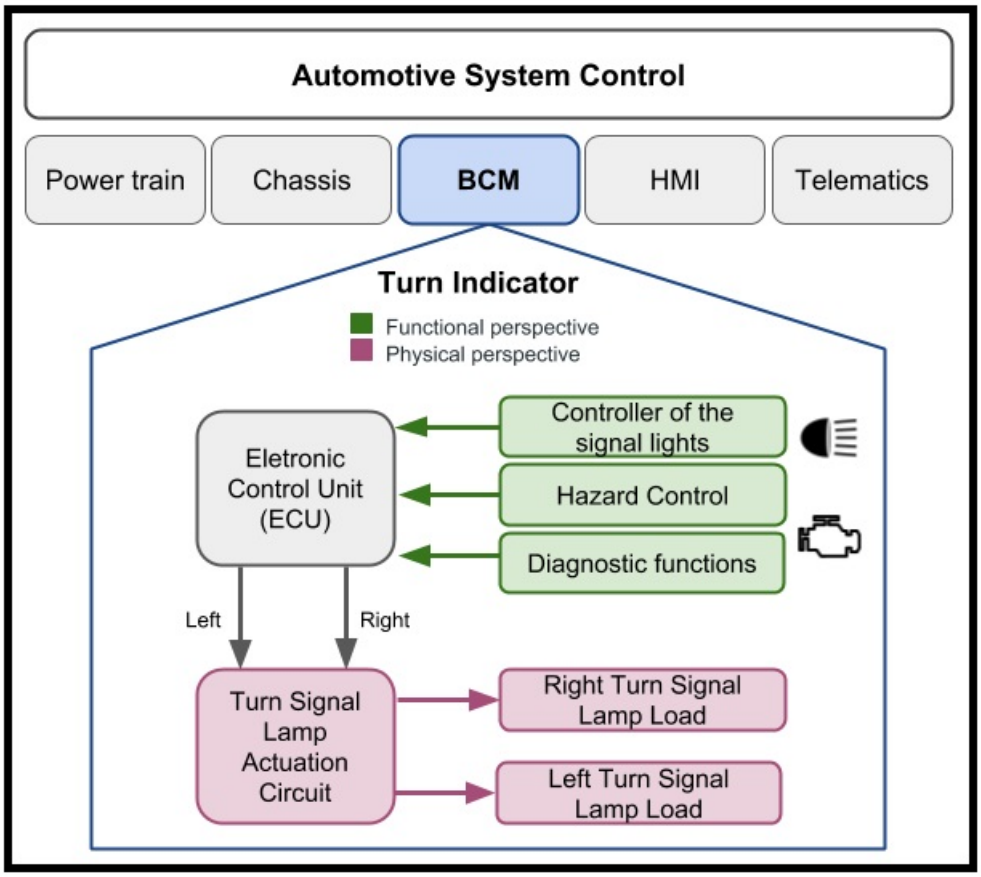

Figure 27 – Features of the Turn Indicator System.

The TIS, from a functional perspective, is responsible for controlling front and rear lights, brake lights, emergency brake lights, indicator lights, hazard indication lights and all the management and diagnostic functions of a vehicle. In addition, from a physical perspective, the turn indicator consists of one or more signal actuators. These components are responsible for controlling the signal lamps of the car, which must cause the signal lights to flash in a configurable frequency. Actuators can be allocated internally or

externally in the cars. From an external view, actuators correspond to the signal lights of the car. On the other hand, internally these can be arranged as visual and acoustic actuators. The latter are, in general, instrument cluster/dashboard feedback lights and feedback tones.

Noteworthy here is that the TIS was chosen in this study for three main reasons. Initially, this research has received the support of experts from automotive development domain. This fact allowed the tailoring of realistic system scenarios and provided a similar representation of one real TIS. Second, the author has found sufficient literature and manuals concerning the domain of study in real contexts. Finally, there were different models already defined in the industrial domain, and these were made available to the author. This fact contributes to a comparative assessment of the proposed models and methodology with similar approaches in use by industry.

6.1.3 Scenario of the Turn Indicator System

Drivers use the lights system to signalize the intention to change the direction of the car. In the right or left directions, it is possible to flash the lights in a hard or soft manner. For safety reasons, these situations must be visually displayed. When drivers want to turn the car in a hard manner, it is necessary to press/activate the flashing system. The car lights continue on until a driver deactivation or until a full car conversion. However, if the driver needs to signalize his changing position by shorter flashing cycle, the car lights are activated in a soft manner. In this latter case, the TIS should control the flashing conditions with a pre-defined time and stop after a given period.

Drivers can also use the car lights to signalize dangerous or abnormal situations. When it is necessary to represent these conditions, the front and rear lights must flash at different frequencies. It means that the driver must be able to activate the hazard function in order to visually indicate hazardous situations. Moreover, every time that the driver needs to use the brake system, to lock/unlock the doors or to trigger the anti-theft protection the light system must flash. Therefore, the car lights must signal each situation. These functions represent different car features, which are independent and controlled by BCM. However, they communicate and generate signals with/to the TIS.

Turn Indicator features operate the following legal and regulatory rules, which are controlled by governmental regulations of each country. This fact forces automotive companies to respect specific regulations, while developing their products. Besides, the Turn Indicator functions also need to follow specific legislation of Original Equipment Manufacturer (OEM).

Figure 28 shows a usage scenario of the Turn Indicator. In this example, features which are related either with the turn indicator function or hazard controllers are shown.

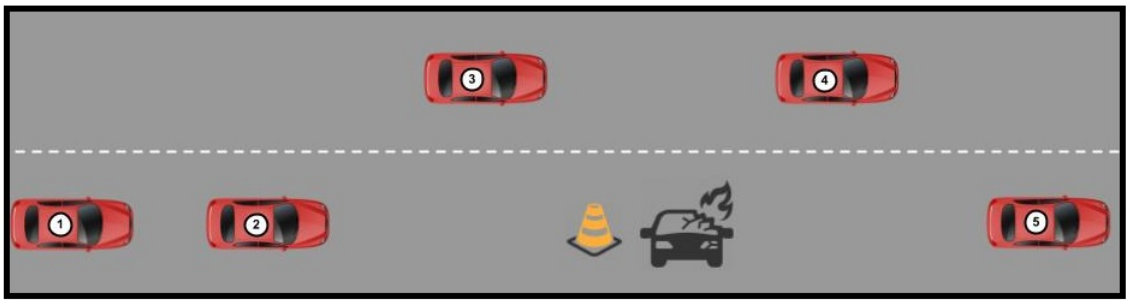

Figure 28 – An Example Scenario of Turn Indicator System.

In Figure 28, the labels 1 to 5 represent distinctive positions of the car based on driver actions as follows:

1. Driver identifies an accident and turns on the hazard lights;
2. Driver needs to change lane and turns the flashing indicator lights in hard manner to the left;
3. Driver turns off the flashing indicator lights;
4. Driver needs to change lane and turns the flashing indicator lights in soft manner to the right;
5. Driver overpasses the anomaly and turns off the hazard lights.

Based on the automotive literature analysis, a set of requirements and constraints related to the Turn Indicator are detailed in order to structure the case study. The subsequent sections are based on general and specific features of TIS and on RTES system concerns. These sections present a detailed overview of the MARTeSys^{ReqD} methodology adoption.

6.2 Design of Architectural Viewpoints of the Turn Indicator System

Architectural design should represent a general comprehension of the system regarding the system requirements. This section outlines how to apply MARTeSys^{ReqD} methodology to design RTES and provides, as results, architectural artefacts of the system under different and complementary perspectives. This contributes to define RTES in a physical and logical perspective, in order to decompose requirements into functional modules, to represent and analyze subsystems iteration, to refine system functions, to verify and validate these systems and so on.

An industrial example of the TIS is modelled in order to apply the proposed study in the description of four architectural viewpoints. Sections 6.2.1, 6.2.2, 6.2.3 and 6.2.4 present the artefacts resulting from the application of the MARTeSys^{ReqD} methodology.

6.2.1 Requirements Viewpoint with the MARTeSys^{ReqD} Methodology

Requirements viewpoint is composed of Requirements Pre-Analysis, High-Level Description of Requirements, Composition of Models using the MARTE Profile, Formal Specification with VSL and Analysis of Requirements. The ollowing sections show the adoption of the proposed methodology along all views of the Requirements viewpoint.

6.2.1.1 Requirements Pre-Analysis

The requirements of the proposed case study are based on the scenario of the TIS presented in Section 6.1.2. The providing of details for these requirements, their functional and non-functional constraints takes into account industrial development processes, of German automotive companies, and also automotive literature analyses. The specification of Turn Indicator requirements is provided as an input to Requirements Pre-Analysis stage. An initial refinement of these requirements is presented on Table 8.

Requirement Description	Type	RFN Type /Concern	MARTE Package
The turn indicator controllers operate in accordance with specific automotive rules. Legality and regulatory criteria are related to the target market(s).	D	-	-
The Turn Indicator system controls its lights with a cyclic time of 1600 ms per full bright-dark cycle.	F	Timing	Time
The final version of the Turn Indicator controllers have to be tested. It is expected at least 175,000 cycles for each unit of multipurpose passenger vehicles.	D	-	-
The turn indicator services can be related to embedded, distributed and real-time constraints.	-	-	-
The turn indicator system performs coherently the prioritization of the different indicator functionalities.	F	-	-
The turn indicator system coordinates the turn situations even when the hazard lights are operational.	F	Distribution	GRM
The turn indicator system parameterizes bright-dark times and flashing frequencies of lights.	F	-	-

The turn indicator system has to identify when the indicator lever[1] is pressed (activated) in a hard/soft manner. The system signals the direction changes of the vehicle when the driver wants to turn the car.	F	-	-
The turn indicator functions have an overall response time of 540 ms.	NF	Timing	NFP/- Time
The turn indicator system needs to check if there is power in the system before starting any system function.	F	-	-
The turn indicator system indicates, internally and externally, all vehicle turns to the drivers.	F	-	-
The system considers a parameterizable time span to turn on, to the right or left side, the indicator lights. Therefore, if this parameterizable interval is not reachable the outputs turn indicator left or right will be switched based on another parameter.	NF	Time	NFP/ Time -
The turn indicator system has to identify when the indicator lever is pressed (activated) in a soft manner. After this, it controls the right or left lights of the vehicle with a predefined number of flashing light cycles. The flashing lights indicate lane changes of the car with a fast turn flashing.	F	-	-
The turn indicator system configures the soft flashing lights for three consecutive cycles. The same configuration is performed when the driver touches the lever for less than 1000ms.	F	-	-
The turn indicator system turns and keeps the indicator lights on until their cancelation by the driver. This will always happen when the indicator lever is pressed for a longer time (not in a soft way).	F	-	-
The turn indicator system turns the turn indicator lights off automatically after a full right or left conversion.	F	-	-
The turn indicator system can indicate anomalies and hazardous situations related to the vehicle. Hazardous occurrences activate hazard lights and can turn on a sonorous warning to the driver.	F	-	-

The hazard functions have an overall response time of 540 ms - response time of turn indicator system.	NF	Timing	NFP/-Time
The Body Control Module (BCM) is responsible for activating the hazard control lamps. BCM communicates with the hazard switch when the function Hazard Indication is active.	F	-	-
The function Hazard Indication must be available after processor failure.	NF	Safety	NFP
The turn indicator system receives signals from BCM. These signals are related to the anti-theft protection function. In this case, the turn indicator system configures the car lights to indicate the arm/disarm state with one flashing cycle.	F	-	-
The turn indicator system sets the lights to indicate the triggered alarm of the anti-theft protection. When an external threat occurs all lights are activated and flashing.	F	-	-
The turn indicator system receives signals, from BCM, which are related to the lock/unlock function. The turn indicator system flashes all lights if the central lock is activated/deactivate	F	-	-
The turn indicator system receives signals, from BCM, which are related to the brake function. When a break action occurs, the turn indicator system activates the brake lights. The lights continue flashing as long as the braking event occurs (is valid) plus 5 sec.	F	-	NFP
The turn indicator system can manage different failure events of integrated components on vehicle level.	F	-	-
The turn indicator system detects internal failure events.	F	-	-
The turn indicator system gathers internal failure events.	F	-	-
The turn indicator system stores failure events.	F	-	-
The turn indicator system sends internal failure events to the diagnostic function OBD2-interface on vehicle level.	F	-	-

Table 8 – Artefacts of the Requirements Pre-Analysis - **Requirements Specification**.

Table 8 describes the first analysis of the Turn Indicator requirements. The column Type presents a prior classification of the system requirements. It allows one to describe if a specific requirement has functional, non-functional or domain type. In Type column, the acronyms D, F and NF, respectively, relate to the domain, functional and non-functional requirements. It is also possible to identify if one specific requirement has a non-functional concern. The main non-functional concerns of RTES are extensively discussed in Chapter 2 and they can be related to timing, reliability, safety, performance, security, embedded, among other features. The third column shows this classification. Finally, the last column can specify or suggest the MARTE packages, which are able to contribute to the design of one requirement.

These previous analyses contribute toward understanding the Turn Indicator requirements. Moreover, they allow for the separation of real-time and embedded concerns of the systems while classifying the group of requirements.

6.2.1.2 High-Level Description of Requirements

The activity of High-Level Description of Requirements refines the system requirements specified in a tabular way. Table 9 shows a high-level description of the requirements while it performs their categorization. The proposed categorization considers the similarity criteria of these requirements.

On Table 9, the declaration of each requirement is improved in order to identify possible ambiguities. The strategy to refine these requirements adopts some key-words as, for example, *Shall*, *Must*, *Are applicable*, *Responsible* and *Will* to refine their definition. Furthermore, it can express priority criteria as part of its future development.

The first column of Table 9 describes the context of one requirement and correlates it in a similar group of system features. For this case study, the six contexts are named as Regulatory Requirements (**RR**), Vehicle Functions (**VF**), Turn Indicator (**TI**), Hazard Indicator (**HI**), BCM Events (**BCM**) or Diagnostic Functions (**DI**).

Context	ID	Requirement Declaration
RR	R1	The feature[2] turn indicator **will** respect the legal automotive regulatory standard of the target market(s).
	R2	The feature turn indicator **must** be operated with a cycle time of 1600 ms per full bright-dark cycle.

[1] Lever Indicator is also so-called comfort flashing, one-touch flashing or highway flashing.

	R3	The turn indicator controllers **shall** be tested with at least 175,000 cycles for each unit that is installed on a multipurpose passenger vehicle.
VF	R4	The feature turn indicator **must** coordinate embedded and real-time functions of indicator controller.
	R5	The feature turn indicator **shall** perform the prioritization of different indicator functionalities, even when other indicator functionalities have not been deactivated.
	R6	The feature turn indicator **shall** precisely and correctly prioritize the turn indicator functions. This requirement relates to the given precedence to turning event, even when the hazard lights are operational.
	R7	The feature turn indicator shall be able to parameterize, in an individual way, the bright-dark times.
	R8	The feature turn indicator **shall** be able to parameterize, in an individual way, the flashing frequencies of lights.
TI	R9	The turn indicator **shall** recognize the different lever events in order to provide the right turn direction.
	R10	The feature turn indicator **shall** visually indicate the direction changes of the vehicle when the indicator lever is pressed in a hard/soft manner.
	R11	The turn indicator feature **must** process itself with a maximal response time of 540 ms.
	R12	The turn indicator feature **must** process its features in a maximal period of 560 ms.
	R13	The feature turn indicator **shall** check the ignition signal before any turn indicator function. This requirement checks if an ignition signal, via clamp 15, is available.
	R14	The system **shall** consider a parameterizable time span to turn on the flashing lights. Therefore, if this parameterizable interval is not reachable the outputs of turn indicator, to left or right side, are switched based on another parameter.
	R15	The feature turn indicator **shall** enter into operation when the indicator lever is pressed (activated) in a soft manner.
	R16	The feature turn indicator **shall** enter into operation when the indicator lever is pressed (activated) in a hard manner.
	R17	The feature turn indicator, when activated in a soft manner, **shall** be defined by a predefined number of flashing light cycles in order to indicate lane changes of the car.

	R18	The feature turn indicator **shall** flash the turn indicator lights for three consecutive cycles when the indicator lever is pressed (activated) in a soft manner or for less than 1000 ms.
	R19	The feature turn indicator, when activated in a hard manner, **shall** continue to flash the turn indicator lights on until a cancelation by counter-activation of the indicator lever.
	R20	The feature turn indicator, when activated in a hard manner, **shall** turn off the turn indicator lights after a full right or left conversion.
HI	R21	The feature turn indicator **shall** visually indicate hazardous situations from the vehicle lights.
	R22	The hazard feature **shall** activate the hazard lights.
	R23	The hazard feature **shall** activate sonorous warning to the driver.
	R24	The hazard feature **must** process fully with a maximal response time of 540 ms - response time of the turn indicator feature.
	R25	The hazard feature **must** process its features in a maximal period of 540 ms - response time of the turn indicator feature.
	R26	The BCM **shall** activate the hazard control lamp in the hazard switch when the function hazard indication is active.
	R27	The hazard feature **must** be available after processor failure.
BCM	R28	The feature turn indicator **shall** configure the indication lights with one flashing cycle in order to show the arm state of the anti-theft protection function.
	R29	The feature turn indicator **shall** configure the indication lights in order to show the disarm state of the anti-theft protection function with one flashing cycle.
	R30	The feature turn indicator **shall** set the lights indicator in order to indicate the triggered alarm of the anti-theft protection in case of a vehicle external threat with all flashing light signals activated.
	R31	The feature turn indicator **shall** indicate the central locking status with all flashing light signals activated.
	R32	The feature turn indicator **shall** indicate an emergency braking situation by turning on the red flashing lights as long as the emergency braking event is valid plus 5 seconds.
DF	R33	The feature turn indicator **must** manage different failure events of the integrated components on vehicle level.
	R34	The feature turn indicator **shall** detect internal failure events.
	R35	The feature turn indicator **shall** gather internal failure events.
	R36	The feature turn indicator **shall** store internal failure events.

	R37	The feature turn indicator **shall** send internal failure events to the diagnostic function OBD2-interface on vehicle level.

Table 9 – Artefacts of the High-Level Description of Requirements- **Requirements Categorization**.

As elucidated in Chapter 4, this activity aims to improve and standardize the requirements specification. The proposed categorization is helpful for creating clusters of correlated requirements. This contributes toward the discovering of system services and scenarios linked to RTES. Moreover, the adopted keywords clarify the description of atomic requirements and these can highlight priority and relevance criteria in the RTES specification. This study considers the atomic requirements of Table 9. The refinements performed in the Requirements viewpoint is essential in the proposed study, since it provides the background for the following design activities. Architectural design, with MARTeSys^{ReqD} methodology, considers the requirements of Vehicle Functions, Turn Indicator and Hazard Indicator scenarios. Moreover, the feature described by the requirement R31, of the BCM Event scenario, is also considered.

Figure 29 depicts the Use Case diagram of the TIS. This architectural model presents, in principal, the system scenarios as a group of functions (use cases), and allows for the representation of external entities that influence each scenario. Thus, this view defines relationships between use cases, the system actors and their connections.

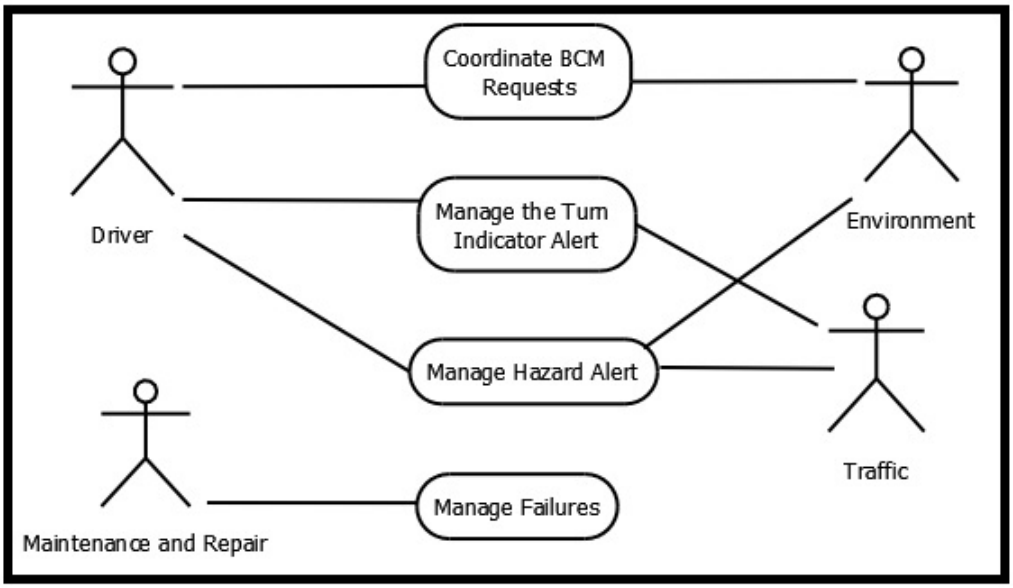

Figure 29 – Artefact of High-Level Description of Requirements - **Use Case Diagram**.

[2] Feature means, in this context, a specific module of the system that is responsible and able to perform a specific functionality.

Scenarios of the TIS are mainly based on the main categories of Table 9. The use cases are related to the changes in direction control (Turn Indicator category), to the hazard alert management (Hazard Indicator category) and to the failure management (Diagnostic Function category). The categories ,from Table 9, which are not designed as an atomic use case are necessarily included and combined into previous ones. It can be highlighted, as for example, the category of Regulatory Requirements does not define a single use case. However, Regulatory Requirements constrains the scenarios of Manage the Turn Indicator Alert and Manage Hazard Alert.

In Figure 29, the Driver, Maintenance and Repair , Environment and Traffic are the main stakeholders of this scenario. The driver can interact with the system and its turn and hazard functions. The environment actor influences the turn and hazard alerts and its conditions provide insights into the system operation. The traffic actor receives different sonorous and visual interactions of the TIS. Finally, the Maintenance and Repair actor manages the improvement and support to the system failures/changes.

6.2.1.3 Composition of Models using the MARTE Profile

The SysML Requirements diagram is designed, in this activity, to represent system requirements and to detail its main functions. Initially, requirements are atomically and graphically expressed.

The SysML Requirements diagram has been extended, with MARTE annotations, in order that relevant real-time concerns can be attached to the system requirements. The refinement of this model provides the input to the activity of Formal Specification of Requirements with MARTE and VSL. Therefore, Figure 30 combines the design decisions of both views.

Figure 30 has the system constraints formalized by the concrete syntax of VSL. This formalism is followed and adopted to refine existing constraints, as well as to annotate the new ones. VSL provides standard bases to express value specifications in model elements.

6.2.1.4 Analysis of Requirements

The transformation rules proposed in Chapter 4 are adopted to formally analyze timing constraints, safety, deadlock, reachability and critical regions controller of the models, from the outset of the requirements specification. Table 9 tabulates the Turn Indicator requirements, in natural language, using key-words and correlated categories. These requirements are inputs for the formal specification of the system requirements by Timed Automata formalism. Through the Timed Automata model, implemented in UPPAAL, an early analysis of requirements is performed. This analysis applies TCTL to verify and validate properties of interest.

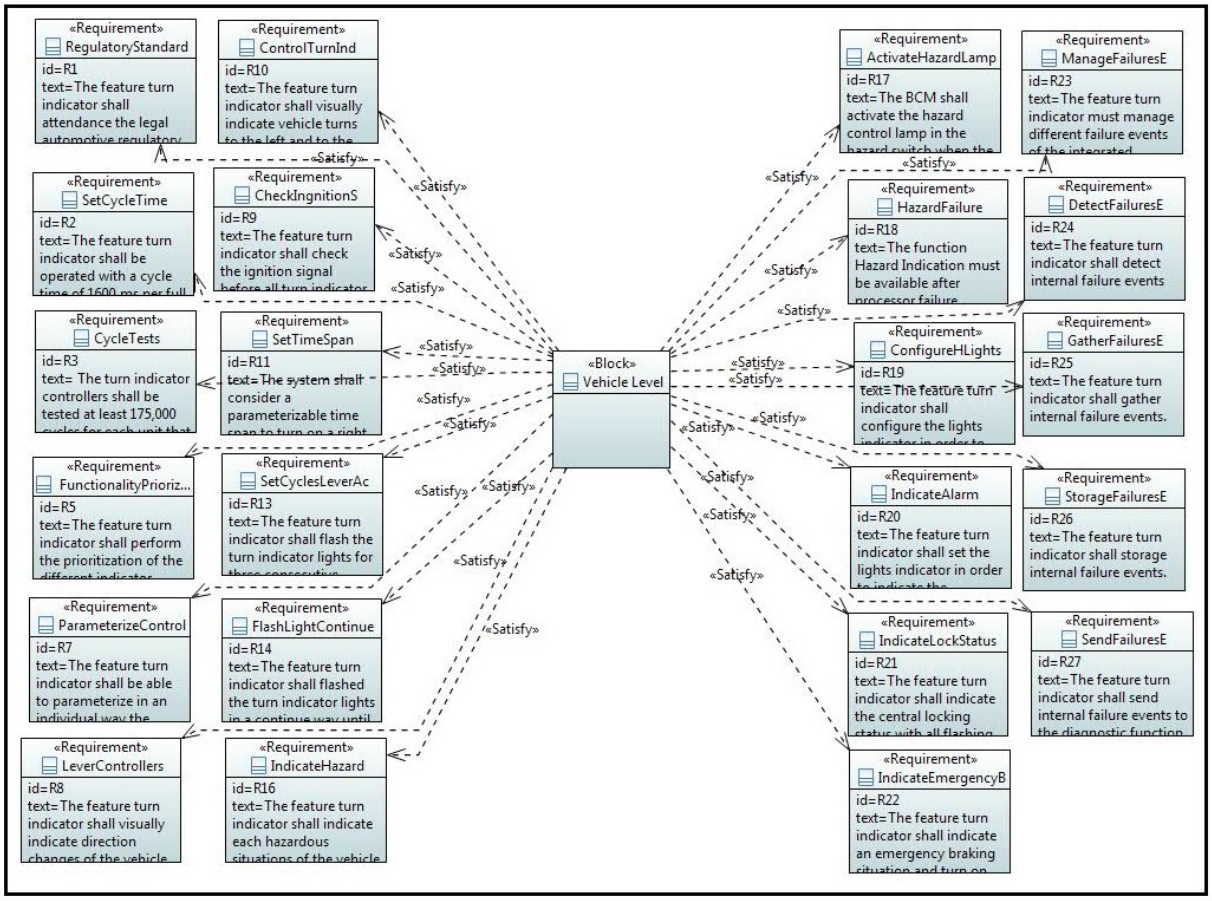

Figure 30 – Artefact of the Composition of Models using the MARTE Profile and VSL Formalism - **SysML Requirements Diagram**.

The MARTeSys^{ReqD} methodology focuses on specific timing RTES concerns and the proposed analysis handles **deadline**, **period** and **events** concerns. Thus, designers could initially (**1**º) search for these RTES concerns in the requirements specification, (**2**º) follow the proposed correlation of natural keywords to Timed Automata formalism and (**3**º) adopt the proposed MARTeSys^{ReqD} Grammar for the NL-TA Transformation (see the proposed Grammar in Chapter 4). Figure 31 depicts the final model, considering the Analysis of Requirements view, focusing on the formalization of specific timing constraints.

As noted from Figure 31, timing constraints already specified, on Natural Language (NL), in Table 9, are formally designed. The overall design model follows the MARTeSys^{ReqD} guidelines for the activity of Analysis of Requirements described in Chapter 4. In accordance with the proposed methodology, **periodic constraints**, for example, are modeled by the following components of the Timed Automata tuple: Clock (P), Guard, Invariant and Reset Operation. This activity considers that the deadline is shorter than the period. Thus, designed models respect the non-functional requirements and constraints specified in the Requirements viewpoint. It considers, in particular, the timing

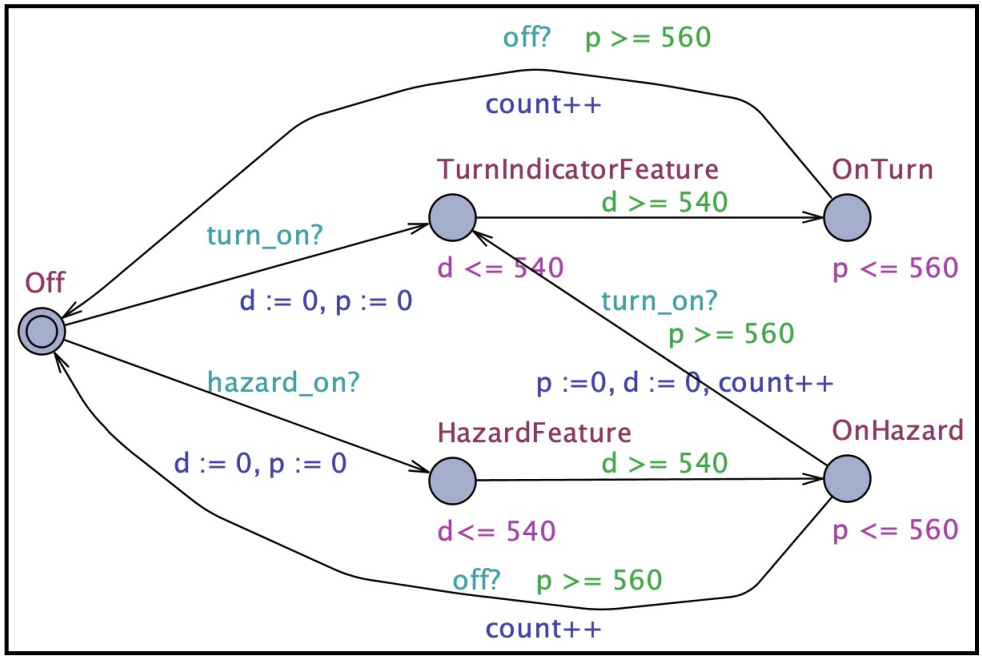

Figure 31 – Artefact of the Analysis of Requirements - The **Timed Automata diagram** of the Turn Indicator System.

constraints specified in the Requirements Pre-Analysis activity and refined in the High-Level Description of Requirements activity.

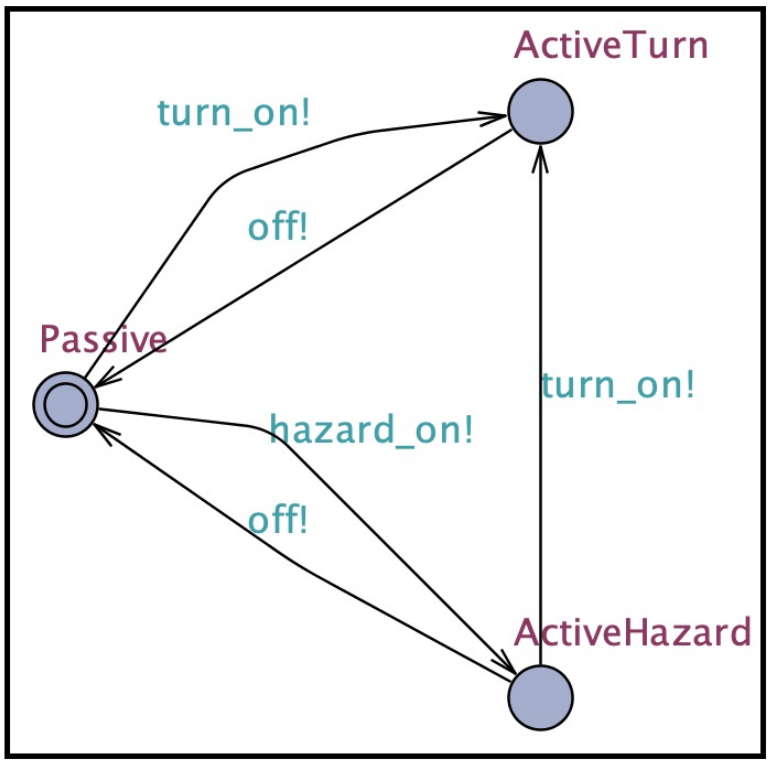

Figure 32 – Artefact of the Analysis of Requirements - The Trigger **Timed Automata diagram**.

The Turn Indicator feature has a periodic constraint value of 560 units of time. Thus, the initial location of the automata (named as "Off") is activated periodically after each 560 units of time. Therefore, all the automata locations, regarding the turn indicator functions (turning the lights on, turning the lights off and hazard flashing), must be able to be periodically activated each 560 units of time.

