Combining SysML and SystemC to Simulate and Verify Complex Systems

Abbas Abdulazeez Abdulhameed

SPIM

Thèse de Doctorat

UFC

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE FRANCHE-COMTÉ

N° | X | X | X |

Combining SysML and SystemC to Simulate and Verify Complex Systems

A dissertation presented by

ABBAS ABDULAZEEZ ABDULHAMEED

and submitted to the

Université de Franche-Comté

in partial fulfillment of the Requirements for obtaining the degree

DOCTOR OF PHYLOSOPHY

Speciality : Informatique

Defended in public on 04 March 2016 in front of the Jury composed from :

Emmanuel GROLLEAU	Rapporteur	Professeur, LISI, ISAE-ENSMA, Université de Poitiers
Vincent POIRRIEZ	Rapporteur	Professeur, LAMIH, Université de Valenciennes
Mamoun FILALI AMINE	Examinateur	Directeur de Recherche CNRS, IRIT, Université de Toulouse
Hassan MOUNTASSIR	Directeur de thèse	Professeur, DISC/Femto-ST, Université de Franche-Comté
Ahmed HAMMAD	Co-Encadrant de thèse	MDC, DISC/Femto-ST, Université de Franche-Comté
Bruno TATIBOUET	Co-Encadrant de thèse	MDC, DISC/Femto-ST, Université de Franche-Comté

UNIVERSITE DE FRANCHE-COMTÈ

UFC
UNIVERSITÉ
DE FRANCHE-COMTÉ

RAPPORT DE SOUTENANCE DE DOCTORAT
Spécialité : INFORMATIQUE

Nom et prénom du candidat : **ABDULHAMEED Abbas Abdulazeez**

Date et lieu de naissance : **14/07/1973 à Bagdad (Irak)**

Date de soutenance : **04 mars 2016** Heure : **14h00**

Nom du Président du jury : Vincent POIRRIEZ

M. Abbas Abdulazeez Abdulhameed a présenté son travail de façon claire. Il a montré sa connaissance d'un domaine très vaste tant théorique que technologique allant de l'ingénierie des modèles aux méthodes de vérification et de validation. Il a exposé ses contributions dans le processus de transformations des modèles d'exigences, exprimés en SysML, vers les modèles cibles : validation avec SystemC et vérification en logique *temporelle*.

Les réponses aux questions diverses et nombreuses ont été honnêtes. Le jury a apprécié la quantité de travail fourni et les compétences acquises durant la thèse.
En conséquence, le jury décerne le titre de docteur en Informatique de l'Université de Franche-Comté avec la mention très honorable à M. Abbas Abdulazeez Abdulhameed.

Noms et signatures des membres du jury

M. Emmanuel GROLLEAU (Professeur des universités) – Laboratoire d'Informatique et d'Automatique pour les Systèmes – Université de Poitiers Rapporteur	M. Vincent POIRRIEZ (Professeur des universités) – Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines – Université de Valenciennes Rapporteur
M. Mamoun FILALI AMINE – (Chargé de recherches, CNRS) – Institut de Recherche en Informatique de Toulouse – Université de Toulouse Membre	M. Hassan MOUNTASSIR (Professeur des universités) – Institut FEMTO-ST – Département Informatique et Systèmes Complexes – Université de Franche-Comté Membre
M. Ahmed HAMMAD (Maître de conférences) – Institut FEMTO-ST – Département Informatique et Systèmes Complexes – Université de Franche-Comté Membre	M. Bruno TATIBOUET (Maître de conférences) – Institut FEMTO-ST – Département Informatique et Systèmes Complexes – Université de Franche-Comté Membre

Le conseil scientifique de l'École Doctorale Sciences Pour l'Ingénieur et Microtechniques a décidé que les félicitations ne seraient plus attribuées.

Fait à : BESANÇON Le : 4 Mars 2016

Acknowledgments

Here I would like to thank everyone, whose help and participation made this thesis possible. First of all, I wish to express the deepest gratitude and respect to my thesis supervisor Dr. Hassan MOUNTASSIR and my co-supervisors Dr. Ahmed HAMMAD and Dr. Bruno TATIBOUET. Words are broken me to express my appreciation. Their skills, their scientific rigidity, and clairvoyance taught me a lot. Indeed, I thank them for their organizations, and expert advice provide me they knew throughout these four years and also for their warm quality human, and in particular for the confidence they have granted to me.

I would nevertheless like to thank more particularly proudly Prof. Messaoud Rahim, professor in the University of Yahia Fares Medea, for his advisers and his scientific expert who helped and guided me a lot for the completion of this thesis.

I extend my warmest thanks to the Minister of Higher Education and Scientific Research in Iraq represented by Campus France, the University of Mustansiriyah, and the University of Franche-Comté, for their co-operation to finance this thesis.
My gratitude and my thanks go to the crew members of VESONTIO for the friendly and warm atmosphere in which it supported me to work. Therefore thanks to Oscar Carrillo, Jean-Marie Gauthier, Pierre-Cyrille Heam, Hamida Bouaziz, Hana M'Hemdi, and to whom I missed their names.
My gratitude and my thanks go to the crew members of DISC for the friendly and warm atmosphere in which it supported me to work. Therefore, Thanks Olga Kouchnarenko, director of DISC (Informatique des systemes complexes) department in Besancon, Dominique Menetrier, Jean-Michel Caricand, Laurent Steck and to all other people if I forget his name. For their unwavering support and encouragement.

Finally, I want to thank my dear friends including Bassam Alkindy, Huda Al-nayyef, Bashar Al-Nauimi, Lemia Louail, who shared my hopes and studies, which made me comfort in the difficult moments and with whom I shared unforgettable moments of events.

Dedication

To my wife Yusra, with my love that gave me strength to go through the most difficult moments of my studies.
I am also addressing the strongest thanksgiving words to my parents, my sisters and their families, my brothers and their families, and to my lovely family, for their support and encouragement during the thesis in long years of studies. Their affection and trust lead me and guide me every day.
Thank you, Mom, Dad, for making me what I am today.

List of Acronyms

BDD	Block Definition Diagram.
EMF	Eclipse Modeling Framework.
MBSE	Model-Based Systems Engineering.
IBD	Internal Block Diagram.
INCOSE	International Council on Systems Engineering.
OMG	Object Management Group.
PD	Parametric Diagram.
RD	Requirement Diagrams.
SCV	SystemC Verification Standard.
SCNSL	SystemC Network Simulation Library.
SE	Systems Engineering.
SMD	State Machine Diagram.
SPIN	Simple Promela Interpreter.
SysML	Systems Modeling Language.
UML	Unified Modeling Language.

Contents

I

Context, Motivations and Related Works

1

Introduction

Contents

Nowadays, the use of models to design complex industrial systems increases continually. These models help in verifying the correctness of applications, in the earlier phases, by analyzing the possible failures and risks.

Systems Engineering (SE) [Blanchard et al., 1990], is an interdisciplinary field that emerged as an efficient way to manage the development complexity. The most prominent of the model is the application of accurate visual modelling systems and best practices to SE activities. Throughout the System Development Life Cycle (SDLC), SE activities include requirements analysis, verification, functional analysis, allocations, performance analysis, and system architecture specification. Model-Based Systems Engineering (MBSE) [Wymore, 1993], is a methodology for designing systems using computer models. MBSE focuses on defining customer needs and required functionality early in the development cycle. A model specification proceeds with design synthesis and system validation, while it considered the entire problem, including functional and non-functional requirements, verification, tests, and costs. Over the last few decades, MBSE has become especially important in the analysis and synthesis of critical and complex systems.

The Systems Modelling Language (SysML) [OMG, 2012], was proposed by the International Council on Systems Engineering (INCOSE)[1], and was standardized by the Object Management Group (OMG). SysML is a graphical modelling language that allows modelling to specify and design complex systems. It models the software side of such systems as well as their hardware side. SysML is well suited for describing the system at early design phases, and this is due to its simplicity and its graphical notations. However, SysML is a semi-formal language that lacks the formal semantic to support validation and verification techniques.

In a model-based development process, verification and validation techniques are supported to better understand models and to evaluate model properties that are stated implicitly in the model. The verification and validation processes assist the correct specifi-

[1] http://www.incose.org/

cation of models and exploring other modelling alternatives.
However, a specification using SysML in the process of verification and validation is missing. Many technologies and methods that are used in verification and validation stages of systems do already exist and have been standardized, from formal verification, simulation, and/or with testing techniques [Kleijnen, 1995], [Debbabi et al., 2010].

SystemC [Aynsley, 2006], is an open-source system-level design language based on C++ that has its own simulation kernel. By combining SysML with SystemC, we benefit from SysML for describing complex systems and from the simulation capabilities provided by the SystemC environment. This can be achieved by transformation of SysML diagrams into SystemC models.

Simulation is a conventional technique for the analysis of specifications of complex systems [Kelton et al., 2000]. Simulation-based approaches ensure that a finite number of user-defined system paths meet the wanted specification. It is approximately inexpensive regarding execution time, but it only validates the behaviour of a system for one particular computation path. For that, simulation is a technique that enables to verify some specific execution traces of a system.

Model-checking [Clarke et al., 1999] is an automated technique that, given a finite-state model of a system and formal properties, systematically checks whether this property holds in that model. Providing a methodology for formal verification of SysML specifications with model-checking based techniques is very appropriate for ensuring the validity of the designed system, fundamentally regarding their behavioural requirements.

In this thesis, we propose combining contributions to validate SysML specifications using simulation and verification techniques. This chapter is organized as follows, Section 1.1 presents the motivations of this thesis. Then, Section 1.2 describes the problem statement. Section 1.3 lists the objectives of this dissertation. Later, Section 1.4 and 1.5 provides a general overview of the proposed approach and describes the main contributions of this thesis. Finally, Section 1.6 presents the thesis overview.

1.1/ Motivations

The correct design of complex and critical systems remains challenging for the engineers. Bugs in a design that are not detected in early design stages can be extremely expensive. Therefore, the development of complex and critical systems requires verification and validation techniques to enable the detection and the correction of system models. For every modelling stage, we need to make sure that the system is implemented in conformity with the initial requirements.

Formal methods are techniques that are used to model complex systems as mathematical objects. Building an accurate mathematical model of a complex system, it is practicable to verify the system properties in a more exact style than practical testing[Bodeveix et al., 2005]. Our aim is the combination of formal methods with engineering application in such a way that all development, reasoning and analysis is an application of formal methods. Formal techniques can be applied to design verified models, but there is no clear reason.

Therefore, formal methods must be applicable exclusively to the evolution of information systems. Verification techniques are proposed to verify and validate the system designs.

However, formal descriptions can be difficult to construct, to read and to understand, especially for non-experts. SysML is a semi-formal language, which appears to offer an interesting compromise, especially when we provide for this latter a methodology for verifying and validating systems designed with it.

In this thesis, we use SysML, SystemC and model-checking techniques. SysML to target systems described in a high level of abstraction. SystemC environment to validate properties by simulation. And model-checking to verify system properties with formal methods.

SysML is a modelling language that permits to obtain a specification of a complex systems including requirement, structural and behavioural parts.

SystemC language is a preferred alternative to the traditional languages and its simulation kernel is an important aspect designed to evaluate the system behaviours through simulations.

The model-checking technique enabled to verify properties over a formal model by ensuring satisfy every possible state of the system. Therefore, its applied combination with transformation function requirements to verify and validate complex systems specified by SysML model. In fact, the increasing importance of critical applications has stimulated research work on modelling techniques that combine SysML with model-checking tools, such as SPIN (Simple Promela Interpreter) [Holzmann, 1997] and UPPAAL [Larsen et al., 1997].

1.2/ Problem Description

To design complex systems, we need a methodology supported by a high-level language for specifying and modelling the different aspects of the system including requirements, structure, and behaviour.

SysML is a UML profile for specifying and modelling complex systems, and formal semantics to provide verification and validation of systems design. To combine verification and validation with SysML, the design phases of the development of complex systems constitutes the main issue of this thesis. Verification and validation of SysML specifications need to adopt well-established verification techniques such as formal verification and simulation. The solutions for the problematic of thesis consists in combining simulation and formal verification methods with SysML.

- How to identify, semi-formalize, and structure informal complex system requirements using SysML models?
- How SysML can be combined with SystemC, to validate non-functional requirements of the complex systems?
- How SysML can be combined with model-checking to provide formal verification of system functional requirements?

1.3/ Objectives of thesis

This work intends to support the application of the MBSE paradigm in the sub-areas of model-based design, as well as model-based verification and validation. It combines SysML specifications with SystemC to simulate modeled behaviors. Moreover, SysML is combined with model-checking methods to apply formal verification and validation techniques. The strategy developed in this work enables to create system design models from semi-formal specifications like SysML, which is converted and implemented in a formal environment. We use formal verification to check requirements of the complex system model.

The verification and validation process is done by simulation for non-functional properties and by formal verification for ensuring the satisfaction of functional requirements. Based on various definitions available in this thesis, we define the process of designing and creating a system model specification that can be simulated and formally verified. Through specific conditions we can evaluate the behaviour of the corresponding real system.

In this thesis, we define the set of activities related to the specification and the verification approach used to assess the correction of complex systems design. The improvements, brought by SysML, have allowed increasing its popularity in the industrial and academic environment. A SysML specification of a system is represented by Requirement Diagrams, Behavioral Diagrams, which describe the dynamic operation of the system such as the State Machine Diagram (SMD), and Structural diagrams describe the system in static mode. In the later, blocks are modeled by three diagrams, the Block Definition Diagram (BDD), which defines the architecture of the blocks and their performed operations, and the Internal Block Diagram (IBD) with the Parametric Diagram (PD) used to define the ports of each block and connectors between them linking their ports.

The design refinement of a system is an important concept in SysML. While it is based on a development method that can begin from an abstract level, towards more detailed levels that can end in an implementation from SystemC environment to simulation or from model-checking to verification.

1.4/ Contributions

In this thesis, we propose contributions that aim to design, verify and validate complex systems. These contributions enable to answer the challenges presented in the above section. We present in Figure 1.1 a diagram representing our contributions (numbered as 1, 2, 3, and 4) to SysML specifications, SystemC environment and model-checking.

The first one is oriented towards the definition of a SysML-based approach by analyzing requirements, it identifies the concepts and the type of use that such language has to support. Such requirements are addressed by the model for requirements and specifications. We create a formal model of the problem domain, which describes the behavioral and structural characteristics of the environment where the problem is discovered.

Premier, we have to create the SysML diagrams for specifying the system structure, requirement, and behavior. Then, the SysML diagrams are mapped into SystemC modules. We execute the simulation with SystemC code in order to obtain the trace of behavior.

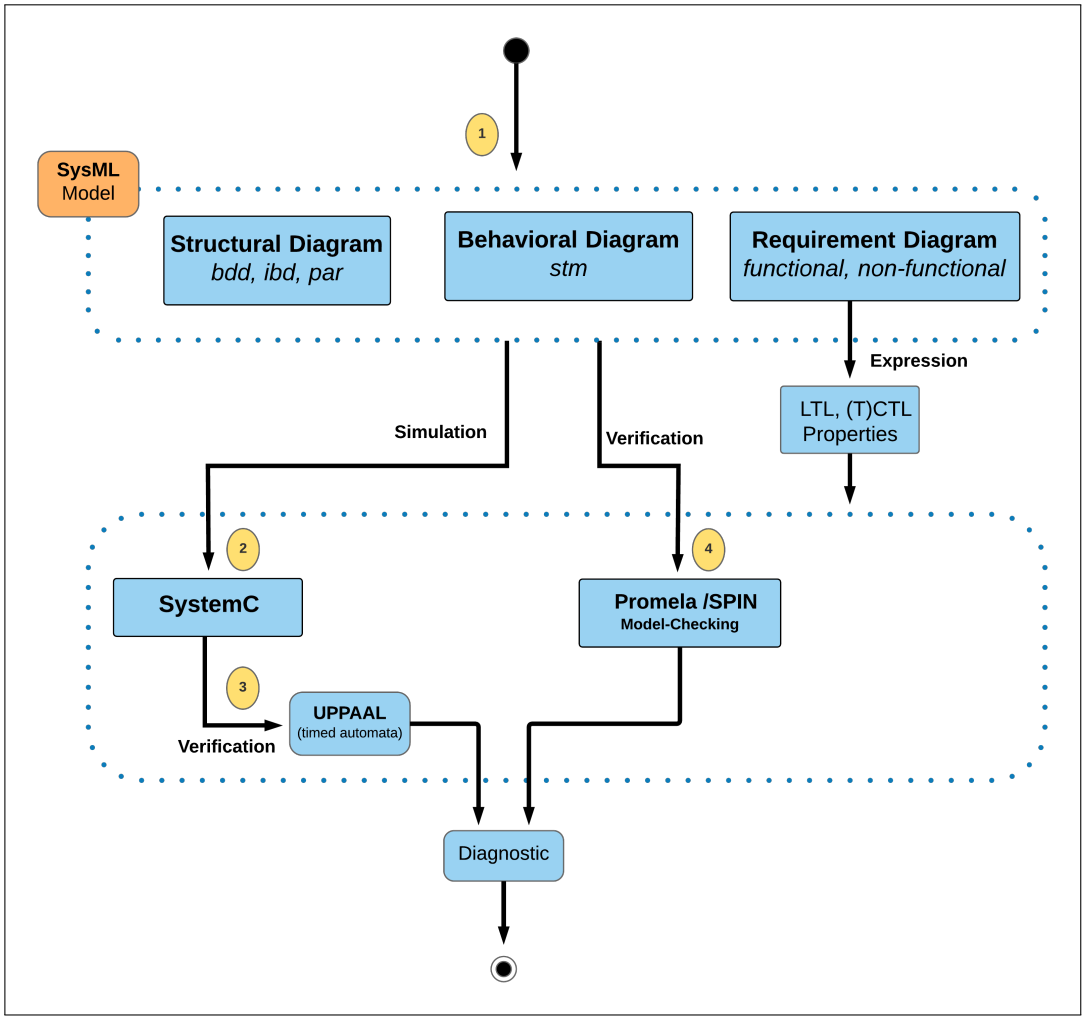

Figure 1.1: The proposed of approach

This trace is then analyzed to validate the system. Then, the derived SystemC specification is mapped into model-checking UPPAAL. Properties to verify by model-checking are derived from SysML Requirements Diagram.

Starting from these basic concepts, we propose an approach that consists in using particular SysML diagrams and elements for describing different aspects of the requirements. According to the SysML specification, the language can also be combined with other systems to define properties that SysML alone cannot express. Since SysML lacks a support for expressing temporal properties expressed in Linear Temporal Logic (LTL) [Pnueli, 1977] or Time Computation Tree Logic (TCTL) [Alur et al., 1990], we need to combine it with a formal language for the specification and analysis of complex systems.