Actions "turn_on?" and "hazard_on?" represent the trigger events of this model. Figure 32 shows the system events. From the state *Off*, if the "turn_on?" event is true, the state *TurnIndicatorFeature* runs. As depicted in the Figure 31, the invariant "d $\leq$ 540" and the guard "d $\geq$ 540" forces the activation of the turn indicator. Thus, the state *OnTurn* is activated from $d, p \leq 540$ to $p \leq 560$, that is, $540 \leq p \leq 560$ units of time. After this and by adopting exclusively **Guard**, **Invariant** and the **Reset Operation** components, as stated in MARTeSys^{ReqD} methodology, it is possible to accomplish the periodic controllers of the Turn Indicator. After this, action "off?" becomes true, (once the state *ActiveTurn*, from Figure 32, sets "off!" action as true), the state *Off*, in Figure 31, is reached again. Figure 32 depicts the trigger automata for the Turn Indicator feature (*ActiveTurn* state) and the Hazard (*ActiveHazard* state). In UPPAAL, these two automata are synchronized by channels synchronization that allows the interaction via the actions within the channels. Chapter 7 depicts the model checking validation of artefacts from the Analysis of Requirements. In order to accomplish this, it applies Uppaal tool and TCTL equations to verify safety, reachability, deadlock and timing constraints, especially the period and deadline, of designed models.

From the Requirements viewpoint the first models of Functional viewpoint are designed. In this research, the scenarios of the Use Case diagram and the system categorization provide the main system services. Therefore, each group of services, into one specific use case, is related to one structural block (in the first level of Functional viewpoint).

6.2.2 Functional Viewpoint with the MARTeSys^{ReqD} Methodology

Functional viewpoint aims to provide another view of the TIS that focuses on structural modelling of the system services. Figure 33 shows the adoption of SysML Block Diagram and provides a complete structural perspective of the system components.

These early views of the Functional viewpoints are responsible for documenting the system structural context with focus on system services. Another refinement of the Functional viewpoint is necessary to contemplate and detail the block relationships as, for example, hierarchic and dependence relationships. Therefore, Figure 34 provides refinement of the Functional view through the Internal Block diagram.

Figure 34 shows structural design of categories specified in Table 9. However, some

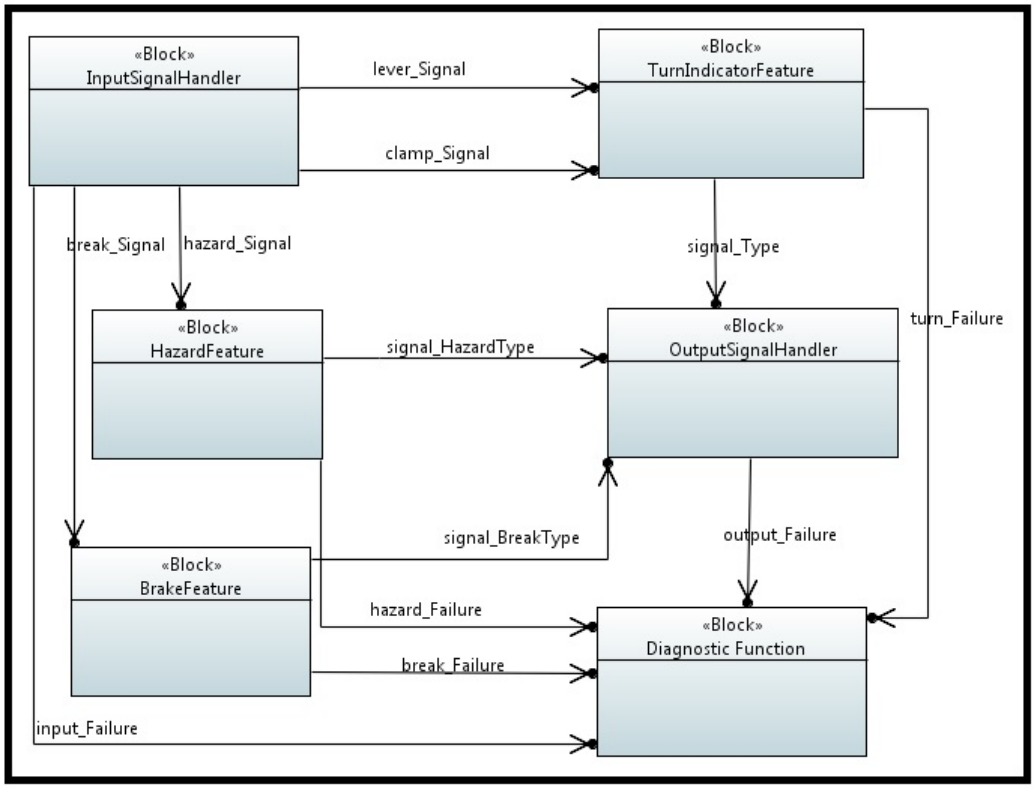

Figure 33 – First Refinement of Functional Viewpoint - **SysML Block Diagram**.

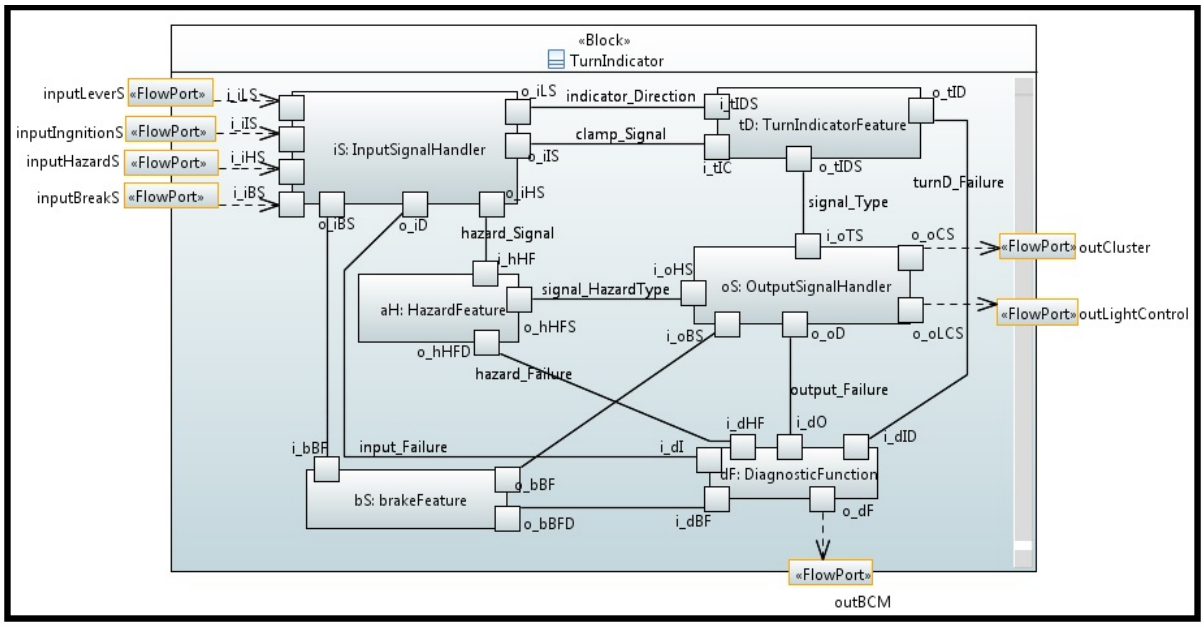

Figure 34 – Second Refinement of Functional Viewpoint - **SysML Internal Block Diagram**.

categories are not considered in this refinement. As an example, it is noteworthy that BCM's communication for the turn indicator features and BCM diagnostic failures are not described. Thus, from this point, the design activities detail essential services of the Turn Indicator along architectural viewpoints and system implementation. The cho-

sen categories as, for example, the **Hazard Feature** and **Turn Indicator Feature**, constitute the fundamental features of the studied system.

The focus of this second view, of Functional viewpoint, is to define the system components and their communication interfaces. These models are refined again, in the Functional viewpoint, in order to annotate or refine non-functional information of architectural models. Figure 35 shows the application of the MARTE profile in order to refine Functional Models.

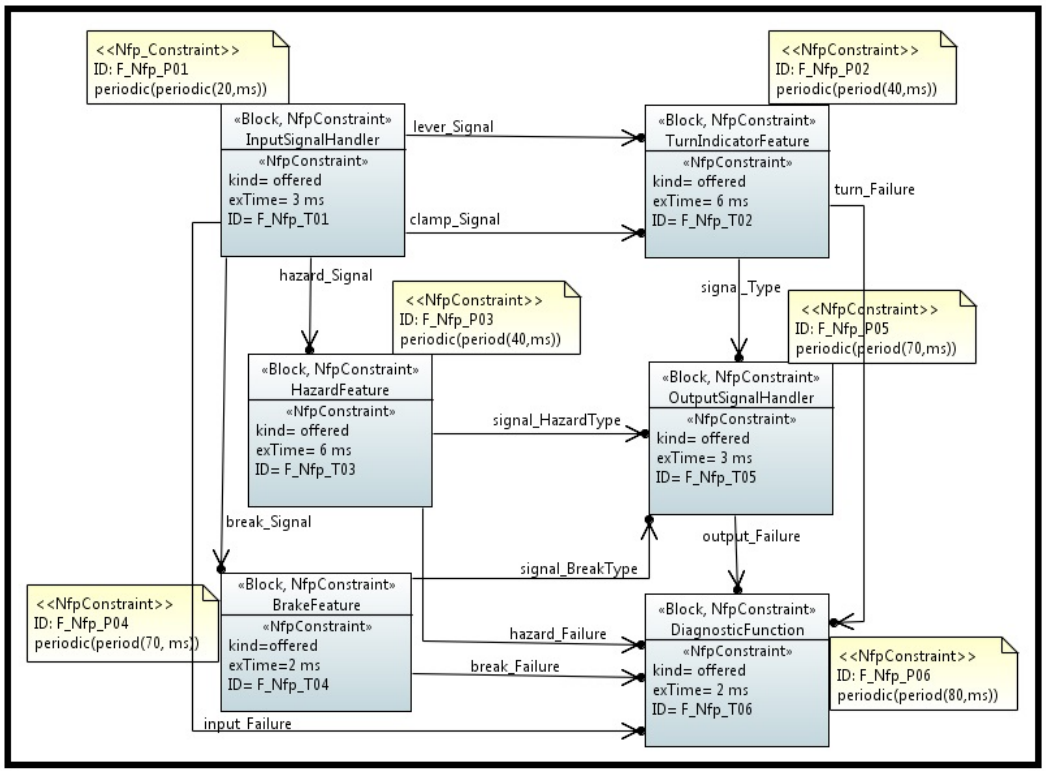

Figure 35 – Third Refinement of Functional Viewpoint - **SysML Block Diagram with MARTE annotations**.

Figure 35 describes another refinement of Figure 33. This first model is enriched by a $\ll NfpConstraint \gg$ stereotype. Each functional block has a period and deadline. The specified time refers to the partitioning of the TIS in components with their own timing constraints. These constraints were specified on Table 9. This stereotype aims to apply a condition or restriction to modeled elements. In functional models, these constraints are applied to each structural block in order to provide early timing estimations. The premature annotation and analysis of timing constraints can contribute to the correct development of the turn indicator features. From a previous project analysis, in the automotive domain, an estimation was delivered for the worst-case-execution (execution time/deadline) of the turn indicator features. As noted from Figure 35, blocks **InputSignalHandler**, **TurnIndicatorFeature**, **HazardFeature**, **BreakFeature**, **OutputSignalHandler** and **DiagnosticFunction** must respect, in total, the deadline of 21 ms to

complete their execution. These annotations respect the requirements specification, performed in Sections 6.2.1.1 and 6.2.1.2 , and constrain future deadlines of the implemented system.

Additionally, these blocks are constrained by periodic timing information. These constraints represent the activation period of turn indicator features and how often their trigger could be activated. These timing constraints are also refined in the following viewpoints. Moreover, the timing annotations are input to the quantitative evaluation of Chapter 7. The scheduler adopts the period of 20 ms to check whether there is an input signal from the feature turn indicator.

6.2.3 Logical Viewpoint with the MARTeSys^{ReqD} Methodology

As described in Chapter 4 and 5, it is important to define how artefacts of the Functional viewpoint are designed in the Logical viewpoint. Section 4.6.1 depicts the proposed mapping to the functional as well as to logical models of TIS.

Figure 20 represents the mapping between functional and logical blocks of the TIS. As seen from Figure 20, the *TurnIndicatorFeature* and *HazardFeature* functional blocks are combined into one single Logical block. This initial design decision affects how Logical viewpoint components are hereafter designed.

Figure 36 depicts a logical overview of the TIS and its respective logical components for each functional block. Thus, the Functional blocks **InputSignalHandler**, **TurnIndicatorFeature**, **HazardFeature** and **OutputSignalHandler** are refined in Figure 36.

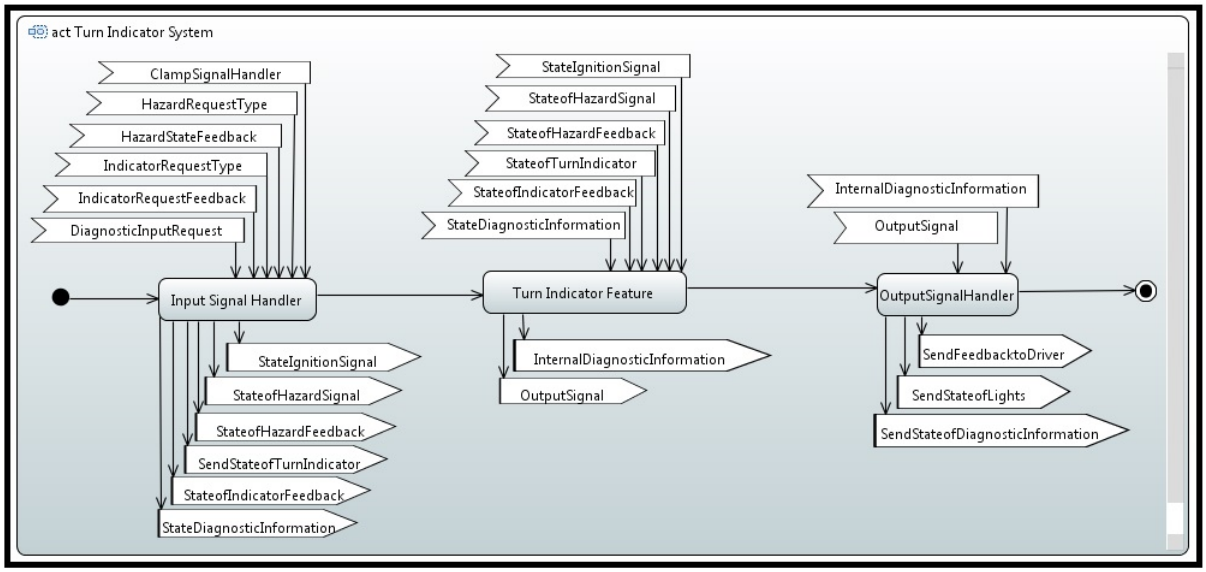

Figure 36 – First Refinement of the Logical Viewpoint: Turn Indicator Model - **SysML Activity Diagram**.

The activities *Input Signal Handler*, *Turn Indicator Feature* and *Output Signal Handler*, modelled in Figure 36, are composite activities. These global activities specify

broader activities being subdivided into distinctive actions. Therefore, Figures 37, 38 and 39 show another refinement of the Logical viewpoint and detailed behavior and actions of these composite activities. The models are individually designed in order to increase the legibility and detail level in their definition.

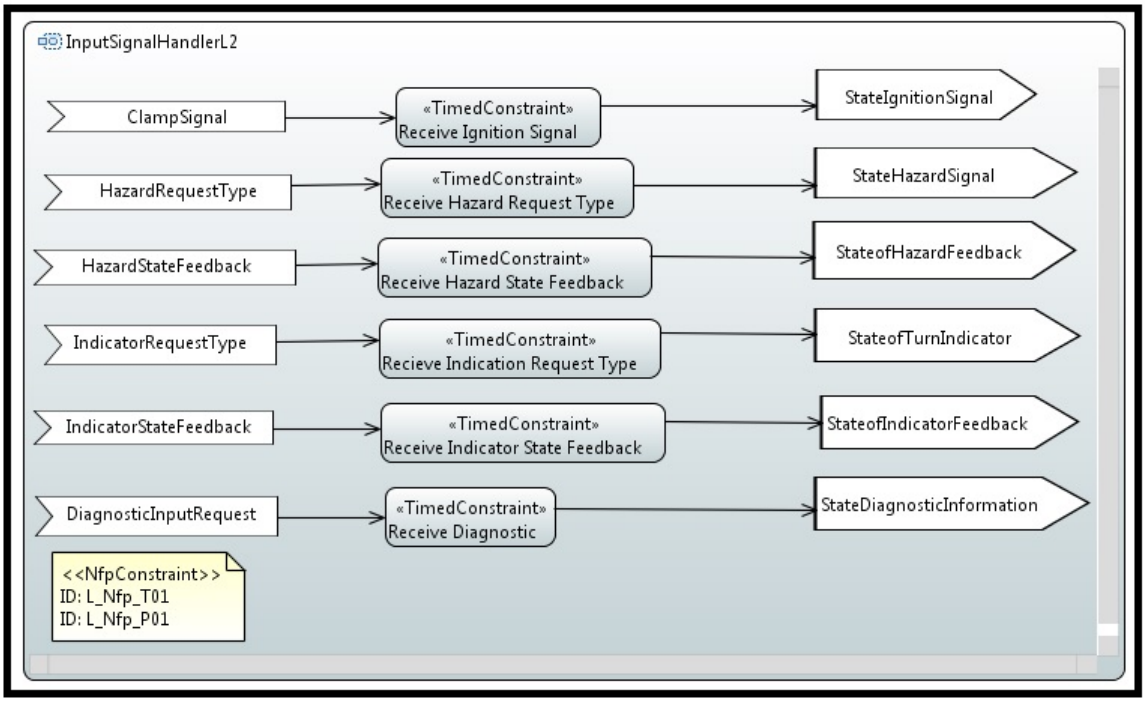

Figure 37 – Second Refinement of the Logical Viewpoint: Input Signal Handler - **SysML Activity Diagram**.

Figure 37 presents the internal activities of *InputSignalHandler* activity. It describes the actions related to hazard and turn requests from the driver, diagnostic receiving and power evaluation (from the clamp signal). Furthermore, this diagram describes external system signals (events), which trigger the initial system states. Moreover, it models the internal signals, which trigger the states of *TurnIndicatorFeature* diagram (Figure 38).

Figure 38 highlights the sub-activities of *TurnIndicatorFeature* activity (from Figure 36). This model shows the internal actions of turn and hazard controllers in a rational and interconnected group of actions. Thus, it details all possible states for signaling the turn directions and the hazard lights. Moreover, this diagram depicts the generated internal events such as *InternalDiagnosticInformation* and *OutputSignal*, which are input for the model in Figure 39 model. Figure 39 describes the actions of the output controllers of the TIS. It highlights the signals generated for the instrument cluster and the light control unit. These models also depict actions and internal signals for the BCM diagnostic module.

MARTE stereotypes refine the activities of the proposed models with timing information and constraint to trace these along the viewpoints. The artefacts from the Logical viewpoint should pass by different types of verification and validation, such as simu-

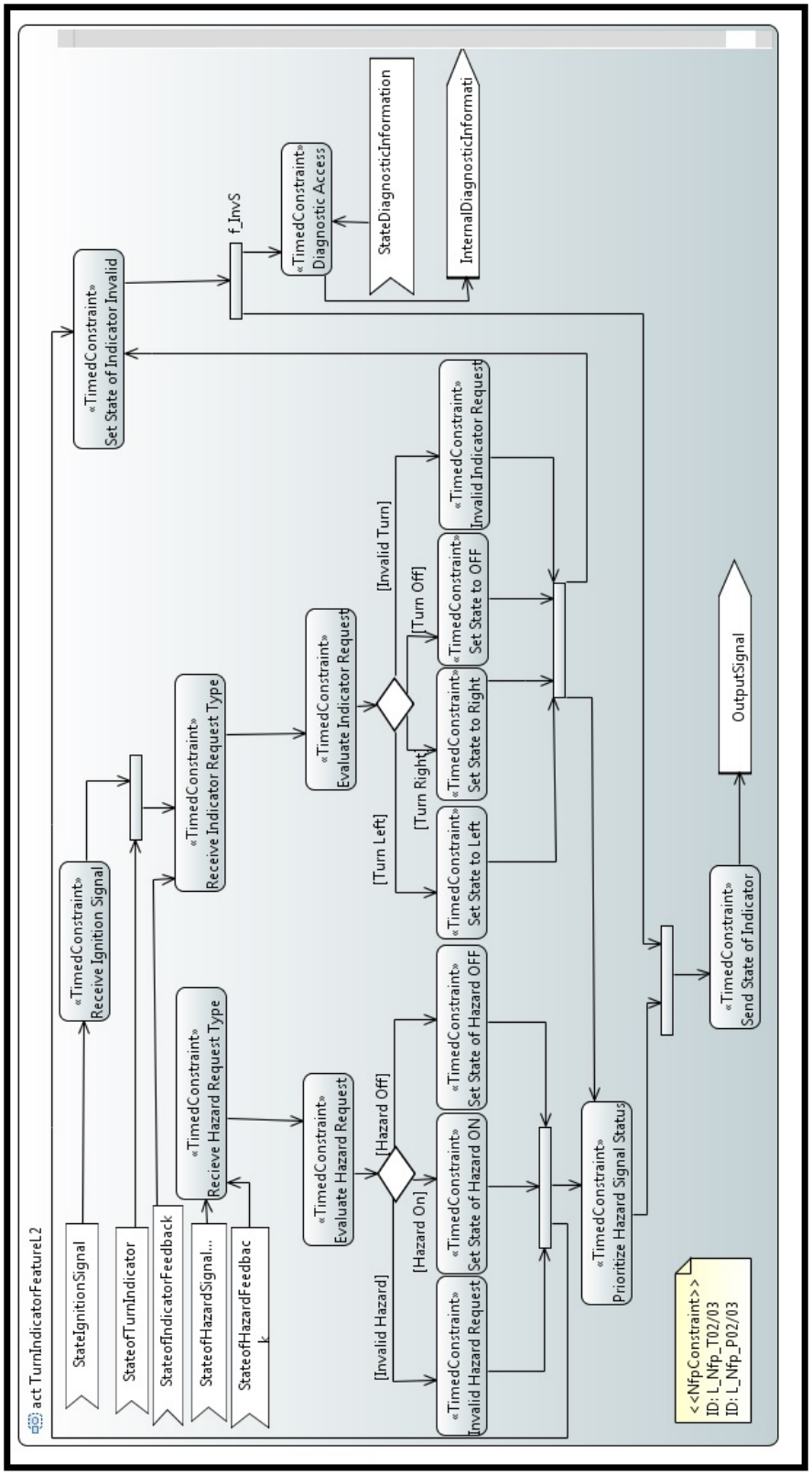

Figure 38 – Second Refinement of the Logical Viewpoint: Turn Indicator Features - **SysML Activity Diagram**.

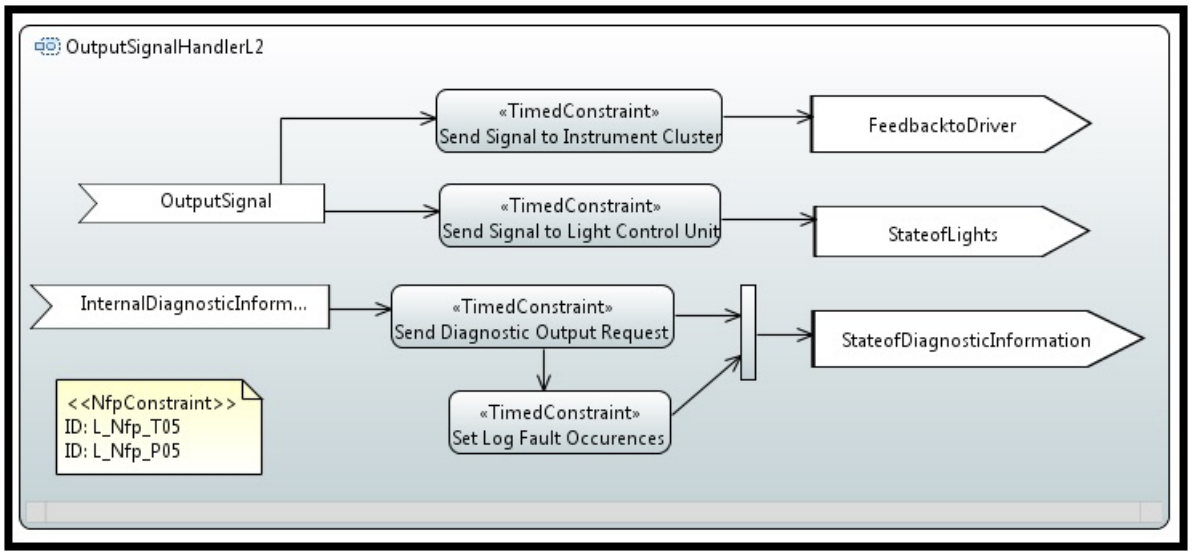

Figure 39 – Second Refinement of the Logical Viewpoint: Output Signal Handler - **SysML Activity Diagram**.

lations and formal checking of their critical constraints [142], [18]. Therefore, the $\ll TimedConstraint \gg$ stereotype allows one to refine the functional blocks in a more specific manner. There exists the possibility, through the attributes of $\ll TimedConstraint \gg$, for example, to declare the timing type of one activity (duration or instant) and the value of this attribute. From these annotations, distinctive analysis can be performed.

Adoption of these annotations describes non-functional characteristics, which constrain and trigger system behavior. This strategy contributes to the expressiveness of RTES behavior models and also provides guidelines for future design activities. Artefacts of this viewpoint will be further refined in the Technical viewpoint.

6.2.4 Technical Viewpoint with the MARTeSys^{ReqD} Methodology

Based on all SysML Activity diagrams, from logical viewpoint artefacts, the Technical viewpoint models are generated. Technical models describe the design decisions, at high level of granularity, in order to show how software and hardware components should be developed.

The Technical viewpoint should represent for each technical component their related resources. In this case study, the technical models depict the main distribution of Turn Indicator components between software or hardware elements.

Figures 40, 41 and 42 show the second refinement and the design decisions of the Technical viewpoint. The design decisions regarding how the system implements technical components, are highlighted, in this view, by adopting swimlanes and MARTE stereotypes.

The adoption of swimlanes and MARTE stereotypes allows for the description of the

design decisions proposed in MARTeSys^{ReqD} methodology. Here, MARTE stereotypes can describe how system components, such as *ClampSignal*, are defined by hardware/software components (it is considered as an analog input in the system). Moreover, the swimlanes description produced design decisions, which split (cluster) the system activities into more general software or hardware containers.

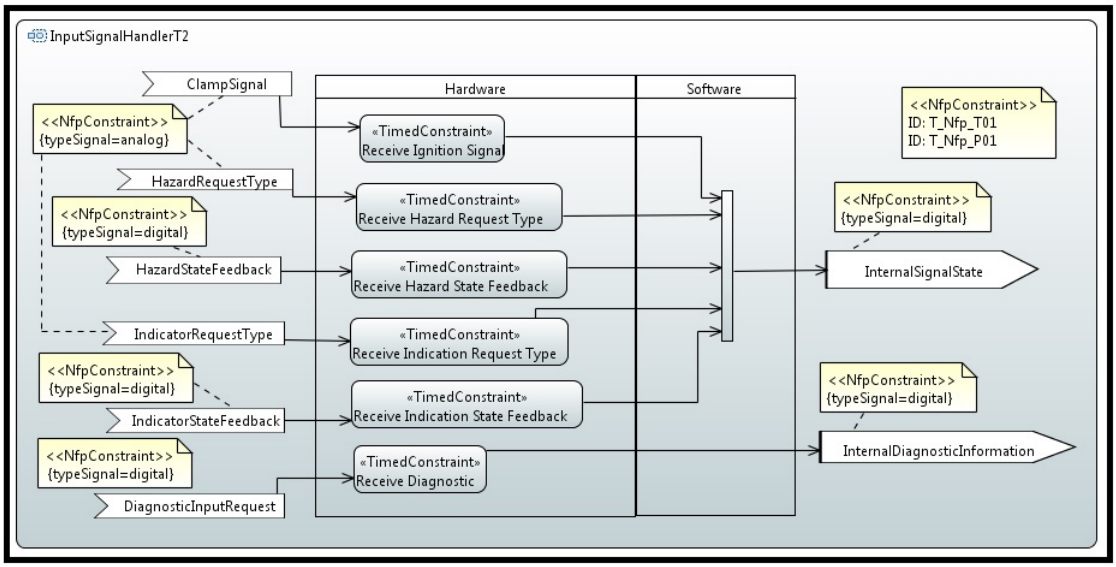

Figure 40 – Technical Viewpoint: Input Signal Handler.

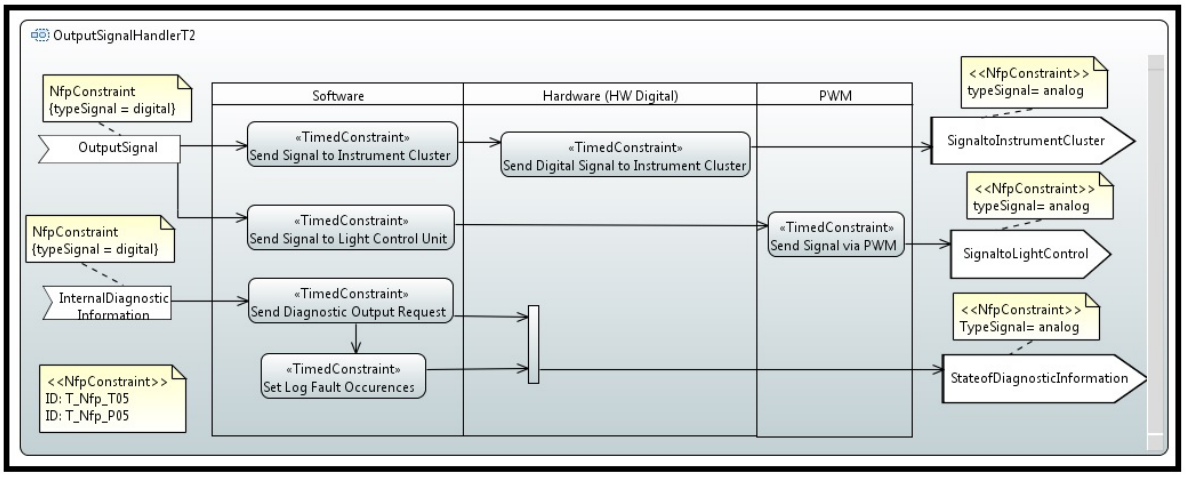

Figure 41 – Technical Viewpoint: Output Signal Handler.

Figure 40 shows that the digital and analog signals, obtained from the sensors, are mainly caught by a hardware element. An analog "ClampSignal", for example, can have a specific hardware/actuator device to check its conditions and transform it into a digital signal. The $\ll NfpConstraint \gg$ stereotype annotates information about the type of each system signal.

Figure 41 depicts the design decisions for the output signal handler. As noted, here activities can be implemented as logical components or as hardware embedded compo-

nents such as Pulse-width Modulation (PWM) control. In the case of PWM, a swimlane can also be used to represent a more specific software/hardware container of the technical solution. Figure 41 shows how the digital signals, described in Figure 39, can be distributed among swimlanes and transformed into analog signals. The main interactions of the TIS with lights, instrument cluster and the BCM are shown.

Figure 42 shows the main features of the turn indicator. In this model, the hazard and the turn features are stored in a software swimlane. It shows that both system features should be able to interpret the "InternalSignalState" in order to logically control the hazard behavior or the turn indicator behavior. The internal and digital "output" signals are input to the Output Signal Handler model.

From the Technical viewpoint models, the global system blocks can be designed. This model allows one to describe how system components interact and show globally the main architectural decisions for system development.

6.3 Model and Unit Design Models

Figure 43 defines the Global System Architecture of the TIS. Here, internal/external interfaces of each component are described in a high abstraction level. In this view, black boxes can represent software or hardware elements. MARTE stereotypes sre employed to add a semantic to these components in order to highlight different resources, such as processors and devices, and processing units.

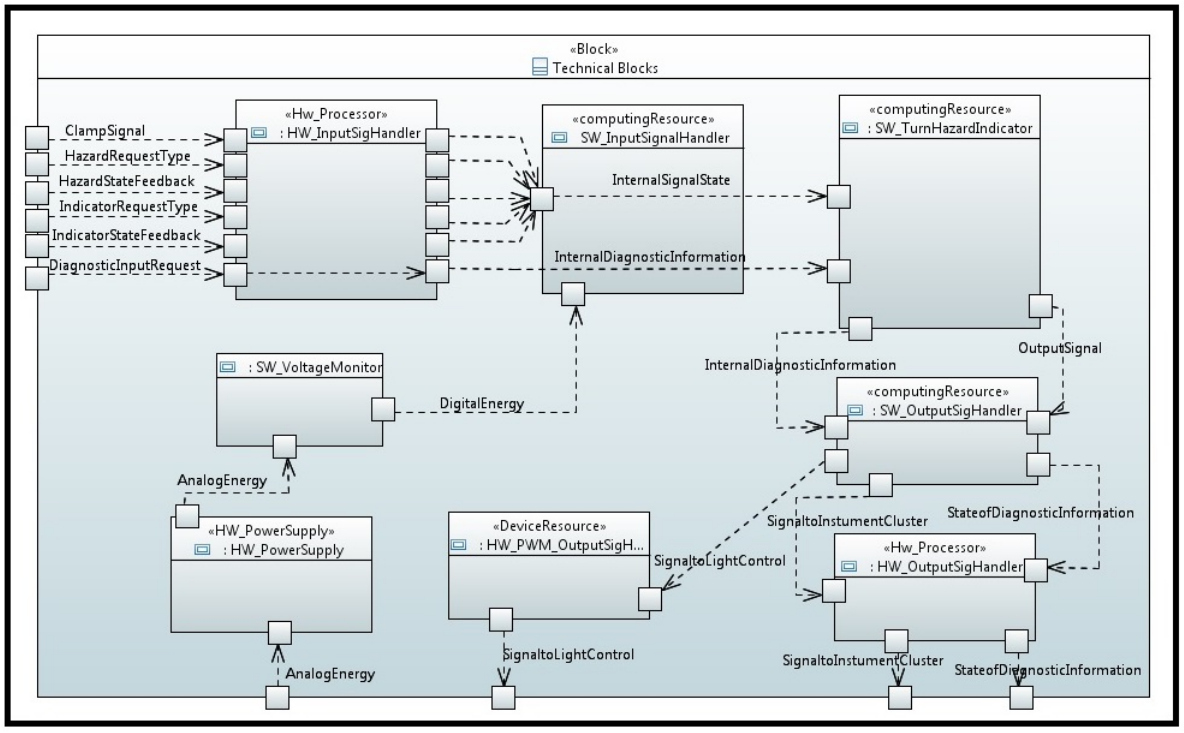

Figure 43 – Global System Architecture of Turn Indicator.

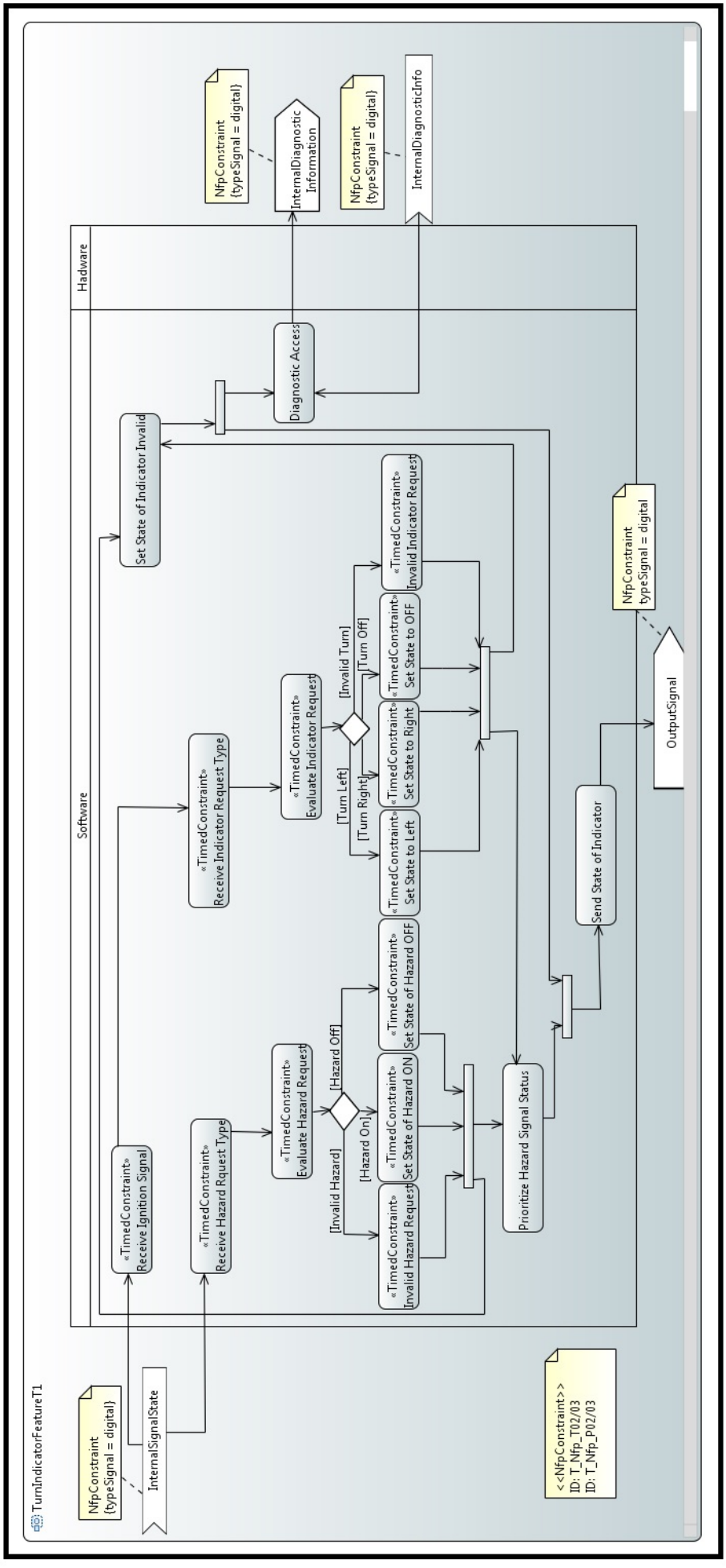

Figure 42 – Technical Viewpoint: Turn Indicator Features.

Different architectural levels are consistent with each other and the constraints imposed at different architectural levels have been maintained and refined throughout the architectural definition. These real-time embedded concerns are traced from Requirements, Functional, Logical and Technical viewpoints and described, in an implementation view, in Model and Unit Design (MUD) phases. Figure 44 describes the Classes diagram of the case study in accordance with the system Functional viewpoint, along with the expected software components (as described in Figure 43).

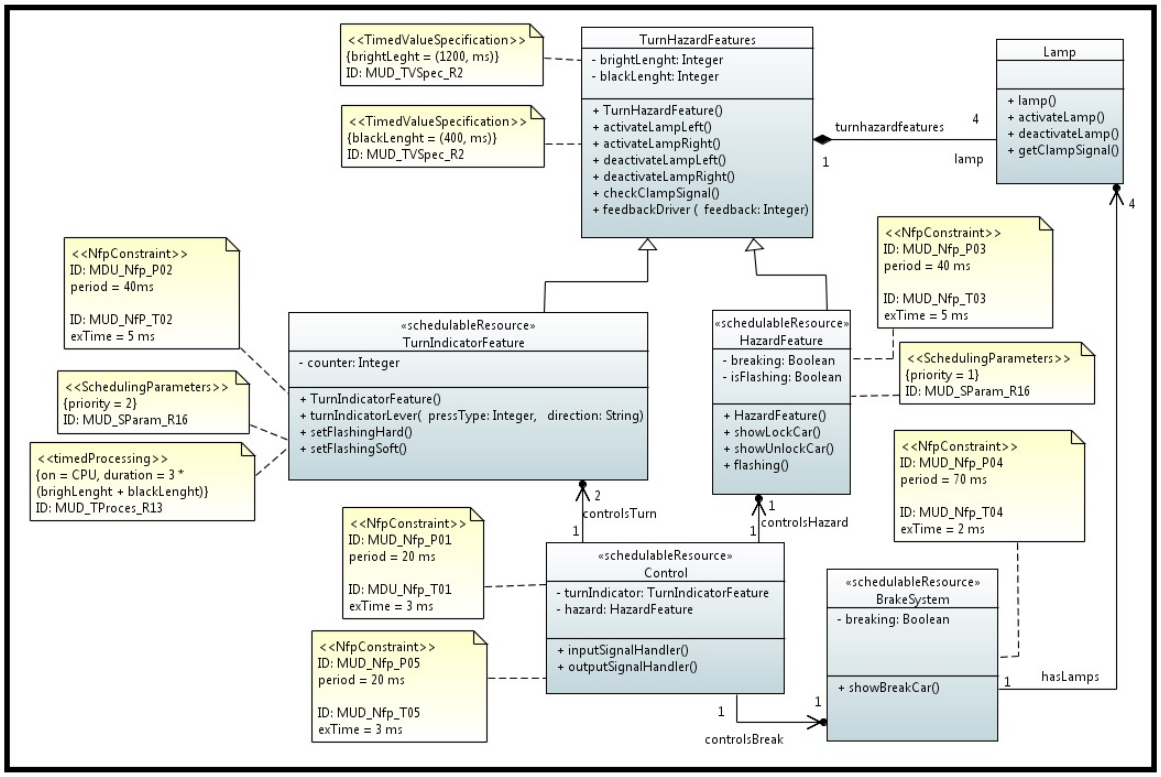

Figure 44 – Class Diagram of Turn Indicator with MARTE Constraints.

Figure 44 depicts the main system classes and their associations. These classes are named as *TurnHazardFeatures*, *Lamp*, *TurnIndicatorFeature*, *HazardFeature*, *Break* and *Control* as they are posteriorly elucidated. As noted in Figure 44, the $\ll TimedValueSpecification \gg$, $\ll schedulingParameters \gg$, $\ll SchedulableResource \gg$, $\ll timedProcessing \gg$ and $\ll NpfConstraint \gg$ constraints were added to the UML Class diagram. These stereotypes allow for the defining of information regarding precedence, resource, timing and processing constraints in low abstraction level.

The strategy to describe and refine the MARTE constraints, from high-level abstraction models, allows one to trace non-functional concerns at development models and contributes to their further analyses. In this thesis, an automatic translation performs the generation of graphical and textual constraints to code. Thus, from the architectural viewpoint models through the MUD view it is possible to specify, model and analyze different RTES concerns.

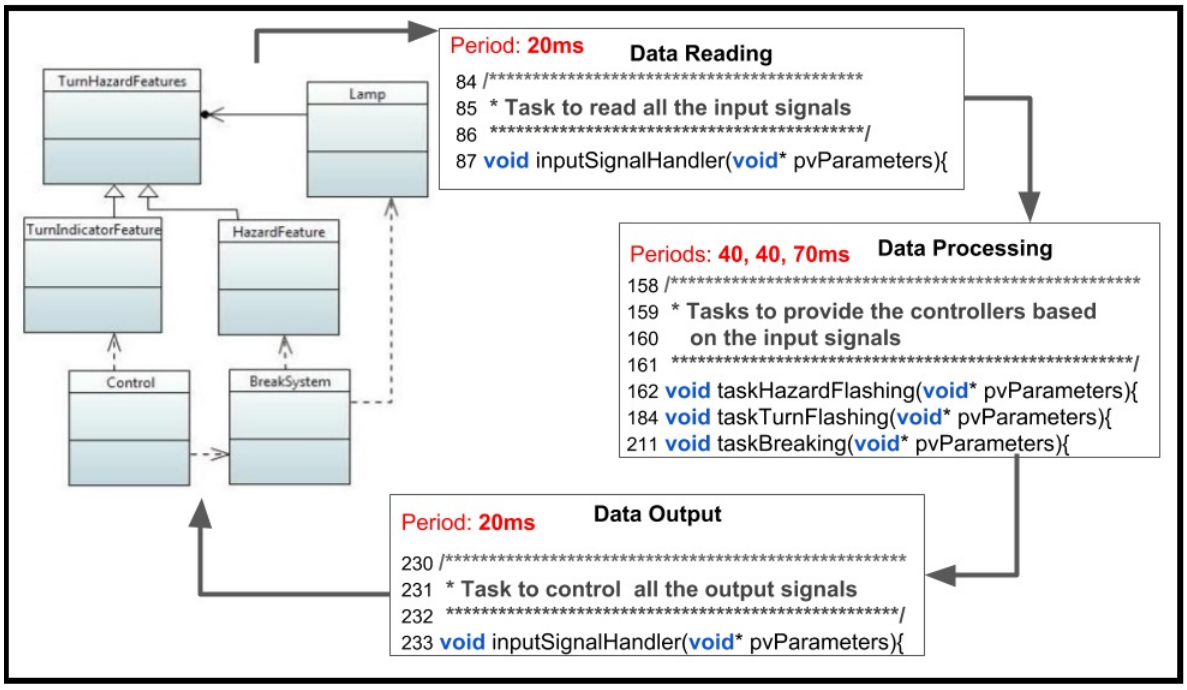

Figure 45 – Tasks of Turn Indicator System.

Figure 45 depicts the overall system tasks and describes how the classes of TIS are defined in the code implementation. Annex D, defines the implementation of the proposed case study based on MUD models. Section D.0.2 provides an explanation of concepts to extract the TIS specification and to perform the implementation of timing requirements. Section D.0.1 specifies the toolbox adopted by MARTeSys^{ReqD} Methodology.

6.4 Tracing Real-Time Embedded Systems Constraints to Implementation Models

The elaborated strategy to trace RTES constraints, from MARTE profile annotations, allows for their refinement over the Requirements, Functional, Logical and Technical viewpoints. These constraints formally annotated by VSL are transferred from the models and strategically positioned into the code in order to contribute to development activities. Moreover, after development of the system, it is possible to perform the empirical evaluations.

Figure 46 shows slices of the architectural viewpoint models and of the source code of the TIS case study.

Figure 46 represents, on the left-hand, the *TurnHazardFeatures* classes and their constraints. These constraints, enumerated with labels 1 and 2, are automatically attached in the code as recommendations to software developers. This scenario is presented on the right-hand of Figure 46. These comments are generated and placed in the code. There-

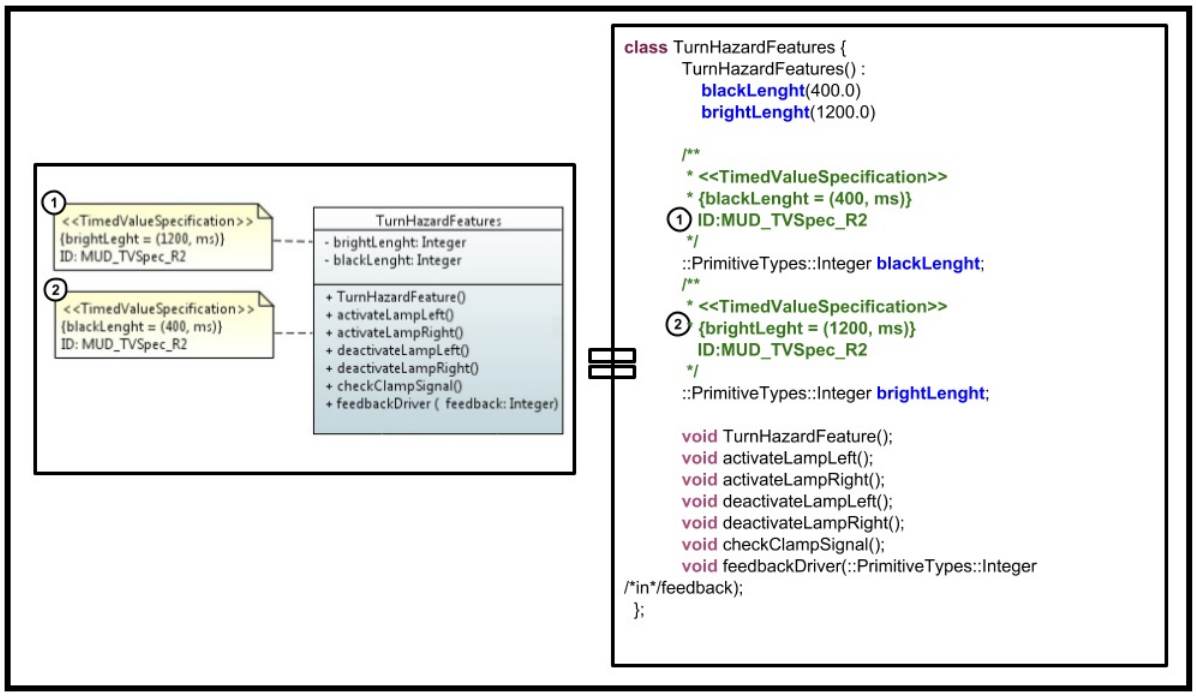

Figure 46 – Tracing Non-Functional Constraints to System Implementation.

after, developers must implement the system and include these comments and guidelines in the system realization (development). The studies developed in [184], [64], [122], can be integrated into the proposed methodology in order to guarantee that these comments and guidelines (proposed as VSL constraints recommendations) are implemented.

RTES are characterized by the correctness of their logical results and especially by the capability to hold non-functional concerns along the development stages. The contribution described in this section can guide developers in code implementation, minimize possible human errors and accelerate development activities, since these constraints are already closer to the final code. Therefore, it has the purpose of contributing to the understanding, definition and consistent implementation of non-functional concerns of RTES.

The elaborated strategy to trace RTES constraints, from MARTE profile annotations, allow for their refinement over the Requirements, Functional, Logical and Technical viewpoints. These constraints formally annotated by VSL are transferred from the models and strategically positioned in the code in order to contribute to development activities. Moreover, after the development of the system, it is possible to perform the empirical evaluations.

Automatic translation performed here is able to generate graphical constraints, from viewpoint models, as textual information in the code. This transformation allows highlighting RTES properties from the architectural models, to be used as formalized guidelines by developers directly from the generated code. As Figure 45 depicts, the system is fully developed based on the generated code allowing measurements and the empirical

evaluation of non-functional properties of the system.

6.5 Contributions

This chapter presented the use of the MARTeSys^{ReqD} methodology for the development of the TIS. The four viewpoints depict the design decisions and shows the main architectural artefacts highlighting how RTES concerns were addressed. A systematic and traceable representation of these concerns, in different views, can contribute to fulfilling the general objective of a system (system requirements). The MARTE profile stereotypes make explicit several embedded concerns and non-functional properties, and highlights necessary information at different development views.

It is supposed that this strategy decreases the level of complexity in the real-time and embedded development process. The main contributions of this chapter were previously discussed in [100] and [101] and these are summarized as following:

1. Four different system viewpoints are presented in this research. Models at the design level describe real-time and embedded requirements in early system development process. These models allow one to depict concerns of system stakeholders, to validate properties that are important to the system under different perspectives and description levels.

2. The presented methodology is able to represent the general architecture of the system and its views, to highlight the interaction between design components, and to describe communication interfaces through viewpoints of the system. This architectural description has to describe RTES in accordance with a functional, non-functional and embedded perspective of the system.

3. A subset of MARTE constructors is applied to the system models in order to produce possible representation of the main functions and embedded concerns in design models. The proposed strategy highlights how to annotate non-functional information in RTES models at different abstraction levels.

4. This research study represents models that can be evaluated. Therefore, it is possible to develop analyses that are relevant to RTES as, for example, temporal and performance analyses.

Chapter 7

Evaluation of the MARTeSys^{ReqD} Methodology

This chapter describes the strategies applied to analyzing and evaluating non-functional constraints of the architectural design. Furthermore, it presents the results of the MARTeSys^{ReqD} qualitative evaluation. The main strategies employed to evaluate the proposed study are empirical validation through simulation of specific system scenarios, model checking and qualitative evaluations.

7.1 Quantitative Evaluation of the MARTeSys^{ReqD} Metodology

Quantitative Evaluation involves the analysis of traceability, verification and fulfilment of RTES constraints along the development cycle. Therefore, it is necessary to provide the system development, in this study, due to the fact that:

- **It traces the RTES constraints to the final development stage**: the proposed methodology performs refinements of non-functional concerns along the four viewpoints: Requirements, Functional, Logical and Technical. Moreover, it provides an automatic generation of MARTE stereotypes, from graph models, to system deployment models. The steps to this transformation are shown in Figure 8, and these consider the Global Architectural design and MUD artefacts. The tracing of these constraints allows one to connect information of design artefacts to system developers along the system viewpoints.

- **It presents the description of RTES concerns in a low granularity level**: adoption of VSL formalism in the architectural models allows one to formalize the annotation of non-functional properties in architectural models as described in

Chapter 2. A strategy to code generation, from MUD models, performs a representation of these constraints by VSL formalism. Therefore, annotations of RTES constraints is performed in a standard and formally defined syntax for to the developers.

- ❑ **It allows for the measuring of timing concerns, from the architectural models, from a dynamic perspective**: strategies to check tasks constraints, through runtime behavior of tasks, can contribute to estimate the execution time values of non-functional constraints. As shown in Section 7.1.1, empirical measurements are performed here by simulation of system scenarios and their timing constraints. Other strategies for analyzing time were also present in Chapter 6 and it shows a formal verification of temporal system constraints.

- ❑ **It provides a strategy to verify the constraints in different perspectives**: it is possible to check the constraints from the architectural design models, as for example, the Class diagram constraints, against the implemented/executed constraints. In this case, it is possible to evaluate the real values of timing constraints, which were previously annotated by MARTE stereotypes.

- ❑ **It contributes to maintenance of timing constraints at different viewpoints**: it is possible to figure out possible discrepancies between the predefined values of constraints and those that come from the automated verification activities. Refinements can be proposed when there are inconsistencies between the initial values, which were modeled by designers, and the simulated values, after dynamic execution, of constrained values. Besides, it is necessary to improve the models or to evaluate the design decisions regarding the software or hardware platform.

Annex D provides an overview of the system development. The following section defines the empirical evaluation of timing constraints and the simulation results.

7.1.1 Empirical Evaluation of MARTE Constraints

In this thesis, simulation and measurements of constraint values are performed in order to check correctness, consistency and feasibility of initial timing annotations. The empirical validation considers the runtime behavior of the tasks. It allows for timing measurements in the design and development models. Moreover, it enables, through simulation techniques, to validate the constrained runtime behavior. These measurements are performed regarding execution time and scheduling policy of each task. Here, the empirical evaluation of RTES constraints aims to answer **two questions**:

1. Do the system tasks maintain their deadline?

2. Does the periodic task set fulfil the requirements of the scheduling policy?

Simulation of RTES constraints can support correctness of design decisions, favor quantitative analyses of non-functional concerns and contribute to refinements of these constraints even after system implementation. Furthermore, it contributes to investigation of empirical evaluation questions.

System implementation respects and follows the traced constraints from the Class diagram model of Figure 44. From these annotations, timing analysis of annotated constraints are performed dynamically. Table 10 provides details of these constraints, for each model element, and presents the results of timing and precondition simulations. The empirical analyses consider the execution time of the system tasks and the simulation results are based on 1500 execution samples. Figures 35 and 44 provides an overview of the constraint annotations of MARTeSys^{ReqD} architectural models.

The FreeRTOS scheduler always executes the highest priority task first. In this implementation, it is assumed that the period of each task defines its priority. Here, tasks with shorter periods have the higher priority. Therefore, for different values of period, the Rate Monotonic (RM) scheduling is followed. However, tasks with the same priority (period) are selected by the policy of Round-Robin (RR) scheduling. Table 11 shows the assignment of tasks τ_1, τ_2, τ_3, τ_4 and τ_5 for the processor regarding the fixed-priority/period constraints.

The Rate Monotonic scheduling has the following rules for scheduling a task set: tasks creation is static and these have the same arrival time, tasks with shorter periods are assigned higher priorities, no resources sharing between the tasks and it deals with independent tasks. However, the task set of the proposed study case is not independent. As an example of dependence relationship of the Turn Indicator tasks, one can state that: τ_2 depends on τ_1, τ_3 depends on τ_1 and so on.

In order to deal with precedence constraints, by RM scheduling, the dependent task set was transformed into an independent one. This happens by an adequate modification of timing parameters. The modification of release times and deadlines forces a situation where any task cannot start before its predecessors and cannot preempt their predecessors. Given the dependent tasks τ_2 and τ_1 , for example, with the respective release times r_2 and r_1; and computation times C_2 and C_1. The new release time for τ_2 is $r_2^* = max(r_2, r_1+C_1)$. Besides, the new deadline of τ_1 is replaced by $d_1^* = min(d_a, d_2 - C_2)$. The scheduling of Turn Indicator tasks considered in Figure 47 respects these modified parameters for all task sets. It is supposed that the new deadline and release time of τ meet the system constraints.

The case study proposed in Chapter 6 shows that the input and output tasks have the shortest period due to the constant evaluation of the input signals and the slow access of Turn Indicator actuators. Table 11 shows that the *inputSignalHandler* and

Functional Blocks	*Blocks: Constraint Type/ Attributes*	*Related Classes*	*Related Task - Constraint Type / Attributes of the Class*	*Simulation*
InputSignal-Handler	≪ *NfpConstraint* ≫: period(20, ms); exTime = 3 ms.	Control	**inputSignalHandler():** ≪ *NfpConstraint* ≫: period(20, ms).	**Period** = 17,66 ms; **exTime** = 46,18 us
TurnIndicator-Feature	≪ *NfpConstraint* ≫: period(40, ms); exTime = 5 ms.	TurnIndicator-Feature	**taskTurnFlashing**/ TurnIndicatorFeature(): ≪ *NfpConstraint* ≫: period(40, ms). **setFlashingHard():** ≪ *schedulingParameters* ≫: priority = 2. **setFlashingSoft():** ≪ *timedProcessing* ≫: duration = 3* (brightLenght + blackLenght); ≪ *Scheduler* ≫: schedPolicy = 2.	**Period** = 35,33; **exTime** = 3,79 us
HazardFeature	≪ *NfpConstraint* ≫: period(40, ms); exTime = 5 ms.	HazardFeature	**taskHazardFlashing**/ HazardFeature(): ≪ *NfpConstraint* ≫: period(40, ms) **flashing():** ≪ *schedulingParameters* ≫: priority = 1.	**Period** = 35,33; **exTime** = 5,50 us
BreakFeature	≪ *NfpConstraint* ≫: period(70, ms); exTime = 2 ms.	BreakSystem	**taskBreaking**/ showBreakCar(): ≪ *NfpConstraint* ≫: period(70, ms).	**Period** = 70,66; **exTime** = 4,08 us
OutputSignal-Handler	≪ *NfpConstraint* ≫: period(20, ms); exTime = 3 ms.	Control	**TurnHazardFeature():** ≪ *TimedValueSpecification* ≫ **brightLeght** = 1200 ms and **blackLeght** = 400 ms; **outputSignalHandler():** ≪ *NfpConstraint* ≫: period(20, ms).	**Period** = 17,66; **exTime** = 94,09 us

Table 10 – MARTE Constraints Simulation.

outputSignalHandler tasks have priority 3, which guarantees these the highest priority of the task set. Tasks *taskTurnFlashing* and *taskHazardFlashing* have priority 2 and follow the Round-Robin scheduling, as these have equal values for their periods. Finally, *taskBreaking* has the smallest priority. Figure 47 represents an initial scheduling of the system tasks in accordance with the rules of the Rate Monotonic policy.

In Figure 47, tasks τ_1 and τ_2 represent the *inputSignalHandler* and *outputSignalHandler* and receive highest priority. Task τ_3 is the *taskTurnFlashing* and τ_4 is the *taskHazardFlashing* and these have the same period/priority. Finally, task τ_5 represents the *taskBraking* and due to its higher period, it has the lower priority. Tasks priority, in accordance to their period, is depicted on Table 11. As presented in Figure 47, tasks

Task	Period	Priority	Scheduling Policy
τ_1	20	3	RM and RR between τ_1 and τ_5
τ_2	40	2	RM and RR between τ_2 and τ_3
τ_3	40	2	RM and RR between τ_3 and τ_2
τ_4	70	1	RM
τ_5	20	3	RM and RR between τ_5 and τ_1

Table 11 – Tasks and their Expected Scheduling Policy.

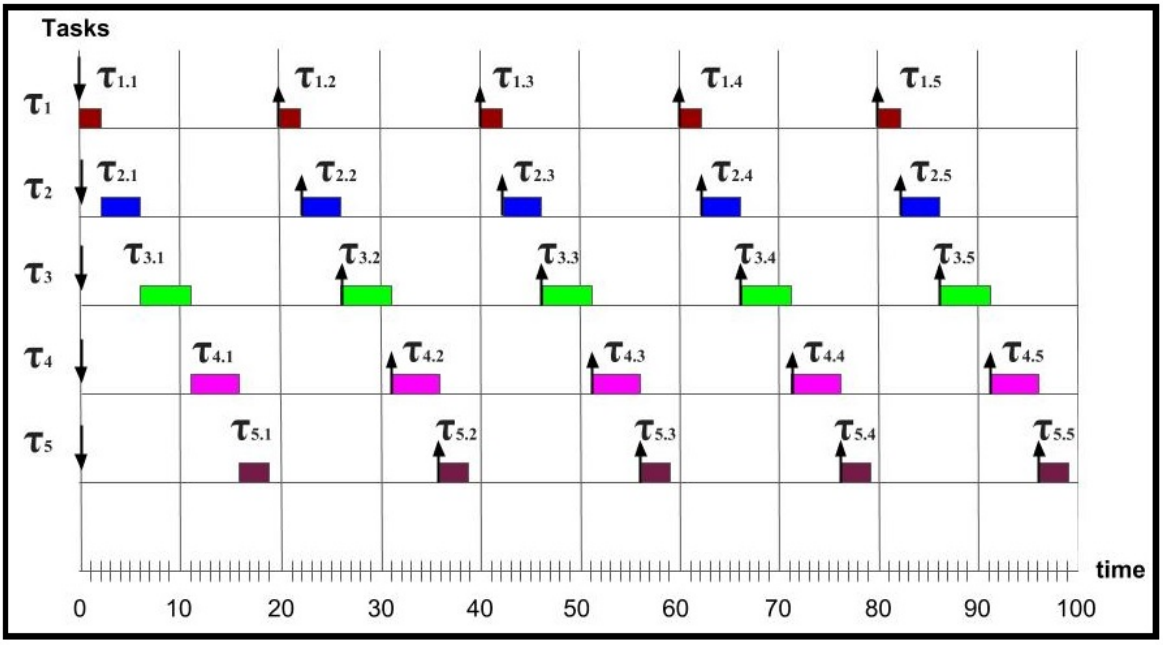

Figure 47 – Rate Monotonic Scheduling of the Tasks of the Turn Indicator Example.

run in accordance with their period and all the instances of tasks finished their execution within the expected deadline.

For the simulation activities, different timing measurements for the execution time were performed by *timestamp* function. Basically, the function *micros*() of FreeRTOS catches the initial and final time execution of each task and, from the difference of these values, generates an automatic log of 1500 samples. Based on this log, the average over the execution time is calculated. Figure 48 shows system tasks and the average for their time execution values in microseconds. As it can be observed in Figures 44 and 48, there are discrepancies between the annotated values and the simulated ones. However, this strategy is useful for checking plausibility between design assumptions in different abstraction levels (higher abstraction levels models versus development models.

The deadlines of each task (**question** 1), constrained as execution time, were overestimated in architectural and MUD models and these are not reached in the system realization. Simulation results show a discrepancy rate on a millisecond scale, constrained in the models, to a microsecond scale in the execution phase. Table 10 described the values for assumptions, of architectural viewpoints artefacts, and their respective measured

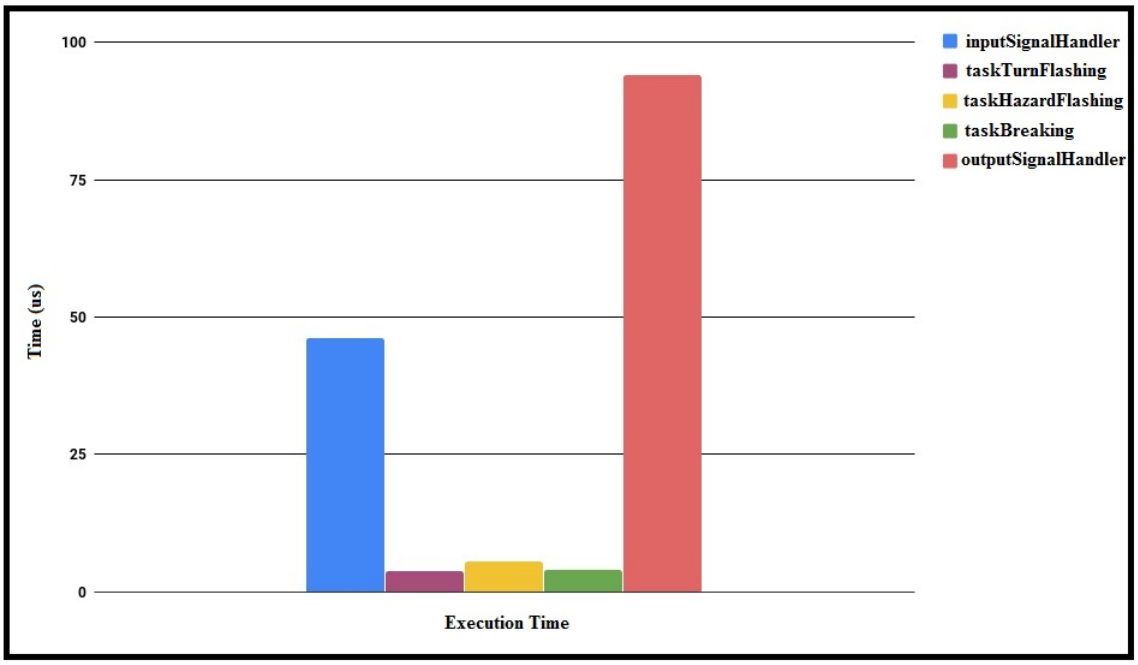

Figure 48 – Execution Time Simulation of Tasks.

results.