The significant role among the complex systems design team members as an unambiguous, precise, and concise specification. Syntax and semantics of SysML are semi-formal, and thus it gives us the opportunity to verify the performance principles formally at an early stage of the combined systems design process.

SystemC specifications are executable, and thus they can be validated by simulation for different scenarios using the C++ compiler. The specifications can be verified formally

using model-checking tools such as Promela/SPIN using non-time properties or UPPAAL using timed automata.

Manual or automated theorem proving of SysML models can be prepared such as LTL and (T)CTL. These formal verifications are increasing the confidence in the correctness, of the final created system. Moreover, SystemC environment can make validation cases which can be used for simulation.

In summary, the thesis contributions are as follows:

- We have defined a set of semantic and transformation syntax rules to combine SysML specification with SystemC.
- We have defined a set of semantic and transformation syntax rules to combine SysML specification with Promela/SPIN.
- We have defined a set of rules to specify transformation by graphical model-checking and developed an algorithm to generate SPIN or UPPAAL descriptions from SysML requirements (requirement diagrams).
- We have developed a SystemC model for the automatic generation of SystemC specifications to simulate and validate the system.
- We have applied our approach in a case study specification to validate its feasibility.

The general process of our approaches consists of several stages. In first place, the modelling with SysML diagrams that will be the source models for the transformation. Moreover, a combination of SysML diagrams with SystemC executable specification is presented. It describes a transformation from SysML structure diagrams and behaviour diagrams to SystemC code, based on XMI files and XSLT style sheets.

We transform SysML specifications Promela models to validate the designed systems by using the model-checker SPIN. The requirement properties are converted to LTL formula and verified with SPIN.

The major proposals can be summarized as follows:

- We propose an approach that consists in using particular SysML diagrams and elements for describing different aspects of the specification from functional or non-functional requirements of the problem domain. A traffic light system is taken as a reference case study and is used to demonstrate our practical application (see Contribution 1 on figure 1.1) [Berrani et al., 2013].
- We propose an approach to validate SysML specifications by simulation. In this approach, a sub-part of SysML diagrams is transformed into SystemC executable models, and then these are used to simulate the system properties to validate the satisfaction of its non-functional requirements (see Contribution 2 on figure 1.1) [Abdulhameed et al., 2014a].
- We compare the techniques and tools that may be used to provide verification of SystemC models. Moreover, we classify the existing methodologies according to their capabilities and integration with the SystemC environment, we used techniques to transformation SystemC model to Uppaal for verification (see Contribution 3 on figure 1.1) [Abdulhameed et al., 2014b].

- We propose an approach to validate SysML specifications using formal verification. In this approach, SysML state machine diagrams are transformed into Promela models, and then these are used to verify the functional requirements of the specified system. These requirements are expressed as properties in temporal logic formulas and then verified in SPIN model-checker (see Contribution 4 on figure 1.1) [Abdulhameed et al., 2015].

1.5/ Publications

The work presented in this thesis has been already published in national and international conferences. In the following, we list the references for the published articles:

- Abdulhameed, A., Hammad, A., Mountassir, H., & Tatibouet, B. (2014, January). An approach based on SysML and SystemC to simulate complex systems. In Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on (pp. 555-560). IEEE.
- Abdulhameed, A., Hammad, A., Mountassir, H., & Tatibouet, B. (2014, Juin). An Approach Combining Simulation and Verification for SysML using SystemC and Uppaal. In CAL 2014, 8éme conférence francophone sur les architectures logicielles, page 9 pages, Paris, France.
- Abdulhameed, A., Hammad, A., Mountassir, H., & Tatibouet, B. (2015, April). An approach to verify SysML functional requirements using Promela/SPIN. In Programming and Systems (ISPS), 2015 12th International Symposium on (pp. 1-9). IEEE.

1.6/ Outline of the thesis

The chapters of this thesis are organized as follows:

- In Chapters 2 and 3, we present a bibliographic review containing various proposals and methodologies for modelling and design of complex systems using SysML and SystemC. These chapters also present the fundamental concepts and definition for the development of the following chapters.
- In Chapter 4, we present a traffic light system, taken as a reference case study and used to demonstrate our practical application. We use particular SysML diagrams and elements for describing different aspects of the requirements specification.
- In Chapters 5 and 7, we show the application of the methodology and the advanced tool in some case studies, and an evaluation of the obtained results.
- In Chapter 6, we present a comparison between the techniques that can be used to provide verification of SystemC models. The SystemC design is transformed into Uppaal automata.
- In Chapter 8, we present the conclusion of our work and we outline the future works of the thesis.

2

Related Works

Contents

2.1/ Introduction

The objective of this chapter is to exhibit the related works of this dissertation. This chapter is divided into three parts modelling, validation, and verification of complex systems. First, we discuss the relevant work of different languages to specify complex systems. Second, we present the relevant work of validation by simulation of complex systems using SystemC. Third, we focus on works to verify SystemC designs using such environment and existing transformation processes to perform model-checking. Finally, we present a summary of the related work.

There are three levels of component specification:

- The informal specification describes the component in the natural language that needs to be transformed into code by an engineer.
- The semi-formal specification uses several descriptive languages or other representation. In this specification, a component behavior description is less ambiguous. The majority of specification standards today are semi-formal, as for instance SysML and UML.
- Formal specification permits to describe the behavior of a given computing system following some accurate approach. A specification is a model of a system that contains a description of its requirements and what needs to be implemented.

2.2/ Modelling of Complex Systems

In this part, we present challenges for integrating and evaluating the difference between semi-formal and formal specification techniques. In addition, we present several spec-

ification formalisms, similarities and differences, as well as possibilities for combining such techniques. The first critical aspect of component-based software development is the specification that describes the functionality and behavior of the component.

2.2.1/ Semi-Formal Languages

The OMG defines UML as follows: ``The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schema, and reusable software components''. According to [Rumbaugh et al., 1999], UML is a general graphical modelling language for the development of systems.

UML is the leading modelling language in the field of systems engineering and has gained much tool support. The tool support for UML influences positively its usability and readability, which enables the efficient building of systems. Despite the fact that UML is mostly used with object-oriented systems, it has the capability to develop other types of systems through its flexible extension mechanism or profiles.

SysML [OMG, 2012] is a graphical modelling language based on UML and was adopted by the OMG, It is a graphical modeling language for specifying, analyzing, and designing complex systems.

2.2.2/ Formal Languages

In [Mueller, 2013, Snook et al., 2006], formal specification is specified written in a formal language where a formal language is either a rigorous mathematical model or simply a standard programming or specification language. By particular application, formal specifications are idea execution by code evaluation. In all states, formal specifications are for mental performance by code review and for passing the specification around to members of a design team. In most states only subsets of formal specification languages, e.g. Z and Vienna Development Method (VDM) [Jackson, 1985], are machine executable. The formal method indicates the application of at least one formal specification language.

The term formal method tends to elicit the strong reaction from software engineers and computing academics. Either a response of enthusiasm extending to the claim the formal techniques will become the newly accepted of software engineering, or a reaction of doubt about the research bias of much formal methods work.

Formal methods are often used during system design when the degree of confidence in the prescribed system behavior, extrapolated from a finite number of the test is high. Formal methods are frequently applied in the design of complex concurrent or reactive systems.

In [Abrial et al., 1998], the B method/language is considered to be a combination of ``Pascal and Z'' with some extensions for refinement. The B method/language is based on a hierarchical stepwise refinement and decomposition of a problem. After initial informal specification of requirements, an abstraction is made to capture, in a first formal specification, essential properties of a system.

For instance, these could be the main safety properties in a safety critical system. This

high-level abstract specification is made more concrete and more detailed in steps. The specification can be refined either by modifying the data structures used to represent the state information and/or by changing the shapes of the operations that act upon these data structures.

In [Mason et al., 2004], the authors defined effective traceability in systems engineering environments. The issue is raised by the fact that engineers use a range of notations to describe complex systems. From natural language to graphical notations such as statecharts to languages with formal semantics such as SystemC, VDM-SL [Plat et al., 1992]. Most have tool support, although a shortage of well-defined integration approaches succeeds to inconsistencies and limits traceability between their corresponding datasets and internal models.

2.2.3/ Hardware Description Languages

In this section, we introduce the VHDL-AMS, Verilog-AMS, Modelica, and SystemC Hardware Description Languages. These languages are used in the industry to model complex systems. Moreover, the comparison between different hardware description languages concerning the abstraction level of the description.

SystemC [Aynsley, 2006], is an open-source system-level design language based on C++ that has its own simulation kernel. SystemC, as an object-oriented approach to modeling systems, is easily modifiable to augment its. The core language consists of macros for modeling fundamental components of hardware designs, such as modules and signals. SystemC also provides hardware-oriented data types like 4-valued logic and arbitrary-precision integers.

VHDL-AMS is the result of an IEEE effort to extend the VHDL language to support the modeling and the simulation of analog and mixed-signal systems. The effort culminated in 1999 with the release of the IEEE standard 1076.1-1999 [Shahdad, 1986]. Verilog-AMS, on the other hand, is intended to be an extension of the Verilog HDL language also to support the modeling and the simulation of analog and mixed-signal systems. Verilog is a digital HDL that has been released in 1995 as IEEE standard 1365-1995.

Modelica [Fritzson et al., 1998] is an object-oriented language for describing Differential Algebraic Equation (DAE) systems combined with discrete events. Such models are ideally suited for representing flows of energy, materials, signals, or other continuous interactions between system components.

MATLAB stands for MATrix LABoratory [Houcque, 2005], Matlab is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. Furthermore, MATLAB is a modern programming language environment, it has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB an excellent instrument for complex systems.

In [Bonanome, 2001], the authors present the comparison between different hardware description languages. In general, Verilog is better suited for structural designs, as it permits better control of modules within the same abstraction layers, even though it lacks basic hierarchy management. SystemC nature is behavioral, which can make it harder to synthesize than Verilog.

Verification process in larger designs is shorter for SystemC, as no C/C++ simulation needs

to be implemented, hence decreasing the time to market. In Figure 2.1, we see that SystemC allows engineers to design both hardware and software components. It is possible to focus on the actual functionality of the system more than on its implementation details. It allows consistent changes to the design, enabling an efficient evaluation of different architecture alternatives including the separation between the hardware and software implementation. SystemC is also characterized by a high-performance speed. Note that this high speed is not only due to the SystemC simulation engine itself, but it is also effected by the high abstraction level used for SystemC environments for system descriptions.

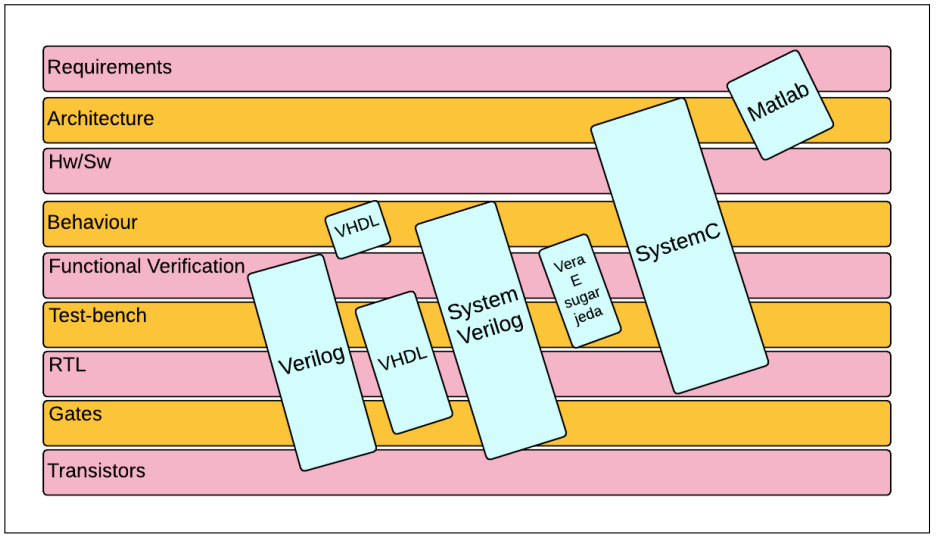

Figure 2.1: Hardware Description Languages and Abstraction Levels

2.3/ Validation of SysML Designs

In this section, we present related work about the combination of SysML with SystemC. The employment of the application of formal analysis techniques and particularly model-checking to SysML/ UML has been a very dynamic field of research in recent years.

In [Bhaduri et al., 2004], the authors give a comprehensive survey of research that applies model-checking to state machines, in which different model checkers including SMV, FDR [Leuschel et al., 2001] (Failures Divergences Refinement) and Promela/SPIN have been employed to achieve the verification of the properties of the designed systems. We additionally present related work concerning UML/SysML model translation to such languages.

In [Vanderperren et al., 2012], the authors defined a design methodology and development flow for the hardware, based on a UML4SystemC profile and encompassing different levels of abstraction. Both SystemC/C profiles are same groups of modeling constructs designed to lift the programming features including structural and behavioral characteristics of the two coding languages to the UML modeling level.

In [Prevostini et al., 2007], the authors present an approach required to some methodologies that will support to minimize the time and improve the quality of design. SysML-based designs are attracting since they permit to model complex systems using platform-

independent languages. Therefore, it is desirable to design a system without knowing which part of the system will be performed in hardware or software.

In [Jain et al., 2012, Riccobene et al., 2012], the authors propose SystemC profile, which is a consistent set of modeling constructs designed to lift both structural and behavioral attributes of the SystemC language to SysML level. It provides means for software and hardware engineers to improve the current industrial complex systems, design methodology joining the capabilities of SysML and SystemC to operate at system-level by including events and time attributes. The combination is based on a mapping from SysML to the SystemC for the structural and behavioral aspects. The refined co-design flowing starts from a SysML description at a high abstraction level of design. Moreover, it proceeds through a series of refined SystemC models to lower abstraction levels of design.

In [Mura et al., 2008, Espinoza et al., 2009], the SysML and Modeling and Analysis of Real Time and Embedded (MARTE) profiles are tested and compared. Both profiles are well suited for the description of embedded systems, although focusing on different aspects. Their goal is to use SysML to describe systems at a high level of abstraction and to provide diagrams for requirements specification.
There are more software tools not presented here that encapsulate the scope of SysML to SystemC code generation, such as:

- The Altova UModel [Scholtz et al., 2013], that designs application models and generates code and project documentation, and then refines designs and completes the round trip by regenerating the code. That makes visual software design practical for any project.
- The Enterprise Architect software [Nikiforova et al., 2012], supports advanced MDA transformations using easy to edit transform templates, with generation and reverse engineering of source code for SystemC language, this can quickly develop detailed solutions from abstract models.
- Artisan Studio [Bombino et al., 2012]. The software for all model driven development, they use models to communicate design decisions in SysML, with leveraging Automatic Code Synchronizer (ACS) and Transformation Development Kit.

2.3.1/ Verification of SysML Designs

The calling for the application of formal analysis techniques and particularly model-checking to SysML/UML specification is a very active field of study. The technical use of functional verification is mainly manual and empirical. The limits of these practices are being developed by the increase in complexity of the systems being validated. Another difficult challenge is posed by the need to keep the pace with continuously evolving requirements.
In [Bhaduri et al., 2004], the authors give a comprehensive survey of researchers that apply model-checking to state machines. With different model checkers including SMV, FDR [Leuschel et al., 2001] and SPIN have been employed to achieve the verification of the properties of the designed systems.

Latella et al. [Latella et al., 1999] show a translation method from UML state machines to Promela. They only enable a model to contain a single state machine. In another translation by Mikk [Gnesi et al., 2002], the input model is not UML state machines,

except Statecharts, that is a related formalism with different semantics. The work in [Goldsby et al., 2006] shows a tool called vUML that converts UML to Promela.

Most of these works have the limitation that no data characteristics can stay associated with objects or state machines. For that, there is no action language, and the only possible effect of a transition is to send signals with no parameters.
The Hugo project [Knapp, 2002] supports SPIN as a back-end to verify UML models. The initial Promela translation was only possible for small models and the current version of the tool follows ideas similar to these in vUML. To the best of our knowledge, the translation is undocumented. The OMEGA [Ober, 2004], project has created a set of tools focusing on real-time properties. The method is based on translating UML to the ``IF'' intermediate language that has several model-checking back ends.

The Rhapsody UML verification environment [Schinz, 2004], continues model-checking of UML state machines by evaluating the models into the input language of the Verification Interacting with Synthesis (VIS) explanatory model checker. All UML constructs are approved, and the action language is a subset of C++. The specializations that are not compatible include postponed events and do activities.

In [Beato et al., 2004, Kwon, 2000, Hai-yan et al., 2001], the authors define a model transformation from UML Statecharts to Symbolic Model Verifier (SMV) input language through an intermediate language and verify the system properties specified in CTL by invoking SMV. The current events are including the formal verification of a simplified directory-based cache coherence protocol in UML Statecharts.

In [AlRawashdeh et al., 2014, Al Obisat, 2012], the work concerns the ability to improve the definition of UML models in the logic of formal methods, where the model checker for theorem provers can run on these graphical diagrams. As this analysis, are combined conversion tool described Hugo/RT into our tool MUML that can support to map the model specifications and properties into Promela.

In [Pedroza et al., 2011, Apvrille et al., 2013], the authors defined a SysML-based environment named AVATAR. It can capture both safety and security related elements in the same SysML model. TTool, an open-source SysML/UML Toolkit, provides AVATAR editing capabilities. The designer may formally and directly verify those properties with the well-established UPPAAL and ProVerif toolkits, respectively. TTool supports AVATAR, including its security extensions. Therefore, the TTool is the open-source toolkit from category, transforming a SysML/AVATAR model into timed automata, and model-checking the latter using UPPAAL [Ma et al., 2014].

In [Miyamoto et al., 2012], the authors proposed the automatic conversion from the early stages design by UML to Promela including LTL formulae. This combination enables to perform model-checking from UML models. Therefore, model-checking can be easily performed without re-encoding the model and providing numerical formulae.

In [Ando et al., 2013] the authors proposed a formalization of SysML state machine diagrams in which the diagrams are translated into (CSP#) processes that could be verified by the PAT model checker.

In [Hammad et al., 2002] the authors define a model transformation from extracted graphic elements from B specification to make it more understandable to the beginner. These visual elements are explained in a UML statechart diagram. In every B specification,

the machine is associated to a package of UML classes and every variable of the machine is associated to a class.

In object-oriented modelling, a statechart diagram is connected with a class of the model. So, one obtains two additional views of the development:

- The UML view that describes in a synthetic and intuitive way the various aspects of the future system allowing a perfect understanding. Moreover, documentation of the model builds allows the validation of the B specification by an expert of the domain, not mastering B.
- The B view that defines in a precise and rigorous way all the components of the system allowing a fine analysis and understanding of the model without ambiguity.

For now, this study is limited to the extraction of diagrams from a subset of the B language. In [Bousse et al., 2012] the authors show a method consisting in a harmonization of SysML concepts with a recognized subset of the B method, using semantic similarities between both languages. This work is a first experiment limited to simple data structures and without making real use of the B method refinement abilities. The both languages propose must include coordination to preserve semantics during the transformation, but such protection is not proven.

Moreover, this work lacks the effective support of traceability using SysML technique also errors classified by the B method cannot be clearly combined to the primary SysML model. Additional work will extend this Validation and Verification (V&V) approach to other Domain Specific Modeling Languages (DSMLs) and formal methods, with a particular concentrating on bidirectional conversion for traceability and reflecting identified mistakes on the original model.

2.4/ Verification of SystemC Designs

We introduce simulation and model-checking techniques and their advantages for the verification and validation of systems during the design phases. We use these techniques to provide a verification of SystemC designs which is a subject of many research.

Formal verification is the act of proving or disproving the correctness of a system with respect to a certain formal specification or property. In model-checking, the specification is in the form of a logic formula, which is determined with respect to a semantic model provided by an implementation. We present these libraries and their applications to verify SystemC designs and some techniques tools.

2.4.1/ With SystemC Environment

The SystemC libraries developed to provide verification are limited and do not allow the verification of temporal properties. Considering this issue, a verification library is built on the top of SystemC libraries to enable the verification of temporal properties.

In [Große et al., 2007], the SystemC Verification Standard (SCV) library provides a common set of APIs for transaction-based verification, constrained randomization, weighted

randomization, and exception handling. By using a generic data, it enables the library to manipulate arbitrary data types in a consistent way. It includes C/C++ built-in types, SystemC environment and user-defined composite types and enumerations. The areas covered by SCV are as follows:

- Data Introspection (similar to Verilog PLI but for C++ data structures).
- Randomization and management for reproducibility of simulation runs.
- Constrained randomization and Weighted randomization.
- Transaction Monitoring and Recording.
- Sparse Array Support.

In [Kasuya et al., 2007], the Native SystemC Assertion (NSCa) is developed to implement reliable verification support in SystemC. The assertion engine contains temporal assertion primitives similar to SystemVerilog Assertion (SVA) [Bustan et al., 2012], as well as primitives to construct assertions for transaction-level models at higher levels of abstraction. The NSCa package includes a total coverage analysis ability so a user can measure the effectiveness of property classification checks.

NSCa implements two types of assertion coverage mechanism. Moreover, Assertion Path Coverage (APC) and Assertion Activation Coverage (AAC), works at a property level. This ability in NSCa collects different levels of statistics that can be used by users to recognize further the failures of assertions. The information provided by AAC is how many times a property has been activated, and the property check passed or failed and checked failed when simulating the property.

In [Oliveira et al., 2012], the System Verification Methodology Library (SVM) is upper-level of TLM library for SystemC, which is based on the OVM-SC library. SystemC application of an Open Verification Methodology (OVM) [Glasser, 2009], was enhanced to the Universal Verification Methodology (UVM) [Piccolboni et al., 2014].

The SVM Library improves the OVM/UVM for SystemC, by adding features based on the OVM for SystemVerilog version 2.2.1. The libraries provided are the Assertion, Randomization/Constraints and Coverage, to support advanced RTL/TLM for SystemC. The outline details of the functional coverage implementation of the SVM as a SystemC library, which is based in the following areas covered in SVM are the following:

- Checked expressiveness a formal hierarchical composition of a functional coverage metric shall be enabled.
- Interoperability to collect and evaluate functional coverage with no modification of the SystemC kernel are allowed.
- Model environment dependencies the functional coverage facility shall not rely on the SCV.
- TLM verification multiple specific sampling per delta cycle shall be supported to allow coverage of all and post transaction event coverage.
- Assertions are characterizing behavior between two events.

In [Haedicke et al., 2012], the Constrained RAndom Verification Environment (CRAVE) is a new library for constrained random simulation generator. The structure of CRAVE has been designed to fit C++ and SystemC ordinarily. To overcome the limitations of SCV, the CRAVE library provides the following characteristics:

- Constraint specification of API is automatic specify random variables, and random objects have been expanded.
- Dynamic constraints and data structures constraints can be controlled dynamically at run-time.
- Enhanced usability inline constraints can be formulated and modified incrementally at run-time.
- BDD-based and SAT/SMT-based techniques have been combined for constraint-solving.

In [Tabakov et al., 2012], the authors present CHIMP. It is a tool for assertion-based dynamic verification of SystemC models, by using LTL. The assertion declares a property about the entire execution trace of the Model-Under-Verification (MUV). The trace property is an LTL formula interpreted over the infinite trace. However, a simulation cannot be run for infinite time. The simulation traces for SystemC model is finite. Therefore, the monitor that is created from a trace property can have three possible results, ``PASS, FAIL, and UNDETERMINED''.

The latter case occurs when there is some future obligation that is not satisfied with the finite simulation. Each assertion is converted to a C++ monitor class. Through the execution of the MUV, a single case of each monitor class is created in the illustration phase before the simulation phase begins. The number of monitor cases is equal to the number of trace properties to verify.

In [Ferro et al., 2010], the authors present ISIS, which enables the runtime Assertion-Based Verification (ABV) [Barnat et al., 2013], ABV is a simulation environment during a particular simulation run, and ever a cycle when both the read and the write signals active. ABV of SystemC TLM virtual platforms ``timed or untimed, clocked or unclocked'' for verifying behavior and requirements. This technique has also been improved to support the PSL modeling that enables the use of ``global'' auxiliary variables in assertions.

The prototype tool called ``ISIS'', which implements all these features, is an academic tool that answers the need for ABV at the system level. Given temporary properties that capture the intended requirements, ISIS automatically instruments the SystemC TLM design with ad hoc checkers.

Through simulation, the checker provides information about the satisfaction of the assertion. Properties that the design under development must verify can be specified by Property Specification Language (PSL) [Foster et al., 2005], which is used to define logical and temporal properties. It inputs PSL assertions and performs the automatic construction of TLM-oriented SystemC monitors to build a structure from basic components.

The monitors are automatically linked to the designs. This instrumented design is compiled using the SystemC library of primitive monitors. SystemC simulator can then be run on this combination of modules, the monitors inform about the satisfaction of the properties during simulation.

2.4.2/ Translation to Model Checking and Tools

In this part, we provide a brief display of the concepts that use the SystemC for formal verification.

In [Große et al., 2010], the authors present SystemC Induction-based VErifier for transaction-level models (SCIVER), a formal property checker for SystemC TLM models using simple assertions. SCIVER supports the verification of high-level properties such as the effect of performance or whether the execution is only started after a certain event. The properties are specified using a variant of PSL with support for TLM primitives. They convert the SystemC model together with the property deduction to a C model. After that, the induction based verification method is applied to the C model.

The SCIVER is not only enabled to detect a property violation but also to prove its absence. The SystemC TLM model produces the formalization of the design specification. This first TLM model is ordinarily untimed, and then will be sequentially refined by combining timing information to a timed TLM model, that in turn is repeated down to RTL. The different types of properties and the respective monitoring logic are explained.

- Simple safety properties: concern values of variables of the TLM model at any time during the execution.
- Transaction properties: can be used to reason about a transaction effect.
- System-level properties: focus on the order of occurrences of event notifications and transactions.

In [Razavi et al., 2011], the authors present SYSFIER, to formalize SystemC semantics and to provide a mixed environment for modeling and verification of SystemC designs. The parts of this mixed environment are called Afra, the formalized semantics of SystemC in the context of Rebeca semantics. Rebeca (Reactive Objects Language) is an object and event driven based modeling language. The main components of Afra are:

- Sytra creates Rebeca models from SystemC models based on the advanced formalism.
- KaSCPar is used to analyze the SystemC models, the output of this parser is an XML file describing the SystemC model.
- The Rebeca model is then created using this XML file.
- Modern is a direct model checker of Rebeca.
- SyMon (SystemC Model-checking Engine) is a verification engine customized for verification of Rebeca models obtained from SystemC codes.

In [Cavada et al., 2014], the authors present KRATOS. These software model checker for SystemC. Kratos permits to verify safety properties, in the form of program assertions by supporting verification in two ways. Firstly, by relying on the translation from SystemC designs to sequential C programs. Kratos can perform model-checking over the resulting C programs utilizing the symbolic lazy predicate abstraction technique.

By implementation algorithm called ESST (Explicit-Scheduler/Symbolic-Threads) also, S3ST can combine specific state techniques to deal with the SystemC Scheduler with Symbolic techniques to support the ``SC_Threads''. The Kratos is built on top of NuSMV and MathSat and uses state SMT-based techniques for program abstractions and refinements.

In [Chou et al., 2012], the authors present Symbolic Data and Suspended Status (SDSS), the formulation interpreting SystemC designs with timed language constructs as Kripke structures. By this formulation, it applies symbolic model-checking to the Bounded Model Checking (BMC) techniques. Within BMC, the transition system is increasingly significant degrees that may extend to time and allows to constraint resources.

The SystemC is scheduler logic as formulas directly. Alternatively, the embed the algorithm of SystemC scheduler in SDSS to enumerate further all possible scheduling of run threads. SystemC designs converted into timed language constructs to Kripke structures and proposes a symbolic model-checking approach for verifying SystemC designs. The main variation is that the scheduler is not included in the encoding of SDSS.

In [Fernandez et al., 1996], the authors present Construction and Analysis of Distributed Processes (CADP). These tools include many apparatus useful for formal verification and feedback. The main instrument is an explicit model checker. CADP provides numerous tools, among them:

- A step-by-step, interactive, and random simulator.
- A model checker that creates an explicit representation of the state space.
- Property checker for various temporal logics.
- Equivalence checker and LTS minimization tools.

The typical entry point for CADP is the language LOTOS. The ISO standard LOTOS is a process algebra used to specify concurrent asynchronous processes by communicating and synchronizing rendezvous on gates. The verification structure is based on a new SystemC/TLM front-end for CADP, called ``TLM.open''.

The ``TLM.open'' front-end consists of a C/C++ library implementing two interfaces. First, TLM.open produces and executes a subset of the OSCI SystemC library, including modules, events, TLM ports, and processes (``SC_METHOD'' only). Second, ``TLM.open'' implements the OPEN/CASAR interface.

The CADP verification toolbox is optimized for asynchronous processes. SystemC/TLM models use asynchronous processes, but SystemC programs model a system at a lower level of abstraction and use synchronous processes. To verify synchronous processes symbolic model checker based on BDD or SAT, are in general more efficient than CADP. Also, ``TLM.open'' can be used for absolute SystemC programs, but is not the most effective tool.

2.5/ Summary

In this chapter, we presented some related works concerning the proposals of this thesis. We showed that semi-formal specifications utilized several specific languages or other

representation to design. [Antonis et al., 2008] base their work SysML requirements like we do in this thesis. They use Event-B to verify requirements and we use UPPAAL model-checker. In another translation by Mikk [Gnesi et al., 2002], the input model is not UML state machines, but Statecharts, which is a similar formalism.

The are utility through combining SysML with SystemC. The attempts by using the destination SysML is to display system architects, the [Riccobene et al., 2012], the synthesis is based on a mapping from the SysML to the SystemC for the structural and behavioral aspects. The SystemC libraries developed for providing a verification are limited and do not allow the verification of temporal properties. Though, its current provider is limited to SystemVerilog language that often requires the confusing use of multi-language simulation environment in particular for interaction with SystemC–based TLM models. In [Oliveira et al., 2012], the library SVM is upper-level of TLM library for SystemC. The efficiency of state exploration and model-checking methods depend heavily on the size of the state design.

II

Scientific Context

3

Related Concepts

Contents

3.1/ Model Verification and Validation

One key goal of model-based development is to enable analysis of the system, therefore ascertaining the property of the system already at model level. Instances of such properties are deadlock freedom, time consistency and limited storage resources. While building up a concurrent object-oriented application, deadlock freedom of the interaction is frequently a dominant requirement. The timing constancy is of importance for real-time systems. On that level, it must be assured that certain calculations are enabled before specified time span. Verification and Validation focus on verifying requirements properties for different purposes. Properties to be verified differ according to the nature of models e.g. non-functional or functional and to the stage in the evolution process (e.g. specification or code generation time) [Cao et al., 2011].
Furthermore, there are also limitations depending on the applied verification and validation techniques from testing, model-checking, formal proof, and runtime verification. However, there is no one-to-one association between requirements properties and purposes of techniques. For verification of properties, a suitable formal verification tool (e.g. model-checker) has to be chosen capable of verifying the characteristics associated with the requirement properties [Adrion et al., 1982].

Verification is the process of ensuring that the model behaves as designed, generally by debugging for dynamic verification or through animation. Verification is necessary but not sufficient for validation, and a model may be verified but not valid. For validation of properties, tests, check the compatibility of the system under test to a specification by simulating and testing.

Validation confirms that no significant difference exists between the model and the real system which is model reflects reality. Validation can be achieved through sta-

tistical analysis, for instance, markov-chain or stochastic models. Additionally, apparent validity may be obtained by having the model reviewed and supported by an expert [Oberkampf et al., 2010].

However, when the application, validation is often combined with verification, particularly during data analysis used for the system being modeled. If a comparison of system analysis and model results indicates that the effects made by the model is close to obtaining from the system, then the implemented model is taken up to be both a verified implementation of the assumptions and a valid representation of the system.
To specify, design and implement complex systems, it is necessary to decompose them into subsystems. A system may be composed of hardware and/or software parts. These heterogeneous systems can be modeled by SysML [Friedenthal et al., 2008], which is based on UML. To implement these systems, we use MDA (Model Driven Architecture) [OMG,] approach to transform PIM (Platform Independent Model) into PSM (Platform Specific Model), like SystemC [Aynsley, 2006], Modelica [Fritzson et al., 1998], or VHDL [Shahdad, 1986].
A model-driven transformation is a set of mapping rules that describe how parts of a given source model map to their corresponding parts in a target domain model. SysML models objective is to enable the description of a system at a high level of abstraction while providing graphical views of its requirements, structure and behaviour. A model can be used to design embedded HW/SW systems that support some methods diagram to a complexity of modern designs such as abstraction, project and design reuse. The transformation SysML-based modelling into SystemC [Bombino et al., 2012] environment is allowed to enabling specified the static and the dynamic system analysis.

SystemC is a language standardized by the ANSI. It consists of several C++ classes (libraries) that enable the description of concurrent systems in an event-based paradigm. It can describe systems at executable specification level [Riccobene et al., 2009].

The main purpose of modelling activity is validation. Model validation techniques involve simulating the model under recognized input conditions and comparing the model output with the system output. This model intended for a simulation study is a mathematical model developed with the help of simulation software. SysML is lacking formalization for the required validation. The combined use of SysML and SystemC is a good way to satisfy the needs of simulation.
The issue is as follows: how simulation and verification can be combined to validate the characteristics of some parts or of the whole system, its functionality or its performance.

Simulation can help to validate performance requirements, but the proof of system functionalities requires the use of formal verification methods. Combining SysML and SystemC is not enough to validate these systems. Sometimes, it is necessary to validate functional requirements by using techniques like model-checking. So far, SysML and SystemC do not supply tools for verification activities.

In this approach, the combination of the model-checking Spin/Uppaal [Zervoudakis et al., 2013], allows to complete the process of validation. The Spin/Uppaal is an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata.

3.2/ Systems Engineering and MBSE

Systems Engineering (SE) is a structured technical design and management process used in the design, development, production and operation of large-scale complex systems. It concerns conception, design, prototyping, implementation/realization, and administration. Throughout these phases, SE is a multidisciplinary approach that focuses on the system as a whole. Like many other engineering disciplines, SE is supported by some systems concepts and standards.

For instance, the primary mission of the INCOSE [Honour, 1998] is ``to advance the body politic of the technical production and practice of systems engineering in industry, academia, and government by promoting interdisciplinary, face approaches to produce technologically appropriate solutions that meet societal needs''.
The design aspect of SE focuses on finding viable solutions to given problems in the setting of a generic solution domain. Therefore, it can be described as many tasks relating to the subsystem decomposition, the target hardware platform, the data storehouse, and the equivalents. In addition, SE permits to identify the proposed solution the design model usually details, the required or existing structural and behavioral aspects that are created during a requirement analysis phase.

MBSE philosophy has started to play a significant role in the definition of system model characteristics. The integration of MBSE methodology within the project of complex systems has found a prolific environment in the context of SE. The MBSE models functions for an efficient management of all the phases that characterize a system. MBSE provides a framework to facilitate sharing. In the future, MBSE also holds the potential to automate the construction of integrated analyses models.

In engineering and sciences, models emphasize certain properties of interest to efficiently and practically communicate or identify results. In this context, models are complex system expressions of designs. MBSE uses a graphical language to generate and record details about system requirements, design, analysis, verification, and validation. Despite the current focus of MBSE to represent the structure, function and behavior of systems, there is a need of verification and validation using formal executable models from specifications. In doing so, MBSE has the following benefits:

- Integration of multiple modelling domains across system life cycle.
- Development of a formalized practice of SE through the use of models.
- Improvement of communication between stakeholders.
- Enhancement of knowledge capture and reuse.

3.2.1/ SysML

SysML is a modelling language specified by the OMG. It is a graphical modelling language, with semi-formal semantics, which purpose is to improve UML-based complex systems development processes with the system engineering method. SysML is a UML profile [Hause et al., 2010]. The environment Specification and provides a standard modeling language to support the specification, analysis, design, verification and validation of a broad range of complex systems which are not necessarily software based. SysML

modifies UML diagrams, such as the class diagram and the composite diagram, which become the block definition diagram and the internal block diagram respectively. Therefore, SysML does not include the UML object diagram, timing diagram, and deployment diagram.

3.2.2/ SysML Environment

A standard system modeling language, such as SysML, is required to express fundamental systems engineering concepts. Such concepts include system composition, interconnections with interfaces, functions, state-based behavior, and parametric aspects of a system. The language has a capacity to express system concepts in the form of model. Moreover, in productivity improvements through is reuse of models across projects and throughout the lifetime cycle. Other benefits are the ability to automate tasks such as change impact analysis, which is increased assurance that the data is valid and complete.

The main focus of the SysML community has been on the integration of SysML models into other engineering tools. There has been significant progress integrating system models with software design, particularly for software developers with UML models since SysML is based on UML profile. Also, considerable progress has been made integrating SysML with engineering analysis and simulation using various integration methods and instruments.

SysML has been combined extensively with requirements management tools. There has been progress integrating SysML with product lifecycle management and hardware design tools [dos Santos Soares et al., 2011]. SysML brings semantics to the UML metamodel. The creation is defined logical formalism that can be supporting the model for a broad range of analytical abstraction and model-checking. Validation requires the model to be logically consistent. In addition, it permits to investigate the effects of a requirement design change or the assessment of how a failure could propagate through a system. The language and tools must also mix with a various range of equation solvers and execution environments that incorporate the capture of quantitative information.