The Arduino microcontroller and its functions/libraries allowed for the implementation of the system. The function *digitalRead* is adopted to read the signals that came from the digital input ports and the interaction between physical and logical components is pretty simple. These interactions and physical/logical signals, which are exchanged between software and hardware components were described in the Technical viewpoint. However, for the actual RTES, this function should be developed following the chosen architecture. This may justify fewer discrepancies between the proposed constraints assumptions, over the execution time values, and the respective simulation results in the developed case study. However, even with lower simulation results in terms of tasks deadlines, it was possible to show that the system tasks can hold their deadline.

As it can be noticed in Figure 48, from the executed code, tasks *inputSignalHandler* and *outputSignalHandler* need a higher time of execution, since these are manipulating external events and controlling physical components. On the other hand, the other tasks can have shorter deadlines, since these perform logical actions to schedulability policies. Although there were discrepancies in the measured scales, both in the results presented in Figure 47 and those from Figure 48, the confirmation is reached that tasks deadlines, of design constraints, are respected in the simulation stage.

Through the adoption of equations, it is possible to prove feasibility, by RM policy, of one task set. Furthermore, these contribute toward proving the previous statements. Equation 15 describes the utilization factor U of τ. This value shows the fraction/quantum of processor time necessary to execute the task set. In this study, U^m refers to

the utilization factor of the simulated task set. Besides, U^a corresponds to the same utilization factor, but it describes the values of the architectural models assumptions.

$$U = \sum_{i=1}^{n} (C_i/T_i) \tag{15}$$

The Rate Monotonic guarantees the feasibility of an arbitrary set of periodic tasks if the total processor utilization U does not exceed the Utilization least Upper-Bound [185]. Equation 16 shows the Least Upper Bound (U_{ulb}) of τ. Here, U_{ulb} describes the least upper bound to the Central Process Unit (CPU) utilization, that is, maximum utilization factor over all task sets.

$$U_{lub} = n * (2^{1/n} - 1) \tag{16}$$

It is also important to highlight that RM guarantees that an arbitrary set of periodic tasks is schedulable if the total processor utilization U does not exceed the value of 0.69. This value is adopted to compare and prove that U^a and U^m fulfil the (**question** 2) regarding the empirical evaluation.

Function 17 shows that the utilization factor should be less or equal the U_{lub}. Attendance to this inequality ($U \leq U_{ulb}$) is sufficient, but not necessary to guarantee the feasibility of a given task set.

$$\sum_{i=1}^{n} (C_i/T_i) \leq n * (2^{1/n} - 1) \tag{17}$$

In RTES development several periodic tasks can run concurrently, due to preemption permission, and these tasks can have individual non-functional requirements. The preemption permission allows tasks to execute concurrently by the scheduler, in one single core processor. Tasks of the case study are periodic constrained in order to ensure that the turn indicator behaves as hard real-time systems and provides deterministic guarantees for the task set. Figures 49, 50, 51, 52 and 53 depict individually and graphically the simulation results of the timing constrained tasks.

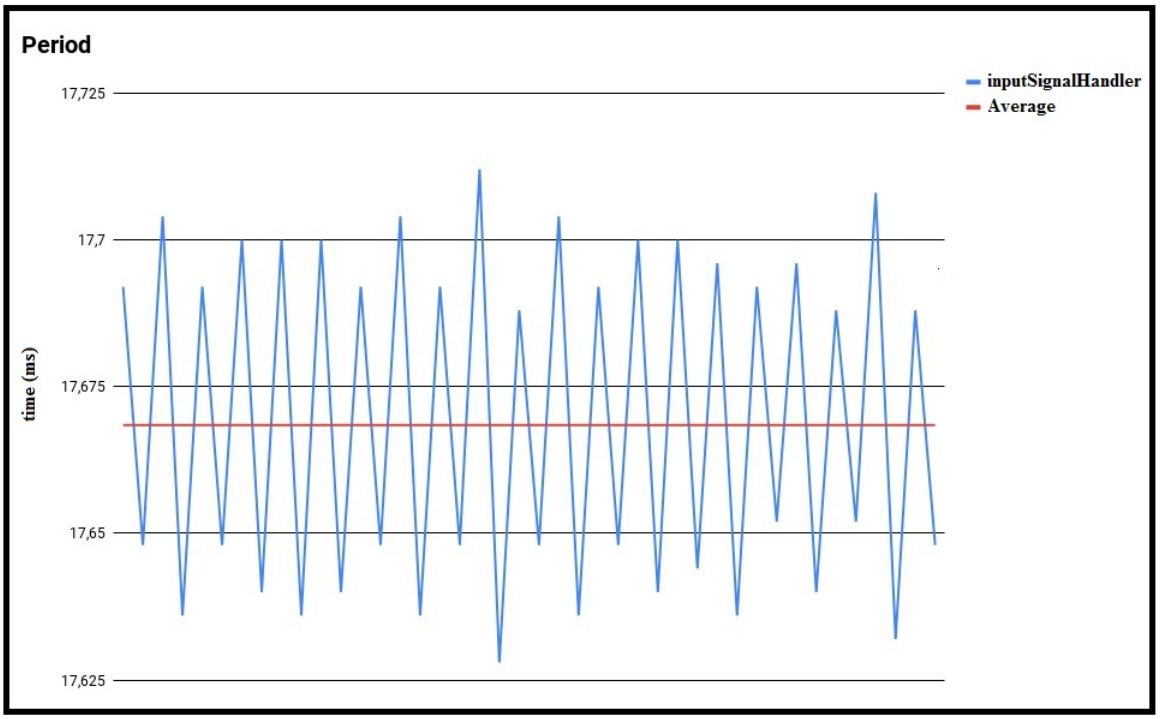

Figure 49 – Period of Input Signal Handler Task.

Figure 49 shows the simulation results of the *inputSignalHandler*. Table 10 shows that initial period of τ_1 is equal 20 ms. However, from the simulation results the period of τ_1 is on average equal to $17,66$ ms. For each value of the simulated execution time, measured in microseconds, $1,1ms$ was added in order to adjust the simulation results to the same/equivalent scale. This strategy simplifies the results evaluation and does not bias the overall evaluation of the tasks assumptions. For τ_1, the processor utilization $U_1^m = 1,14/17,6 = 0,064$.

Similar to *inputSignalHandler*, the *outSignalHandler* has the highest priority from among the other tasks. Figure 50 shows that the task τ_2 respects the constrained period (see Figure 44) for all of the 1500 simulated and measured samples. For τ_2, the measured processor utilization $U_2^m = 1,19/17,6 = 0,067$.

The tasks *taskTurnFlashing* and *taskHazardFlashing* have the same constrained period of 40 ms. Figures 51 and 52 present the simulated results of these tasks and these show that the real period occurs within of $35,33$ ms. In this case, τ_3 and τ_4 were executed respecting their predefined deadlines. Moreover, as depicted in Figure 44 and Table 10, the τ_3 and τ_4 tasks have a scheduler constraint that shows a higher priority for τ_3. Following the FreeRTOS scheduler, it is undestood that these tasks are executed by the Round-Robin scheduler as these have the same period. In the proposed implementation, the priority of τ_3 under τ_4 is implemented in a logical manner. In this case, a hazard action can always be replaced by a turn indicator occurrence. The measurements of processor utilization of τ_3 and τ_4 are, respectively, $U_3^m = 1,1/35,3 = 0,031$ and $U_4^m =$

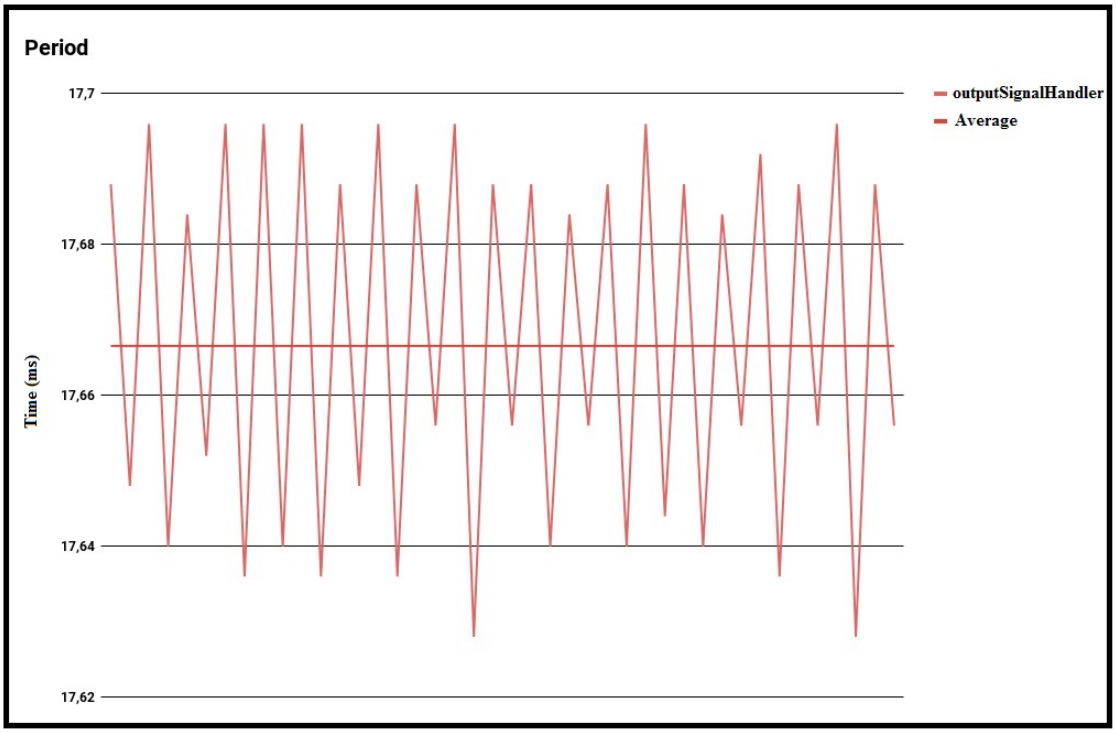

Figure 50 – Period of Output Signal Handler Task.

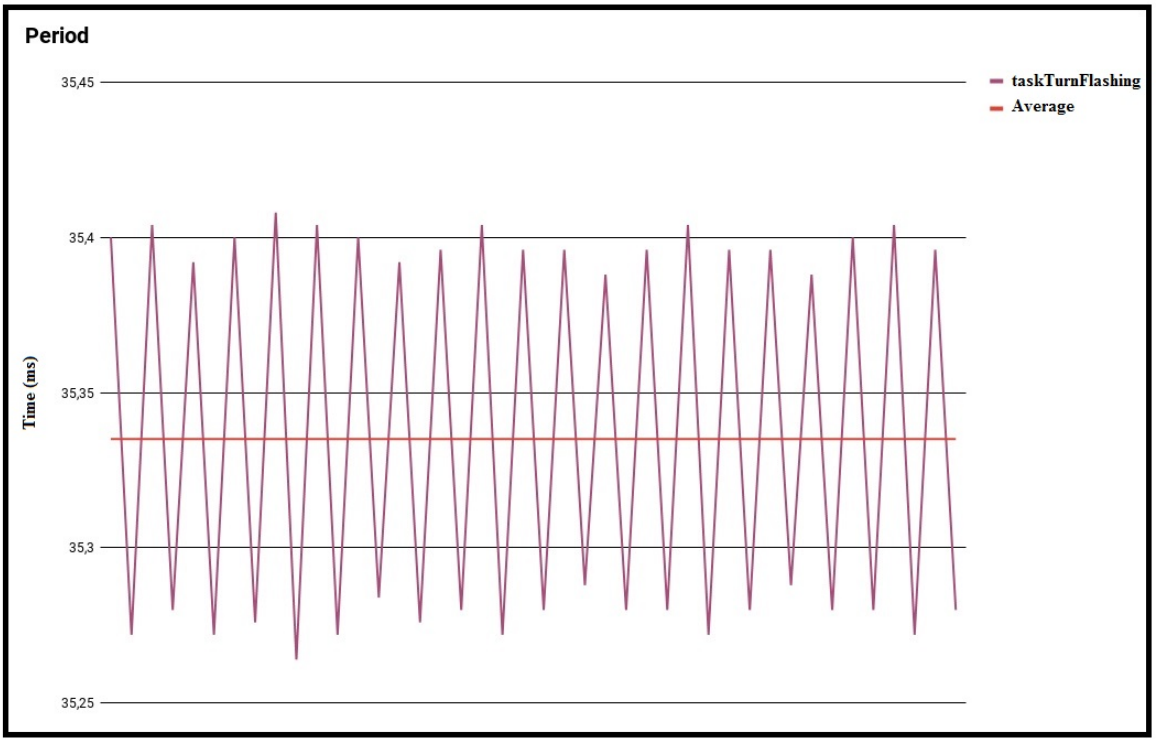

Figure 51 – Period of Turn Flashing Task.

$1,1/35,3 = 0,031$.

Figure 53 represents on average the period of 70 ms of task τ_5. These results fit exactly with the initial constraint values of *taskBreaking*. Moreover, $U_5^m = 1,10/70 = 0,01$ and it

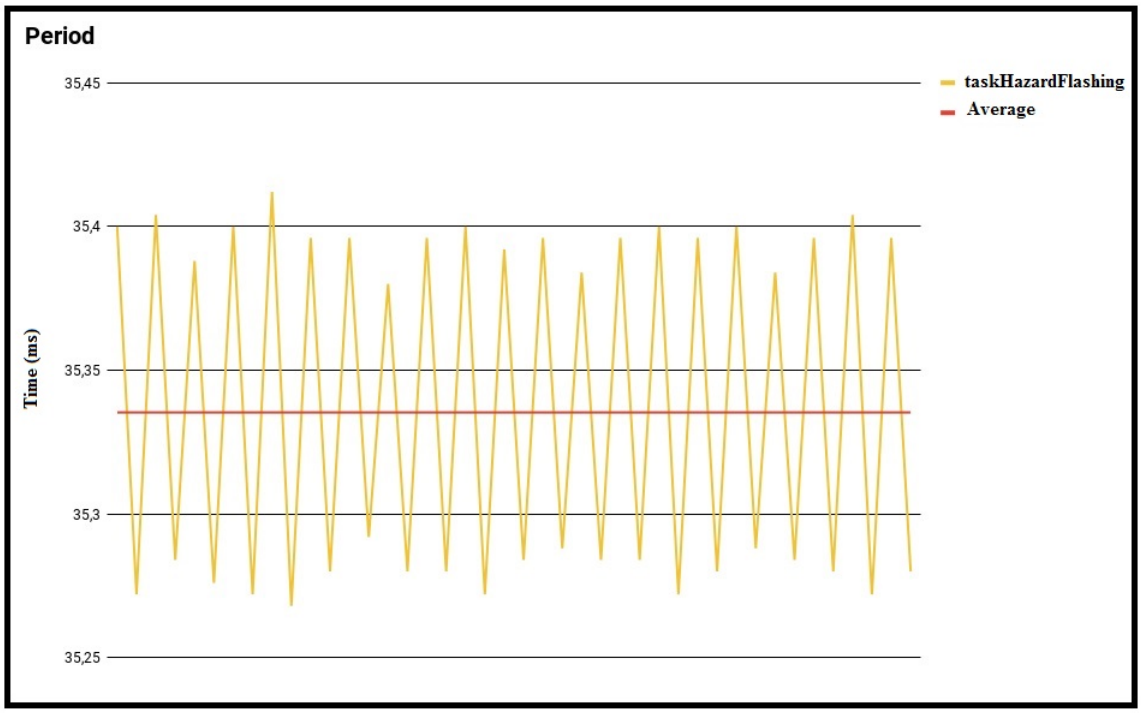

Figure 52 – Period of Hazard Flashing Task.

also confirms the schedulability results of Figure 47 regarding to Rate Monotonic policy.

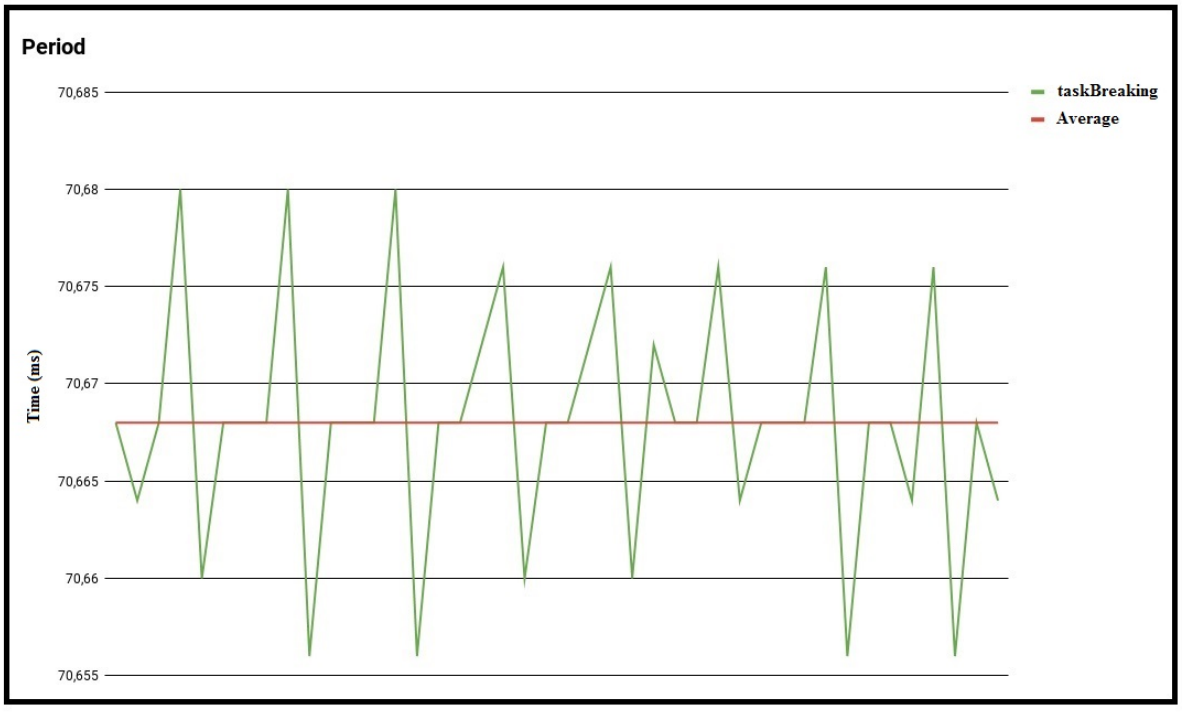

Figure 53 – Period of Breaking Task.

Measured values of U_1^m, U_2^m, U_3^m, U_4^m and U_5^m are in accordance/respect the sufficient $U \leq 0.69$, and it provides the feasibility guarantee of the task set. Based on the measurements of the simulated tasks, $U^m \leq U_{lub}$. It shows that all the real deadlines,

from the simulation results, are met. This fact confirms the previous results of the Rate Monotonic scheduling in accordance with the tasks period, depicted in Figure 47, and the execution time simulations.

$$U^m = \begin{cases} U_1^m + U_2^m + U_3^m + U_4^m + U_5^m \leq 5 * (2^{1/5} - 1) \\ 0,064 + 0,067 + 0,031 + 0,031 + 0,01 \leq 5 * (1,14 - 1) \\ 0,203 \leq 5 * 0,14 \\ 0,2 \leq 0,7 \end{cases}$$

The same evaluation can be performed considering the models assumptions, defined by MARTE constraints, regarding the processor utilization factor (U^a). Based on the model constraints, previously described on Table 10, thus the following calculation can be made:

$$U^a = \begin{cases} U_1^a + U_2^a + U_3^a + U_4^a + U_5^a \leq 5 * (2^{1/5} - 1) \\ 2/20 + 4/20 + 5/40 + 5/40 + 3/70 \leq 5 * (2^{1/5} - 1) \\ 0,1 + 0,2 + 0,1 + 0,1 + 0,04 \leq 5 * (1,14 - 1) \\ 0,54 \leq 5 * 0,14 \\ 0,5 \leq 0,7 \end{cases}$$

Therefore, one notes $U_{lub} \geq U^a \geq U^m$. Hence, the U^m of the measured time (C_i^m)(by simulation) are better than estimated ones (C_i^a). This proves that both timing constraints and their respective execution measurements have optimal processor utilization among all fixed priority assignment. Besides, it confirms that computation times of task sets respect the guarantee level of processor utilization.

From the analysis of the results on Table 10, one notes that tasks meet the deadlines specified in the architectural models. It can be said that the first and second questions, which were proposed in this section answered adequately here. All the system tasks are able to hold their deadline (**question 1**) and periodic task set can fulfil the requirements of the scheduling policy of Rate Monotonic scheduler (**question 2**). However, in terms of the execution time of each task the use of processor, can be optimized since on average each task left the processor idle for few periods of time.

Noteworthy here is that, even with discrepancy simulation values, this methodology adds value to RTES design. These quantitative measurements allow for the verification of traced constraints and their plausibility regarding the design assumptions. In addition, it also presents a manner by which one can check if architectural constraints in the design model can be reached. The proposed simulation confirms that the RTES constraints described in previous design viewpoints, and traced at different abstraction levels, can be reached and simulated in the final development phase.

7.2 Early Evaluation of MARTE Constraints of Architectural Viewpoint

This section applies model checking solutions and the UPPAAL tool to automatically verify designed artefacts. Figures 31 and 32 provide the inputs to the proposed analysis.

Timed Computational Tree Logic (TCTL) formula allows for the evaluation of timing constraints, which were mapped from requirements specification to Timed Automata model. Moreover, one can also validate non-functional constraints, such as safety, reachability, deadlock and insistence of critical regions. Figure 54 describes the overall TCTL equations for validating and verifying the Turn Indicator model and its constraints.

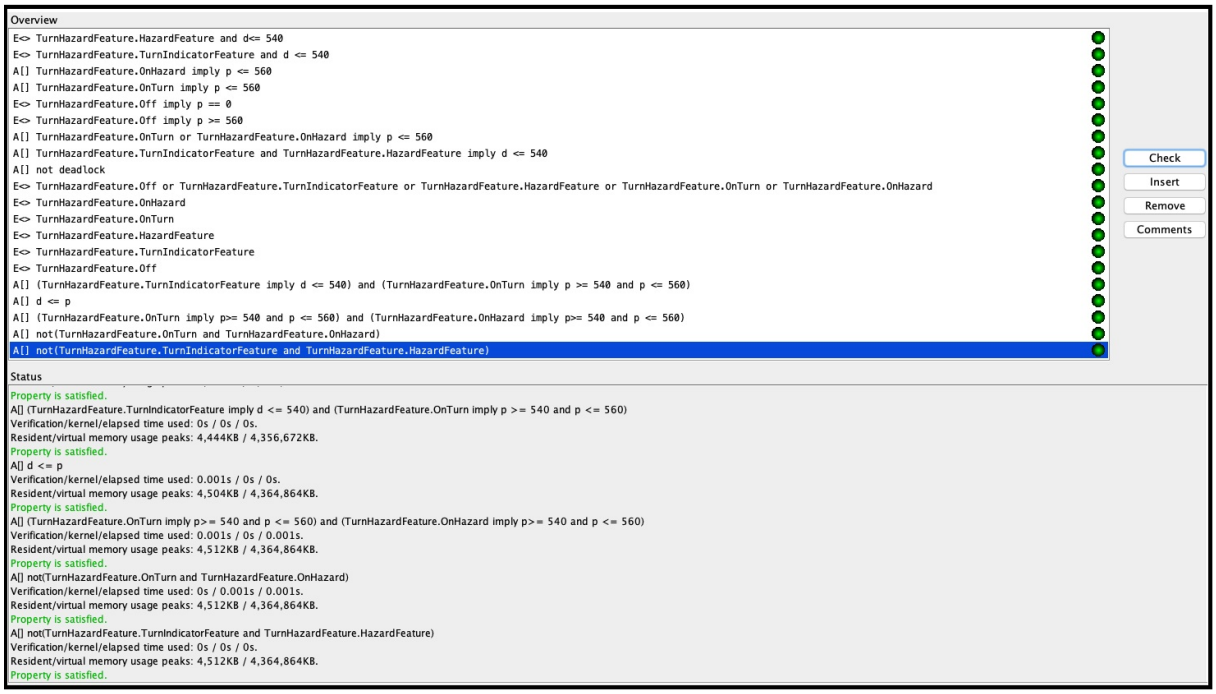

Figure 54 – The Proposed TCTL Specifications and the Validation Results.

Figure 54 includes, in its upper section, all the proposed TCTL formulae. These formulae use the branching time, through its quantifiers $\forall$ and $\exists$, and clock variables/constraints to verify the MARTeSys^{ReqD} designed models. In Figure 54, the lower part describes the results of the TCTL verification.

The studies in this thesis propose five distinctive evaluations in order to perform the initial verification of RTES properties. The verification is performed by model checking the model against the TCTL formulae. Here, path formulae and state formulae are checked in UPPAAL through the TCTL query language. According to [136], "state formulae describe individual states, whereas path formulae quantify over paths or traces of the model". Therefore, a group of TCTL equations is proposed to verify (**1**°) if periodic constraints are respected, (**2**°) if deadline constraints are respected, (**3**°) reachability

property, (**4**°) safety properties, and (**5**°) the absence of simultaneous participation in critical regions.

1. **Periodic constraints**: This verification aims at showing that the system runs periodically. In addition, it must prove that all periodic states always respect the proposed period. The last TCTL formula, for example, is able to prove the states *OnTurn* and *OnHazard* are active in the period p, where $540 \leq p \leq 560$. Moreover, after $p \geq 560$ the state *Off* is reached and the system runs and all states are once again ready for activation in the $0 \leq p \leq 560$ interval.

   - ❏ A[ ] TurnHazardFeature.OnHazard imply $p \leq 560$
   - ❏ A[ ] TurnHazardFeature.OnTurn imply $p \leq 560$
   - ❏ A[ ] TurnHazardFeature.OnTurn or TurnHazardFeature.OnHazard imply $p \leq 560$
   - ❏ E<> TurnHazardFeature.Off imply $p \geq 560$
   - ❏ E<> TurnHazardFeature.Off imply $p == 0$
   - ❏ A[ ] (TurnHazardFeature.OnTurn imply $p \geq 540$ and $p \leq 560$ p) and (TurnHazardFeature.OnHazard imply $p \geq 540$ and $p \leq 560$)

2. **Deadline constraints**: This verification aims at proving that the designed model always hold the constrained deadline. The TCLT formula $A[]d \leq p$ checks if all the states constrained by the deadline respect the global period. As noted in Figure 54, this property is satisfied.

   - ❏ E<> TurnHazardFeature.HazardFeature and $d \leq 540$
   - ❏ E<> TurnHazardFeature.TurnIndicatorFeature and $d \leq 540$
   - ❏ A[ ] TurnHazardFeature.TurnIndicatorFeature and TurnHazardFeature.HazardFeature imply $d \leq 540$
   - ❏ A[ ] $d \leq p$

3. **Reachability property**: This verification aims at showing whether a given state is always reachable from the initial automata state. This property is valid if there exists a path, starting at the initial state, to each automata state.

   - ❏ E<> TurnHazardFeature.OnHazard
   - ❏ E<> TurnHazardFeature.OnTurn
   - ❏ E<> TurnHazardFeature.HazardFeature
   - ❏ E<> TurnHazardFeature.TurnIndicatorFeature
   - ❏ E<> TurnHazardFeature.Off

4. **Safety properties**: This verification proves that an undesirable system state, or blockade condition, will never happen. In this study, due to the importance of the turn indicator controllers, it is important to guarantee that the system does not run into a deadlock.

   - ❑ A[ ] not deadlock

5. **Absence of simultaneous participation in critical regions**: This verification proves that excludent states cannot be activated at the same time. This analysis verifies that the priority state *OnTurn* never happens simultaneously to *OnHazard* state.

   - ❑ A[ ] not(TurnHazardFeature.OnTurn and TurnHazardFeature.OnHazard)
   - ❑ A[ ] not(TurnHazardFeature.TurnIndicatorFeature and TurnHazardFeature.-HazardFeature)

The proposed verification of the timing constraints and the validation of RTES properties closes the contributions of the MARTeSys^{ReqD} methodology to the early analysis of RTES models. This thesis provides guidelines for the formal verification and validation of artefacts in the Requirements viewpoints. It considers a formal grammar to transform natural language specification to the Timed Automata formalism. From the designed models, one can analyze the mentioned RTES constraints in order to increase the reliability and correctness of the proposed methodology. The TCTL equations are useful to check and prove the correctness of the models and the respective MARTeSys^{ReqD} formal grammar.

7.3 Qualitative Evaluation of the MARTeSys^{ReqD} Methodology

The MARTeSys^{ReqD} methodology was adopted in four case studies as in ADC, TIS, RTMS and IPS. The Qualitative Evaluation performed here considers the MARTeSys^{ReqD} application in the TIS.The main objective of this qualitative evaluation, performed by experts in automotive and RTES development, is to analyze the adequacy level of the MARTeSys^{ReqD} methodology for specification, design and validation of non-functional concerns in these systems.

In order to evaluate the decisions in the present research study, as well as provide guidelines to improve the views, two interviews were performed with different experts. These professionals are from different knowledge backgrounds and all are involved in automotive systems development. The first interview was performed from September/2017 to

October/2017 with five interviewees and the second from August/2018 to January/2019 with eleven interviewees.

The first qualitative evaluation was performed in the initial stage of the methodology development. In this first evaluation, the main objectives were to check if the guidelines for design Requirements, Functional, Logical and Technical viewpoints in RTES domain were appropriate and if the MARTE annotations are helpful in this domain. Even the qualitative approach to evaluate the proposed methodology was refined, in line with the research study level of advancement and maturity.

7.3.1 Data Gathering

The first and second qualitative evaluation are based on interviews, questionnaires and measurement and qualitative analyses, from the perspective of the author, as well as from the feedbacks from interviewees. Figure 55 depicts the main steps of the data gathering process in the proposed evaluation.

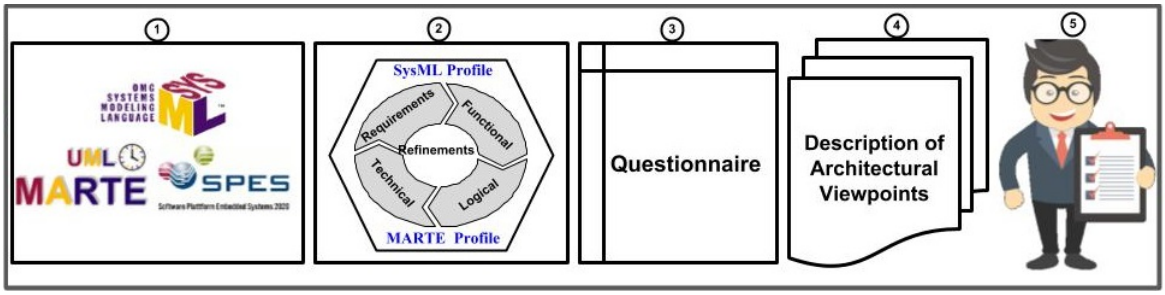

Figure 55 – Main Steps to Perform the Qualitative Evaluation.

As described in Figure 55, the qualitative evaluation of the MARTeSys^{ReqD} methodology starts with an overall description (explanation) of the SysML and MARTE profiles. Here, it was necessary to provide an overall explanation of the SysML profile and its diagrams especially those adopted in this thesis. Furthermore, a brief definition of the *CoreElements*, *NFP*, *Time*, *GRM*, *HLAM*, *GCM*, *HRM* and *SRM* packages is conducted, which aids in comprehending of their adopted stereotypes.

Following this, in the 2^{nd} step, the MARTeSys^{ReqD} methodology is presented. It depicts the concept of viewpoints, views, architectural design and refinements and shows how these define and contribute to the studies in this thesis. Moreover, a textual document with the expected inputs and outputs of the Requirements, Functional, Logical and Technical viewpoint is delivered to each reviewer.

In the 3^{rd} step, the questionnaire and its questions are presented to the interviewees. Due to the fact that the qualitative questions are written in natural language, it is important to describe the meaning of the question in order to avoid misunderstandings. The proposed questionnaire is based on the IEEE 29148 [81] standard for the RE process,

on the ISO/IEC/IEEE 29148:2011 [81], ISO/IEC/IEEE 42010:2011 [37], ISO/IEC/IEEE 29148:2011 [81] and on the research papers [186], [76].