3.2.3/ SysML Architecture

SysML diagrams are divided into three pillars i.e. structure, behavior, and requirement. Structural pillar provides the hierarchical picture of a model and gives the guideline regarding the application of block, parts, connectors, and ports. The behavioral pillar includes of data flows, interactions, activity flow and state machine. Moreover, pillar contains sequence modeling.
SysML is defined by nine diagrams, classified into three subgroups: Structural, Behavioral and Requirement diagrams. Figure3.1 describes this categorization and the modification degree of the diagrams on their UML counterparts. As specified on the following diagram, SysML reuses a subset of UML2.3 (UML4SysML) and defines its extensions. Therefore, SysML includes nine diagrams instead of the thirteen diagrams from UML2.3, making it a smaller language.

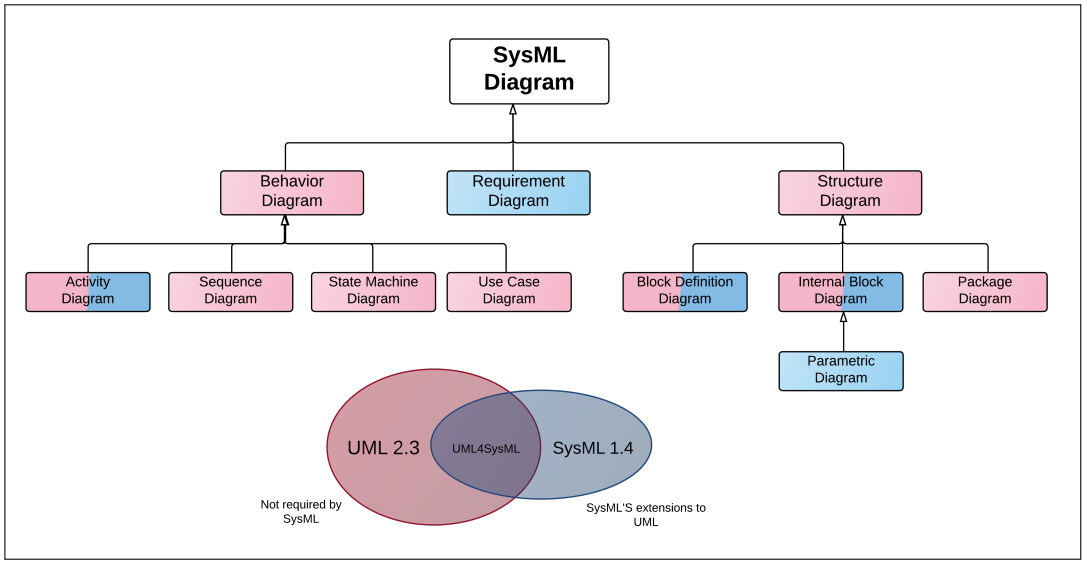

Figure 3.1: SysML Diagram category

Requirement Diagram

The Requirements Diagram (RD) displays requirements, packages, other classifiers, test cases, rationales, and relationships. The possible relationships available for requirements diagrams are containments, deriveReqt and requirement dependencies ``Copy'', ``Refine'', ``Satisfy'', ``Trace'', and ``Verify''. RD is used to reflect the relationships of other models. It assists in better organizing requirements and also shows the various kinds of relationships between different requirements.

An extra advantage of using this diagram is to standardize the way of specifying requirements through a defined semantics. As a direct consequence, SysML allows the reproduction of requirements as model parts, which mean that requirements are part of the system architecture [Laleau et al., 2010]. The RD provides modeling constructs to represent text-based requirements and relate them to other modeling elements. These requirement modeling constructs are intended to supply a bridge between traditional requirement management tools and other SysML models.

Block Definition Diagram

The Block Definition Diagram (BDD) is used to define block characteristics in terms of their structural and behavioral features, such as properties and operations. To represent the state of the system, and its behavior the basic structural element are aiming at specifying hierarchies and interconnections of the system to be modeled. A block is specified by parts and flow ports. The physical elements of the block are referred to Parts, and the interfaces of the block referred to Flow ports [Hause et al., 2010].

The constraint block permits to define constraints, such as equations or inequalities, while specifying the involved parameters and variables. The mathematical relationship can be used to constrain value properties of blocks. The purpose of constraint blocks is as follows:

- To assess the validity of system values in an operating system.
- To perform engineering analysis during the design stage of the life cycle.

The variables are called constraint parameters. They represent quantities, and so they are typed most often by value types.

Internal Block Diagram

The Internal Block Diagram (IBD) is based on UML composite structure diagrams and includes restrictions and extensions as defined by SysML. An IBD captures the internal structure of a block in terms of properties and connections between properties. A block includes properties so that its values, parts, and references to other blocks can be specified.

However, whereas an IBD created from a block will only display the inner elements of a classifier (``Parts'', ``Ports'', and ``Connectors''), each property is described as a part, and ports are used to specify allowable types of interaction. They are connected by the interactions between them, such as software operations, discrete state transitions, flows of inputs and outputs, or continuous interactions. That particular block is the context of the diagram SysML permits any property (part) shown in an IBD display compartments within the property (or part) symbol. There are two types of ports in SysML:

- Standard port specifies the services that a block provides to the environment, i.e., with other blocks, as well as the services that the owning block requires from its environment. The specification of the services is checked by typing the standard port by the provided and/or required interfaces.
- Flow ports are interaction points through which the information flows are exchanged between blocks. The interaction points between block and parts supports the integration of behavior and structure, a flow specification is created to express the type of data that can flow through the port.

Parametric Diagram

The Parametric Diagram (PD) enables to express mathematical relationships between parameters. In the PD, the ``Ports'' are constraint parameters and the ``Connections'' are binding connectors. A constraint property is an instance of a constraint block. Its constraint parameters are next bound to other constraint parameters or properties of blocks. The semantics of a binding connector indicate a mathematical equations between the block properties or constraint parameters being connected. This mathematical equation is a causal relationship.
The PD is a SysML specific modeling method that allows the combination of constraints or equations into the model for analysis purpose. These constraints are illustrated by parameters and rules that describe the evolution of these parameters related to each other.

State Machine Diagram

The State Machine Diagrams (SMD) are used to specify the state-based behavior of any component whose behavior can be expressed. They describe possible sequences of states and actions through which the modeled element can proceed during its lifetime as the results of reacting to discrete events [Machida et al., 2011]. The SMD defines a set of concepts that can used for modeling discrete behavior through finite state transition systems. The state machine describes the behavior as the state history of an object in terms of its transitions and states. A block includes operations so that its values, parts, and references execution blocks can be specified.

Through the (``Transition'', ``Entry'', and ``Exit''), the states are specified along with the associated event and guard conditions. In addition to expressing the behavior of a part of the system, state machines can also be used to express the method protocol for parts of the complex systems. These two kinds of state machines are referred to here as behavioral state machines and protocol state machines. The state machines contain one or more regions that include vertices (``states'') and transitions. A composite state has nested states that can be sequenced or concurrent [OMG, 2012].

3.2.4/ Benefits of using SysML

The SysML model differs from conventional drawing tools in the following three specific ways:

- SysML presents the capture and description of numerical values and quantities for the application of International Organization for Standardization (ISO) Quantities, Dimension, Units, and Values standards. It enables any SysML model of a system to be checked to guarantee that the units are complete and consistently defined.
- Conventional drawing and simulation tools provide text and diagram based documentation of models, but they require the semantics and detail provided by SysML. The strength of SysML is the robust semantics and detail captured for formal specifications. It becomes significant when using the SysML model as a source of information for analysis and simulation tools.
- Once a system is distributed in SysML the model provides a consistent body of knowledge about the system. The SysML model can be used to interface and interoperate with other tools and data sources.

3.3/ Simulation and validation with SystemC

Simulation is a common technique for the analysis of specifications of complex systems [Rowson et al., 1994]. Simulation can be defined to confirm the eventual real behavior of the selected system model. It is used for performance optimization by creating a model of the system and functions. We can analysis the estimation and assumption of the real system by using simulation results. Simulation is relatively inexpensive in terms of execution time, but it only validates the behavior of a system for one particular computation path. Simulation models are intensively used to solve problems and to assist designers.

The description of a design may be simulated to produce and study its behavior. The simulation is not a complete representation of the operation, design, analysis, transactions and outputs generated.
The simulation has described the evaluations limiting those representations, and other abilities needed to satisfy the user requirements. A simulation has provided an interactive, graphical environment for modeling, simulating, and analyzing dynamic systems. It enables rapid construction of virtual prototypes to explore design concepts at any level of detail with minimal effort.

SystemC [Aynsley, 2006, Boutekkouk, 2010], is an open-source system-level design language based on C++ that has its own simulation kernel. SystemC has a run-time scheduler that handles both the synchronization and scheduling of concurrent processes. The designers can apply object-oriented capabilities to hardware design. SystemC allows to work at a higher level of abstraction, enabling extremely active, more dynamic architectural trade-off analysis and design [Mello et al., 2010]. There are five major extensions that SystemC provides to model hardware:

- A notion of time.
- Support for hardware data types.
- Module hierarchy and organization.
- Concurrency.
- Communication between different modules and processes.

3.3.1/ SystemC Language Architecture

The architecture of SystemC environment is illustrated in Figure3.2. The core of the language is composed of three layers. The lower layer is a Discrete-Event (DE) event driven simulator. SystemC model implementation is defined to allow the development of different simulators with the same functionality. The middle layer defines classes that provide the semantic, structural, functional, communication and data typing facilities required for hardware and software systems modeling and design.

About the top part of the core of SystemC language, several language extensions are not yet standardized. These extensions are being developed by Open SystemC Initiative (OSCI) working groups. SystemC environment uses specific construction and executes the simulation kernel whose semantics is the first subject covered by the IEEE standard.

Time may be assigned to processes for performance analysis purposes, this timing does not accurate cycle but rather describes the time to generate or consume data or to model buffering or data access. The behavior of the interfaces between modules is described using communication protocols. These classes of models are used to explore architectures, for evidence of algorithms and performance modeling and analysis.

The SystemC processes execute concurrently and may suspend on ``wait ( )'' statements. Such processes require their private, independent performance unit called ``sc_thread''. When the only signal triggering process is the clock signal, we obtain clocked thread process call ``sc_cthread'' certain processes do not require an independent execution stack. Processes that cannot be suspended on ``wait ( )'' statement are termed ``sc_method''.

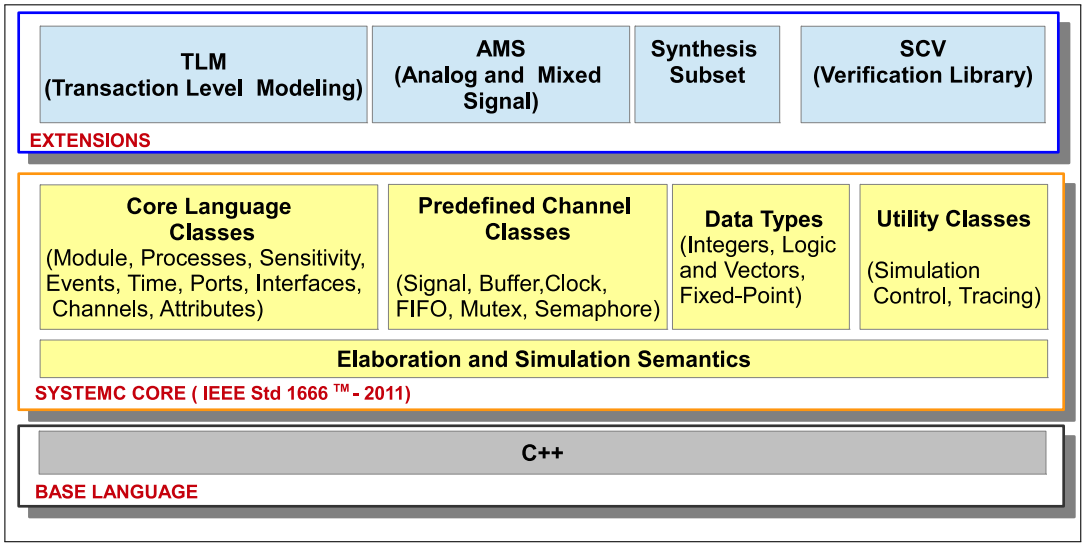

Figure 3.2: Architecture of SystemC platform

These processes are executed in zero simulation time and return control back to the simulation kernel.

Modules are defined by deriving from the SystemC library class ``sc_module''. Modules define connection points called ``ports'' as data members. A port is instantiated from the class ``sc_port'' or a class derived from this category. Ports can be connected through channels. Several primary channels are defined in the SystemC class library and are derived from the class ``sc_prim_channel''. The module hierarchy of a model is created during the execution of the explanation phase of the model utilizing objects instantiated from classes that derive from ``sc_module'', ``sc_port'', or ``sc_prim_channel''. All these building blocks have the same base class called ``sc_object''. When this program is compiled and linked with the SystemC library, an executable version of the model is produced.

This executable can be used to simulate and to verify the model dynamically. A SystemC program should not define the main function because this main function is defined inside the SystemC library. The program should define a ``sc_main'' function instead. This function must create and initialize the module hierarchy and call the SystemC library function ``sc_start'' to start the simulation of the model.

During the execution of a model, the elaboration phase starts with the performance of the main function the library. This function performs some initializations and then calls the ``sc_main'' function which is defined by the developer. This function creates and initializes the module hierarchy by instantiating the top-level module and channel objects and their connections. The constructors of these objects are executed, and these constructors can create and initialize submodules, ports, processes, channels, with their connections.

Listing 3.1 presents the SystemC modules illustrated in Figure3.3. The modules represent hardware blocks of the SystemC model. In this example, ``Adder'' is the module name. The module consists of the input ports ``x, y'', and the output port ``z''. Moreover, SystemC processes describe the implementation of the module. A module may have more than one

SystemC process. For this particular example, ``calculate'' is the SystemC process. There are three different types of SystemC processes. The kind of the process and its sensitivity list defined in the constructor of the module. A ``sc_method'' process is a SystemC process that is called by the SystemC scheduler whenever a signal changes in its sensitivity list. It cannot be suspended and resumed. A ``sc_thread'' on the other hand, is a SystemC process that can be suspended and resumed through ``wait ( )'' calls and event notifications. There are two files ``.h'', for ports, functions, variables, and processes declaration and one ``.cpp'', for process and functions implementation, ``systemc.h'' designates the SystemC library file. The code can be seen in Listing 3.1.

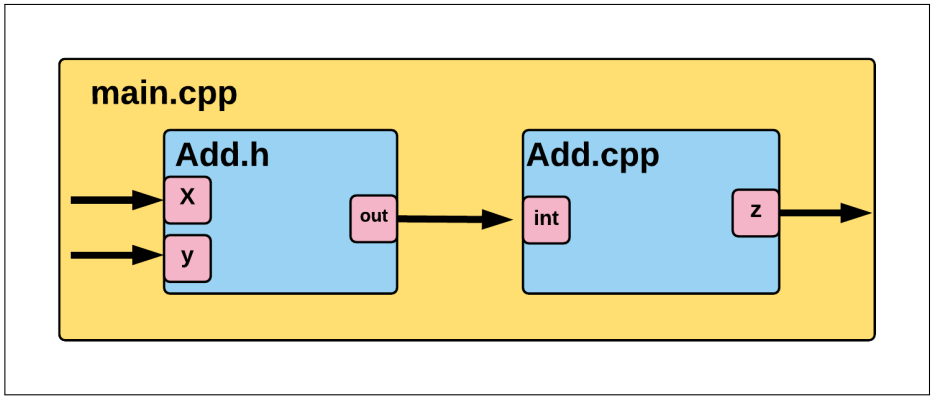

Figure 3.3: Declaration ports with a module

Listing 3.1: adder.h with adder.cpp

```
// file (adder.h)
# include "systemc.h"
SC_MODULE(Adder){
        sc_in<int> x;
        sc_in<int> y;
        sc_out<int> z;
        void calculate();
SC_CTOR (Adder){
        SC_THREAD(calculate);
        sensitive <<x<<y;
        }
}

//file (adder.cpp)
# include adder.h
void Adder::calculate(){
        z= x + y;
        wait();
}
```

Transaction-level modeling

Transaction-level modeling (TLM) [Cai et al., 2003], is a technique for illustrating a system by using function calls that define a set of events over a set of channels. TLM is a library implemented as a layer on top of SystemC. It is flexible enough to model components and systems at many different levels of abstractions:

- Loosely-timed: the model uses transactions corresponding to a complete read or write across a bus or network to physical hardware. That provides timing at the level of the individual transaction.
- Approximately-timed: the model breaks down transactions into some phases corresponding much more closely to the phasing of particular hardware protocols.

However, careful choice of the abstraction level and associated methodology is necessary to ensure practical gains for design teams. In a TLM-based design flow, the system is a principal model in high-level to check the functionality disregarding details related to the target architecture.

Register Transfer Level Modeling

The Register Transfer Level (RTL) [Calazans et al., 2003]. Is a high-level executable design allowing to simulate the behaviors of the complex systems efficiently before synthesizing the RTL hardware description generating from SystemC TLM specifications. The RTL is a modeling style that corresponds to digital hardware synchronized by clock signals. This modeling technique is used within languages such as Verilog and VHDL. Therefore, it supports practical hardware synthesis tools. In the RTL style, all communications between processes occur through signals. The processes may either represent sequential

logic, in which case they are sensitive to a clock edge, or they may describe combinational logic, in which case they will be sensitive to all inputs.

The ports in RTL module correspond to wires in the real-world operation of the module. The RTL domain, on the other hand, describes the clock cycle by data flow of the hardware at the register level and can be synthesized. Therefore, the design process often implies a manual translation step from behavioral to RTL with baseline testing to verify proper operation of the RTL design. A high-level executable design allows to simulate the behaviors of the complex system efficiently before synthesizing the RTL hardware description generating verification from SystemC TLM specifications.

3.3.2/ SystemC Simulation Environment

The SystemC simulation systems [Benini et al., 2003], kernel handles the scheduling and synchronization of SystemC processes. The simulation execution depends on two types of procedures that are scheduling and the event handler routines. The scheduling operation is a significant task since it creates and classifies events in time. In SystemC, the event utilizes three data structures: state variables, event list, and clock.
A simulation process is intended to model a specification entity in the simulation with a well-defined behavior. The behavioral description of the entity is encapsulated by the process, defining the actions performed by the process throughout its lifetime. Figure 3.4 describes the execution semantics of the scheduler.

The execution of the scheduler can be split into two major phases the ``sc_start( )'' and the ``sc_stop( )''. The simulation starts with ``sc_start( )'', also stops when the simulator both has no more events to process or when ``sc_stop( )'' is called.

The simulation begins with the initialization phase when every process runs once, then alternates between evaluation and update phases. When no such available to operation controls exists, the simulator starts the ``Update phase'', signal values are updated to the values computed during the evaluate phase.

At this point, if there are any suspended delayed notifications, the simulator enters ``Delayed notification phase'' to define that processes are available to run due to the delayed notifications and responses to the ``Evaluate phase''. Unless, if there are timed notifications, the simulator step inside ``Timed notification phase'' anywhere it advances the current simulation time to the initial pending timed notification and reemerges the ``Evaluate phase''. If there are no timed notifications, the simulation is finished. It is during the evaluation phase that processes run, and during the update phase that signals values and other primitive channels are updated. The scheduling algorithm implies the existence of three loops resulting from immediate, delta, and timed notifications, as follows:

- The immediate notification loop is restricted to the evaluation phase.
- The delta notification loop takes the path of the evaluation phase, followed by the update phase and go back to the evaluation phase. This loop advances the simulation by one delta cycle.
- The timed notification loop takes the path of the evaluation phase, followed by the update phase and back to the evaluation phase. This loop advances simulation time.

The order in which runnable processes execute is undefined. Immediate notification results from a call to the ``sc_event::notify ( )'' event with no argument. A delta notification results from with a zero delay. A timed notification results from a call to ``sc_event_timed ( )'' with a delay greater than zero.

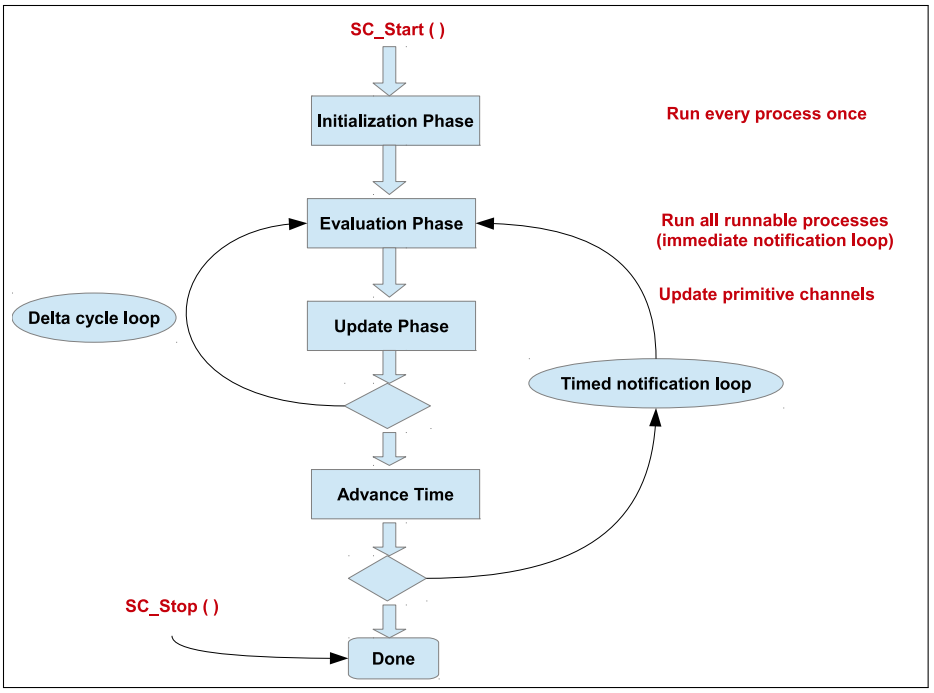

Figure 3.4: SystemC Simulation Kernel

3.4/ Verification with Model-Checking

Model-checking [Sreemani et al., 1996, Blanc et al., 2010] is an automated technique that, given a finite-state model of a system and formal properties, systematically checks whether this property is satisfied in that model. Model-checking is a software verification technique that uses the model of the software application and tries to verify certain properties on the software model. Model-checking process is composed of the designed system model, properties definition, running the model checker and analyzing the results.
Model-checking verification techniques use models representing the possible system behavior in a mathematically precise and unambiguous manner. Verifying an accurate model of the system, usually leads to the detection of ambiguities, incompleteness, and inconsistencies in informal system specifications. Such problems are only discovered at a much later stage of the design. A model checker is usually composed of three main parts:

- Property specification language: based on a temporal logic.
- Model specification language: a formal notation for encoding the system to be verified as a finite-state transition system, i.e., the model.
- Verification procedure: an exhaustive intelligent search of the model state space that determines whether the specification is satisfied or not.

Validation methods have both advantages and limitations. Model-checking is used to find a number of errors with a low cost of application compared to techniques like mathematical demonstration. The latter requires the knowledge of mathematical approaches to provide and support tools.
Model-checking defines a set of algorithms for verifying a formula written in temporal logic. The behavior of a complex system is specified by a transition system. The properties to check are of two types:

- Safety properties: stipulate that something bad will never happen.
- Liveness properties: stipulate that something suitable will eventually occur.

3.4.1/ Temporal Logic

Temporal logic [Clarke et al., 1986], is a logical structure for specifying and verifying the correctness of computer programs. However, due to their correspondence to natural language, their expressivity, and the structure of off-the-shelf algorithms for model-checking, temporal logic has the possibility to affect several other areas of engineering.

The requirements or specifications are expressed as Linear Temporal Logic (LTL) [Pnueli, 1977] formulas. LTL is the leading technique for the specification of temporal rules. And the Computation Tree Logic (CTL) [Alur et al., 1990] is the base of logical formalisms for program specification and verification because of its intuitive syntax and its very reasonable complexities. The LTL extends propositional logic with the four operators ``$always(\Box A)$'' (condition holds always in the future), ``$Eventually(\Diamond A)$'' (condition holds sometime in the future), ``$NextXA$'' (condition holds in the next cycle), ``$Until(A \cup B)$'' (condition A holds until condition B, afterwards do not care), ``$Or(A \parallel B)$'', ``$And(A\&B)$'', and ``$Not(!A)$''.

Spin provides an automatic translator from LTL formulae into Buchi automata. In case the system violates a property, the trace of actions leading to an invalid state, or a cycle is reported. The incorrect trace can be replayed, on the Promela source, by a guided simulation.

3.4.2/ Promela

PROMELA (PRocess/PRotocol MEta LAnguage) [Mikk et al., 1998] is a verification modeling language that can describe hardware and software systems. It is used to build Promela models that can be interpreted by the SPIN model checker. SPIN can generate a C program that can perform a verification of the model. Various safety and liveness properties can be verified such as deadlocks, non-executable code and finding non-progress cycles.

Promela programs consist of three different elements: processes, message channels, and variables. Promela code can be interpreted by the C preprocessor and therefore it is also allowed to use macros in Promela models. Formerly a Promela model is parsed by SPIN, and the C preprocessor first expands macro calls. The main part of a Promela model consists of proctypes with statements. An instantiation of a proctype referred to as a process, which created on initialization of a run, or other processes can dynamically create it. Statements are executed in sequential order inside a process, but processes can interleave with each other.

Listing 3.2 presents a Promela module named ``SenderRe'' with a list of messages (mtype) ``msg0, msg1, ack0, ack1'', and the channels ``to_sndr, to_rcvr''. Moreover, the processes (proctype) is called ``Sender, Receiver'' which are sensitive to ``mtype.'' For each Promela module we have used a label, named ``again'' in each proctype and a ``goto'' statement, with the usual semantics.

3.4.3/ Model-Checkers Tools

The formal validation of information systems specification is of particular interest in MDE and programming, for that the object at synthesizing an implementation of a system from models. Therefore, if the synthesis algorithms are correct, one only needs to validate the models to produce conventional systems. The information systems of MDE specification languages usually does not have any dedicated model checker and the check is a long process. Although, several model checkers already exist, and it is simple to choose a specialized existing tool with support in the model-checking domain.

Different types of temporal logics are used by model-checking tools to express properties of a system as logical formulas to be verified on the system design. Most of the model-checking tools use either linear time or branching time logics. The requirements properties that can be expressed in either of these logics can also be expressed in the other one also. However, some of the tools use different types of logic to formulate the properties.

Listing 3.2: Promela file SenderReceiver.pml

```
mtype = { msg0, msg1, ack0, ack1 };
chan    to_sndr = [2] of { mtype };
chan    to_rcvr = [2] of { mtype };

active proctype Sender()
{
again: to_rcvr!msg1;
to_sndr?ack1;
to_rcvr!msg0;
to_sndr?ack0;
goto again
}
active proctype Receiver()
{
again: to_rcvr?msg1;
to_sndr!ack1;
to_rcvr?msg0;
to_sndr!ack0;
goto again
}
```

The specific state model checkers, like CADP [Fernandez et al., 1996], SPIN, and FDR2, use an explicit representation of the transition system associated with a model specification. The NuSMV symbolic model checker represent the transition system as a boolean formula. However, limited model checker, like NuSMV [Cimatti et al., 2000] and Alloy consider traces of a maximal length of the system and represent them using a boolean formula.
The constraint satisfaction model checkers is used logic programming to verify the formula. The SPIN, CADP, NuSMV and ProB [Leuschel et al., 2003] are supports linear time also called LTL is a temporal modal logic with modalities that refer to time.
The Branch time also called CTL is branching-time logic. Its model of time is a structure like a tree where the future is not determined. There are different ways in the future that can be followed as the selected future path. In CTL, it can be specified that when an initial state is true, then all possible executions of a program avoid some undesired state or condition.

SPIN

SPIN (Simple Promela Interpreter) [Holzmann, 1997] is a model checker tool for automating the verification of modeled systems. SPIN is used to trace logical design errors in distributed systems design. Such designs are operating systems, switching systems, data, and communications protocols, additional concurrent algorithms, for railway signaling protocols.
The tool can check the logical compatibility of a specification. It reports possible deadlocks, flags incompleteness, unspecified receptions, and unwarranted assumptions about the relative speeds of processes. It can be accepted as a full LTL model-checking system, supporting all correctness requirements expressible in linear time temporal logic. However, it can also be used as an efficient verifier for more basic safety and liveness properties.

UPPAAL

UPPAAL [Havelund et al., 1997], is a verification tool for a Timed Automata-based modeling language. In addition to heavy clocks, the tool supports both simple and complex data types like delimited integers and arrays as well as synchronization through shared variables and actions. The specification language supports deadlock, safety, liveness, and response properties.
UPPAAL is able to generate characteristic traces witnessing a submitted safety property [Behrmann et al., 2004]. UPPAAL supports three options for diagnostic trace generation: some trace leading to the goal state, the shortest trace with the minimum number of transitions, and fastest trace with the shortest accumulated time delay.

3.5/ Model-Driven Engineering

Model-Driven Engineering (MDE) [Kent, 2002] is an approach that focuses on creating and exploiting domain models specific to application domains rather than on algorithmic concepts. A model is meant to abstract notions of the physical world and is an instance of a metamodel that defines an entire class of models. Models can be used as means to understand better the problem domain, but can also be a part of a more complex process of automated code generation. According to the level of detail, the code can be generated from the models, ranging from system structures to complete. The plan is to make sure that each working system uses the most appropriate language for the application domain. Further, the necessary is manipulating models to combine different domains perspectives. The main idea of the approach is that a software system is specified at various levels of abstraction using different modelling languages and that this specification is iteratively transformed into a particular model or implementation.

3.5.1/ Eclipse Modeling Framework

Eclipse Modeling Framework (EMF)[1] is integrated to the Eclipse Development Environment, which creates tools based on structured models for code generation. The metamodels are created using the EMF Ecore tool. The input tool for the metamodel receives an XML file based on the XMI structure generated previously.

EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. Many tools in modelling, metamodelling and MDE are based on EMF. In particular, we used Ecore tools for metamodelling. A powerful Eclipse plugin for the Papyrus[2] modeler is available to model any kind of EMF model, especially UML and related modeling languages as SysML.

3.5.2/ Model Transformation with ATL

Model Driven Architecture (MDA) [Soley et al., 2000] defines the approach to Information Technology system specification that separates the selection of functionality from the specification of the implementation of that functionality on a particular technology

[1] http://www.eclipse.org/modeling/emf/
[2] http://www.eclipse.org/papyrus/

platform. MDA approach enables a model that determines some system functionality to realize on multiple platforms through additional standards for mapping. MDA is defined by the OMG, which promotes the use of the MOF (Meta Object Facility). The MOF is a formalism of metamodel description.

Many languages and tools have been proposed to specify and execute transformation programs. OMG issued the Query/View/Transformation request for proposal (QVT) [Gardner et al., 2003], to define a standard transformation language. Despite a final specification, the area of model transformation continues to be a subject of intense research. In parallel with the OMG process some model transformation approaches have been proposed both by academia and industry. The models, constructs, tool support, and modeling approaches, distinguish the proposals with a certain suitability for a certain set of problems. However, the demand for model transformation is increasing due to the high demand for tool integration, reconfigurability feature, and information reusability.

The models in MDA can be of metamodels of description or metamodels of the transformation. MDA advocates the development of three types of models:

- Computation Independent Model (CIM): this model represents the highest level of abstraction. It defines the system requirements and the context in which it will operate while the details of the software structure and realization are hidden or not determined.
- Platform Independent Model (PIM): this model describes the specifications of the system, but does not show details of the use of its platform or of a particular technology.
- Platform Specific Model (PSM): this model describes the details and characteristics absent from the PIM. It must be modified to specify the implementation of the system in a single technology platform.

As these different types of models describe various levels of abstraction of the similar system, MDA supports the use of transformation mechanisms allowing the transformation of the CIM to PIM and the PIM to PSM. A model-to-text transformation is an algorithm that accepts an instance of a metamodel and generate text in an appropriate concrete syntax. We use MDA concepts and techniques in our work to combine SysML with SystemC or SysML with Promela.

The Atlas Transformation Language (ATL) was introduced by the Atlas Group and the TNI-Valiosys Company [Bézivin et al., 2003]. ATL aims at providing a practical implementation of the MOF/QVT standard. Therefore it provides a transformation engine that allows transforming any given source model to a specified target model.

However, to perform the transformation, the user has to specify an appropriate ATL program based on some correct relation and implementation metamodels. ATL is based on rules that are either matched in a declarative way or called in an imperative way. In addition to rules, ATL provides so-called helpers (similar to Java methods for instance) in a declarative style.

These helpers are used later in the rules to implement the real transformation. ATL supports unidirectional model transformation. However, it is possible to implement bidirectional transformation through the explicit implementation of both transformation sides. In ATL transformation, the source model has read-only access, while the target model

has write-only access. This kind of transformation execution is called ``source-target-execution''. For every source, the target elements are produced. Traceability sections are also created in this state. In the second state, all the bindings for the created target elements are executed. ATL resolution algorithm and execution of lazy rules are applied if necessary. The algorithm does not suppose any rules order, target elements creation for a source, and target elements initialization.

3.5.3/ Code genration with Acceleo

Acceleo[3] is an implementation of the OMG Model to Text Language (M2T) standard and an implementation of the MOF Models to Text Transformation (MOFM2T) standard [OMG, 2008]. It is a template-based code generator with an advanced IDE. The framework defines its language. The generator files consist of modules and templates. The Acceleo can be interpreted as an object oriented language where the modules are the classes, and the templates are the methods inside a module. Acceleo is a language code generator which allows generating structured file from an EMF model, the output is a text that can be a programming language or other formalism. Acceleo requires defining an EMF metamodel and a model conforming to metamodel that will result into text.

3.6/ Conclusion

In this chapter, we presented SysML since it becomes one of the most used modeling languages in the MBSE field and since it meets the needs of engineers to describe all aspects of complex systems. SysML is a language intended for the systems engineering community based in UML and allows designers to model not only structural and behavioral properties but also to organize the requirements.

SystemC is a modeling language that can be used to describe embedded systems at different abstraction levels. An open-source SystemC simulator is available. However, simulation is not the only purpose for which a SystemC model can be used.

Model-Checking can be used to determine the validity of formulas written in some temporal logic on a behavioral model of a system. Model-Checking tools are effective as the debugging helper for industrial designs and are fully automated.

[3] www.eclipse.org/acceleo/

4

The Traffic Light Case Study

Contents

4.1/ Introduction

In this chapter, we propose to use SysML language, to define the architecture of requirements that relate to the safety, verify, and WSN energy consumption of the crossroads system. This system is the pivot case study of this thesis.
A SysML specification of a system is described by requirement diagrams, structural diagrams, and behavior diagrams. Our approach is based on processing an incremental refinement from an abstract level toward more detailed levels. In our case, it is a question of replacing an abstract block in a specification by a composition of blocks preserving its structural properties and its behavioral properties. Structural diagrams of SysML describe the system in static mode, and behavioral diagrams describe the dynamic operation of the system. We note that the term used in SysML for components modeling is block, three diagrams, namely the BDD, IBD, and PD, enable to define and instantiate blocks. The BDD defines the architecture of the blocks with their operations. The IBD, is used to define the ports of each block and to connect them through their ports. The PD enables to express mathematical relationships between parameters. In Figure 4.1, we show the position of the contribution presented in this chapter, regarding the contributions of this thesis.

4.2/ Functional and Non-functional Requirements

The requirements can be classified depending on the kind of condition or capability that they describe. The classification is not standardized, but it is agreed that functional requirements specify a function that a system or system component has to perform, and that non-functional requirements specify how well the system should perform its intended functions.