These IEEE standards contributed in defining the questions of the qualitative evaluation. These provide, respectively, insights on the properties that should be supported by the modeling languages and the main foundations, which need support through the architecture description languages.

Two questionnaires are proposed and these are composed, respectively by 13 and 14 qualitative questions. These questions are mainly related with correctness, completeness, consistency between design artefacts and the required system, ambiguity, conformity between models, legibility, description of abstraction levels, description of multiple views, expressiveness, relevance to the field of study, relevance of the architectural design to RTES constraints, applicability or value of the methodology. An explanation of the evaluated concepts of the qualitative questions is as follows:

- **Correctness**: MDSE approaches must provide architectural design artefacts, which are able to describe the system services. Correctness of one design/development artefact shows/presumes the right representation of system description in accordance with the expected functionalities. Moreover, the correctness can also be related to the correctness in the adoption of methods and tools. In the latter case, correctness considers the right application of the semantic and syntax of each diagram/methodology element. Therefore, this concept has a great impact on the project development, influencing the reliability and integrity of the developed system.

- **Completeness**: An architectural description is complete whether or not the entire system behavior has been reached in design activities [187]. Here, this concept relates to the analysis of the artefacts from the viewpoints in order to confirm that the system works as intended.

- **Consistency between design artefacts and the expected system**: The conceptual models of an architecture description ,which are modelled in different viewpoints, must be consistent with each other. Moreover, models on a high abstraction level must be consonant with those in a low abstraction level. In this case, all the artefacts from the system refinements are defining the same system requirements through different viewpoints.

- **Ambiguity**: Architectural models should not express the same system (subsystem), in one specific view, under double or ambiguous, understandable contexts. It means that one specific artefact which is described, at different abstraction levels and viewpoints, must not have a contradictory meaning in these viewpoints.

- **Conformity between models**: This concept describes how the models conform with requirements of clients. In this case, artefacts from Requirements, Functional, Logical and Technical viewpoints provides the expected inputs and outputs of the system.

- **Legibility**: Architectural models must be understandable and intelligible to humans. It is assumed that model elements have a clear and comprehensible semantic and syntax.

- **Description of abstraction levels**: This concept evaluates if the system is designed under distinct perspectives. These perspectives/levels must be complementary and consistent with each other. It means that system services can be modelled or defined under distinctive view through different refinements and levels.

- **Description of multiple viewpoints**: A viewpoint is specification of the conventions for constructing and using a view. It formalizes a pattern or template to develop individual views by establishing the purposes, details and techniques to create a view. The viewpoint determines the languages to be used for describing the view, and any associated modeling methods or analysis techniques to be applied to these representations of the view [37]. Evaluation of this concept is noted through the proposed methodology, by the adoption specifics and coherent viewpoints for designing RTES.

- **Expressiveness**: An architectural description must communicate the overall and complete idea of the system. Expressiveness shows a measure of effectively conveying meaning by a specific representation. An expressive methodology can ensure that the design models are meaningful enough to represent the desired domain.

- **Relevance to the field of study**: This concept analyzes how much the viewpoint descriptions, based on refinements of SysML and MARTE, are relevant and can contribute to RTES development.

- **Relevance of the architectural design to RTES constraints**: The MARTE constraints can contribute to the modelling and designing of non-functional concerns of RTES. Evaluation of these constraints and their descriptions aims to check the contributions of the proposed methodology to architectural viewpoints design.

- **Applicability or value of the methodology**: This evaluated concept aims at analyzing the impact and importance of the proposed methodology, its guidelines and its expressiveness, while suggesting design decisions for practical RTES development.

The mentioned questionnaire and its questions are based on the previous concepts. For each question, an answer must be provided considering the follow interval: **fully agree**, **agree**, **partly agree**, **neutral**, **partly disagree**, **disagree** and **totally disagree**.

In the 5 step, the results of the MARTeSys^{ReqD} methodology adoption are presented. The artefacts of architectural design are explained in a high abstraction level and, then, delivered to the interviewees.

Finally, the interviewees are given a period of two hours to analyze the architectural models and to fill in the questionnaire while providing their impressions and criticism of the proposed methodology. The steps concerning qualitative evaluation were performed individually and it takes one hour and thirty minutes to execute the $1-5$ steps and two hours to the 6 step. In the end, the questionnaire complemented with the perception of an expert is analyzed. The complete questionnaire and its analysis are presented in Sections 7.3.2 and 7.3.3.

7.3.2 First Qualitative Evaluation

The initial qualitative evaluation was performed in September and October of 2017 and it received contributions from five experts in RTES development. Table 12 shows the attendees of this first qualitative evaluation.

As noted from 12, the group of attendees is composed of experts from both academy and industry. The first interview was performed while the MARTeSys^{ReqD} methodology was being refined and architectural models were incomplete. However, the intention of this initial analysis was to obtain different insights and contributions for the definition of a representative and seamless architectural description of RTES.

The questionnaire takes into account the perception of experts relating to the application of the MARTeSys^{ReqD} methodology and its fulfillment of significant concepts to RTES development. Table 15, Annex F, represents a summary of the proposed questionnaire. The first column describes questions regarding RTES concepts, while the other columns maintain the interviewees identification and their evaluation.

Expert Identification	Company	**Experience**	**Knowhow Description**
Expert 1 (E1)	Oldenburg University	20 years	Embedded system, design methods, RTOS and HW/SW co-design.
Expert 2 (E2)	Hella Company	8 years	Embedded engineering and automotive development.
Expert 3 (E3)	Hella Company	6 years	Real-time and embedded scheduling with AUTOSAR standards.
Expert 4 (E4)	University Paderborn/C-LAB	5 years	Embedded system researcher.
Expert 5 (E5)	Hella Company	7 years	Requirement engineering processes.

Table 12 – Attendees of the First Qualitative Evaluation.

The correctness criteria in the 2^{nd} question, Table 15 of Annex F, received one neutral and one answer of partial disagreement. This fact relates to the level of difficulty to comprehend the stereotypes of the MARTE profile. Through interviewees $E2$ and $E4$ it is not possible to evaluate the proposed semantic of one model element without an understanding of the concepts that were adopted to define it. The author decided to improve the definition of the MARTE stereotypes and elucidate, using a group of guidelines, on how one specific real-time concern can be annotated by MARTE constructors/stereotypes.

The completeness criterion (3^{rd} question) was not well evaluated. This may be due to the fact that the proposed methodology and its application were still under development when this initial evaluation was performed. In this stage, only the models and partial refinements from Requirements, Functional and Logical viewpoint were available.

The results of 5^{th} and 6^{th} questions present some problems regarding consistency and conformity in adoption of this methodology. In the opinion of the author, this relates directly to the fact that models or diagrams that represent the same system service were described by a distinct name in different views/refinements. A standard nomenclature and description of the models in different viewpoints can contribute to consistency and conformity of the architectural models.

The qualitative evaluation carried out is important to evaluate the design decisions, already taken in this research, since it provides different insights on their adequacy level and relevance. In addition, results allows for improvements and adequacy of the research to be made, since the first qualitative evaluation performs an initial evaluation of the MARTeSys^{ReqD} methodology. In general, the results presented are positive and describe the decisions previously made for the Requirements, Functional and Logical views, which for the greater part of cases meet the recommendations of standards [81] and [37].

As checked through the 11^{th} and 12^{th} questions, adoption of the MARTE profile in the model elements was evaluated as highly relevant to the domain of study. Here it is important to highlight that the adoption of MARTE constraints, in the different design levels, was well evaluated with the assumption that these will add value to RTES design. The answers to the 12^{th} and 13^{th} questions shows the contribution of the proposed methodology in terms of temporal aspects definition and to the early design of non-functional properties.

7.3.3 Second Qualitative Evaluation

The second qualitative evaluation aims to evaluate the MARTeSys^{ReqD} methodology and its adoption along the RTES design. It had considered design artefacts from four architectural viewpoints. This evaluation follows the main steps described in Figure 55, and this is performed by the same interviewees of the first evaluation plus six new ones from the RTES area.

Table 13 describes the attendees of this evaluation and tabulated personal information about interviewees. As this depicts Annex F, this information is declared in the questionnaire by each interview within the topic "Personal Questions about the interviewees". These data can contribute, for example, to the analysis and understanding of their evaluations of one criterion. Thus, the research interests, experience and the knowledge level regarding methodologies and the profiles adopted in the MARTeSys^{ReqD} methodology were all checked. Table 13 uses the acronyms **L1**, **L2** and **L3** to indicate, respectively, the knowledge level in the SysML profile, MARTE profile and SPES methodology. Furthermore, L1, L2 and L3 can be classified as **High (1)**, **Medium (2)** and **Low (3)**.

Expert Identification	**Company**	**Expe-rience**	**Job Position**	**Knowhow Description**	**L1**	**L2**	**L3**
Expert 1 (E1)	Oldenburg University	20 years	Professor	Embedded system, design methods, RTOS and HW/SW co-design.	1	1	1
Expert 2 (E2)	Hella Company	8 years	System Engineer	Embedded engineering and automotive development.	2	1	1
Expert 3 (E3)	Hella Company	6 years	Functional Safety Engineer	Real-time and embedded scheduling with AUTOSAR standards.	1	2	2
Expert 4 (E4)	University Paderborn/-C-LAB	5 years	Researcher	Embedded system researcher.	2	1	2
Expert 5 (E5)	Hella Company	7 years	Requirements Engineer	Requirements engineering processes.	1	2	2
Expert 6 (E6)	HSHL[3]	3 years	Researcher	Embedded Systems and Internet of Things (IoT).	1	1	1
Expert 7 (E7)	HSHL	15 years	Professor	Model-based Engineering of Embedded Systems.	1	1	1
Expert 8 (E8)	INGenX Technologies	25 year	Start-up Founder and CEO	Start-up founder and CEO/, System Engineering Coach, System Safety Expert (TUV Sud).	2	1	3
Expert 9 (E9)	UFU[4]	9 years	Researcher	Software and systems engineer .	1	1	2
Expert 10(E10)	FIB[5]	5 years	Professor	System Specification and Formal Analysis.	1	2	3
Expert 11(E11)	FIB[5]	5 years	Product Manager	System Specification and Formal Analysis.	1	2	3

Table 13 – Attendees of the Second Qualitative Evaluation.

Second round of the qualitative analysis considers an improvement of the questionnaire and its questions were explained or described with greater clarity. Moreover, it receives two new qualitative questions. Annex F, Table 16, presents the refined questionnaire.

[3] University of Applied Science Hamm-Lippstadt.
[4] Federal University of Uberlândia.
[5] Federal Institute of Brasilia

Thus, the 14^{th} question aims at obtaining the opinion of the interviewees about the feasible and realistic contribution of the proposed methodology in real environments/RTES projects. The last question seeks to evaluate the effectiveness of the proposed strategy to trace design artefacts at distinct abstraction levels. Even though the first analysis had mostly favorable opinions, regarding MARTeSys^{ReqD} methodology, from the experts, it is understood that this number of interviewees is not very high. Therefore, the decision has been made to apply the proposed interview over a wider range of people.

In this evaluation, the interviewees also analyze the architectural models and complete refinements of each viewpoint. A new interview and questionnaire application were performed in order to get a global and definitive overview of the adequacy and usefulness of the proposed study. Annex F presents a results overview of the second qualitative evaluation regarding criticism from the interviewees.

The analyses of the questionnaire answers indicates a significant improvement in the level of adequacy of MARTeSys^{ReqD} to RTES design. Thus, the conclusion is reached that **Relevance to the Field of Study**, **Relevance of MARTE constructors** and **Practical Applicability** is either better evaluated (when it considers the same interviewees of the first and second evaluation) or "Fully Agree"/"Agree" as ranked by the interviewees.

Noteworthy from the comparison between Table 15 (Annex F) and Table 16 (Annex F), that interviewees that attended both evaluations either improved their evaluation or maintained that attained in the first qualitative evaluation. It means, when considering the final results of Table 16, there were no decreasing values for the qualitative question assessments.

Highlighted also is an improvement in the analysis of some criteria, which were not well evaluated in the first qualitative evaluation. The questions 5^{th} and 6^{th}, on Table 15, for example, presented some problems regarding ambiguity and conformity in adoption of this methodology. However, in the second qualitative evaluation both criteria have a higher ranking regarding their contribution.

The correctness criteria expressed by the 2^{nd} question, Table 15 (Annex F), initially received one neutral and one partial disagreement answer. However, correctness is evaluated once again as "fully agree" or "agree", on Table 16 results, for the majority of interviewees.

Considering exclusively the 1^{st} and 3^{rd} interviewees, which performed the first and second qualitative evaluation, emphasis can be placed on the noteworthy improvement in their evaluation regarding: completeness, consistency, ambiguity, expressiveness and relevance of the methodology. In both case, the raking increased by 2 or more levels of acceptability.

Other important facts, in the second qualitative evaluation, are the proposition of the personal questions to the interviewees and the 14^{th} and 15^{th} qualitative questions. The

evaluation of the knowledge level regarding the SysML profile, MARTE profile and SPES methodology by the interviewees allow for a better evaluation, by the thesis author, of the answers and criticism given by the interviewees.

The 14^{th} qualitative question provides some insights regarding the real applicability of the proposed study. As illustrated on Table 16, the majority part of the interviewees evaluate that MARTeSys^{ReqD} adds value to architectural design and development of RTES.

Considering the 15^{th} qualitative question, ones that the proposed strategy for tracing non-functional requirements/constraints along the architectural viewpoints needs improvements. In this thesis, traceability is provided by identification labels placed on each annotated constraint. It allows for a full trace of these constraints at different abstraction levels. However, the automatic trace and support by an automatic tool can improve track/link of these constraints.

7.4 Contributions

The contributions of this chapter are published in [102], [103] and [104] and these manly relate to the quantitative and qualitative evaluation of the MARTeSys^{ReqD} methodology adoption. As initial contributions, emphasis is placed on the tracing of constraints along of the development phase and their dynamic evaluation. The empirical verification, presented in this chapter, contributes to proving that **(1)** there is a consistency between the architectural constrained models and the final model implementation regarding periodic tasks, and **(2)** to checking the need for refinements (in system or hardware level) of the RTES constraints. Finally, it **(3)** also indicates that constrained parameters, which are related to real-time schedule/precedence constraints can be satisfied and evaluated.

A qualitative evaluation is also performed here by industrial and academic experts of RTES development. This evaluation allows for for quantifying, by means of expert analyses, the main contributions of the MARTeSys^{ReqD} methodology for RTES development. Considering the final qualitative evaluation, one notes a positive evaluation by the RTES experts regarding the completeness, consistency, ambiguity, expressiveness and marked relevance and applicability of the MARTeSys^{ReqD} methodology in RTES development. Moreover, this evaluation describes the criticism of these experts regarding its usefulness, adequacy and relevance in this domain. In the first qualitative evaluation, criticism regarding the correctness, completeness and conformity provided important feedback as to the improvement of the proposed methodology. The second qualitative evaluation highlights the need for improvements in the traceability criteria for RTES constraints. A broader discussion about this topic is presented, as future research topic, in Chapter 8.

Chapter 8

Conclusion and Future Work

This chapter discusses the contributions achieved through the proposed research, addresses future research topics and lists the research activities that lead to this thesis.

8.1 Main Results

The contributions and main results of this thesis are summarized below and these consider the challenges in the developing of RTES, the proposed assumptions and research questions of Chapter 1.

Proposition of a methodology to perform requirements specification and architectural design of Real-Time Embedded Systems. The main contribution of this thesis relates to the development and analysis of a methodology that covers different phases of RTES design. Initially, constructors, stereotypes and enumerations of the MARTE profile were traced and linked with specific and non-functional concerns from the RTES domain. From this collected data and the combined use of the SysML profile, Timed Automata and SPES guidelines the MARTeSys^{ReqD} methodology is accomplished. MARTeSys^{ReqD} methodology is based on viewpoints, refinements and granularity levels. Based on author knowledge and the state of the art analyses, the understanding is reached that MARTeSys^{ReqD} addresses expressive bases in RTES development, while considering specificities of their specification and architectural design.

Definition of a standard and formal strategy to describe design decisions. The MARTeSys^{ReqD} methodology adopts the concepts of architectural viewpoints and refinements. The models can be enriched by MARTE constraints strengthening their expressiveness. However, more than found through only the MDSE approach, this research study presents a group of formal guidelines to apply the proposed study into the distinctive RTES domains. These guidelines allow (**1**°) to collaborate to the understanding of the design steps, (**2**°) to estimate the time complexity function to system design and (**3**°) to perform early estimation of system complexity. Formalization of the MARTeSys^{ReqD}

methodology considers formal input sets for the architectural formalization. Here different algorithms are applied to define the architectural viewpoints and their refinements. This formalization helps RTES engineers to quantitatively measure the design processes. Besides, it contributes toward predicting the design efforts and the scope of the project based on the number of design artefacts.

Rational mapping of constructors to annotate concerns, at different abstraction levels, in a specific RTES methodology. This research study contributes to map RTES concerns, which are important in RTES development. Moreover, the proposed mapping allows one to link each outlined concern to an atomic MARTE stereotype. The MARTE profile contains hundreds of different types of stereotypes. However, the MARTE specification does not describe how to concretely correlate its stereotypes and annotations with the global system requirements. MARTE constructors are specified in generic fashion, which makes the applicability of the MARTE profile difficult in RTES specification and design. The author of this thesis claims that the relationship between atomic RTES concerns and the MARTE profile constructors contributes to architectural design and viewpoints of these systems. In general, the presented leveling allows to strengthen the specification, modelling, design, validation and maintenance of RTES, since it addresses different concerns and properties of RTES in the initial development stages.

Description of formal and standard strategies for handling timing constraints since early and along architectural design. This study contributes with two strategies to analyze, verify and validate timing constraints of RTES. The analysis of these constraints is performed in (**1**°) early design steps using transformation rules and Timed Automata formalism. Indeed, this study provides a formal grammar thatspecifies formal rules to be applied in order to transform requirements specification, in natural language, to Timed Automata. UPPAL is used here to validate and verify models of Requirements viewpoint considering period, deadline and event occurrences of a RTES specification. In addition, (**2**°) timing constraints are also verified in the final design steps. In this case, a strategy to automatically trace the MARTE constraints from Model and Unit Design models to the code is defined. Thus, it adopts the Papyrus tool for code generation, definition and representation of RTES constraints at a low granularity level. The tracing of these constraints, in the code, allows to connect design artefact information to system developers. Moreover, as explained in Chapter 7, it allows one to simulate timing concerns and their respective assumptions, from the architectural models, in a dynamic perspective.

It provides traceability patterns between artefacts of the Requirements Specification and Architectural design. This research defines a pattern for writing all model constraints adopting the VSL formalism and labeled annotations. Thus, each constraint annotation identifies the correlated viewpoint and its type regarding the MARTE

specification. The same standard is applied in different refinements and allows a model checker to automatically verify if periodic and deadline constraints are consistent with the architectural design assumptions.

8.2 Research Challenges and Limitations

The proposed thesis studies still present risk and limitations regarding the following topics:

The time analysis performed here is based on estimations. The author applies empirical methods to measure the accuracy of these estimations. Thus, the accuracy of the approach is also dependent on the knowledge or experience of the engineer. Thus, the value of these analyses is as good as the engineering predictions. However, the MARTeSys^{ReqD} methodology allows to perform analyses and estimations from initial requirements specification considering timing constraints. This study is able to contribute to industrial projects, since it provides quick answers and verification of the system properties.

This study does not consider human interactions in the complexity analysis of architectural design. The design complexity analysis of the overall process is performed from the formalized architectural viewpoints. This strategy allows one to perform the analysis impact and timing estimation while designing the models and before developing further components of the system. Therefore, it does not propose a strategy to measure human activities that relate to the RTES design. Measurements concerning creative solutions, engineer expertise, and time to learn and apply one specific solution or method were not considered by the proposed study.

The number of refinements for each viewpoint are not defined in the proposed methodology. The final low granularity levels of the are not defined, that means, the "stop refinement level" is not specified in the MARTeSys^{ReqD} methodology. The main reason behind this, it that this design decision is wholly dependent on the engineering experience and the project type. Nevertheless, the proposed methodology and its detailed specification supports functional and non-functional requirements specification and provides guidelines to RTES designers.

The MARTeSys^{ReqD} methodology applies VSL to specify the model annotations. However, this formalism is not checked in relation to semantical correctness of the developed application. **The qualitative analysis performed in this study considers a small scope of experts in the RTES development**. The contributions of the methodology needs to pass through the evaluation and criticism of a more representative number of stakeholders.

The design steps are not fully supported by tools or scripts. However, the concepts and the MARTeSys^{ReqD} methodology guidelines are fully covered in the thesis

scope and can be applied/followed in different RTES domain, techniques, design strategies and tools.

8.3 Future Research

This section highlights future research. Thus, refinements and new directions that may be considered regarding the scope of this thesis are considered as follows:

- MARTeSys^{ReqD} methodology provides MDSE background to specify and design RTES. The design decisions were formalized in order to adopt the methodology in a consistent and structured manner. However, a metamodel has not defined that expresses the global syntax of the MARTeSys^{ReqD} methodology. Thus, metamodel development may clarify the methodology description and depiction. In addition, it allows to detail the methodology infrastructure and its correlate languages and processes.

- Different techniques, languages and tools can be adopted to deal with the complexity of RTES. This aims at supporting the design of both hardware, software and mechanical elements, as well as provide more expressive and extensive descriptions for the fundamental requirements of RTES. The languages and diagrams adopted in MARTeSys^{ReqD} methodology do not cover all the RTES design. For example, the values of the physical environment and mechanical parts are not designed or simulated in this thesis. An extension of MARTeSys^{ReqD} methodology could contribute to the entire system design and their refinements along the development cycle.

- The proposed study depicts software and hardware behavior of RTES. The connection between the behavioral hardware descriptions to high-level synthesis is not described in the thesis. The realization of this scenario approximates the design artefacts to the final hardware components.

- In RTES development, the intrinsic properties to this type of application, such as temporal requirements, performance, synchronization and parallelism must be analyzed, understood, described, elicited and designed. This study considers the design and validation of timing constraints along architectural viewpoints. Thus, it is necessary to combine other strategies to address and validate other RTES constraints.

- An extension of the MARTeSys^{ReqD} methodology to generate test cases to all viewpoint artefacts is another open research topic. It allows for combining and integrating into one single methodology of automatic test case generation. Thus, this combination can further prove the correctness of the design by simulation.

- MARTeSys^{ReqD} methodology depicts all conceptual correlations between viewpoints. Nevertheless, there is a gap regarding the automatic linkage of the final models of one viewpoint to another viewpoint. Fully automatic or partial transformations, depending on manual design decisions made by engineers, between viewpoints can be a further extension of this thesis.

- The architectural viewpoint artefacts are not fully transformed to the system implementation view. Another extension of the thesis is to define mapping rules to transform these artefacts to models of existent tools. It means that these tools would generate the complete system code. Nevertheless, these tools probably require further refinement of the imported models.

- Many directions can be followed regarding the application of formal methods. An important contribution to the early design analysis of Requirement models is automatic TCTL generation. Considering the MARTeSys^{ReqD} methodology, the TCTL can be automatically generated for the formal models, while describing period, deadlines and events of RTES.

- Different types of formalism can be integrated in the MARTeSys^{ReqD} methodology. As a direct result of this further extension, other timing constraints such as jitter and offset can be verified by model checking approaches, as that performed for deadline, period or events.

- Apply the proposed methodology in distinctive domains.

All these above described extensions show the impact of the thesis, since it has already closed some gaps as described in Chapter 3 and opens further research topics.

8.4 Bibliographic Production

This thesis provided the following contributions in form of scientific publications:

1. "*Model-Based Requirements Specification of Real-Time Systems with UML, SysML and MARTE*" published in the International Journal on Software and Systems Modelling [92].

2. "*Annotating SysML Models with MARTE Time Stereotypes for Requirements Specification and Design of Real-Time Systems*" [94] published in the 7^{th} *IEEE Workshop on Self-Organizing Real-Time Systems.*

3. "*An Analysis of the Value Specification Language Applied to the Requirements Engineering Process of Cyber-Physical Systems*" [98] presented in the 4*th IFAC Symposium on Telematics Applications* and published on Journal IFAC-PapersOnLine.

4. "*A Model-Based Engineering Methodology for Requirements and Formal Design of Embedded and Real-Time Systems*" published in the 50*th Hawaii International Conference on System Sciences (HICSS)* [95].

5. "*Applying MARTE Profile for Optimal Automotive System Specifications and Design*" [93], published in the 50*th Hawaii International Conference on System Sciences (HICSS).*

6. "*A Technique to Architect Real-time Embedded Systems with SysML and UML through Multiple Views*", [96], published in the 19*th International Conference on Enterprise Information Systems (ICEIS).*

7. "*Guidelines for using MARTE profile packages considering concerns of real-time embedded systems*", [100] , published in the *International Conference on Industrial Informatics (INDIN).*

8. "*Multi-formalism in Different Levels of Abstraction for Requirements Engineering and Design of Real-Time Systems*", [97], published in the *PhD Forum at Design, Automation and Test in Europe (DATE).*

9. "*SPES Methodology and MARTE Constraints in Architectural Design*", [101], published in the *IEEE International Symposium on Computers and Communications (ISCC).*

10. *An Approach to Formalization of Architectural Viewpoints Design in Real-Time and Embedded Domain*, [99], published in the 21*th IEEE Computer Society Symposium on Object/Service-Oriented Real-Time Distributed Computing (ISORC).*

11. *Model-Based Design Methodology for Early Evaluation of Real-time and Embedded Constraints*, [102], published in the *International Conference on Industrial Informatics (INDIN).*

12. *An Approach for Architectural Design of Automotive Systems using MARTE and SysML*, [103], published in 14*th International Conference on Automation Science and Engineering.*

13. *Non-Functional Constraints Annotation to Real-Time and Embedded System Design*, [104], published in the *VIII Brazilian Symposium on Computing Systems Engineering.*

14. *An Approach for Architectural Design of Automotive Systems using MARTE and SysML* [188], published in the *14th International Conference on Automation Science and Engineering.*

15. *Ein Modellierungsansatz für eine Systemarchitekturbeschreibung von Automotive-Systemen mit MARTE und SysML*, [189], published in *Automatisierungstechnik Journal*.

16. *A Methodology to Early Design and Evaluation of Real-Time Embedded Systems considering Non-Functional Constraints*, [190], published in the *PhD Forum at PhD Forum at 38th Design Automation Conference (DAC)*.

17. *A Proposal to Trace and Maintain Real-time Embedded Constraints*, [191], published in the *6th International Embedded Systems Symposium*.

ANNEX **A**

Description of the Value Specification Language

A.1 Introduction

Value Specification Language (VSL) is an extension of concepts of "*Value Specification*" and "*DataType*" from UML. VSL was created to complement and customize the UML metamodel with new marked values to value properties and stereotype attributes. VSL standard provides definitions for the **abstract syntax** (MOF compatible metamodel) and for **concrete syntax** (textual grammar) while adds new constructors to specify and write MARTE expressions.

VSL is used for specifing values of constraints values, properties and attributes of stereotypes. Generally, VSL specification is directly related to non-functional RTES requirements and properties. Moreover, VSL can be used to express marked values and to define constraints for any UML element which is associated to a value specification.

In VLS, Value Specifications express textual values of model elements. Expression 18 shows the possible value types which can be specified by VSL formalism.

$$\begin{aligned} <value-specification>:: \quad &=<literal> \mid <enum-specification> \mid <interval> \mid \\ &=<collection> \mid <tuple> \mid <choice> \mid <expression> \mid \\ &=<time-expression> \mid <obs-call-expression> \end{aligned} \tag{18}$$

As depicted in Expression 18, a value specification can be simple literal, as a number, or can be a complex expression which involves variables and operations. A value specification could be a literal value (*LiteralSpecification*), a composite value (*IntervalSpecification, CollectionSpecification, TupleSpecification*), an expression (*Expression*) or

a time value/expression (*TimeValueSpecification, TimeExpression*). A short description of each value specification is performed below:

- **Literal**: A literal is a fixed value in the code. It describes how to describe atomic values as, for example, integer, real number, strings, boolean, data time and enumeration types. VSL presents strict production rule for each of these atomic values.

- **Enumeration Specification**: An enumeration specification describes a UML enumeration literal. VSL provides a formal notation to name an literal enumeration.

- **Intervals**: The interval values are able to describe ordered sets of value specifications. An interval is specified between two specific values: the minimum and the maximum value.

- **Collections**: Collections presents a way to combine value specifications into items collection. Collection is a set of individual value specifications separated by commas.

- **Tuples**: Tuple denotes structured values of different types. It allows to describe values that are the same of tuple data type. The elements of a tuple are named tuple items and consist of a pair of item name and their associated values separated by an equal symbol.

- **Choice values**: Choice value specifications denotes the value of a choice data type. It contains the name of one of the attribute members (chosen alternatively), which determines the chosen data type and a value that conforms to the chosen data type.

- **Expression**: An expression is defined in VSL by a simple constant or variable (call and declaration), or it can be a compound expression formed by the combination of expressions through operator calls. VSL presents a strict production rule for each of these atomic values. Additional details can be found in [2].

- **Expression**: An expression is defined in VSL by a simple constant or variable, or it can be a compound expression formed by the combination of expressions through operator calls. VSL presents a strict production rule for each of these atomic values. Additional details can be found in [2].

- **Time Expression**: The VSL Time Expression model of VSL allows to formalize different temporal and non-functional expressions on models and are important for providing different and rigorous standards for representation of expressions. Section A.2 provides an overview of the Time Expression model due to its importance to the MARTeSys^{ReqD} methodology.

Understanding each value specification is important to define functional constraints modeled elements. It allows to formally formulate more complex expressions which usually groups one or more expressions of a value specification in architectural models. Moreover, the adoption of Time Expression, in MARTeSys^{ReqD} methodology, allows a specialized syntax for writing expressions and specifications of time values in the model elements.

A.2 Time Expression

A Time Expression is described in Equation 19. In general, Time Expression enables to formalize intervals (minimum and maximum) and event duration, as well as the distance considered between consecutive events (see Expressions 26 and 27). Moreover, Time Expression model allows to express events occurrence (see Expression 24), detail specific event durations (observe Expression 20), describe conditional events occurrence and, also, specify possible variations in events (Expression 25).

$$< time - expression > ::= \quad < duration - expr > | < instant - expr > | \\ < jitter - expr > \tag{19}$$

A **DurationExpression** is a temporal expression to evaluate an event as a duration value. Its syntax is defined by expression 20:

$$< duration - expr > ::= \quad (< real - literal > | variable - call - express >) | \\ < duration - obs - expr > | (' (' < instant - obs - expr > \\ ' - ' < ' instant - obs - expr > ')') \tag{20}$$

A Duration Expression can also be decomposed into the following expressions 21, 22 and 23:

$$< variable - call - express > ::= \quad < variable - name > | \tag{21}$$

$$< duration - obs - expr > ::= \quad < duration - obs - name > \; ['['< occur - index - expr >']'] ['when' < condition - expr >']' \tag{22}$$

$$< instant - obs - expr > ::= \quad < instant - obs - name > ['['< occur - index - expr >']'] ['when' < condition - expr >] \tag{23}$$

Expressions 21, 22 and 23 need the definition of the following atomic values for an expression:

$$\begin{aligned} < variable - name > ::= \quad & [< namespace >' .'] < body - text > \\ < duration - obs - name > ::= \quad & [< namespace >' .'] < body - text > \\ < instant - obs - name > ::= \quad & [< namespace >' .'] < body - text > \\ < occur - index - expr > ::= \quad & < value - specification > \end{aligned}$$

An **InstantExpression** allows to describe temporal expressions to annotate instant time values. Expression 24 depicts the syntax of the InstantExpression:

$$\begin{aligned} < instant - expr > ::= \quad & (< datatime - literal > | < variable - call - expr >) \\ ::= \quad & ['+' < duration - obs - expr >)| \\ ::= \quad & (< instant - obs - expr > ['+' < duration - expr >]) \end{aligned} \tag{24}$$

One datetime-literal provides the following specifications:

$$\begin{aligned} < datetime - literal > ::= \quad & (< data - string > [daystring]) \\ ::= \quad & |(< time - string > [data - string >] \\ ::= \quad & [day - string])|(< daystring >]) \end{aligned}$$

A **JitterExpression** describes a variation (jitter) of events occurrences. These occurrences must be detailed to enable the control of possible variations or delays system events/actions. Expression 25 describes different possibilities to describe a jitter.

$$\begin{aligned} < jitter - expr > ::= \quad & ('jitter('< instant - obs - expr >')') | \\ ::= \quad & ('jitter('< instant - obs - expr > \\ ::= \quad & '-' < instant - obs - expr >')') \end{aligned} \tag{25}$$

Expressions to define temporal instants or event durations are respectively denoted in Equations 26 and 27. Both expressions describe a special type of interval specification and it marks temporal expressions with lower and upper limits:

$$\begin{aligned} < instant - interval > ::= \quad & ('['|']') < instant - expr >' .' \\ ::= \quad & < instant - expr > ('['|']') \end{aligned} \tag{26}$$

$$\begin{aligned} < duration - interval > ::= \quad & ('['|']') < duration - expr >' .' \\ ::= \quad & < duration - expr > ('['|']') \end{aligned} \tag{27}$$

Often, when specifying real-time systems, one needs to represent time cardinality such as delays, events duration, clock time, chronometric time and logical time in model elements. Understanding the Time Expression model is primordial, in this study, once it allows timing concerns of RTES following a standard formalism, proposed by the MARTE profile, with completeness and less ambiguity.

ANNEX **B**

Algorithms Formalization to MARTeSys^{ReqD} Methodology

This Annex complements the contributions of Chapter 5. The following sections details the proposed algorithms to formalize the MARTeSys^{ReqD} design decision to the Requirements, Functional and Logical viewpoints.

B.1 Requirements Viewpoint

Algorithm B.1 High-Level Description - Algorithm to group requirements in cohesive categories

```
1   {
2   1         numberOfRequirements = |R|;
3   1         numberOfCategories = |C|;
4   n         for (j= 0; j < numberOfRequirements;  j++){
5   n           aux = 0;
6   n*m       for (i = 1; i < numberOfCategories; i++){
7   n*m         if (ReqTable[j][0] has high cohesion with c[i]){
8   A≤n*m        c[i] = c[i] + ReqTable[j][0] + "R" + j;
9   A≤n*m        aux = 1;
10  –              }
11  –            }
12  n            if (aux == 0){
13  B≤n*m      for (i = 0; i < numberOfCategories; i++){
14  B≤n*m        if (c[i] == empty){
15  C≤B          c[i] = ReqTable[j][0] + "R" + j;
16  C≤B          break;
17  –                 }
18  –              }
19  –            }
20  –         }
21  }
```

Algorithm B.1 adds one requirement r_i in a category c_i if there is similarity between them. For each requirement, from *ReqTable*, an evaluation must be done in order to

cluster this requirement in a category c_i. Cohesive criterium means that requirements with common responsibilities must compose the same group. Therefore, requirements in a single category describe and are correlated to the same system service. This design decision is formalized, in Algorithm B.1, through evaluation if "$aux = 1$" then r_i must be categorized in an already created category or if "$aux = 0$" it must generate a new category. Moreover, it is created a sequential label or ID reference to describe each requirement of one category. This identification allows a simple manner to name each requirement.

Algorithm B.2 is a second refinement of High-Level Description of Requirements. This view employs the categorization performed in Algorithm B.1 to represent system requirements in a graphical manner. It adopts the Use Case diagram to show the group of related functions which is able to setup atomic use cases. Algorithm B.2 creates the actors and relations between the use cases.

Algorithm B.2 High-Level Description - Algorithm to design the Use Case diagram from system categories

```
{
1 numberOfCategories = |C|;
1   create the actors;
m for (i = 0; i < numberOfCategories; i++) {
m   create sc[i];
    }
m for (i = 0; i < numberOfCategories; i++) {
m   create the associations to sc[i];
m   if (sc[i] has a relationship with an actor )
A<m  link sc[i] to the actor;
-   }
}
```

The last refinement to the High-Level Description of Requirements is presented in Algorithm B.3 and it adopts the SysML Requirements diagram.

Algorithm B.3 High-Level Description - Algorithm to model requirements and their relationships

```
{
1 numberOfRequirements = |R|;
n for (j= 0; j < numberOfRequirements;  j++){
n   ReqDiagram[j] = a graphical representation of ReqTable[j][0];
-   }
n  for (j= 0; j < numberOfRequirements;  j++){
n*n    for (i= 0; i < numberOfRequirements;  i++){
n*n       if (ReqDiagram[j] has a relationship ReqDiagram[i] and i <>j)
A<(n*n)     create the association from ReqDiagram[j] to ReqDiagram[i];
    }
  }
}
```

Algorithm B.3 describes the design of the SysML Requirements diagram and the definition of requirements relationships. *ReqDiagram* element shows one specific requirement (from ReqTable). The modeled requirements can be related with each other by derive, satisfy, copy, verify, refine and/or trace relationships. Models from Algorithm B.3 are used, as an input, to the Composition of Models using the MARTE Profile (Algorithm B.4) and to Formal Specification with VSL (Algorithm B.4).

Algorithm B.4 MARTE Composition and VSL Specification - Algorithm to apply in Requirement Models MARTE stereotypes and to formalize their description

```
1  {
2  1 numberOfRequirements = |R|;
3  n for (j= 0; j < numberOfRequirements;  j++){
4  n   if  (ReqDiagram[j]  needs to be detailed through a non–functional information){
5  A≤n  ReqDiagram[j] is labeled by a MARTE stereotype;
6       }
7  n     if (ReqDiagram[j] has ≪ NfpConstraint ≫ label)
8  A≤n     Formalize the ≪ NfpConstraint ≫ of ReqDiagram in accordance with VSL syntax;
9     }
10 }
```

Algorithm B.4 describes the last refinements of Requirements viewpoints. Each $ReqDiagram_j$ should be checked in order to discover and detail related non-functional and real-time properties. In the cases where it is necessary to highlight these properties, in this level of abstraction, a MARTE stereotype can be adopted. The $\ll NfpConstraint \gg$ stereotype allows to add textual information to model elements and it provides a manner to annotate different RTES constraints. The last part of Algorithm B.4 shows a strategy to formalize a $\ll NfpConstraint \gg$ declaration by VSL standard.

The Analysis of Requirements activity is formalized, in MARTeSys^{ReqD} methodology, by proposed Grammar to the NL-TA Transformation.

Proposed Grammar to the NL-TA Transformation

The following rules details the grammar for the NL-TA Transformation. This grammar specifies formal rules which must be applied to transform requirements specification, in natural language, to Timed Automata.

$$< \textbf{root} > \; ::< construct > \; | < construct >< root > \; .$$

$$< \textbf{construct} > \; ::< location > \; :: \; | < location >< edge > \; :: \; | < location >< edge >< construct > \; .$$

$$
\begin{aligned}
< \textbf{location} > \; & ::< label > \mid < clock\_constraint > \\
& :: \mid < label >
\end{aligned}
.
$$

$$
\begin{aligned}
< \textbf{edge} > \; & ::< guard > \\
& :: \mid < guard >< action > \\
& :: \mid < guard >< reset > \\
& :: \mid < guard >< action >< reset > \; . \\
& :: \mid < action > \\
& :: \mid < action >< reset > \\
& :: \mid < reset >
\end{aligned}
$$

$$< \textbf{label} > \; ::< name > \; .$$

$$< \textbf{clock\_constraint} > \; ::< clock >< relation >< constant > \; .$$

$$
\begin{aligned}
< \textbf{guard} > \; & ::< clock\_constraint > \\
& :: \mid < clock\_constraint >< logic\_relation >< guard >
\end{aligned}
.
$$

$$
\begin{aligned}
< \textbf{action} > \; & ::< name > \text{“!”} \\
& \mid < name > \text{“?”}
\end{aligned}
.
$$

$$
\begin{aligned}
< \textbf{reset} > \; & < clock > \text{“ :=}''< constant > \\
& \mid < clock > \text{“ := ”} < constant >< reset >
\end{aligned}
.
$$

$$< \textbf{logic\_relation} > \; :: \text{“}AND\text{”}|\text{“}OR\text{”}|\text{“”} \; .$$

$$< \textbf{name} > \; ::< STRING > \; .$$

$$< \textbf{clock} > \; ::< name > \; .$$

$$
\begin{aligned}
< \textbf{relation} > \; & :: \text{“ } > \text{ ”} \\
& |\text{“ } \geq \text{ ”} \\
& |\text{“ } = \text{ ”} \\
& |\text{“ } < \text{ ”} \\
& |\text{“ } \leq \text{ ”}
\end{aligned}
\quad .
$$

$$< \textbf{constant} > \; ::< INTERGER > \; .$$

B.2 Functional Viewpoint

Algorithm B.5 describes the second refinement to the Functional viewpoint. It describes the refinement of the Block Definition diagram model through the Internal Block diagram. In this view, the number of internal blocks is the same of functional blocks, however this refinement adds external interfaces or communications channels to the blocks. In this algorithm, the variable *numberSignals* defines the number of system signals. Therefore, these signals represent the global directed edges of the model. In this viewpoint, internal ports and flow ports are adopted in order to represent the communication and types of relations between blocks. The $mIBD$ matrix stores the internal blocks which have any association with an input or output edge.

Algorithm B.5 Second Refinement - Algorithm to model the Internal Block Diagram with internal and flow ports

```
1  {
2  1 numberSignals= |S|;
3  1 signals[numberSignals] = vetor with the overall signals;
4  1 numberInternalBloc = |FuncBlock|;
5  1 mIBD = matrix[numberInternalBloc][2];
6  n_s for (i=0; i <= numberSignals; i++){
7  n_s * n_ib for (j= 0; j <= numberInternalBloc; j++){
8  n_s * n_ib if (signals[i] is output of funcBlock[j]){
9  A≤n_s * n_ib mIBD[i][0] = funcBlock[j];
10 A≤n_s * n_ib break;
11 –      }
12 –    }
13 n_s * n_ib for (j= 0; j <= numberInternalBloc; j++){
14 n_s * n_ib   if (signals[i] is input of funcBlock[j]){
15 B≤n_s * n_ib     mIBD[i][1] = funcBlock[j];
16 B≤n_s * n_ib   break;
17 –       }
18 –    }
19 n_s   if ((mIBD[i][0] <> empty) and (mIBD[i][1] <> empty)) {
20 C≤n_s  create a internal port to intExpBlock[i][0];
21 C≤n_s    create a internal port to  mIBD[i][1];
22 C≤n_s    create an association from  mIBD[i][0] to mIBD[i][1];
23 –      }
24 n_s         if ( mIBD[i][0] <> empty) {
25 D≤n_s   create a flowport to mIBD[i][0];
26 D≤n_s  create an association from  mIBD[i][0] to flow port;
27 –             }
28 n_s  if (mIBD[i][1] <> empty){
29 E≤n_s    create a flowport to  mIBD[i][1];
30 E≤n_s    create an association from  mIBD[i][1] to flow port;
31         }
32   }
33 }
```

It is worth to mention that during the modeling activities each functional block from $mIBD_{i,j}$ is modeled by syntax of Internal Block diagram. Similar to Algorithm 5.3, in Algorithm B.5, lines $6-17$ create a matrix of internal blocks ($mIBD$) from the functional

blocks. However, this matrix has the specificities of internal blocks components, where for each line the first column can store an internal output block to $signal_i \in S$. Moreover, the second column can store an input internal block to $signal_i \in S$.

In Algorithm B.5, lines $18-30$ show how to create relationships to $mIBD$ elements. Initially, internal ports and their respective associations can be defined for each block (lines $18-22$). Lines $23-26$ show that flow ports and their associations can be created from internal blocks of the $mIBD_{i,0}$ elements. In this case, it creates a relationship to the elements of first column. Finally, lines $27-30$ present the design of flow ports and their associations from the internal blocks of the $mIBD_{i,1}$ elements.

The third and last abstraction level of Functional viewpoint is detailed in Algorithm B.6. This refinement aims to qualify the RTES models with well formed descriptions of non-functional properties. A group of guidelines to specify RTES concerns is detailed in Annex B and it presents how to use constraints of the MARTE profile. Therefore, *CoreElements*, *NFP*, *Time*, *GRM*, *HLAM*, *GCM*, *HRM* and *SRM* packages can have their stereotypes adopted to refine the models.

Algorithm B.6 Third Refinement - Algorithm to annotate MARTE constraints in the functional blocks

```
1  {
2  1 numberFuncBloc = |FuncBloc|;
3  n_b for (i= 1; i <= numberFuncBloc; i++;){
4  n_b   if  (funcBlock[i]  needs to be detailed through a non-functional information){
5  A≤n_b funcBlock[i] is labeled by a MARTE stereotype;
6            }
7  n_b     if (funcBlock[i] has ≪ NfpConstraint ≫ label)
8  A≤n_b    Formalize the ≪ NfpConstraint ≫ of funcBlock[i] in accordance with VSL syntax;
9            }
10 }
```

As it can be observed in Algorithm B.6, a stereotype or $\ll Nfpconstraint \gg$ constraint, from the MARTE profile, can be annotated in a functional block while refining their properties. This refinement allows to highlight timing properties in early stages of the architectural design.

B.3 Mapping between Functional Viewpoint to Logical Viewpoint Models

Mapping between Functional Viewpoint and Logical Viewpoint is performed in order to show how final models of Functional viewpoint can be refined in the initial models of Logical viewpoint. This section shows how to link the design decisions of Functional viewpoint to Logical viewpoint.

Algorithm B.7 shows the mapping of functional models to logical models in a $N : M$ relation. Here, $M \leq N$, where N describes the functional blocks and M the logical blocks. This algorithm formalizes the main design decisions and shows that functional blocks, with common control features, can be mapped by one logical block.

Algorithm B.7 Mapping viewpoints - Algorithm to map Functional viewpoint to Logical viewpoint

```
1  {
2  1 counter = 0;
3  n_b for (i= 0; i<=|FuncBlock|; i++){
4  n_b   hasSimilarity = false;
5  n_b*(n_b-1) for (j= i+1; j<= numberFuncBloc; j++) {
6  n_b*(n_b-1) if (FuncBlock[i] has similar control functions with FuncBlock[j] AND LogBlock
       does NOT contains FuncBlock){
7  A≤n_b*(n_b-1)  LogBlock[counter] = LogBlock[counter] + FuncBlock[j];
8  A≤n_b*(n_b-1)  hasSimilarity = true;
9  –        }
10 –    }
11 n_b   if (hasSimilarity == false){
12 B≤n_b*k for (k = 0; k <counter; k++){
13 B≤n_b*k  if (FuncBlock[i] has similar control functions with LogBlock[k]  AND LogBlock does
       NOT contains FuncBlock){
14 C≤B       LogBlock[k] = LogBlock[k] + FuncBlock[j];
15 C≤B       hasSimilarity = true;
16 –        }
17 –      }
18 – }
19 n_b if (hasSimilarity == false) AND LogBlock does NOT contains FuncBlock{
20 D≤n_b  LogBlock[counter] = FuncBlock[i];
21 D≤n_b    hasSimilarity = true;
22 – }
23 n_b   if (hasSimilarity){
24 E≤n_b counter++;
25 –    }
26 – }
27 n_l for (i= 1; i <=|LogBlocks|; i++){
28 n_l   check constraints compatibility;
29 n_l   update LogBlock[i] with MARTE constraints of FuncBlock[i];
30 – }
31 }
```

Algorithm B.7 aims to provide the group of logical blocks of RTES. These blocks are the basis to the development of Logical viewpoint and here they are represented by the *LogBlock* vector/element. The functional blocks ($FuncBlock$) are compared (lines $3-10$) to check if one block i has any similarity with the others $i+1, i+2, ...i < |FuncBlock|$ blocks. This first design steps shows that if a functional block i can be grouped with one or more functional blocks they are clustered in *logBlock[counter]*. After this evaluation, if block i was not allocated as one logical block, two scenarios are still possible: **(1)** block i is similar with one of the already composed logical block (lines $11-18$) or **(2)** block i has no similarity with the other blocks and it provides a singular logical block (lines

$29 - 22$). The variable counter defines the final number of logical blocks to the Logical viewpoint. Algorithm B.7 describes an initial reading and grouping of functional blocks to logical blocks. However, it is possible, for example, to compare the logical blocks again and refine the similarity criteria by using the lines $3 - 10$ of Algorithm B.7.

In Algorithm B.7, a functional block should be replicated without modification, in the Logical viewpoint, if this block has not similar control features or if it is an input/output functional block. In addition, all the MARTE constraints from Functional viewpoint are updated, for each logical block, in order to maintain the consistency between each viewpoint.

B.4 Logical Viewpoint

In this section, the set of logical design steps are separately represented for input logical blocks (Algorithm B.8), output logical blocks (Algorithm B.9) and feature blocks (Algorithm B.10). This subdivision can contribute to compression of the design decisions.

Logical viewpoint is represented by different refinements of the Activity diagram. In order to formalize the group of activities and their input/output associations, it is necessary to adopt different data structures as, for example, list data types to describe the models. Therefore, the algorithms to formalize the Logical viewpoint are described in higher abstraction level than the previous one from the Requirement and Functional viewpoints. This decision aims to facilitate the comprehension of the design decisions regarding the Activity diagram and MARTE profile annotations. The proposed description is sufficient enough to specify the MARTeSys^{ReqD} methodology allowing its manual adoption to model the Logical viewpoint.

In Algorithm 5.4, each Activity diagram is placed in one list of activities. Lines $8 - 17$ describe the control/creation of three possible types of Activity diagrams: ***actI*** (Activity diagram to one input logical block), ***actO*** (Activity diagram to one logical output block) and ***actGen*** (Activity diagrams to features blocks). It is important to highlight that every position *actGen* list can contain one specific Activity diagram, from the features blocks, which can be composed by different sub-activities.

Algorithm B.8 describes the group of steps to be performed in order to create the Logical viewpoint to input blocks. These logical blocks comes from the mapping between Functional and Logical viewpoints.

Algorithm B.8 Logical viewpoint:Input - Algorithm to describe the design of the Activity diagram to the Input Logical blocks

```
1  {
2  ne      foreach (signal in externalSignal){
3  ne         if (signal associates with one ACTIVITY) {
4  A≤ne        if (activity of this signal is NOT created){
```

```
5   B≤A     create activity for signal;
6   B≤A     store each new activity in activityList;
7   –           }
8   A≤ne     create an association from signal to the activity;
9   –        }else{
10  ne−A    create the new element for signal;
11  ne−A    link signal to the new element;
12  ne−A    create activity(ies) to the new element;
13  ne−A    create an association from this new element to the activity(ies);
14  ne−A    store the activity(ies) in activityList;
15  –       }
16  –     }
17  na    foreach (activity in activityList){
18  na      if (activity is end node){
19  D≤na     if (internalSignal of the activity is NOT created) {
20  E≤D        create internalSignal for activity;
21  E≤D        store each new internal signal in internalSigList;
22  –         }
23  na         create an association from activity to the internalSignal;
24  –        }else{
25  na−D      link activity to the new element;
26  na−D      create activity(ies) to the new element;
27  na−D      create an association from this new element to the activity(ies);
28  na−D      store the activity(ies) in activityList;
29  –        }
30  –     }
31  }
```

Structure "foreach" is applied in the algorithms of Logical viewpoint and it presents, at higher abstraction level, the incremental checking of all elements of a group/list. Algorithm B.8 shows that external signals are considered as inputs or star point for this diagram. It means that every external signal must have one specific activity to detail its behavioral features. These signals are identified by MARTE stereotypes in Algorithm B.6. The list of external signals (*externalSigList*) is already initialized. On the other hand, the list of internal signals (*internalSigList*) is generated along the design of the Activity diagram.

In Algorithm B.8, lines $1-7$ formalize how external signals can create/generate activities to the Input Activity diagram. In algorithms B.8, B.9 and B.10, a "new element" can be modelled and replaced by an activity, a decision node, a join node, a fork node or a final node. Modelling this " new element" depends of design decisions of engineers. However, the proposed algorithms provides strategies to formalize the design, from a new element, to the other subsequent activities and their associations. Consequently, lines $8-15$ show how to create internal signals from the final/end activities/nodes of *activityList*. Lines $15-20$ describe the design of new activities and its associations.

Algorithm B.9 represents a group of instructions to model the feature blocks and its internal behavior. In this algorithm, the setting of rational actions is defined in order to design a general Activity diagram while considering internal signals, fork, merge, decision nodes and atomic activities of the system.

Algorithm B.9 Logical viewpoint: Feature Blocks - Algorithm to describe the design of the Activity diagram to the Feature Logical blocks

```
1  {
2  n_i foreach (signal in internalSigList){
3  n_i   if (activity of this signal is NOT created) and (signal is NOT linked with a MERGE/
       FORK/DECISION node){
4  A≤n_i  create activity to the signal;
5  A≤n_i  store the new activity in activityList;
6  A≤n_i  create an association from signal to the activity;
7          }
8  n_i  switch(signal){
9  n_i     must be merged:{
10                if(mergeNode does not exist){
11                create a mergeNode to the signal;
12              }
13          create an association from signal to mergeNode;
14          if (next element is ACTIVITY){
15            create activity(ies) to the mergeNode;
16            create an association from mergeNode to activity(ies);
17            store the activity(ies) in activityList;
18          }else{
19             link mergeNode to the new element;
20             create activity(ies) to the new element;
21             create an association from this new element to the activity(ies);
22             store the activity(ies) in activityList;
23           }
24          };
25  n_i     must be decided: {
26            if (decisionNode does not exist){
27              create decision element to the signal;
28            }
29            create an association from signal to decisionNode;
30            if (next element is ACTIVITY) {
31              create activities to the decisionNode;
32              create an association from decisionNode to the activities;
33              store the activities in activityList;
34            }else {
35              link decisionNode to the new element;
36              create activity(ies) to the new element;
37              create an association from this new element to the activity(ies);
38              store the activity(ies) in activityList;
39             }
40          };
41  n_i     must be forked: {
42            if (forkedNode does not exist){
43              create fork element to the signal;
44            }
45            create an association from signal to forkNode;
46            if (next element is ACTIVITY){
47              create activity(ies) to the forkNode;
48              create an association from forkNode to the activity(ies);
49              store the activity(ies) in activityList;
50            }else {
51              link forkNode to the new element;
52              create activity(ies) to the new element;
53              create an association from this new element to the activity(ies);
54              store the activity(ies) in activityList;
```

```
55          }
56        };
57  ni    default:
58          create an association from signal to the activity;
59      }
60    }
61  na foreach(activity in activityList){
62  na   switch(activity next step ){
63  na     decision node: {
64           create an association from the activity to the decisionNode;
65           if (next element is ACTIVITY) {
66             create activity(ies) to the decisionNode;
67             link decisionNode to the activity(ies);
68             store activity(ies) in activityList;
69           }else {
70             link decisionNode to the new element;
71             create activity(ies) to the new element;
72             create an association from this new element to the activity(ies);
73             store activity(ies) in activityList;
74            }
75         }
76  na     fork node: {
77           create an association from the activity to the forkNode;
78           if ( next element is ACTIVITY) {
79             create activity (ies) to the forkNode;
80             link forkNode to the activity(ies);
81             store activity(ies) in activityList;
82           }else {
83             link forkNode to the new element;
84             create activity(ies) to the new element;
85             create an association from this new element to the activity(ies);
86             store activity(ies) in activityList;
87            }
88         };
89  na     merge node: {
90           create an association from the activity to the mergeNode;
91           if (next element is ACTIVITY) {
92             create activity(ies) to the mergeNode;
93             link mergeNode to the activity(ies);
94             store activity(ies) in activityList;
95           }else {
96             link mergeNode to the new element;
97             create activity(ies) to the new element;
98             create an association from this new element to the activity(ies);
99             store activity(ies) in activityList;
100          }
101        };
102 na     atomic activity:{
103          create new activity;
104          create an association from activity to the other activity;
105          store activity in activityList;
106        };
107        final activity:{
108          create internalSignal;
109          create an association from activity to the internalSignal;
110          store internalSignal in internalSigList;
111        };
112     }
113   }
```

```
114 }
```

As it can be observed in Algorithm B.9, all possibilities to design an internal signal and a control node are considered.The formalization of Logical viewpoint follows a more general reasoning to the behavioral system design. It considers the possible design steps, named as "next element", to each: activity, decision node, merge node and a fork node. Thus, "next element" can be a decision node, a merge node or a fork node. Lines 3 and 8 depicts that from an internal signal a new activity or a "new element"/ association can always be created for each internal signal.

Algorithm B.9, lines $2-6$ formalizes how to create a new atomic activity for one input *signal* and how to link these elements (see lines 3 and 5). However, there are some cases where an internal signal can be associated with: merge nodes, decision nodes or fork nodes. Lines $7-59$ formalize these possible conditions through the switch structure. Lines $8-23$ depict the design steps to associate an internal signal to a merge node. In this case, if the merge node does not exist it is necessary to create this model element and link them (lines $9-12$). Moreover, after designing the merge node it is necessary to model the next model element and associate them. The mentioned next element can be an activity, a fork or a decision node (lines $29-39$ formalize these design steps). The same group of instructions, with different models elements, to an internal signal are formalized to create a decision node (lines $24-39$) and a fork node (lines $40-55$) for an internal signal. Finally, the default case to this internal signal is described in line 56. This default case defines an association between an internal signal and an already created activity.

Algorithm B.9 also formalizes the decisions which must be considered for modelling an atomic activity. For each activity on *activityList* is necessary to model the "next model element". It means that the different possibilities in the design of the Activity diagram must be evaluated. In this case, the next model element can be: a decision node (lines $62-74$), a fork node (lines $75-87$), a merge node (lines $88-100$), an atomic activity (lines $101-105$), a final activity (lines $106-110$). In all the possible cases **(1)** it is created an association between the activity and model element and **(2)** there is a checking of the successor model element and its creation. For example, if the next design element to one specific activity is a decision node, it is necessary to associate these elements. After this, it is necessary to model the element which is related to the decision node. In this case, it can be either a new atomic activity (lines $64-68$) or a new element: decision, fork or merge node (lines 68 - 73). In both cases, a new activity is modelled and it is stored in *activityList*.

Evaluation of activities is performed in the overall *activityList*, and it generates a list of internal signals. These signals come from each final activity of the activity list, and they are input to design the output blocks. In Algorithm B.9, the modelling activities are

performed until the storing of the activity (ies) in *activityList*, that is, the next activity level. This looping occurs until the modelling of final activity (ies).

Algorithm B.10 describes the steps to be performed in order to design the Logical viewpoint to the output blocks. This algorithm considers a set of activities for each *internalSignal*. Therefore, the list of *internalSignal* is the input of Algorithm B.10. It is important to mention that this list is provided in Algorithm B.9. The overall activities of the logical output models are analyzed in order to generate the list of external signals (*extenalSignal*).

Algorithm B.10 Logical viewpoint: Output - Algorithm to describe the design of the Activity diagram to the Output Logical blocks

```
1  {
2  ni foreach (signal in internalSigList){
3  ni   if (signal associates with one ACTIVITY) {
4  A≤ni  if (activity of this signal is NOT created){
5  B≤A    create activity(ies) to the signal;
6  B≤A    store each new activity in activityList;
7  –    }
8  A≤ni create an association from signal to the activity;
9  –   }else{
10 ni–A   create the new element for signal;
11 ni–A   link signal to the new element;
12 ni–A   create activity(ies) to the new element;
13 ni–A   create an association from this new element to the activity(ies);
14 ni–A   store the activity(ies) in activityList;
15 –    }
16 – }
17 na foreach (activity in activityList) {
18 na   if (activity is a end node){
19 C≤na  if (externalSignal of this activity is NOT created){
20 D≤C    create externalSignal to this activity;
21 D≤C    create an association from activity to the externalSignal;
22 D≤C    store externalSignal in externalSigList;
23 –   }
24 E≤na  if (activity is going to one already created externalSignal){
25 F≤E  merge association of these signals;
26 F≤E  link merge relation from these activities to the externalSignal;
27 –   }else{
28 E – F    create an association from activity to the externalSignal;
29 –      }
30 –   }else{
31 na–C   link activity to the new element;
32 na–C   create activity(ies) to the new element;
33 na–C   create an association from this new element to the activity(ies);
34 na–C   store the activity(ies) in activityList;
35 –    }
36 – }
37 }
```

Algorithm B.10 starts its design decisions based on the signals of *internalSigList*. It means that each signal is checked and its next model element is created. This model element can either be one activity (lines $3-9$) or one new element (lines 9 - 16). In the

last case, similarly to Algorithm B.9, the new element can be a decision, a merge or a fork node. Moreover, here all the activities are incrementally checked (see lines $17 - 36$) in order to design the Activity diagram to a logical output block. Evaluation of these activities generates an external signal that is stored in $externalSigList$

MARTeSys^{ReqD} methodology also details the outputs of Algorithms B.8, B.9 and B.10 in order to provide another refinement of the Technical viewpoint. Further design decisions must consider the system components and their physical and logical interfaces. In this study, these components are refined by MARTE stereotypes. Therefore, behavioral design and the description of internal and external system signals influence and contribute to definition of the global architecture of the system and its realization.

In this viewpoint, there are a second and a third refinement of the Activity diagram models through constraints of the MARTE profile. Adoption of these constraints can strengthen the expressiveness of models and also refine already annotated constraints. These annotations can add new and refined timing, resources and non-functional information to one element of the Activity diagram. The algorithms to formalize the MARTE annotations, in this viewpoint, are similar to Algorithm B.4 and due to this fact they are not presented here.

ANNEX C

Relating MARTE Profile Constructors and Concerns of RTES

C.1 Introduction

RTES design and specification must consider several types of non-functional requirements that influence the system and the concerns which direct and must be contained within the architectural descriptions. According to [37], "a concern pertains to any influence on a system in its environment, including developmental, technological, business, operational, organizational, political, economic, legal, regulatory, ecological and social influences". Concerns are important for directing architectural decisions of the system, design choices, project management and system implementation.

ISO/IEC/IEEE 42010:2011 [37] is an international standard that addresses several aspects related to the design, analysis and sustainment of system's architectures. Therefore, it is possible to realize different types of requirements and concerns which are pertinent to architecture description of a system. Thus, this specification does not describe how to correlate them between themselves or how to relate them to system global requirements.

Semantic concepts, which are related to the *MARTE Foundations package* and *MARTE Design package*, are presented in the MARTE metamodel through its syntax. An important contribution of this section is the mapping of MARTE stereotypes/constructors to real-time and embedded concerns. Thus, for each RTES concern, already detailed in Chapter 2, one stereotype, from MARTE profile, is suggested to support its definition in architectural models.

C.2 Mapping MARTE Stereotypes to Specify RTES Constraints

MARTE profile defines stereotypes, constructors and annotations in a generic way and without linking them directly to real-time and embedded requirements or system concerns. This fact can make it difficult to apply the MARTE profile, in an intelligible way, in RTES specification and design. Table 14 maps how MARTE constructors can represent and contribute for defining requirements/concerns in model elements. Besides, it facilitates the discovery of MARTE stereotypes which contributes to representation of the overall RTES concerns and clarifies their adoption to the specification and design of RTES.

Non-Functional Requirement/-Concerns	Appropriate MARTE Packages	MARTE Stereotypes Guidelines
Reliability	NFP, Time.	**Accuracy**: $\ll NfpConstraint \gg$, **Maturity**: $\ll NfpConstraint \gg$, **Threshold**: $\ll TimedConstraint$[1] $\gg$, **Fault Tolerance**: $\ll NfpConstraint \gg$, **Recoverability**: $\ll TimedConstraint \gg$, **Compliance**: $\ll NfpConstraint \gg$
Time	NFP, Time.	**Timing Concerns: deadline**: $\ll TimedDurationObservation \gg$, **cost**: $\ll NfpType \gg$, **release time**: $\ll NfpType \gg$, **activation latency**: $\ll TimedDurationObservation \gg$, **start time**: $\ll NfpType \gg$, **end time**: $\ll NfpType \gg$, **event duration**: $\ll TimedEvent \gg$, **instants**: $\ll TimedInstantObservation \gg$, **period**: $\ll TimedDurationObservation \gg$, **Clock**: $\ll ClockType \gg$.
Time	NFP, Time.	**Precision Concerns** [2]**: Jitter**: $\ll TimedInstantObservation \gg$, **Tolerated Delay**: $\ll TimedDurationObservation \gg$, **Freshness**: $\ll TimedInstantObservation \gg$, **Resolution**: $\ll TimedInstantObservation \gg$, **Drift**: $\ll TimedDurationObservation \gg$.