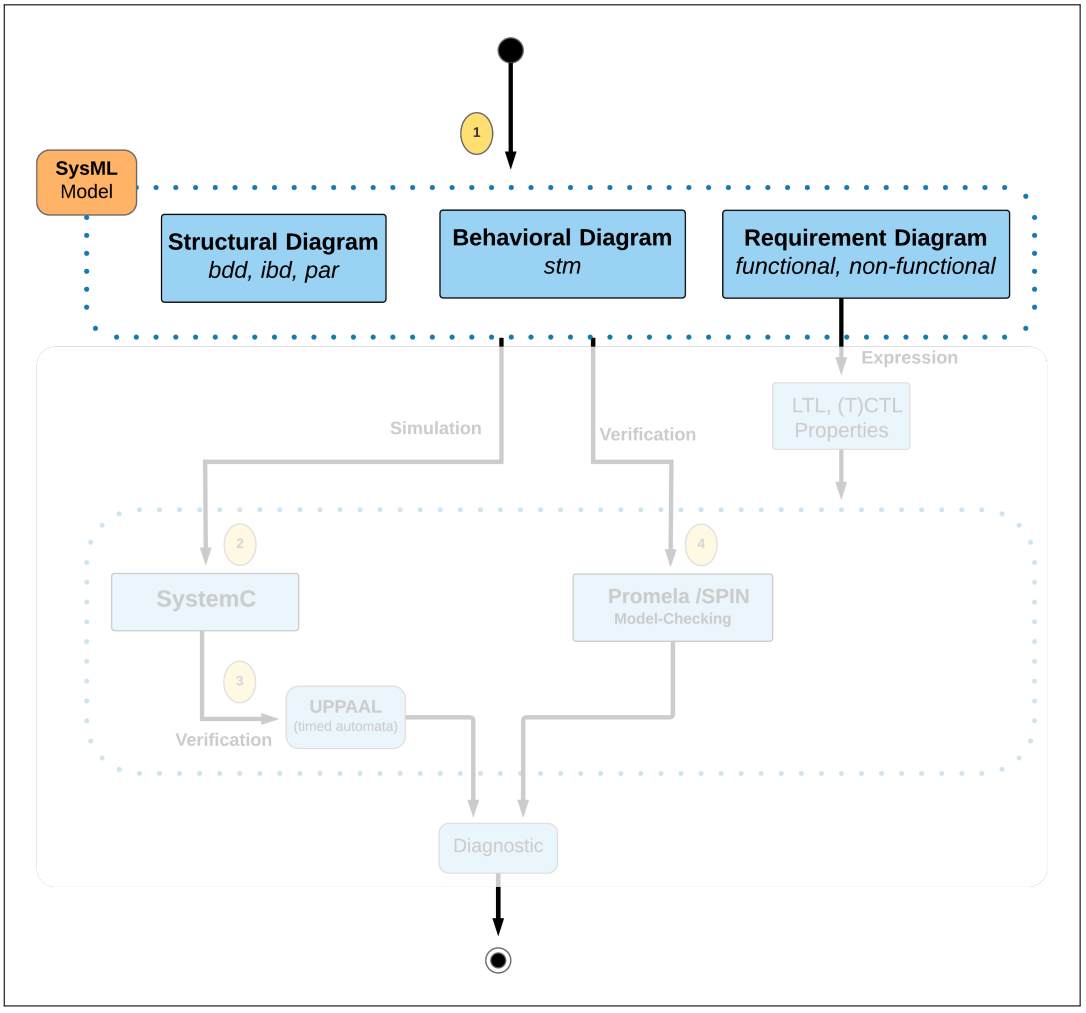

Figure 4.1: The phase one of thesis

One requirement in a software requirements specification may be associated with one or more other requirements in that specification. Relationships can be of a particular type that more accurately defines how the requirements are related. Using unknown relationship types may produce deficiency results in requirements engineering.

For example, during change impact analysis Requirements Engineering (RE) may have to analyze manually all requirements in a software requirements specification. RE is a specific discipline of the software engineering. The RE process is recognized as being the most critical process in software development. The functional and non-functional requirements has the external interface definitions and constraints. Figure 4.2 presents a classification of the possible types of the requirements specification.

4.3/ Requirement Analysis

Requirements analysis [Gunter et al., 2000] is the primary stage in the systems engineering process and software development process. Its include responsibilities that enter into the identification of conditions or requirements to meeting a new design system or

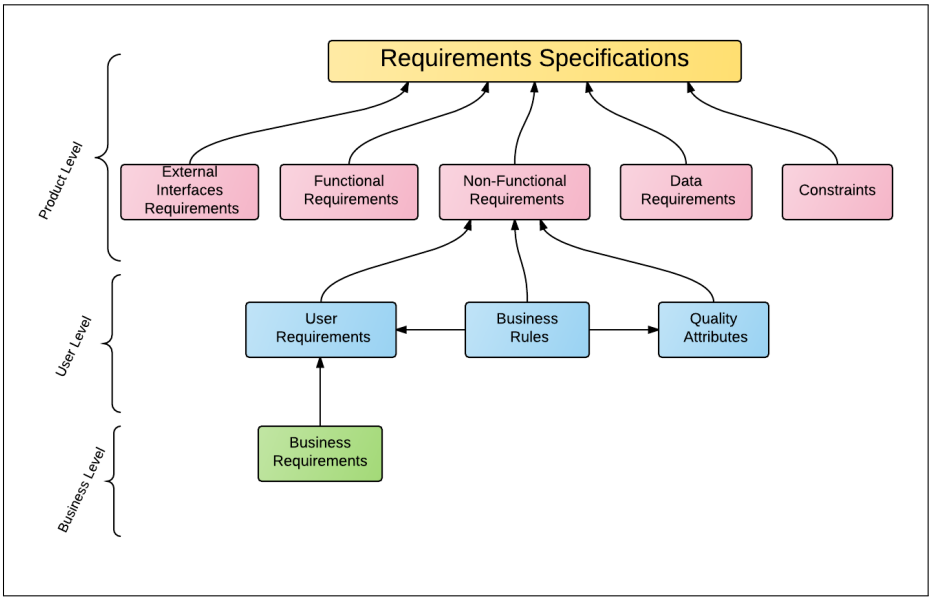

Figure 4.2: A classification of the requirements specification types

change product taking into account. The possible inconsistent requirements of the various stakeholders, such as users or beneficiaries. Requirements analysis is critical to the completion of the system design.

The requirements must be testable, related to identify business needs or opportunities, actionable, measurable, and defined to a level of detail adequate for system design. The requirements of the complex systems can be functional and non-functional. Requirements analysis needs explanation and description of functional requirement and design constraints. Functional requirements specify quantity ``how many'', quality ``how good'', coverage ``how far'', timelines ``when and how long'', and availability ``how often''. Design constraints describe those factors that limit design flexibility, like environmental conditions or limits support against internal or external warnings and contract, regulatory standards.

Requirements express what an application is meant to do. They do not attempt to express how to accomplish these functions. The set of requirements for the system should describe the functional and non-functional requirement so that they are recognizable by system users who do not have detailed technical knowledge. Functional requirements are linked with specific functions, tasks or behaviour the system must do while non-functional requirements are constraints on several attributes of these functions or tasks.

4.4/ SysML Model of Case Study

Vehicular traffic is continuously growing around the world, particularly in large urban areas. The resulting congestion has become a major concern to transportation specialists and decision makers. The passage priority associated with eventual changing way would produce bottlenecks. The solution adopted by traffic operators to manage distribution is signalized systems (tricolor and bicolor lights). The traffic lights installed in crossings are

used to adjust the vehicle movements. They are controlled by a system that synchronizes the color changes of the different junction lights. The traffic-light colors are managed by a controller that depends on the number of vehicles waiting to cross the junction. The time of each phase and the duration of a cycle lights (red-green-yellow) is specified by the traffic center of the town. This center supervises all street intersections. Figure 4.3 illustrates the crossroads environment [Berrani et al., 2013].

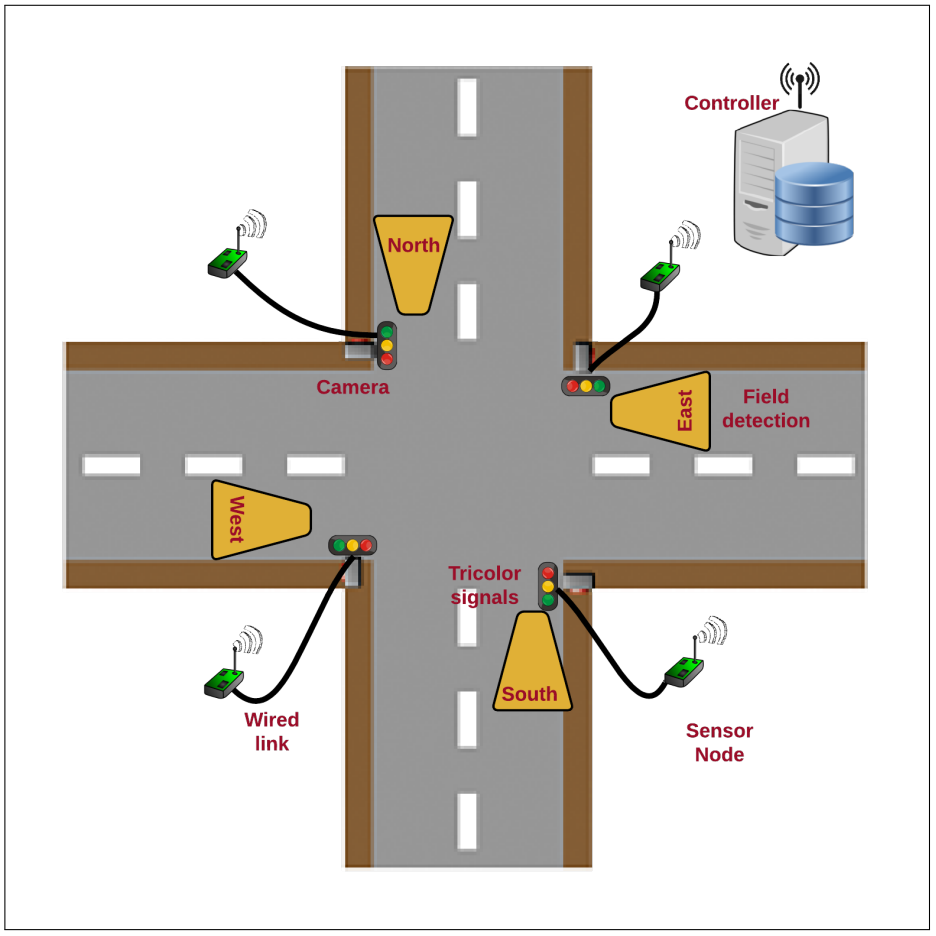

Figure 4.3: Crossroads system environment

The conception of this work, we focused on the Wireless Sensor Network (WSN) energy consumption. For that, we must analyze our case study to concentrate on the study of this parameter. The retained hypotheses that enable us to classify and simplify the studied system are the following:

1. The sensors used on the roads aim to control the traffic lights and interact with the controller.
2. The traffic light colors are red, green and yellow.
3. The system failures are not treated to simplify the case studied here.
4. The electricity network gives energy to the video traffic detection camera and the tricolor signal lights.

5. The video traffic detection camera, detects the vehicles at a distance that is fixed through the devices installations.
6. The image sensor can estimate the vehicle numbers waiting for cross the junction.
7. The pedestrian crossing and the communication between the controller and the traffic center are not studied to simplify the case considered here.

We assume that the crossing has two roads (North with South and East with West) in both directions. The four traffic-light units are designated such as North_Light, South_Light, East_Light and West_Lights, according to figure 4.3. The system specifications are as follows:

1. The system must be economic: The energy consumption should be minimized.
2. The system must be compatible with traffic laws: The control design should be according to the current traffic laws.
3. The system must be efficient:
   - If the light is green, it changes to the yellow color only if there are approaching vehicles on the other road and its light time is completed.
   - All sensor messages pass through a controller.
   - The period of the traffic signal depends on the number of vehicles waiting on every road.
   - This period should be long enough for finishing the queue of vehicles.
4. The system must be safe:
   - The traffic North_Light and South_Light are in the same color, and it is the same for East_Light and West_Lights.
   - When North_Light and South_Light are green, the East_Light and West_Lights are red.
   - When North_Light and South_Light are red, the East_Light and West_Lights are green or yellow.
   - When the controller sends a demand to change color, all crossing lights should change their color concurrently.

WSN Modeling

The WSN modeling is motivated by many factors, which include mistake sensitivity, scalability, sensor network topology, production costs, operating environment, hardware constraints, transmission media and power consumption. These factors are important because they serve to direct the design of the WSN protocols and algorithms. Also, they can be used to compare different WSN architectures.

Furthermore, the possibilities of WSN applications are supported, but the difficulties that platform for display are not limited various and not complexity also. Among the fundamental issues, which the WSN non-functional properties represent, we can specify the energy consumption, the communication security, the automatic configuration, etc.

Through this project, we study the WSN energy consumption that is a large extent dependent on the prototype of the node. These nodes are designed to the plan to maximize their life expectancy. In [Kossiakoff et al., 2011, Odey et al., 2012], the authors described an energy consumption model for WSN. The basic feature of this model is its accuracy in evaluating the energy consumption. Therefore, this model allows the estimation of the overall lifetime of the WSN exactly. For this design, we adopted this energy consumption model in our study.

To model the power consumption of WSN, we have simplified the present power consumption characteristics with the accessible. Power consumption defined according to a similar and linear function in Primary states ``i'' and when in changes between two states ``i'' and ``j''. The energy ``E_i'' consumed through a single Evening to Primary state ``i'' rely on the power consumption ``P_i'' of the implicit electronic circuitry and the time ``T_i'' spent in that state and were modeled as:

$$E_i = P_i \times T_j$$

The transmit state energy consumption model ``E_{tx}'' is given as:

$$E_{tx} = P_{tx} \times T_{tx}$$

The receiver is active and receiving the data packet from the transmitter some distance away. We model the energy consumed when a receiver is active ``E_{rx}'' as:

$$E_{rx} = P_{rx} \times T_{rx}$$

The energy consumption in idle state ``$E_{idle}$'' is modeled as ``E_{tx}'' and ``$E_r$'' but in the absence of payload overhead or decoding cost as in ``E_{tx}''and ``E_{rx}''.

$$E_{idle} = P_{idle} \times T_{idle}$$

To get a complete energy consumption model for the transceiver, this energy consumption should also be factored into our calculation. Energy ``E_{rx}'' in sleep state is given as:

$$E_{slp} = P_{slp} \times T_{slp}$$

We present the Transceiver energy consumption as an aggregation of the energy consumption of the basic states (active and sleep) and the transition states. We present the transceiver energy consumption ``E_{transc}'' as:

$$E_{transc} = E_{tx} + E_{rx} + E_{idle} + E_{se} + E_{slp}$$

Our complete transceiver energy model is shown in 4.8

The Requirement Diagram

This diagram is employed to represent the requirements of the designed system. From the case study, we have identified two main requirements that relate to the safety and the longevity of the system. Figure 4.4 shows the requirement diagram of monitoring junction system. For example, the element (Id=R1) expresses that the traffic lights on both ways that form the junction are different all the time.

The constraint that describes the requirement is considered like an invariant of the block ``Controller_System''. To show how the requirements of the ``CrossRoad'' system are presented in an RD. The RD that contains three requirements ``CrossRoadRequirement'', a test case ``StateMachine_Light'' represented as state machine diagram. Moreover, the block ``Controller_System'' which represent the traffic light system. In this diagram, we show that the requirement ``CrossRoadRequirement'' is composed of the two ``Traffic lights colors'' and ``No_Deadlock''. The state of ID: ``R1 and R2'' are satisfied by the block ``Controller_System'' and verified by the state machine ``StateMachine_Light''.

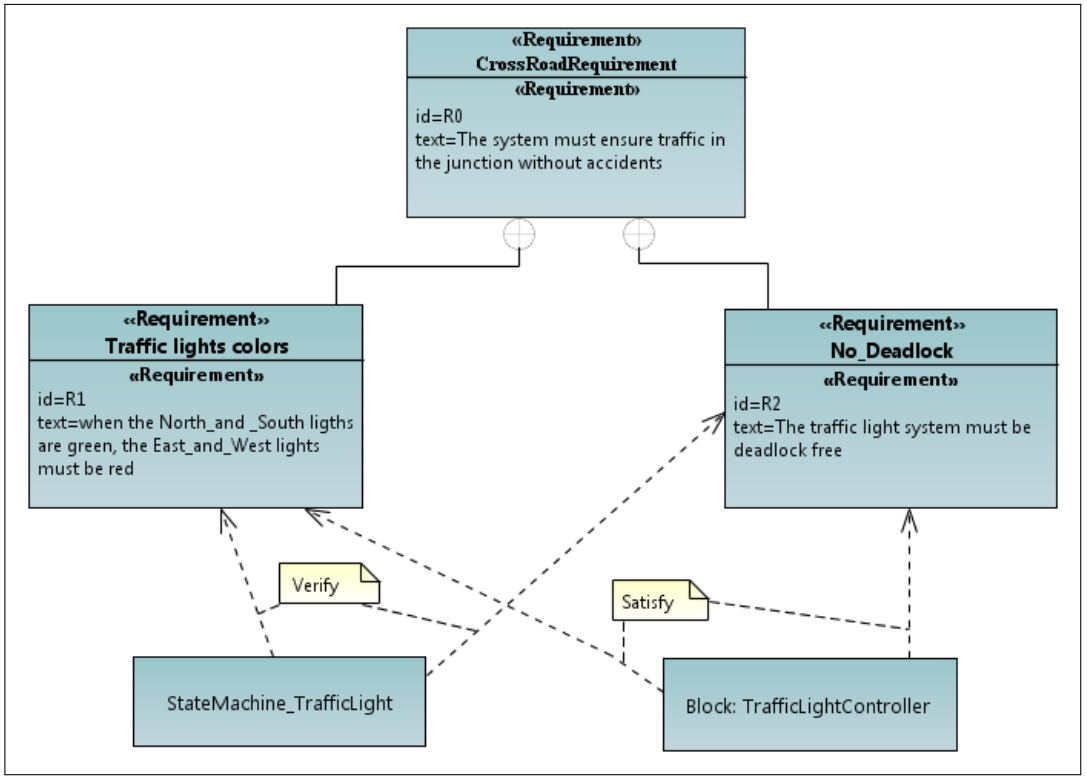

Figure 4.4: Crossroads System Requirement

The Block Definition Diagram

The block definition diagram provides a structural description of the studied system. The main block that represents the cross-road monitoring system consists of six blocks. The first block ``Crossroads'' is the most abstract level of the modeling. It represents the system as a whole. Moreover, it is composed of four sub-blocks ``Controller_System, Vehicles, NorthandSouthLights and EastandWestLights'' and sub-sub blocks ``Timer, Road_-

Sensor, and Camera''. Figure 4.5 illustrates the crossroads top level modeling.

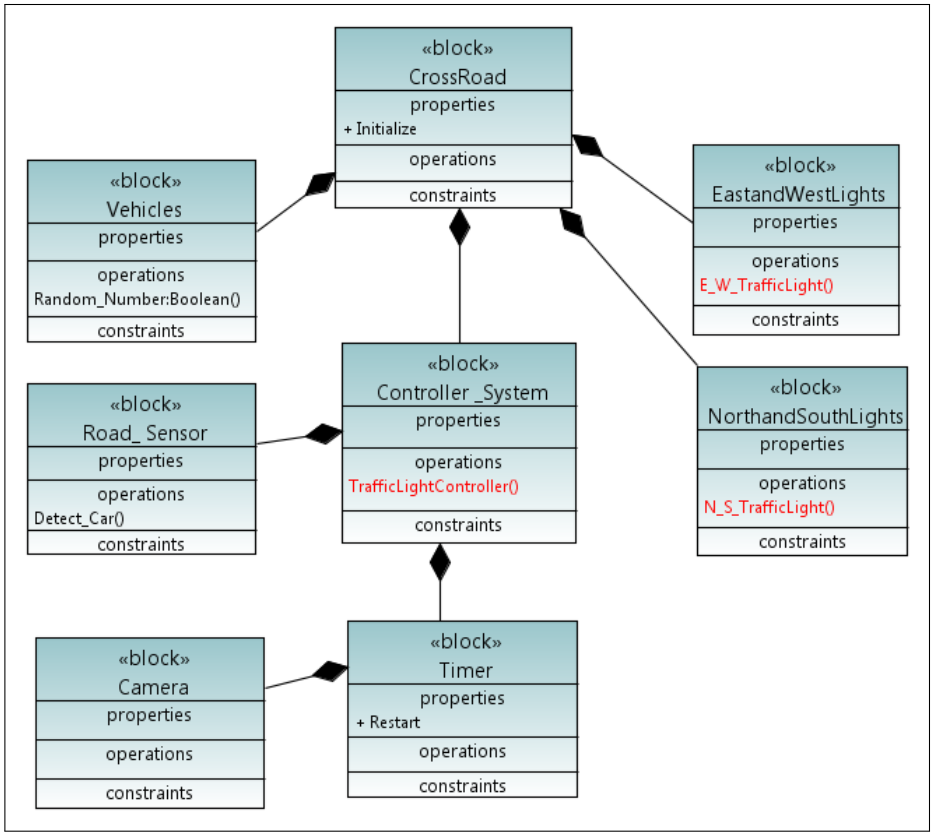

Figure 4.5: Global system structure

This outspread sensor node consists of a simple sensor node, an additional sensing unit ``camera'' and an actuator ``traffic lights''. The simple sensor node block includes several units ``Senser_Memory, Sensor_Processing, Sensor_Battery, and Sensor_Radio'' with sub-block Transmitter and Receiver. Figure 4.6 shows the constraint block of ``Transcevier_-Energy'' of sensor environment.