Performance	NFP, Time, GCM.	**Response Time**: $\ll NfpType \gg$, **Accuracy**: $\ll TimedConstraint \gg$, **Operation Capacity**: $\ll NfpConstraint \gg$, **Throughput**: $\ll NfpType \gg$, **Recovery Time**: $\ll NfpType \gg$.
Safety	NFP.	**Privacy**: $\ll NfpConstraint \gg$, **Minimum Metric Access Level**: $\ll NfpType \gg$, **Maximum Metric Access Level**: $\ll NfpType \gg$, **Tolerance to Failures**: $\ll NfpConstraint \gg$, **Risk Level**: $\ll NfpConstraint \gg$, **Risk Prevention**: $\ll NfpConstraint \gg$, **Access Levels**: $\ll NfpType \gg$, **Redundance**: $\ll NfpConstraint \gg$, **Integrity**: $\ll NfpConstraint \gg$.
Distribution	NFP, GRM, GCM, SRM.	**Concurrency**: $\ll ConcurrencyResource \gg$, **Communication**: $\ll CommunicationResource \gg$, **Tasks Allocation**: $\ll Scheduler \gg$ or $\ll MutualExclusionResource \gg$, **Host Definition**: $\ll ClientServerSpecification \gg$, **Distributed Log**: $\ll SwCommunicationResource \gg$, **Parallelism**: $\ll NfpConstraint \gg$, **Synchronous Process**: $\ll SwSynchronizationResource \gg$.
Interoperability	GRM, HLAM, HRM.	**Comunication Protocols**: $\ll ResourceManager \gg$ or $\ll MutalExlusionProtocol \gg$, **Bus Structure**: $\ll HW\_Bus \gg$, **Inter-Process Communication**: $\ll RtSpecification \gg$.
Security	GRM.	**Control Access**: $\ll ConcurrencyResource \gg$, **Integrity**: $\ll SynchResource \gg$, **Subsystem Integration**: $\ll SwInteractionResource \gg$, **Distributed Log**: $\ll SwCommunicationResource \gg$, **Privacy**: $\ll NfpConstraint \gg$.
Resource Utilization	GRM, HRM.	**Resource Features**: $\ll HwResourceService \gg$, **Policies of Access**: $\ll MutualExclusionProtocol \gg$, **Deadlock**: $\ll NfpConstraint \gg$.
Deadlock	NFP, HRM, SRM.	**Policies of Access**: Software level: $\ll AccessPolicyKind \gg$, **Policies of Access**: Hardware level: $\ll MutualExclusionProtocol \gg$, **Correctness of Outputs**: $\ll NfpConstraint \gg$, **Deadline Satisfaction**: $\ll NfpConstraint \gg$.

Embedded	HRM.	**Layout of the Devices**: $\ll HwComponent^3 \gg$, **Memory Organization**: $\ll HW\_StorageLayout^3 \gg$, **Input Interruption Control**: $\ll InterruptKind \gg$, **Outputs Interruption Control**: $\ll InterruptKind \gg$, **Energy Consumption**: $\ll HW\_PowerSupply \gg$, **Communication Channels**: $\ll HW\_Media^3 \gg$, **Bandwidth**: $\ll HW\_Bus \gg$, **Connections**: $\ll HW\_EndPoint \gg$, **Arbiter**: $\ll HW\_Arbite \gg$, **Memory Size**: $\ll HW\_Storage \gg$, **Power Consumption**: $\ll HW\_PowerSupply \gg$, **Heat Control**: $\ll HW\_CoolingSupply \gg$, **Execution Nodes**: $\ll HW\_Processor \gg$ and $\ll HW\_ComputingResource \gg$.
Concurrency	NFP, GRM, SRM.	**Parallelism**: $\ll SwSynchronizationResource \gg$, **Correctness**: $\ll NfpConstraint \gg$, **Performance**: $\ll NfpConstraint \gg$, **Robustness**: $\ll NfpConstraint \gg$, **Control Access**: $\ll ConcurrencyResource \gg$, **Synchronization**: $\ll SwSynchronizationResource \gg$.
Flexibility	NFP, GRM.	**Capability**: $\ll NfpContraint \gg$, **Transportability**: $\ll NfpType \gg$, **Interchangeability**: $\ll SharedDataComResource \gg$, **Expansion**: $\ll NfpContraint \gg$, **Contraction**: $\ll NfpContraint \gg$, **Effectiveness**: $\ll NfpType \gg$.
Maintainability	NFP, Time.	**Mean Time to Repair**: $\ll TimedConstraint \gg$ and $\ll TimedProcessing \gg$, **Maximum Time to Repair**: $\ll, TimedConstraint \gg$ and $\ll TimedProcessing \gg$, **Maintenance Staff Hours**: $\ll NfpType \gg$, **Modifiability**: $\ll NfpType \gg$, **Evolvability**: $\ll NfpContraint \gg$, **Modularity**: $\ll NfpContraint \gg$, **Skill Levels of Staff**: $\ll NfpContraint \gg$, **Frequency of Maintenance**: $\ll NfpType \gg$, **Complexity Level**: $\ll NfpType \gg$.

Usability	NFP, Time.	**Measures of Performance**: $\ll NfpConstraint \gg$ and $\ll Npf \gg$, **Capabilities of Usage**: $\ll NfpConstraint \gg$ and $\ll Npf \gg$, **Measure of Safety**: $\ll NfpConstraint \gg$ and $\ll Npf \gg$, **Measures of Availability**: $\ll NfpConstraint \gg$ and $\ll Npf \gg$.
Inter-process Communication	GRM, SRM.	**Policies of Access**: $\ll SwMutualExclusionResource \gg$, **Effectiveness**: $\ll NfpType \gg$, **Policies of Control**: $\ll CallConcurrencyKind^4 \gg$ and $\ll SwInteractionResource \gg$.

Table 14 – Adoption of MARTE Packages Stereotypes to represent RTES Concerns.

Table 14 represents the relation of atomic RTES concerns, in the left side, with MARTE profile constructors, in the right side, in order to contribute for the representation of non-functional properties in architectural models. In general, the presented leveling allows to strengthen the specification, modelling, design, validation and maintenance of RTES, once it makes possible to address different concerns and properties of RTES at early stages of their development.

The proposed guidelines show relevant concepts to specification, design and development of RTES. In this domain, these concepts can be related to qualitative, temporal and constrained issues. Thus, it contributes to understand how to describe and refine non-functional concerns in MBE approaches while it contributes to minimize the complexity of their analysis and development.

Employing MARTE stereotypes to the specification, analysis and design of RTES allows early and high level characterization of a concept. Besides, through these MARTE concepts it is possible to set the nature, the semantic and several real-time and embedded features in a model. Moreover, it allows to trace RTES concerns along of different viewpoints.

ANNEX **D**

System Realization

The implementation of the systems features performed, in this study, allows and support the evaluation activities of the MARTeSys^{ReqD} methodology. Thus, a brief explanation regarding the main adopted technologies is performed here. Moreover, this section provides an explanation of concepts to abstract/implement, in the code, software, systems, hardware, mechanical and physical components of the TIS.

D.0.1 Toolbox for Modeling, Simulation and Verification of MARTeSys^{ReqD} Methodology

In this research, the Papyrus [176] tool has been adopted to design activities and to generate source code. The developed code has been embedded in an Arduino microcontroller [192]. Moreover, the real-time controllers as, for example, tasks coordination, have been executed in FreeRTOS environment [193].

Arduino is an open source platform to free electronic prototyping of circuits. The Arduino microcontroller is programmable in C/C++ programming languages and it is accessed by an Integrated Development Environment (IDE). **C++** is a commercial and open-source programming language that enables development of general purpose programs while it supports the Object Oriented Paradigm [194]. Moreover, C++ is often used to develop RTES [13], [195].

Papyrus is an open source tool which allows to model UML and SysML diagrams, as well as representing MARTE profile elements. In this study, Papyrus was used for modeling activities because it can generate code in C, C++, Java and Ada languages, it is open source, and also due to its customization capability.

FreeRTOS is an open-source Real-Time Operationg System (RTOS). It provides facilities to manage tasks (creation, access, elimination), real-time scheduling and full access to FreeRTOS capabilities within classic Arduino environment.

The integrated environment of the **Uppaal** [138] tool is adopted in this study to model and verify artefacts of Functional viewpoint. The tool is designed to verify systems

that can be modelled as networks of timed automata extended with integer variables, structured data types, user defined functions and channel synchronisation [136]. As it is detailed in Section 7.2, the artefacts of Requirements viewpoint are transformed into a correspondent timed automata, by the proposed transformation rules, and verified in the Uppaal tool. The mentioned technologies are important in this study once they contribute to design architectural models, to code generation and to empirical evaluation activities.

D.0.2 System Implementation

The case study can be subdivided into two stages: embedded software development and the configuration of hardware components to simulate the physical interface of the system. For the foregoing stage, analog and digital ports are available to receive and send signals to external devices like LEDs, sensors and actuators. In the presented case study, LEDs, push buttons and beeps have been used to simulate the Turn Indicator features (see Figure 56). In the development phase, an implementation of the architectural models and their imposed MARTE constraints is performed in order to allow further empirical evaluations (Chapter 7) of the proposed case study.

Figure 56 provides an overview of components which controls the input and output interfaces of the case study. These components are combined to prototype the Turn Indicator functions in the Arduino microcontroller. For all the features of the Turn Indicator a set of buttons is available. The frontal and back turn/hazard lights are realized by yellow and red LEDs represent brake lights. The system controllers are mainly related with the coordination of the buttons: to Turn Right/Left in a hard manner, the buttons to Turn Right/Left in a soft manner, the buttons to signalize Hazard and Break Actions, the button to Activate/Deactivate the Clamp, all the correspondent System Lights and Feedback (sonorous beep) to the driver.

Figure 44 presented in Chapter 6 shows the overall class of the system and its constrained tasks. The Class diagram is composed by six classes and they are the bases to the implementation stage. Within the Class diagram and the implementation models these classes are named as *TurnHazardFeatures*, *TurnIndicatorFeature*, *HazardFeature*, *Lamp*, *Control* and *BreakSystem*. During the design steps the functionalities of the TIS are developed according to the architectural models. Here, an automated and partial code generation, from graphical annotations of the Class diagrams to the source code, is performed. Papyrus can generate the main structure of classes, which are already specified by the Class diagram, together with the specified attributes and methods.

The *Control* class is responsible for management of the main system's components. It implements the decisions to control the input and output signals and their processing. The *TurnIndicatorFeature* and *HazardFeature* classes are operated here through their

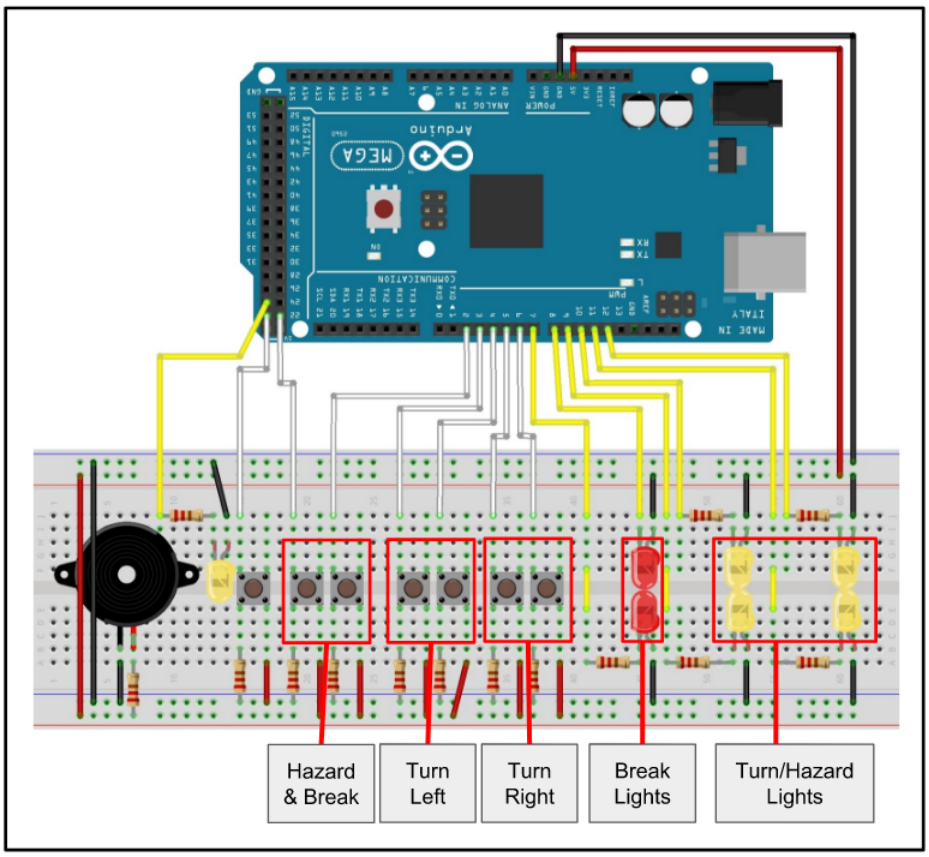

Figure 56 – Prototype of Turn Indicator System.

instances. These instances allow to access methods and properties of these classes.

As it was described above, the system has been developed to be embedded in an Arduino microcontroller. Turn Indicator components have been coupled in the Arduino board as buttons, bips and LEDs. Therefore, in this class the input and output components are logically declared and defined to allow their manipulation in input/output events. *Control* class manages the setup of pins to control the lights, allowing the flashing functions of the case study, and defines the buttons to receive the users commands. In addition, this class is responsible for creating the system's tasks and to create their logical functions. Once this case study is related to the RTES context, it is important to allow parallelism, cooperation and scheduling between the different Turn Indicator features. Figure 45 shows the overall system's tasks. These tasks are related to data reading (*inputSignalHandler*), data processing (*taskTurnFlashing*, *taskHazardFlashing*, *taskBreaking*) and data output (*outputSignalHandler*).

The Turn Indicator features, and consequently its tasks, are triggered by events. For example, when the driver wants to activate the car lights to flash to the left side, he/she needs to produce a turn event. However, in this study, all system events are treated as periodic and static occurrences in order to provide deterministic guarantees regarding response time. This fact allows that non-deterministic system services can always be

treated and executed at specific time intervals. This strategy forces the fulfillment of the constraints regarding the proposed deadlines and periodic executions.

The function *vTaskDelayUntil* performs the control and measurement of period tasks. It has been adopted to record the next period of one task and to ensure a constant execution frequency. Function *vTaskDelayUntil* is based on two parameters: last execution time and the period of repetition. Thus, it was possible to measure the tasks activation, the manner which the schedule should control each task and follow the schedule policy.

The **inputSignalHandler** task is responsible to control all the system's inputs as, for example, the power button, the lever buttons, the hazard button and the breaking button. The turn buttons represent the turn indicator in a hard and in a soft manner, to the left and right sides. As it can be observed in Figures 44 and 45, there is a periodic constraint to this task ($\ll NfpConstraint \gg$ stereotype, period = 20 ms) and it indicates that it executes in pre-defined periods. These constraints and their assumed values comes from the non-functional restriction of requirements specification. The Turn Indicator functions can only run if the system is turned on and it is represented by a component named as *clamp*. The clamp conditions are checked by a button called as *butPower*. For all the input evaluations, the clamp condition must be checked to show if there is power to execute the system functions. The inputSignalHandler task is responsible for the periodic reading of hazard, turn indicator and break inputs.

The **taskTurnFlashing** task is responsible to execute the priority features of TIS. An instance of *TurnIndicatorFeature* class, named as *turnIndicator*, manages all the turn flashing controllers. The object *turnIndicator* can call the *turnIndicatorLever* method in order to identify which signal type is sent from the physical part (buttons). There are two parameters that define a user command/request:

1. *pressType*: defines the flashing type to the system which can be hard or soft;

2. *direction*: defines the turn flashing side which can be right or left. Another method, called "loop", was developed to execute the logic to flashing the lamps (LEDs) based on the results of the *turnIndicatorLever* method. In the "loop" method all the Turn Indicator functions are cyclically executed.

The *taskTurnFlashing* task is also responsible to access the Turn Indicator features which have been defined in the *TurnIndicatorFeature* class. Therefore, prioritization of functions as, for example, when the flash right (in a soft manner) is running and it should be correctly replaced to the flash left (in a hard manner) is performed in this class. The turn indicator features prioritization are performed between all flashing situations and the overall conditions are described below:

- From soft right to soft right; from soft right to soft left; from soft right to hard right and from soft right to hard left;
- From soft left to soft left; from soft left to soft right; from soft left to hard right and from soft left to hard left;
- From hard right to hard right; from hard right to hard left; from hard right to soft right and from hard right to soft left;
- From hard left to hard left; from hard left to hard right; from hard left to soft right and from hard left to soft left.

The *turnIndicator* object operates methods of *TurnIndicatorFeature* class. Consequently, this instance is also constrained by the MARTE constructors which were added in the architectural design models. These constraints were automatically translated to code annotations, from the Papyrus models, and can be visualized as guidelines in realization models. Figure 57 presents an example of non-functional constraint which was attached in the *setFlashingSoft* method.

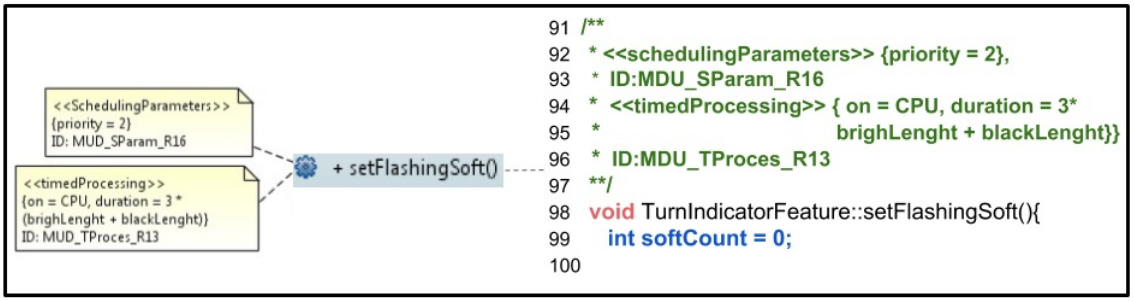

Figure 57 – Example of MARTE Constraints to the System Realization.

Functions of the Turn Indicator have higher priority than hazard functions. The $\ll NpfConstraint \gg$ stereotype declares the needs of preemption of this task concerning Hazard's tasks. In addition, a processing criteria to soft flashing execution is annotated through the $\ll timedProcessing \gg$ stereotype. This constraint has been attached in the related methods.

The **taskHazardFlashing** task executes features related to hazard controllers. This is a periodic task and it executes an object of the class *HazardFeature*. Thus, hazard flashing functions are controlled by this task. This object access the flashing method to control the lights activation and deactivation in accordance with the turn settings definition. As it can be observed in the Class diagram of the Turn Indicator, Figure 44, the *HazardFeature* and *TurnIndicatorFeature* classes are a specialization of *TurnHazardFeatures* class. Thus, the methods to manage (activate and deactivate) the frontal right lamp, the back right lamp, the frontal left lamp and the back left lamp are inherited and

specialized in this class. Moreover, outputs of the *HazardFeature* are specialized in this class and they are also inherited from *TurnHazardFeatures* class.

The **taskBreaking** task shows the behavior of the TIS when an emergency breaking situation occurs. The break events come from the alert breaking feature of Body Control Model (BCM) and the lights signal should be controlled by the Turn Indicator module. Here, breaking event is observed when the *butBreak* button is pressed and the *breakSystem* object, from the *BreakSystem* class, performs the red lights signalization. In addition, a $\ll NpfConstraint \gg$ stereotype to *taskBreaking* object was automatically represented in this class and it constrains the visual behavior of the break lights. As it occurs with the objects of *HazardFeatures* and *TurnIndicatorFeature* classes the object of *BreakSystem* class, controlled in *taskBreaking*, also communicates with *outputSignalHandler* method to send/generate output signals.

The **outputSignalHandler** task represents a periodic task ($\ll NfpConstraint \gg$ stereotype, period = 20 ms) and it is responsible to control all the outputs to hazard, turn indicator and break tasks. This task executes the method *outputSignalHandler* of all objects. In a general way, this method creates a bridge between the logical part of the TIS and their physical parts. This interaction is done by objects of the *TurnIndicatorFeatures* and instances of the *Lamp* classes. The *Lamp* class is the one responsible to control (activate and deactivate) the pins (lights) of the system.

A group of measurement are developed and considered the mentioned tasks, in the following chapter, in to perform simulation and verification activities. The dynamic behavior of these tasks is subsequently checked to allow the prediction and evaluation of the constraints assumptions.

ANNEX **E**

Results of the First Qualitative Evaluation

Table 15 represents a summary of the questionnaires applied in the First Qualitative Evaluation.

Criteria/Interval	**Fully Agree**			**Neutral**			**Totally Disagree**
1. Correctness: Is it possible to observe the right definition of system description (considering the functionalities which must be reachable in the system development)?		E3, E4, E5	E1	E2			
2. Correctness: Is it possible to observe the right definition of system description (considering the right use of the semantic and syntax of each diagram)		E1, E3, E5		E2	E4		
3. Completeness: Is it possible to see the fulfillment of the Requirements viewpoint along of the Functional and Logical viewpoints?			E1, E5		E2, E3	E4	
4. Consistency: Are the views of each system viewpoints consistent between each other?		E4, E5, E2		E1	E3		
5. Ambiguity: Are the views and its respective models non-ambiguous for each viewpoints of the system?		E4, E5	E2	E1, E3			

6. Conformity: Is the Conformity between models observable in the Requirements, Functional and Logical viewpoints of the system?	E5	E2	E1	E4	E3		
7. Legibility: Are the models understandable/readable by humans?	E1, E4, E5	E2, E3					
8. Abstraction Levels: Is the design elaborated under different and complementary abstraction levels?	E1, E2, E5	E3, E4					
9. Multiple Views: Is the architecture described by using multiple viewpoints?	E1	E2, E3, E5	E4				
10. Expressiveness: Are the viewpoints of the system expressive?	E3	E4, E5	E1	E2			
11. Relevance to the Field of Study: How do you describe the relevance of the models to the automotive system design?	E1	E3, E4, E5	E2				
12. Relevance of MARTE constructors: Are the timing features presented in the models, mainly through MARTE elements, able to contribute to the non-functions constraints descriptions?	E2, E4	E5	E1		E3		
13. Relevance of MARTE constructors: The MARTE constraints make clearly the temporal aspects of the system under analysis. Does the early definition of these temporal aspects help the architectural level description?	E2	E3, E4, E5	E1				

Table 15 – Results of the First Qualitative Evaluation.

ANNEX **F**

Results of the Second Qualitative Evaluation

Table 16 represents a summary of the questionnaires applied in the Second Qualitative Evaluation.

Criteria/Interval	**Fully Agree**			**Neutral**			**Totally Disagree**
1. Correctness: Can you observe the correctness of models in the presented architectural viewpoints? This means, is it possible to observe that the models are able to describe and consider the system functionalities.	E1, E5, E6, E8, E11	E2, E3, E4, E9, E10		E7			
2. Correctness: Are the models, elaborated in the architectural viewpoints, correct? This question wants to evaluate if the authors (in your opinion) are adopting the right semantic and syntax to the diagrams design.	E1, E6, E9	E3, E5, E7, E8, E11	E2, E10	E4			
3. Completeness: Can you contemplate the completeness of models along of the architectural viewpoints? This means, it is possible to see the fulfillment of the requirement views throughout functional, logical, logical and technical design.	E6	E1, E8, E9	E2, E3, E5, E10, E11	E7	E4		

4. Consistency: In your opinion, are the system viewpoints consistent between each other? This means, that the system refinements in each abstraction level are defining the same system requirements (through different viewpoints).	E1, E2, E5, E6, E8	E3, E4, E9, E10, E11		E7			
5. Ambiguity: Are the models non-ambiguous for each viewpoints of the system? This means that the presented models do not express the system design under two or more possible understandable context.	E4, E6, E8	E1, E5, E7, E9, E10	E2, E3, E11				
6. Conformity: Can you see the conformity between models of the Requirements, Functional, Logical and Technical viewpoints? In this case, it is possible to highlight that the models are conforming to inputs and outputs necessary (longed/expected) to Requirements, Functional, Logical and Technical viewpoints.	E1, E5, E6, E8	E2, E3, E9, E11	E4, E10	E7			
7. Legibility: Were you able to understand the models and their diagrams? This questions seeks to know if the models intelligible/readable by humans. This means, it is possible to understand the semantic and syntax of the viewpoints which are described in the models.	E1, E4, E5, E8, E9	E2, E6, E7	E10	E3, E11			
8. Abstraction Levels: Is the architectural description elaborated under different and complementary abstraction levels?	E1, E2, E4, E5, E6, E7, E9, E10, E11	E3, E8					

9. Multiple Views: Is the concept of viewpoints and refinements of their views adopted in the architectural description? This means, that architectural models are described through distinct perspectives and they can be used by different proposes and stakeholders.	E1, E5, E6, E7, E8, E9	E2, E10, E11	E3, E4				
10. Expressiveness: Are the viewpoints of the system expressive? That is, can they ensure that the design models are enough expressive and valuable to represent the system requirements?	E3, E8	E1, E4, E5, E6, E9	E2, E7, E11	E10			
11. Relevance to the Field of Study: How do you describe the relevance of the models to the automotive system design? That is, can the models contribute to the automotive system design?	E1, E3, E5, E6, E8, E11	E2, E4, E9	E7	E10			
12. Relevance of MARTE constructors: Are the timing features annotated in the models, mainly through MARTE elements, able to represent early non-functions constraints descriptions?	E2, E4, E6, E9, E11	E1, E3, E5, E7, E8, E10					
13. Relevance of MARTE constructors: The MARTE constraints make clearly the temporal aspects of the system under analysis. Can the definition of temporal aspects and RTES concerns, performed in the models, improve/support the architectural viewpoints description?	E1, E2, E6, E8, E9, E11	E3, E4, E5, E7	E10				

14. Practical Applicability of the MARTeSys^{ReqD} Methodology: In your opinion, the proposed methodology adds value to Real-Time Embedded System (RTES) design? This question, aims to analyze the impact and importance of this methodology and their functional and non-functional guidelines to the practical RTES development.	E1, E2, E6, E5, E8, E9	E3, E4, E7	E10, E11				
15. Traceability Criteria: Are the traceability strategies effective to track/link design elements in different architectural viewpoints?		E1, E6, E8, E10	E2, E4, E9, E11	E3, E5, E7			

Table 16 – Results of the Second Qualitative Evaluation.

F.1 Personal Questions about the Interviewees

The interviewees are requested to answer the following questions:

1. Field of work/research:
2. Time of carrier:
3. Job position:
4. Level of knowledge of SysML profile (High/Medium/Low):
5. Level of knowledge of MARTE profile (High/Medium/Low):
6. Level of knowledge of SPES methodology (High/Medium/Low):

Bibliography

1 SysML, OMG. **OMG - OMG Systems Modeling Language - version 1.4.** [S.l.: s.n.], 2017. Technical Report Formal/2015.06.03.

2 MARTE. **Modeling and Analysis of Real-Time and Embedded Systems (MARTE)- version 1.1., OMG**. [S.l.: s.n.], 2011. Technical Report Formal/2011.06.02.

3 GOMAA, H. **Real-Time Software Design for Embedded Systems**. 1st. ed. New York, NY, USA: Cambridge University Press, 2016.

4 YAN, X.; LI, Y. Real-time simulation of automotive systems based on UPPAAL. In: **2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)**. [S.l.: s.n.], 2017. p. 173–176.

5 WAN, B. et al. A Time-Aware Programming Framework for Constructing Predictable Real-Time Systems. In: **19th International Conference on High Performance Computing and Communications**. [S.l.: s.n.], 2017. p. 578–585.

6 BRAU, G.; HUGUES, J.; NAVET, N. Towards the Systematic Analysis of Non-Functional Properties in Model-based Engineering for Real-Time Embedded Systems. **Science of Computer Programming**, v. 156, p. 1 – 20, 2018.

7 STANKOVIC, J. A.; RAMAMRITHAM, K. (Ed.). **Tutorial: Hard Real-time Systems**. Los Alamitos, CA, USA: IEEE Computer Society Press, 1989.

8 BUTTAZZO, G. C. **Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications**. 3rd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2011.

9 PANUNZIO, M.; VARDANEGA, T. An Architectural Approach with Separation of Concerns to Address Extra-Functional Requirements in the Development of Embedded Real-time Software Systems. **Journal of Systems Architecture**, v. 60, n. 9, p. 770 – 781, 2014.

10 LIU, J. W. S. W. **Real-Time Systems.** 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

11 DARSCHT, P.; PEREIRA, C. E. Modeling Computer-based Real-Time Systems: Which Views do We Need? **Control Engineering Practice**, v. 5, p. 993–998, 1997.

12 SELIC, B. The Theory and Practice of Modeling Language Design for Model-based Software Engineering-A Personal Perspective. In: SPRINGER. **Lecture Notes in Computer Science (LNCS) Series**. [S.l.]: Springer, 2010.

13 LAPLANTE, P. A.; OVASKA, S. J. **Real-Time Systems Design and Analysis: Tools for the Practitioner.** 4th. ed. [S.l.]: Wiley-IEEE Press, 2012.

14 PETTERSSON, P. **Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice.** Tese (PhD Thesis) — Department of Computer Systems Uppsala University, Uppsala, Sweden, 1999.

15 COLNARIC, M.; VERBER, D.; HALANG, W. A. **Distributed Embedded Control Systems: Improving Dependability with Coherent Design.** 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2008.

16 KIRNER, T. G.; DAVIS, A. M. Requirements Specification of Real-Time Systems: Temporal Parameters and Timing-Constraints. **Information & Software Technology**, v. 38, n. 12, p. 735 – 741, 1996.

17 COLNARIC, M.; VERBER, D.; A., H. W. Real-Time Characteristics and Safety of Embedded Systems. In: _____. London: Springer London, 2008. p. 3–28.

18 POHL, K. et al. **Model-based Engineering of Embedded Systems: The SPES 2020 Methodology.** 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2016.

19 BöDE, E. et al. Design Paradigms for Multi-Layer Time Coherency in ADAS and Automated Driving (MULTIC). In: **FAT-Schriftenreihe 302**. 302. ed. [S.l.]: Forschungsvereinigung Automobiltechnik e.V. (FAT), 2011, (FAT-Schriftenreihe).

20 VISHNYAKOV, A.; ORLOV, S. Software Architecture and Detailed Design Evaluation. **Procedia Computer Science**, v. 43, p. 41 – 52, 2015.

21 MARTINS, L. E. G.; GORSCHEK, T. Requirements Engineering for Safety-critical Systems. **Inf. Softw. Technol.**, v. 75, n. C, p. 71–89, 2016.

22 LAPLANTE, P. **Real-time Systems Design & Analysis.** 3th. ed. [S.l.]: Wiley India Pvt. Limited, 2006.

23 VANVLIET, H. **Software Engineering: Principles and Practice.** 3th. ed. Free University, Amsterdam, The Netherlands: John Wiley & Sons, 2008.

24 NOYER, A. et al. A Model-based Framework Encompassing a Complete Workflow from Specification until Validation of Timing Requirements in Embedded Software Systems. **Software Quality Journal**, v. 25, 2016.

25 GIESE, H. et al. **Model-based Engineering of Embedded Real-time Systems.** [S.l.]: Springer, 2011. v. 6100.

26 SANDUKA, I.; OBERMAISSER, R. Model-based Development of Systems-of-Systems with Real-Time Requirements. In: **12th IEEE International Conference on Industrial Informatics (INDIN)**. [S.l.]: IEEE, 2014. p. 188–194.

27 INCOSE, T. O. I. C. on S. E. **INCOSE Systems Engineering Vision 2020.** [S.l.], 2010.

28 SERNA, E. M.; BACHILLER, O. S.; SERNA, A. A. Knowledge Meaning and Management in Requirements Engineering. **International Journal of Information Management**, v. 37, n. 3, p. 155–161, 2017.

29 WALTER, S.; RETTBERG, A.; KREUTZ, M. Towards Formalized Model-based Requirements for a Seamless Design Approach in Safety-Critical Systems Development. In: **Int. Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORC)**. [S.l.: s.n.], 2015. p. 111–115.

30 KELLNER, A. et al. Challenges in Integrating Requirements in Model Based Development Processes in the Machinery and Plant Building Industry. In: **2016 IEEE International Symposium on Systems Engineering (ISSE)**. [S.l.: s.n.], 2016. p. 1–6.

31 CARRILLO, J.; NICOLAS, J. Requirements Engineering Tools. **IEEE Software**, v. 28, n. 4, p. 86–91, 2011.

32 MAALEM, S.; ZAROUR, N. Challenge of Validation in Requirements Engineering. **Journal of Innovation in Digital Ecosystems**, v. 3, n. 1, p. 15–21, 2016.

33 VOGELSANG, A. et al. Functional Viewpoint. In: _____. **Model-based Engineering of Embedded Systems: The SPES 2020 Methodology**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 69–83.

34 OQUENDO, F.; WARBOYS, B.; MORRISON, R. (Ed.). **Software Architecture: The Next Step.**, v. 3047 de **Lecture Notes in Computer Science**, (Lecture Notes in Computer Science, v. 3047). Berlin, Heidelberg: Springer, 2004. 194-199 p.

35 BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice.** 3rd. ed. Boston, MA, USA: Addison-Wesley Professional, 2012.

36 MALAVOLTA, I. et al. What Industry Needs from Architectural Languages: A Survey. **IEEE Trans. Software Eng.**, v. 39, n. 6, p. 869–891, 2013.

37 ISO/IEC/IEEE Systems and Software Engineering - Architecture Description. **ISO/IEC/IEEE 42010:2011(E) - (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)**, p. 1–46, Dec 2011.

38 SHAW, M.; GARLAN, D. **Software Architecture: Perspectives on an Emerging Discipline.** 1th. ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

39 SHAW, M. The Coming-of-Age of Software Architecture Research. In: **23rd International Conference on Software Engineering ICSE**. Toronto, Ontario, Canada: [s.n.], 2001. p. 656–664.

40 MEDVIDOVIC, N.; TAYLOR, R. N. A Classification and Comparison Framework for Software Architecture Description Languages. **IEEE Trans. Softw. Eng.**, IEEE Press, Piscataway, NJ, USA, v. 26, n. 1, p. 70–93, 2000.

41 WEHRMEISTER, M. A.; BERKENBROCK, G. R. AMoDE-RT: Advancing Model-driven Engineering for Embedded Real-Time Systems. In: **16th Int. Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing**. [S.l.: s.n.], 2013. p. 1–7.

42 GÓNGORA, H. G. C.; GAUDRÉ, T.; TUCCI-PIERGIOVANNI, S. Complex Systems Design & Management: Proceedings of the Third International Conference on Complex Systems Design & Management. In: _____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. cap. Towards an Architectural Design Framework for Automotive Systems Development, p. 241–258.

43 LI, M. et al. Model-based Systems Engineering with Requirements Variability for Embedded Real-Time Systems. In: **IEEE International Model-driven Requirements Engineering Workshop (MoDRE)**. Ottawa, ON, Canada: IEEE, 2015. p. 1–10.

44 ESPINOZA, H. et al. Challenges in Combining SysML and MARTE for Model-based Design of Embedded Systems. In: **5th European Conference (ECMDA-FA)**. Enschede, The Netherlands: Springer, 2009. (Lecture Notes in Computer Science, v. 5562), p. 98 – 113.

45 CANCILA, D.; ESPINOZA, H.; PAIGE, R. F. Special Issue on Model Based Engineering for Embedded Systems Design. **Journal of Systems Architecture - Embedded Systems Design**, v. 58, n. 5, p. 177, 2012.

46 WOZNIAK, E. et al. Assigning Time Budgets to Component Functions in the Design of Time-critical Automotive Systems. In: **Proc. of the 29th ACM/IEEE International Conference on Automated Software Engineering**. [S.l.: s.n.], 2014. p. 235–246.

47 SELIC, B. Beyond Mere Logic - A Vision of Modeling Languages for the 21st Century. In: **Proceedings of the 5th International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS)**. Angers, Loire Valley, France: SciTePress, 2015. p. 1 – 9.

48 FEILER, P. H.; GLUCH, D. P. **Model-based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language.** 1st. ed. [S.l.]: Addison-Wesley Professional, 2012.

49 HU, F. et al. Robust Cyber-Physical Systems: Concept, Models and Implementation. **Future Generation Computer Systems**, v. 56, p. 449 – 475, 2016.

50 REINKEMEIER, P. et al. A Pattern-based Requirement Specification Language: Mapping Automotive Specific Timing Requirements. In: **Workshopband Software Engineering**. [S.l.]: Köllen Druck + Verlag GmbH, 2011. (Lecture Notes in Informatics (LNI)), p. 99–108.

51 WEHRMEISTER, M. A. et al. Combining Aspects and Object-Orientation in Model-driven Engineering for Distributed Industrial Mechatronics Systems. **Mechatronics**, v. 24, n. 7, p. 844–865, 2014.

52 GIESE, H. et al. (Ed.). **Modeling Languages for Real-Time and Embedded Systems: Requirements and Standards-based Solutions.**, v. 6100 de **Lecture Notes in Computer Science**, (Lecture Notes in Computer Science, v. 6100). Berlin, Heidelberg: Springer-Verlag, 2007. 129 - 154 p.

53 FAN, X. **Real-Time Embedded Systems.** 1st. ed. [S.l.]: Elsevier, 2015.

54 WAN, J.; CANEDO, A.; FARUQUE, M. A. A. Functional Model-based Design Methodology for Automotive Cyber-Physical Systems. **IEEE Systems Journal**, v. 11, n. 4, p. 2028–2039, 2017.

55 SPT, O. **UML Profile for Schedulability, Performance and Time (SPT) - version 1.1.** [S.l.: s.n.], 2005. Technical Report Formal/2005.07.05.

56 QOS, O. **UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms - version 1.1.** [S.l.: s.n.], 2008. Technical Report Formal/2008.04.08.

57 UML, O. **OMG Unified Modeling Language - version 2.5.** [S.l.: s.n.], 2015. Technical Report Formal/2015.03.01.

58 SILVESTRE, E. A.; SOARES, M. S. Multiple View Architecture Model for Distributed Real-Time Systems Using MARTE. In: **20th Information Systems Development, Reflections, Challenges and New Directions (ISD 2011)**. Heriot-Watt University, Edinburgh, Scotland, UK: [s.n.], 2011. p. 195 – 203.

59 SELIC, B.; GERARD, S. Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE. In: **Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE**. 1. ed. Boston: Morgan Kaufmann, 2014.

60 NI, S. et al. Modeling Dependability Features for Real-Time Embedded Systems. **IEEE Transactions on Dependable and Secure Computing.**, v. 12, n. 2, p. 190 – 203, 2015.

61 SIKORA, E.; TENBERGEN, B.; POHL, K. Requirements Engineering for Embedded Systems: An Investigation of Industry Needs. In: **Proc. of Requirements Engineering: Foundation for Software Quality - 17th International Working Conference REFSQ**. [S.l.: s.n.], 2011. p. 151–165.

62 MORI, M. et al. Systems-of-Systems Modeling using a Comprehensive Viewpoint-based SysML profile. **Journal of Software: Evolution and Process**, v. 30, n. 3, 2018.

63 ALUR, R.; DILL, D. L. A Theory of Timed Automata. **Theoretical Computer Science**, v. 126, p. 183–235, 1994.

64 BECKER, L. B.; PEREIRA, C. E. From Design to Implementation: Tool Support for the Development of Object-Oriented Distributed Real-Time Systems. In: **Proceedings 12th Euromicro Conference on Real-Time Systems**. [S.l.: s.n.], 2000. p. 109–116.

65 BARBIERI, G.; FANTUZZI, C.; BORSARI, R. A Model-based Design Methodology for the Development of Mechatronic Systems. **Mechatronics**, v. 24, p. 833–843, 2014.

66 ASSAR, S. Model Driven Requirements Engineering: Mapping the Field and Beyond. In: **IEEE 4th International Model-driven Requirements Engineering Workshop (MoDRE)**. Karlskrona, Sweden: [s.n.], 2014. p. 1–6.

67 EBERT, C.; PEREIRA, C. E. Measuring the Impact of Real Time Design Techniques. In: **Innovationen bei Rechen- und Kommunikationssystemen, Eine Herausforderung für die Informatik**. [S.l.: s.n.], 1994. p. 348–355.

68 PEREIRA C. E.AND FRIGERI, A.; DARSCHT, P.; HALANG, W. Object-Oriented Development of Real-Time Industrial Automation Systems. **Volume O: Power Plants and Systems, Computer Control**, v. 29, p. 321–326, 1996.

69 FURIA, C. A. et al. **Modeling Time in Computing.** 1st. ed. [S.l.]: Monographs in Theoretical Computer Science., 2012.

70 SCHEEREN, I.; PEREIRA, C. E. Combining Model-based Systems Engineering, Simulation and Domain Engineering in the Development of Industrial Automation Systems: Industrial Case Study. In: **17th International Symposium on Object/Component-Oriented Real-Time Distributed Computing**. [S.l.: s.n.], 2014. p. 40–47.

71 NOYRIT, F. et al. Consistent Modeling Using Multiple UML Profiles. In: PETRIU, D.; ROUQUETTE, N.; HAUGEN, y. (Ed.). **Model Driven Engineering Languages and Systems**. Oslo, Norway: Springer Berlin Heidelberg, 2010. (Lecture Notes in Computer Science, v. 6394), p. 392 – 406.

72 EASTERBROOK, S. et al. Selecting Empirical Methods for Software Engineering Research. **Guide to Advanced Empirical Software Engineering Springer**, Springer, v. 16, p. 285–311, 2007.

73 SHAW, M. Writing Good Software Engineering Research Papers: Minitutorial. In: **Proceedings of the 25th International Conference on Software Engineering**. Washington, DC, USA: IEEE Computer Society, 2003. (ICSE 03), p. 726 – 736.

74 BOOTH, W. C.; COLOMB, G. G.; WILLIAMS, J. M. **The Craft of Research.** 3rd. ed. Chicago: University of Chicago Press, 2008. 1 - 336 p. (Chicago Guides to Writing, Editing, and Publishing).

75 SHAW, M. What Makes Good Research in Software Engineering? **International Journal on Software Tools for Technology Transfer**, v. 4, n. 1, p. 1 – 7, 2002.

76 DYBÅ, T. et al. Qualitative Research in Software Engineering. **Empirical Softw. Engg.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 16, n. 4, p. 425–429, 2011.

77 DAVY, D.; VALECILLOS, C. Summary of a Literature Review of Qualitative Research in Technical Communication from 2003 to 2007. **International Professional Communication Conference**, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 1–7, 2009.

78 WOODSIDE, A. G.; WILSON, E. J. Case Study Research Methods for Theory Building. **Journal of Business & Industrial Marketing.**, v. 18, n. 6/7, p. 493 – 508, 2003.

79 RUNESON, P.; HöST, M. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. **Empirical Software Engineering: An International Journal**, Kluwer Academic Publishers Hingham, MA, USA, Hingham, MA, USA, v. 14, n. 2, p. 131–164, 2009.

80 YIN, R. K. **Case Study Research: Design and Methods (Applied Social Research Methods).** 5th. ed. [S.l.]: Sage Publications, 2015. 1 - 312 p.

81 ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering. **ISO/IEC/IEEE 29148:2011(E)**, p. 1–94, 2011.

82 RIBEIRO, F. G. C.; SOARES, M. S. An Approach for Modeling Real-Time Requirements with SysML and MARTE Stereotypes. In: **15th International Conference on Enterprise Information Systems (ICEIS)**. [S.l.: s.n.], 2013.

83 _____. A Metamodel for Tracing Requirements of Real-Time Systems. In: **16th IEEE Computer Society Symposium on Object/Service-Oriented RealTime Distributed Computing (ISORC)**. [S.l.: s.n.], 2013.

84 RIBEIRO, F. G. C.; MISRA, S.; SOARES, M. S. Application of an Extended SysML Requirements Diagram to Model Real-Time Control Systems. In: _____. **13th International Conference in Computational Science and Its Applications (ICCSA)**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 70 – 81.

85 RIBEIRO, F. G. C. **Modelagem de requisitos de software de tempo-real usando SysML e MARTE.** Dissertação (Master Thesis) — Universidade Federal de Uberlândia, Uberlândia, 2013.

86 HERRERA, F.; MEDINA, J.; VILLAR, E. Modeling Hardware/Software Embedded Systems with UML/MARTE: A Single-Source Design Approach. In: _____. **Handbook of Hardware/Software Codesign**. [S.l.]: Springer Netherlands, 2017. p. 141–185.

87 QUADRI, I. R. et al. MADES: A SysML/MARTE High Level Methodology for Real-Time and Embedded Systems. In: **Proceedings of the 10th Embedded Realtime Software and Systems Conference**. [S.l.: s.n.], 2012. p. 1 – 10.

88 MARQUES, M. R. S.; SIEGERT, E.; BRISOLARA, L. Integrating UML, MARTE and SysML to Improve Requirements Specification and Traceability in the Embedded Domain. In: **12th IEEE International Conference on Industrial Informatics (INDIN)**. Porto Alegre, RS, Brazil: IEEE, 2014. p. 176 – 181.

89 HERRERA, F. et al. The COMPLEX Methodology for UML/MARTE Modeling and Design Space Exploration of Embedded Systems. **Journal of Systems Architecture - Embedded Systems Design**, Elsevier North-Holland, Inc., New York, NY, USA, v. 60, n. 1, p. 55 – 78, 2014.

90 KHAN, A.; MALLET, F.; RASHID, M. A framework to specify system requirements using natural interpretation of UML/MARTE diagrams. **Software & Systems Modeling**, Springer Verlag, 2017.

91 CHEN, B.; X., L.; ZHOU, X. Model Checking of MARTE/CCSL Time Behaviors using Timed I/O Automata. **Journal of Systems Architecture**, v. 88, p. 120 – 125, 2018.

92 RIBEIRO, F. G. C. et al. Model-based requirements Specification of Real-Time Systems with UML, SysML and MARTE. **Software & Systems Modeling**, p. 1 – 19, 2016.

93 _____. Applying MARTE Profile for Optimal Automotive System Specifications and Design. In: **50th Hawaii International Conference on System Sciences (HICSS)**. Waikiloa Village, Hawaii: [s.n.], 2017.

94 _____. Annotating SysML Models with MARTE Time Stereotypes for Requirements Specification and Design of Real-Time Systems. In: **7th IEEE Workshop on Self-Organizing Real-Time Systems**. York, UK.: [s.n.], 2016.

95 _____. A Model-based Engineering Methodology for Requirements and Formal Design of Embedded and Real-Time Systems. In: **50th Hawaii International Conference on System Sciences (HICSS)**. Waikiloa Village, Hawaii: [s.n.], 2017.

96 RIBEIRO, Q. A. D.; RIBEIRO, F. G. C.; SOARES, M. S. A Technique to Architect Real-time Embedded Systems with SysML and UML through Multiple Views. In: INSTICC. **Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 2: ICEIS**. [S.l.]: ScitePress, 2017. p. 287–294.

97 RIBEIRO, F. G. C. Multi-Formalism in Different Levels of Abstraction for Requirements Engineering and Design of Real-Time Systems. In: **PhD Forum at Design, Automation and Test in Europe (DATE)**. [S.l.: s.n.], 2018.

98 RIBEIRO, F. G. C. et al. An Analysis of the Value Specification Language Applied to the Requirements Engineering Process of Cyber-Physical Systems. **IFAC-PapersOnLine**, v. 49, n. 30, p. 42 – 47, 2016. 4th {IFAC} Symposium on Telematics Applications {TA}.

99 _____. An Approach to Formalization of Architectural Viewpoints Design in Real-Time and Embedded Domain. In: **21th IEEE Computer Society Symposium on Object/Service-Oriented Real-Time Distributed Computing (ISORC)**. [S.l.: s.n.], 2018.

100 _____. Guidelines for Using MARTE Profile Packages Considering Concerns of Real-Time Embedded Systems. In: **15th IEEE International Conference on Industrial Informatics, INDIN**. [S.l.: s.n.], 2017. p. 917–922.

101 _____. SPES methodology and MARTE constraints in architectural design. In: **ISCC**. [S.l.]: IEEE, 2018. p. 377–383.

102 _____. Model-based Design Methodology for Early Evaluation of Real-time and Embedded Constraints. In: **INDIN**. [S.l.]: IEEE, 2018. p. 875–881.

103 _____. An Approach for Architectural Design of Automotive Systems using MARTE and SysML. In: **CASE**. [S.l.]: IEEE, 2018. p. 1574–1580.

104 _____. Non-Functional Constraints Annotation to Real-Time and Embedded System Design. In: **SBESC**. [S.l.]: IEEE, 2018.

105 KOPETZ, H. **Real-Time Systems: Design Principles for Distributed Embedded Applications.** 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2011.

106 BURNS, A.; WELLINGS, A. **Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX.** 4th. ed. USA: Addison-Wesley, 2009.

107 WELCH, L. et al. Specification and modeling of dynamic, distributed real-time systems. In: **19th IEEE Real-Time Systems Symposium**. [S.l.: s.n.], 1998. p. 72–81.

108 LEE, D. T. Evaluating Real-Time Software Specification Languages. **Computer Standards & Interfaces**, v. 24, n. 5, p. 395–409, 2002.

109 PEREIRA, T. et al. Towards a Metamodel for a Requirements Engineering Process of Embedded Systems. **2016 VI Brazilian Symposium on Computing Systems Engineering**, p. 93–100, 2016.

110 GAMBIER, A. Real-time Control Systems: A Tutorial. In: **5th Asian Control Conference (IEEE Cat. No.04EX904)**. [S.l.: s.n.], 2004. v. 2, p. 1024–1031.

111 PEREIRA, C. E. On Describing Timing Requirements within a Real Time Object Oriented Requirements Engineering Phase. In: **Proceedings of the Workshop 4: Object Oriented RealTime Systems Analysis and Design Issues**. [S.l.: s.n.], 1993. p. 993–998.

112 BURNS, A.; WELLINGS, A. J. **Real-Time Systems and their Programming Languages: ADA 95, Real-Time Java, ad Real-Time POSIX.** 3th. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., 2001. I-XVI, 1-611 p.

113 FREITAS, E. P. et al. DERAF: A High-Level Aspects Framework for Distributed Embedded Real-Time Systems Design. In: **Proc. of the 10th International Workshop on Early Aspects**. [S.l.]: Springer, 2007. (Lecture Notes in Computer Science, v. 4765/2), p. 55–74.

114 BRAMBILLA, M.; CABOT, J.; WIMMER, M. **Model-driven Software Engineering in Practice: Second Edition.** 2nd. ed. [S.l.]: Morgan & Claypool Publishers, 2017.

115 WHITTLE, J.; HUTCHINSON, J.; ROUNCEFIELD, M. The State of Practice in Model-driven Engineering. **IEEE Software**, v. 31, n. 3, p. 79–85, 2014.

116 STRAETEN, R. V. D.; MENS, T.; BAELEN, S. Challenges in Model-driven Software Engineering. In: CHAUDRON, M. R. V. (Ed.). **Models in Software Engineering**. [S.l.]: Springer Berlin Heidelberg, 2009. p. 35–47.

117 ESTEFAN, J. **Survey of Candidate Model-based Systems Engineering (MBSE) Methodologies.** [S.l.], 2008.

118 FOUAD, A. et al. Embedding requirements within Model-driven Architecture. **Software Quality Journal**, v. 19, n. 2, p. 411–430, 2011.

119 AGNER, L. T. W. et al. A Brazilian Survey on UML and Model-driven Practices for Embedded Software Development. **J. Syst. Softw.**, Elsevier Science Inc., v. 86, n. 4, p. 997–1005, 2013.

120 SOMMERVILLE, I. **Engenharia de Software.** 8th. ed. São Paulo: Pearson, 2011.

121 MILLER, J.; MUKERJI, J. **MDA Guide Version 1.0.1**. [S.l.], 2003.

122 WEHRMEISTER, M. A. **An Aspect-Oriented Model-driven Engineering Approach for Distributed Embedded Real-Time Systems**. Tese (Thesis) — Universidade Federal do Rio Grande do Sul, Porto Alegre, Rio Grande do Sul, 2009.

123 IYENGHAR, P.; NOYER, A.; PULVERMUELLER, E. Early Model-driven Timing Validation of IoT-compliant Use Cases. In: **IEEE 15th International Conference on Industrial Informatics (INDIN)**. [S.l.: s.n.], 2017. p. 19–25.

124 BOOCH, G. et al. **Object-Oriented Analysis and Design with Applications.** 3th. ed. Redwood City, CA, USA: Addison - Wesley Professional, 2007.

125 WEILKIENS, T. **Systems Engineering with SysML/UML: Modeling, Analysis, Design.** 1th. ed. San Francisco, USA: Morgan Kaufmann, 2008.

126 DAUN, M.; TENBERGEN, B.; WEYER, T. Requirements Viewpoint. In: _____. **Model-based Engineering of Embedded Systems: The SPES 2020 Methodology**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 51–68.

127 HOLTMANN, J. et al. Integrated Systems Engineering and Software Requirements Engineering for Technical Systems. In: **Proceedings of the 2015 International Conference on Software and System Process**. [S.l.: s.n.], 2015. (ICSSP 2015), p. 57–66.

128 PETERS, J. et al. A Generic Representation of CCSL Time Constraints for UML/MARTE Models. In: **52nd ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2015. p. 1–6.

129 AMYOT, D. et al. Towards Improved Requirements Engineering with SysML and the User Requirements Notation. **2016 IEEE 24th International Requirements Engineering Conference (RE)**, p. 329–334, 2016.

130 WAYKAR, Y. Importance of UML Diagrams in Software Development. In: **Managelization**. [S.l.: s.n.], 2013.

131 BRIAND, L. C.; LABICHE, Y.; SULLIVAN, Y. O. Impact Analysis and Change Management of UML Models. **Proceedings of the International Conference on Software Maintenance (ICSM)**, p. 256–265, 2003.

132 SELIC, B. A Systematic Approach to Domain-Specific Language Design Using UML. In: **10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 07)**. [S.l.: s.n.], 2007. p. 2 – 9.

133 CHOI, J.; BAE, D. H. An Approach to Constructing Timing Diagrams from UML/MARTE Behavioral Models for Guidance and Control Unit Software. In: **Computer Applications for Database, Education, and Ubiquitous Computing - International Conferences**. [S.l.: s.n.], 2012. p. 107 – 110.

134 CHOI, J.; JEE, E.; BAE, D. H. Timing Consistency Checking for UML/MARTE Behavioral Models. **Software Quality Journal**, v. 24, n. 3, p. 835 – 876, 2016.

135 BENNETT, A. J.; FIELD, A. J. Performance Engineering with the UML Profile for Schedulability, Performance and Time: a Case Study. In: **12th Annual International Symposium on the IEEE Computer Society's**. Los Alamitos, CA, USA: IEEE Computer Society, 2004. p. 67 – 75.

136 BEHRMANN, G.; DAVID, A.; LARSEN, K. G. **A Tutorial on Uppaal 4.0**. 2006.

137 BENGTSSON, J.; YI, W. Timed Automata: Semantics, Algorithms and Tools. In: . [S.l.]: Springer-Verlag, 2004. p. 87–124.

138 UPPAAL, T. Uppaal: http://www.uppaal.org/. Acess em 23.11.2018. 2018.

139 OLIVEIRA, K. S. **Aspectos Iniciais Modelados com uma Extensão da SysML.** Dissertação (Master Thesis) — Universidade Federal de Uberlândia, Uberlândia, 2013.

140 DELLIGATTI, L. **SysML Distilled: A Brief Guide to the Systems Modeling Language.** 1st. ed. [S.l.]: Addison Wesley Professional, 2013.

141 KRUCHTEN, P.; OBBINK, H.; STAFFORD, J. The Past, Present, and Future for Software Architecture. **IEEE Softw.**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 23, n. 2, p. 22–30, 2006.

142 EDER, S.; MUND, J.; VOGELSANG, A. Logical Viewpoint. In: _____. **Model-based Engineering of Embedded Systems: The SPES 2020 Methodology**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 85–93.

143 MALLET, F.; SIMONE, R. Correctness Issues on MARTE/CCSL Constraints. **Science of Computer Programming**, Elsevier, v. 106, p. 78 – 92, 2015.

144 GRASSET, A. Design of Critical Embedded Systems: from Early Specifications to Prototypes. In: **International Symposium on Rapid System Prototyping (RSP)**. [S.l.: s.n.], 2015. p. 38–38.

145 ALBINET, A. et al. The MeMVaTEx Methodology: From Requirements to Models in Automotive Application Design. In: **5th European Congress ERTS Embedded Real Time Software**. [S.l.: s.n.], 2010. p. 1–10.

146 ITEA, E. **The EAST-ADL Architecture Description Language.** [S.l.: s.n.], 2013. Version M2.1.12.

147 AUTOSAR. **AUTOSAR AUTomotive Open System Architecture.** [S.l.: s.n.], 2015. Release R4.2.

148 SAADATMAND, M.; CICCHETTI, A.; SJÖDIN, M. UML-based Modeling of Non-Functional Requirements in Telecommunication Systems. In: **The Sixth International Conference on Software Engineering Advances (ICSEA 2011)**. Barcelona, Spain: [s.n.], 2011. p. 213 – 220.

149 GAMATIÉ, A. et al. A Model-driven Design Framework for Massively Parallel Embedded Systems. **ACM Trans. Embedded Comput. Syst.**, v. 10, 2011.

150 GOMEZ, C.; DEANTONI, J.; MALLET, F. Multi-View Power Modeling Based on UML, MARTE and SysML. In: **EUROMICRO-SEAA**. Cesme, Izmir, Turkey: IEEE Computer Society, 2012. p. 17 – 20.

151 ULBRICH, P. et al. Using MARTE in Code-centric Real-time Projects Providing Evolution Support. In: **Proceedings of the First Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2010)**. [S.l.: s.n.], 2014. p. 25–29.

152 YUNCHENG, S. Research on Modeling and Design of Real-Time Embedded Systems. In: **7th International Conference on Intelligent Computation Technology and Automation**. [S.l.: s.n.], 2014. p. 547–550.

153 QUADRI, I. R.; BAGNATO, A.; SADOVYKH, A. MADES EU FP7 Project: Model-driven Methodology for Real Time Embedded Systems. In: _____. **Embedded and Real Time System Development: A Software Engineering Perspective: Concepts, Methods and Principles**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. p. 57–89.

154 QUADRI, I. R.; BAGNATO, A.; A., S. MADES FP7 EU Project: Effective High Level SysML/MARTE Methodology for Real-Time and Embedded Avionics Systems. In: **7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)**. [S.l.: s.n.], 2012. p. 1 – 8.

155 MARQUES, M. R. S.; SIEGERT, E.; BRISOLARA, L. B. Uma Abordagem para Engenharia de Requisitos Baseada em Modelos no Domínio de Software Embarcado. In: **Workshop em Engenharia de Requisitos, Montevideo, Uruguay, April 8-10, 2013**. [S.l.: s.n.], 2013.

156 IQBAL, M. Z. et al. Applying UML/MARTE on Industrial Projects: Challenges, Experiences, and Guidelines. **Software & Systems Modeling**, v. 14, n. 4, p. 1367–1385, 2015.

157 EBEID, E. et al. Extensions to the UML Profile for MARTE for Distributed Embedded Systems. In: **Forum on Specification and Design Languages (FDL)**. [S.l.: s.n.], 2015. p. 1–8.

158 OUHAMMOU, Y.; GROLLEAU, E.; RICHARD, P. Extension and Utilization of a Design Framework to Model Integrated Modular Avionic Architecture. In: _____. [S.l.: s.n.], 2015. p. 16–27.

159 FERNáNDEZ, D. M.; PENZENSTADLER, B. Artefact-based Requirements Engineering: The AMDiRE Approach. **Requirements Engineering.**, Springer-Verlag New York, Inc., v. 20, n. 4, p. 405–434, 2015.

160 KAHANI, N. AutoModel: a Domain-specific Language for Automatic Modeling of Real-time Embedded Systems. In: **Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings**. [S.l.: s.n.], 2018. p. 515–517.

161 BUCAIONI, A. et al. MoVES: A Model-driven Methodology for Vehicular Embedded Systems. **IEEE Access**, v. 6, p. 6424–6445, 2018.

162 OUHAMMOU, Y.; GROLLEAU, E.; RICHARD, P. From Model-based Design to Real-time Analysis. p. 45–50, 2012.

163 WEBER, R. et al. Technical Viewpoint. In: _____. **Model-based Engineering of Embedded Systems: The SPES 2020 Methodology**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 95–106.

164 KAZMAN, R.; BURTH, M. Assessing Architectural Complexity. In: **CSMR**. [S.l.: s.n.], 1998.

165 BASHIR, H.; THOMSON, V. Estimating Design Complexity. **Journal of Engineering Design**, v. 10, p. 247–257, 1999.

166 KINNUNEN, M. J. **Complexity Measures for System Architecture Models.** Dissertação (Mestrado), 2006.

167 KREIMEYER, M.; LINDEMANN, U. **Complexity Metrics in Engineering Design.** [S.l.]: Springer-Verlag Berlin Heidelberg, 2011.

168 CARD, D. N.; AGRESTI, W. W. Measuring Software Design Complexity. **Journal of Systems and Software**, v. 8, n. 3, p. 185 – 197, 1988.

169 ROSSI, M.; BRINKKEMPER, S. Complexity Metrics for Systems Development Methods and Techniques. **Information Systems**, v. 21, n. 2, p. 209 – 227, 1996.

170 BANDI, R. K.; VAISHNAVI, V. K.; TURK, D. E. Predicting Maintenance Performance using Object-Oriented Design Complexity Metrics. **IEEE Transactions on Software Engineering**, v. 29, n. 1, p. 77–87, 2003.

171 TOMIYAMA, T. et al. Complexity of Multi-Disciplinary Design. **Annals of the CIRP**, v. 56, n. 1, 2007.

172 PANDEY, R. K. Managing software design complexity. **ACM SIGSOFT Software Engineering Notes**, v. 34, n. 1, p. 1, 2009.

173 KOPETZ, H. The Complexity Challenge in Embedded System Design. In: IEEE. **11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)**. [S.l.], 2008. p. 3–12.

174 STEVANETIC, S.; ZDUN, U. Software Metrics for Measuring the Understandability of Architectural Structures: A Systematic Mapping Study. In: **Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering**. [S.l.]: ACM, 2015.

175 LOUCOPOULOS, P.; KARAKOSTAS, V. **System Requirements Engineering.** New York, NY, USA: McGraw-Hill, Inc., 1995.

176 PAPYRUS, T. Papyrus: www.eclipse.org/papyrus/. Acess em 27.11.2018. 2018.

177 PARVIAINEN, P.; TIHINEN, M.; VANSOLINGEN, R. Requirements Engineering: Dealing with the Complexity of Sociotechnical Systems Development. In: MATE, J. L.; SILVA, A. (Ed.). **Requirements Engineering for Sociotechnical Systems**. [S.l.]: IdeaGroup Inc., 2004. Chapter 1.

178 MATE, J. L.; SILVA, A. **Requirements Engineering for Sociotechnical Systems.** 1rd. ed. [S.l.]: IdeaGroup Inc., 2005. 1 - 375 p.

179 HATLEY, D. J.; PIRBHAI, I. A. **Strategies for Real-time System Specification**. [S.l.]: Dorset House Publishing Co., Inc., 1987.

180 KNUTH, D. E. **The Art of Computer Programming: Fundamental Algorithms**. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

181 BARROSO, A. S. et al. An Evaluation of Influence of Human Personality in Software Development: An Experience Report. In: **8th Euro American Conference on Telematics and Information Systems**. [S.l.: s.n.], 2016. p. 1–6.

182 _____. Influence of Human Personality in Software Engineering - A Systematic Literature Review. In: **ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems**. [S.l.: s.n.], 2017. p. 53–62.

183 NAVET, N.; SIMONOT-LION, F. **Automotive Embedded Systems Handbook.** 1st. ed. Boca Raton, FL, USA: CRC Press, Inc., 2008.

184 BECKER, L. B. et al. MOSYS A Methodology for Automatic Object Identification from System Specification. In: **3rd International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC**. [S.l.: s.n.], 2000. p. 198–201.

185 LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. **J. ACM**, ACM, v. 20, n. 1, p. 46–61, jan. 1973. ISSN 0004-5411.

186 HAZZAN, O.; DUBINSKY, Y. Qualitative Research in Software Engineering. **System Desig Frontier**, v. 4, p. 6–17, 2007.

187 DRECHSLER, R. et al. Completeness-driven Development. In: **Graph Transformations**. [S.l.]: Springer Berlin Heidelberg, 2012. p. 38–50.

188 Ribeiro, F. G. C. and Pereira, C. E. and Rettberg, A. and Soares, M. S. An Approach for Architectural Design of Automotive Systems using MARTE and SysML. In: **14th International Conference on Automation Science and Engineering (CASE)**. [S.l.: s.n.], 2018. p. 1574–1580.

189 RIBEIRO, F. G. C. et al. Ein Modellierungsansatz für eine Systemarchitekturbeschreibung von Automotive-Systemen mit MARTE und SysML. **Automatisierungstechnik**, v. 67, p. 490–501, 2019.

190 RIBEIRO, F. G. C. A Methodology to Early Design and Evaluation of Real-Time Embedded Systems considering Non-Functional Constraints. In: **PhD Forum at 38th Design Automation Conference (DAC)**. [S.l.: s.n.], 2019.

191 RIBEIRO, F. G. C. et al. A Proposal to Trace and Maintain Real-time Embedded Constraints. In: **6th International Embedded Systems Symposium**. [S.l.]: Springer, 2019.

192 ARDUINO. Platform arduino, www.arduino.cc/. Acess em 27.08.2018. 2018.

193 FREERTOS. www.freertos.org/. Acess em 27.08.2018. 2018.

194 C++. http://www.cplusplus.com/doc/tutorial/. Acesso em 20.09.2018. 2018.

195 HARBOUR, M. Programming Real-Time Systems with C/C++ and POSIX. 2018.