The Internal Block Diagram

The IBD is a white box view of a block. It explains the system internal structure in terms of ports, parts and connectors. These parts are joined by connectors that connect their ports. IBD represents the internal formation of the Crossroads block. As shown in the diagram, the port management allows moving the direction of Controller System and the Port of other parts i.e. ``NorthandSouthLights, EastandWestLights''. Figure 4.7 shows the IBD diagram.

The Parametric Diagram

The PD is a SysML explicit modeling method that enables the integration of constraints or equations into the model in the analysis plans. These constraints are specified by parameters and rules that explain the evolution of these parameters associated with each other.

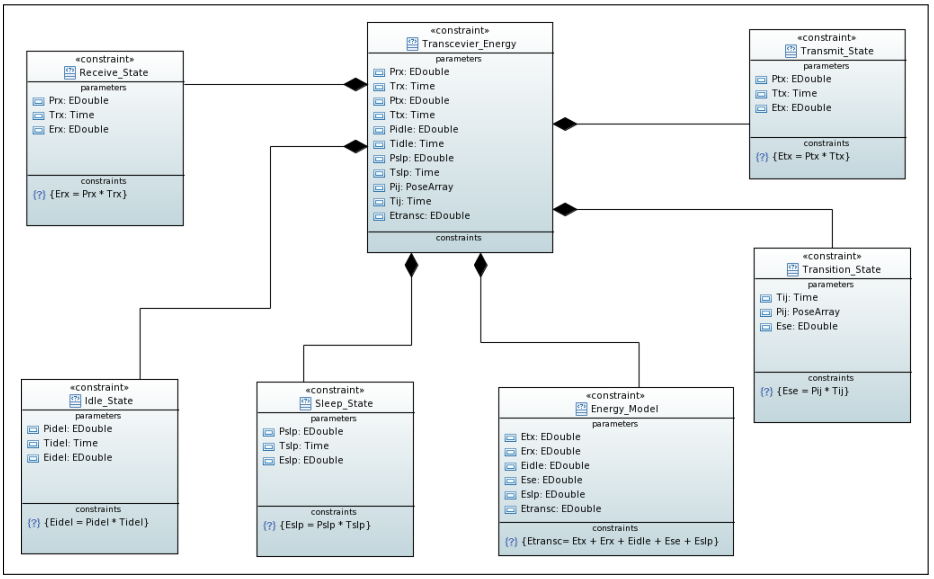

Figure 4.6: Sensor Block constraint

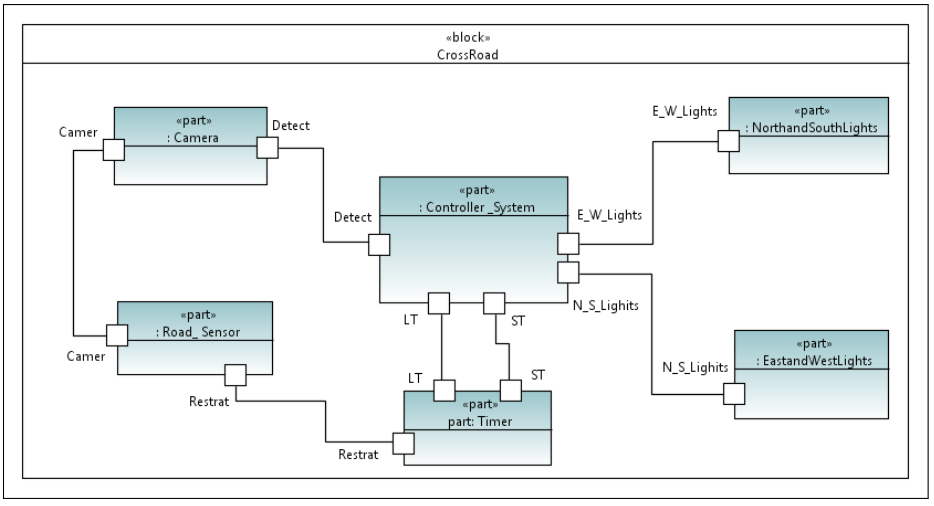

Figure 4.7: IBD of Crossroads

The main goal of our outline is to study the WSN energy consumption. So, we adopted the energy consumption model proposed in [Kossiakoff et al., 2011].

In the framework of this model, we can distinguish several sources of energy consumption, such as the transmitter, the receiver, the sensor, the processor, the memory, and the actuator.

We assume that the electricity networks supply the actuator (traffic light unit) and the sensor unit (additional sensing unit-camera). However, we disregard the energy consumed by the processor because we have not identified operations such as a requisition, the processor does compressions or treatments.

We have also disregarded the energy consumed by the memory because the sensor node does not save data at its level. However, we maintained the energy consumed due to the

data transmission, the data reception and the changing of the operating mode transient energy. In Figure4.8, we present the parametric diagram transmitter that illustrates the energy consumed by this element. Also, this diagram includes the transmitter of transient energy.

Moreover, this diagram includes the communication model that explains the state of the transmission channel between the sensor node and the controller. Further, the parametric diagrams of the processor, the memory, the receiver, and the battery are defined as the parametric diagram transmitter.

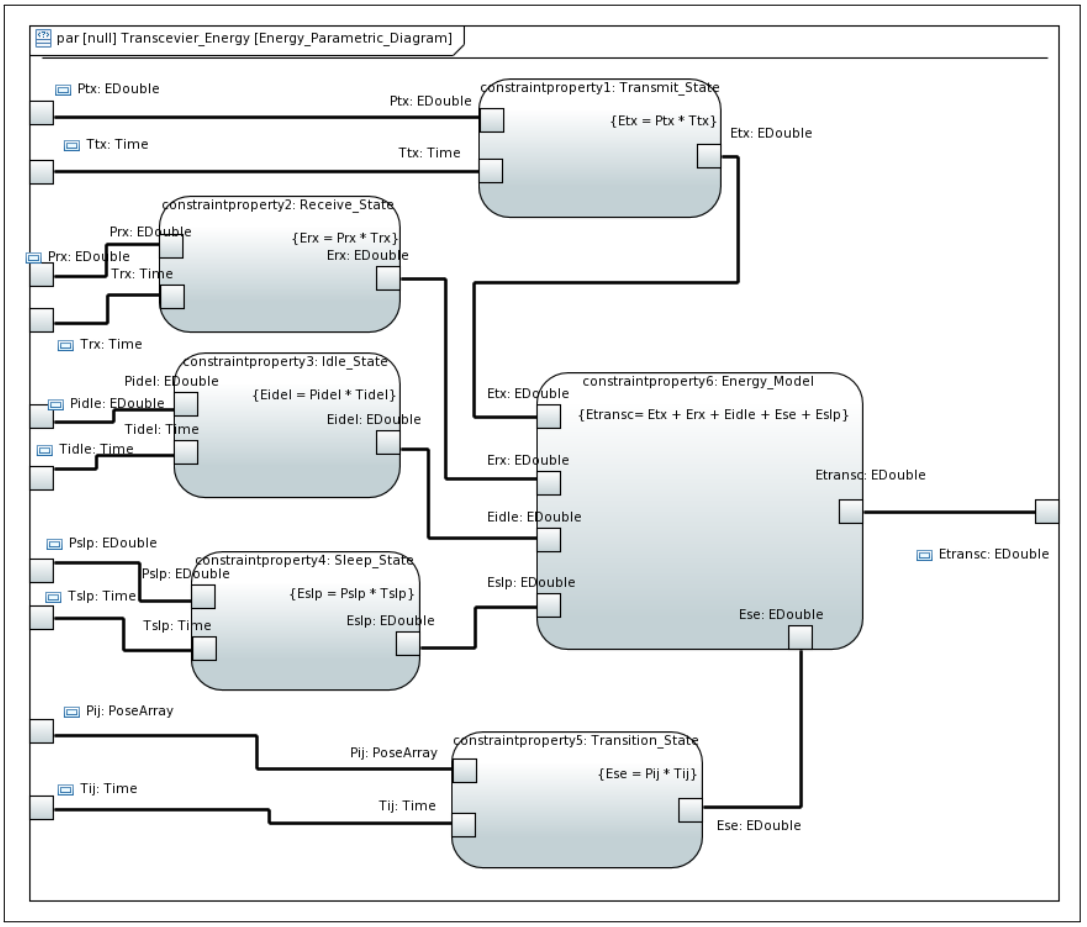

Figure 4.8: PD Sensor constraint

The State Machine Diagram

The SMD describes the behaviour of the SysML block using a state automaton, and it presents the potential sequences of states and actions that a block can handle during its lifetime in reaction to discrete events signals. The behaviour of the``Controller_System'' block is described through SMD. The diagram shows the state of operation ``TrafficLight-Controller''.

The default state is the next state if no transition condition is satisfied as shown in state ``Init_State''. If the state has any unconditional transition, then assigning default state to next state is omitted as shown in case ``NS_G_State''.

Figure 4.9 shows the SMD diagram. The timer for the crossfires is managed by a controller system at each start up. However, the timer initializes a clock that measures the duration of each crossfire color ``red (38 Sec.), yellow (5 Sec.), and green (33 Sec.)''. Then it sends the value to NorthandSouthLights and EastandWestLights, based on the given entries ``NorthRed, North Yellow,.. WestGreen''.

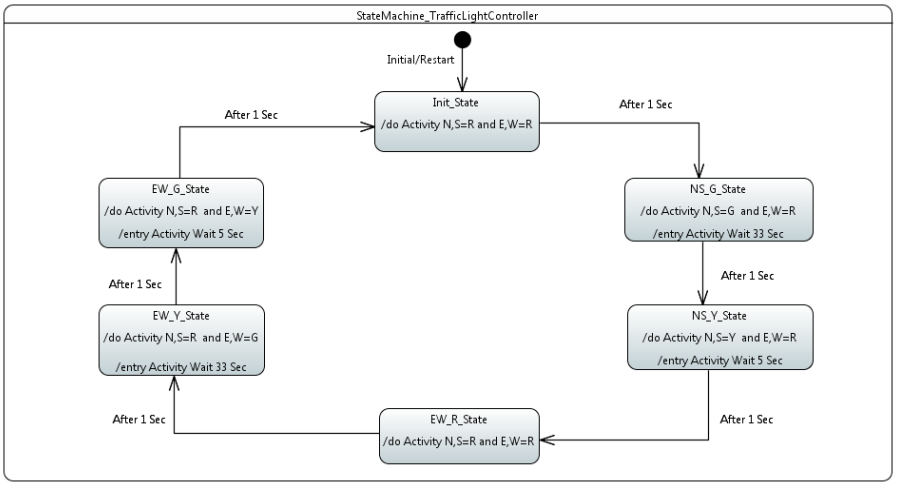

Figure 4.9: SMD of Controller System block

In our case study, we have characterized the operations done by the sensor node components to count the consumed energy. For this, we suppose that the sensor node processing unit receives the lane status number of vehicles on the road from the further sensing unit camera. Following, the processor forwards this data to the transmitter (RF) to send it to the controller.

However, once the receiver (RF) receives a message from the controller, the message will be send immediately to the processor. Then, the processor extracts the command signals from the receiving message and sends them to the actuator trace-lights. During our study, we assumed that for each sending or receiving message, the processor changes its state at least once. This supposition allows us to calculate the processing unit transient energy.

4.5/ Conclusion

We have shown in this chapter, how to semi-formal model except for the SMD specify systems. We created a SysML model which describes the structure and the behavior of the environment and the system. The issue has discovered the requirement properties of the system. Then, we used a refinement relation between SysML system blocks, described by structural and behavioral diagrams. The refinement in SysML is an essential concept, and it is based on the development of a process from an abstract level towards more detailed levels.

III

Contributions

5

Simulating SysML Specification using SystemC

Contents

5.1/ Introduction

The complexity of heterogeneous systems has increased during last years. One challenge of designing these systems is to deal with the application of methodologies based on MDE. Complex systems can be built through different model transformations of their descriptions. In our case, we use SysML as the starting point to describe systems in a high-level of abstraction, and SystemC language, as the target, is chosen as an alternative to the traditional languages, because it has a simulation kernel that is an important aspect that allows the designer to evaluate the system behaviors through simulations. In this chapter, we propose a methodology to validate complex systems specified with SysML language by translating them into SystemC models.

The main objective is to describe a methodology to model and enable the analysis and validation of systems. The models permit the specification of requirements, structure, and behaviors of a system. They may be used to validate the characteristics of some parts or the overall designed system, e.g. its functionality or its performances. In this section, we define system validation by simulation after translation in SystemC of SysML specifications.

The proposed approach is decomposed into three stages:

- First we use the Papyrus modeling tool to create the SysML diagrams ``BDD, IBD, PD, and SMD'' for specifying the complex system requirements, structure, and behavior.
- Second, based on MDE, we propose to use ATL (meta-model based transformation language) to automatically transform SysML models into SystemC models. More-

over, we propose to use Acceleo to generate SystemC code from the SystemC model previously generated.

- Finally, we run the simulation of the SystemC code in order to obtain execution traces. These traces are used to validate the system.

Figure 5.1 summarizes the main steps of the proposed approach.

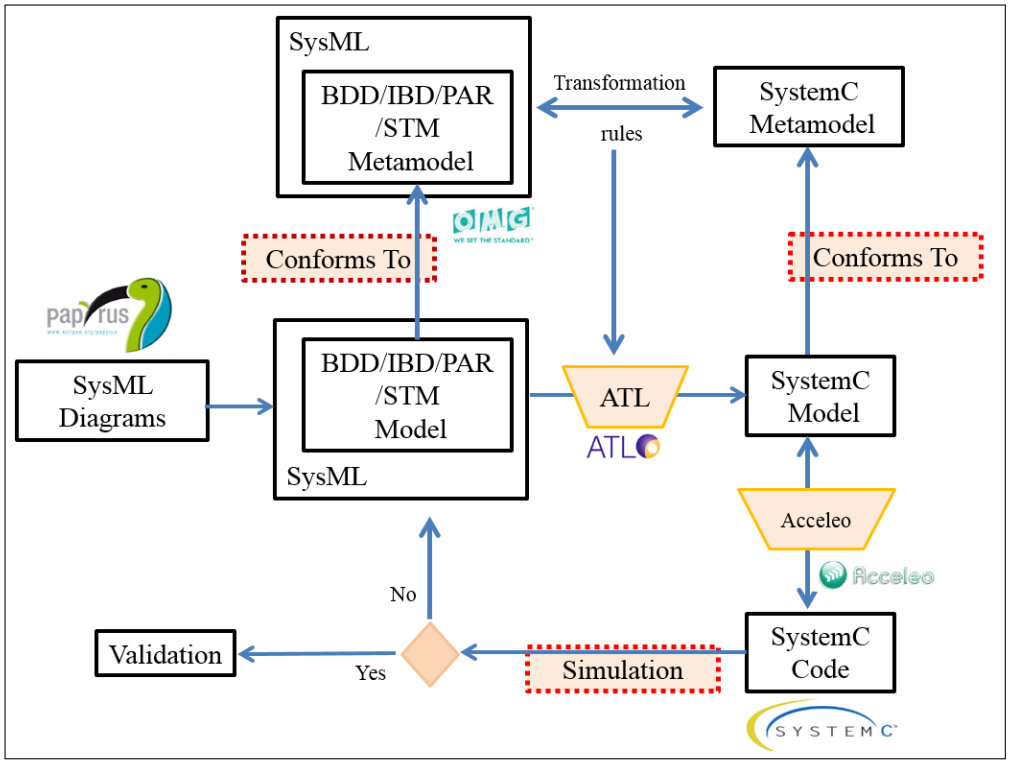

Figure 5.1: Methodology flow for SysML to SystemC transformation approach

5.2/ From SysML to SystemC

In this section, we propose a MDE transformation approach to map SysML into SystemC environment. We focus on defining how and which BDD, IBD, PD, and SMD elements are transformed to formalize behavioral aspects and to preserve the structural representation of the system. Then, we define the mapping rules used to create a SystemC model from the SysML BDD, IBD, PD, and SMD.

5.2.1/ Model /MetaModel Transformation

Model transformation represents the heart of an MDE activity in the development process. A model represents one or more particular aspects of the system under design, maintenance or in an operational context. A model is written in the language of one particular metamodel. The metamodel acts as a candidate to extract some relevant aspects from a system and to disregard all other specifics. There are many possibilities to define

metametamodel. Ordinarily, the definition is reflexive, i.e. the metametamodel is self-defined.
The metametamodel layer forms the foundation for the metamodeling architecture. The primary goal of this layer is to define the language for specifying metamodel. A metamodel is conform to the metametamodel if and only if each of its elements has its metaelement defined in the metametamodel. For example, metametaobjects in the metametamodeling layer are MetaClass, MetaAttribute, and MetaOperation [Othman et al., 2013]. The source model is transformed through ATL transformation rules to generate the target model. Figure 5.2 abstracts the main steps of our approach, and gives an overview of the process of model transformation. In a simple scenario of a model transformation, some transformation rules are defined with respect to both source and target metamodels [Czarnecki et al., 2006]. These transformation rules define the mapping between the model elements of both metamodels. The model transformation engine then reads the source model that is compliant to the source metamodel and executes the transformation rules to create the target metamodel that conforms to the target metamodel. The

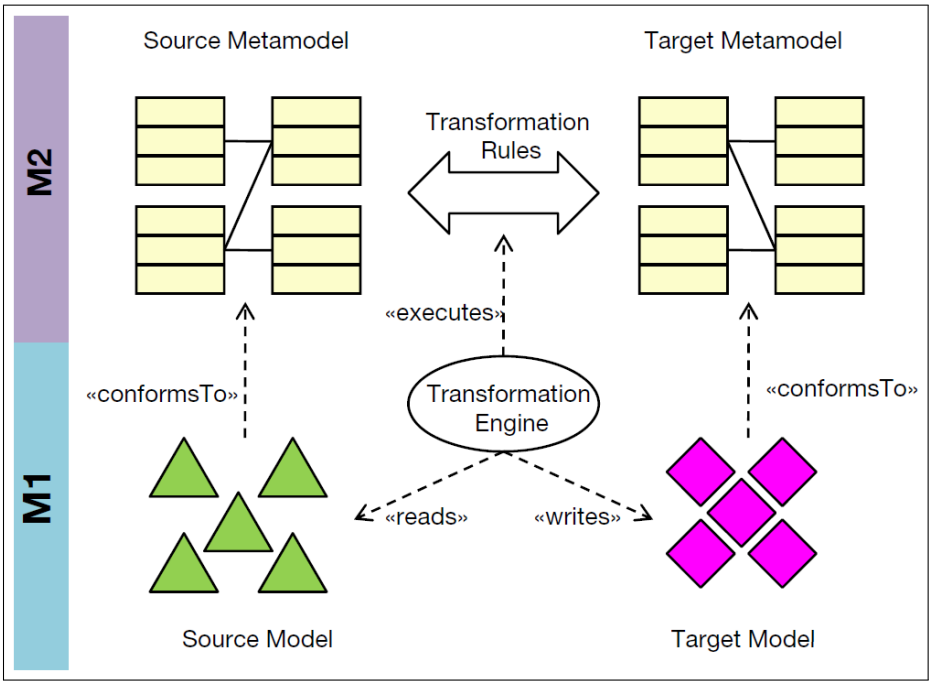

Figure 5.2: Model Transformation approach

model is an abstract view of reality and conforms to a metamodel that precisely defines the concepts presented at this level of abstraction. The metamodeling method means that a metamodel is used to specify the model that consists of SysML diagrams with SystemC environment.

5.2.2/ SysML Meta-Model

SysML meta-model is defined in terms of its underlying UML, on which SysML is based. With the adoption of UML in the SysML specification, UML class diagrams have been used to produce the SysML metamodel diagram throughout this section. These diagrams are the same as if the would be generated if using SysML. The metamodel itself is associated

with the modelling elements within the SysML, how they are constructed and how they relate to one another. The full UML metamodel on which SysML is based is highly complex and, to someone without much SysML or UML experience, can be completely impenetrable. Metamodels are displayed in highly simplified versions of the actual metamodel to support information and to group different phases of the model according to each diagram.

5.2.3/ SystemC Meta-Model

A SystemC metamodel is structured by using modules which are derived from the base class ``sc_module'' [MKuster et al., 2012]. The module can interface with other modules by ports. These ports can be interconnected using channels. A port requires an interface that must be provided by the external channel that is bound to the port. An export, on the other hand, provides an interface which is implemented by the internal channel that communicates to the export. One or more processes define the behavior of a module. As can be seen in Figure 5.3, all building blocks for the module hierarchy are derived from the base class ``sc_object''.

5.2.4/ Model Transformation Technology

The general process of our approach consists of several stages and the modelling with SysML diagrams that will be the source of model transformations. In this work, we consider the transformation of four diagrams: BDD, IBD, PD, and SMD. Principal, run is based on model transformation ``Model2Model'', by exploiting the ATL language [Vieira et al., 2014]. ATL tools implemented in an Eclipse plugin support the fundamentally associated tasks: editing, compiling, executing, and debugging. The application of this methodology with ATL is mainly based on:

- The definition of the source and target metamodel.
- The definition of transformation.
- The definition of the source model that conforms to source metamodel.

An ATL transformation module has many input models and one output model particularly. It contains some of the rules that describe the mapping from source elements to target elements. ATL defines two different types of rules the ``called rules'' and the ``matched rules''. A matched rule allows matching some of the model elements of a source model and to generate from them some distinct target model elements. Compared to matched rules a called rule has to be invoked from an ATL imperative block to be executed. ATL necessary code can be defined within either the action block of matched rules or the body of the called rules. In ATL, models and model types are bound to concrete models and metamodels at run time. ATL does not perform any type checking at compile time. Just at run time, ATL resolves meta-classes and properties by their name in the bound metamodel. An ATL is composed of two sections:

- The ``from'' section, which defines constraints on the source element.
- One or more ``to'' sections, which defines how target elements are initialized from source elements.

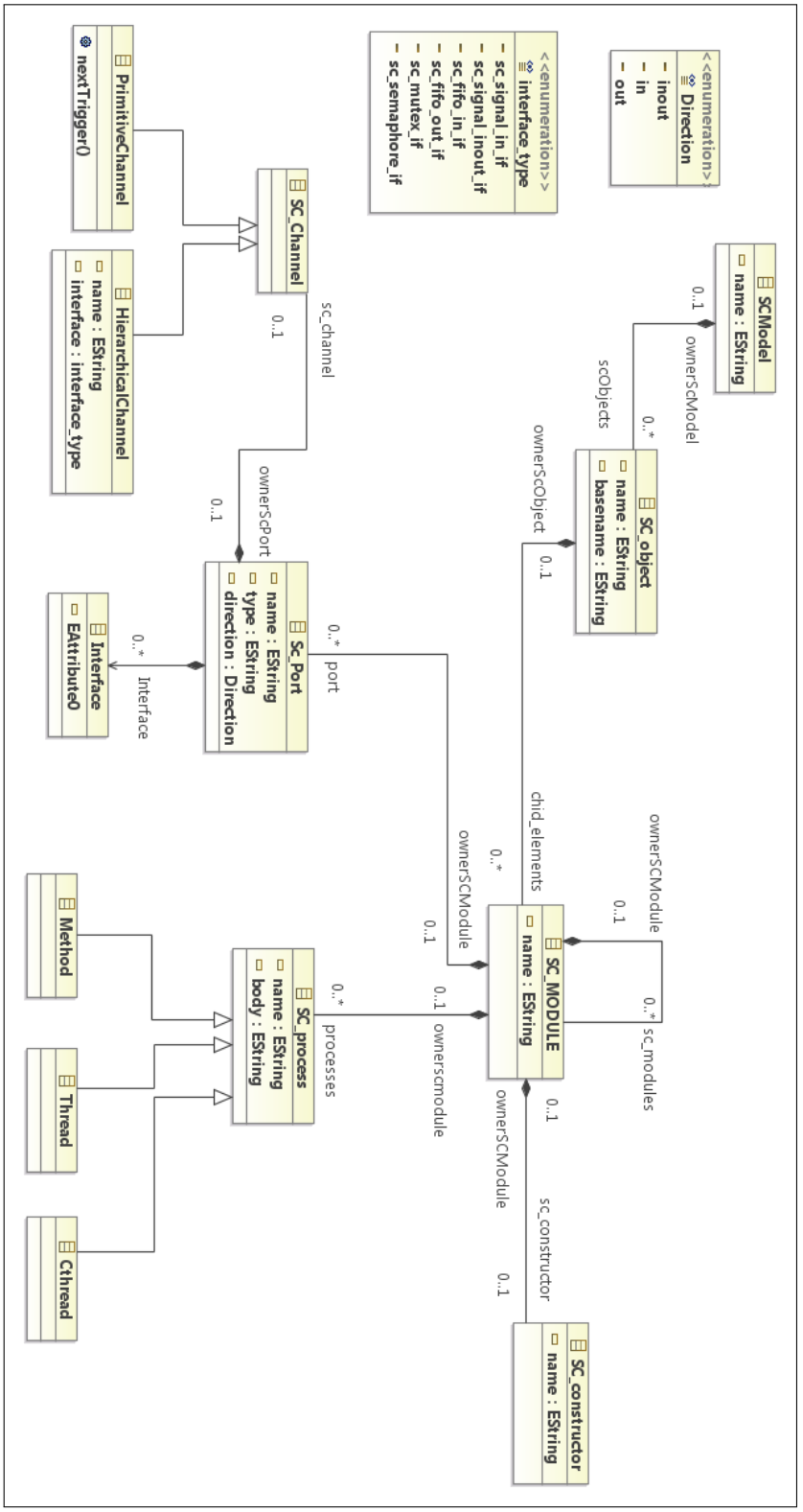

Figure 5.3: Metamodel of SystemC Model

5.2.5/ Transforming SysML into SystemC

UML and SysML models element are the sources to generate SystemC model elements. The ``SysML2SystemC'' module has one output model named "OUT" of model type ``MM-SystemC'' and one input model ``IN'', which is also of model type ``MMSysML'' and ``MMUML''. Though creating a target item ``from'' source part, ATL retains a traceability link between the two elements. This link is used to initialize a target item in the ``to'' match as seen in Listing 5.1.

Listing 5.1: Rule UML/SysML to SystemC transformation module

```
-- @nsURI MMSysML=http://www.eclipse.org/papyrus/0.7.0/SysML
-- @nsURI MMUML=http://www.eclipse.org/uml2/3.0.0/UML
-- @path MMSystemC=/22-07-ATLpro/Metamodels/SystemC.ecore
module Transformation;
create OUT : MMSystemC from IN : MMSysML, IN1 : MMUML;

rule Model2SCModel{
        from sysml: MMUML!Model(
        sysml.oclIsTypeOf(MMUML!Model)
        )
        to scModel: MMSystemC!SCModel(
                name <- sysml.name
        )
}
rule Package_BDD2SystemC_Main {
 from
          BDD :MMUML!Package(
                BDD.oclIsTypeOf(MMUML!Package)
           )
 to
          Top : MMSystemC!SC_object(
                  name <- BDD.name,
                  ownerScModel <- BDD.getModel()
        )
}
......

rule StandarPort2SystemC_SC_Port{
from
         standarport : MMSysml!FlowPort
to
        sc_port : MMSystemC!Sc_Port (
                name <- standarport.name,
                type <- standarport.type.name,
                direction <- standarport.direction,
                ownerSCModule <- standarport.owner
        )
}
```

5.2.6/ Rules for Transformation

The SysML diagrams are created using the Papyrus [Lanusse et al., 2009] tool. Papyrus is a graphical tool that captures SysML diagrams. We begin with a system description given

by the RD, BDD, IBD, PD and SMD. These will be the input of our transformation. This model will describe the requirements, structural and behavioral information about the system.
For the target language, in this case SystemC, there are different possible translations of the considered semantics with the behavioral concepts of SystemC environment. The following translation rules were chosen:

1. Structural view by a SysML BDD and an IBD of the top-level block used to encapsulate the overall hierarchical design. Moreover, the IBDs for the design of each compound block synchronization with the associated BDDs for the block types definition. The basic mapping between SysML and SystemC is:
   - SysML Blocks → SystemC Modules.
   - SysML Flow Ports → SystemC ports.
2. Behavioral view by a SysML SMD of the overall system functionality associated with the top-level block to model input, output, sequences, and conditions, for coordinating the inner blocks behaviors.
3. A SystemCThreads are used to allow parallel states activation semantics.
4. A boolean signal represents the activation of each state, more than one state can be active at the same time.
5. A SystemCThreads are sensitive each one to another by notification to represent every event trigger that can fire transition from the associated state.
6. The variable last trigger that identifies the last trigger fired as an enumerated value is instantiated.
7. The trigger fire is implemented as an event notification and as a change of the last trigger variable.
8. In SystemCThreads, condition statements are used to represent the guards used in SMD.

All the states are sensitive when a trigger is released, and the active triggered transition is executed if the corresponding guarding conditions are true.

Through these concepts, we may apply basic mapping between SysML elements and SystemC elements. This mapping is defined in the Table 5.1.

5.2.7/ SystemC Model Transformation to SystemC Code

Acceleo is a tool that implements the MOF Model to Text Transformation Language (MOFM2T) standard [Specification, 2008]. MOFM2T is a transformation language that takes some structured model as input and produces a textual output. Acceleo is a code generator that enables to generate structured file from an EMF model [Lazăr et al., 2010]. The output is a text that can be a programming language or other formalism. Acceleo requires defining an EMF metamodel, and a model conforming to the metamodel that

Table 5.1: Mapping between SysML BDD, IBD, PD, and SMD with SystemC

SysML elements	SystemC Model
Package, constraint Block	Package, Module-Core, Class
Requirement	Boolean expression, Assertion
Flow-Specification, Value-type	Channel, type
Flow-Property, Flow-port	Interface- Channel
Connector flux	Equation channel
Constraint Property	Equation
Operation	Event, Processes
State	State of the process(case statement)
Peudostate	State of the process(condition statement)
Transition	Action of the process(action statement)
Do activity	Action of the process(event statement)

will result in text. The transformation language is defined using templates. For example, Acceleo can take an Ecore model generated by SystemC as input, and generate native SystemC application code by template transformation.

We have the metamodel and model of SystemC for the purpose of code generation. In the first line of Acceleo code, we import the metamodel so that the generator knows the structure of our model. The concept to define Acceleo is called template, and it is the smallest unit identified in a template file. To allow setting the main reference for the workflow to collect information from the necessary to model code generation.

To perform code generation from models, we have used the Acceleo technology. As seen in Listing 5.2, this language uses an approach based on templates, which can be seen as a piece of code that creates reserved namespaces containing expressions on entities. The source models are being used to inject model information in predefined templates. They are ordinarily implemented as modules of meta code and expansion rules, that select and print string reverse with valid code semantics or non-executable text such as XML.

5.3/ Validation by Simulation

Simulation approaches are at the heart of many methodologies. Simulation techniques are traditional and beneficial tools for debugging, verifying and validating systems. They are implemented sequentially at each phase in the design flow. A set of simulation models is constructed to represent behaviors of various components or the whole system. Through the implementation of these simulation models, the result values for given inputs patterns are created and observed. The correctness and quality of output values are evaluated to ensure that specified requirements have been affected in the models. These results can also help designers to explore and trade off between different designs alternatives through simulation experiments. The behavior of the simulation does not depend on the distribution in which the processes are executed at each step in simulation time.

Listing 5.2: Acceleo which produces the output SystemC module

```
[comment encoding = UTF-8 /]
[*** The documentation of the module generateSCModel.
*/]
[module generateSCModel('http://www.femto-st.fr/disc/systemC.ecore')]
[file (aSCModel.name+'.h', false, 'UTF-8')]
[for(aSCObject : SC_object| aSCModel.scObjects)]
......

[template public generateSCObject(aSCObject : SC_object)]
public SC_module [aSCObject.name/] {
 [for(aSCModule : SC_MODULE | aSCObject.chid_elements)]
          [generateSCModule(aSCModule)/]

[/for]
}
[/template]

[template public generateSCModule(aSCModule : SC_MODULE)]
SCModule [aSCModule.name/] = new SCModule();
[for(aSCModule : SC_MODULE | aSCModule.sc_modules)]
         [generateSCModule(aSCModule)/]
[/for]
[for(aSCPort : Sc_Port | aSCModule.port)]
         [generatePort(aSCPort)/]
[/for]
[for(aProcesses : SC_process | aSCModule.processes)]
         [generateProcesses(aProcesses)/]
[/for]
[/template]

[template public generatePort(aSCPort : Sc_Port)]
sc_[aSCPort.direction/]<[aSCPort.type.toString()/]> [aSCPort.name/];
[/template]

[template public generateProcesses(aProcesse : SC_process)]
sc_[aProcesse.name/]<[ownerscmodule.ownerSCModule/],

[/template]
```

5.3.1/ SystemC Simulation

The SystemC simulation kernel relies on the notion of delta cycles. A delta cycle is composed of an evaluation phase and an update phase. The kind used for modeling primitive channels cannot change immediately. By dividing the two stages of assessment and update, it is possible to guarantee determinism. The event happens at given simulation time. The time starts at zero and moves forward only, time increments are based on the default time unit and the time resolution. Three main concepts are being used as following:

- Initialization: is the first step in the SystemC scheduler. Each process is performed once during initialization, and each thread process is executed until a wait statement is encountered.

- Elaboration: is defined as the execution of the ``sc_main()'' function from its entry point to the first invocation of ``sc_start()''.

- Simulation: SystemC simulator regulates the timing and the order of process execution, deal with event notification and manages updates to channels. The supports concept of ``delta cycles'', which consists of the execution of evaluation and update phases. The number of delta cycles for every simulation time will depend on the same simulation.

In SystemC, the featured event strategy typically uses three data structures from the state variables, the event list, and the clock. The simulation execution depends on two types of procedures by the called scheduling and the event handler routines. The scheduling operation is an important task since it creates and classifies events in time. A simulation process is intended to model an appropriate entity in the simulation with a well-defined specific behavior. The behavioral description of the object is encapsulated by the process, defining the actions performed by the process throughout its existence.

System-level design techniques have proposed to use high-level abstraction methods to design hardware and software concurrently in a unified environment. SysML model and simulation are key techniques to describe, validate, analyze and verify complex systems. In various SysML model and simulation approaches, the SystemC environment has become the real standard.

5.3.2/ SystemC Network Simulation Library

SystemC Network Simulation Library (SCNSL) [Fummi et al., 2008] is interposed between SystemC and standard C++ libraries and the classes that the developers will realize.
The SystemC libraries and SCNSL start from system specifications. The primary model can be described at different levels of abstractions as the RTL and the TLM levels or with combinations of parts of different level. On this initial model, we can build the verification systems that will establish the validity of the final product. From this initial model, the process continues to improve the system, developing the final product in all three domains. SCNSL is an efficient tool, both for flexibility and performances, in comparison to the ones already in existence.

5.3.3/ Traces Generation

The Value Change Dump (VCD) trace format [Windisch et al., 2013], is the only trace format supported by basic SystemC standard. This trace format records value changes of complex systems over time. The SystemC standard defines a programming interface or API (Application Programming Interface) to finish.

The processing is performed only once to generate a ``trace.vcd'' file containing the sequence of instruction and data addresses from execution, and this trace file is then used repeatedly as the input to the more detailed simulation. This file trace can then be used with the tool such as GTKwave to visualize the activity of the model in the form of timing diagrams. The main advantage of this trace format is to be supported by the majority of complex systems design tools.

5.4/ Experiments with the case study

In this section, we will discuss the case study presented in Chapter 4, proving the effectiveness of our approach to specify and validate the behavior of road intersection signals. The tooled approach, previously presented, was used. Figure 4.5 illustrates the crossroads top-level modeling. We have identified two main requirements that relate to the safety and the longevity for power consumption of the system in Figure 4.4 that shows the requirement diagram of monitoring junction system. The BDD includes six blocks the first block, named CrossRoad, represents the system. As a whole, it is composed of three sub-blocks (``Controller System, NorthandSouthLights, and EastandWestLights'')and sub-sub blocks (``Timer, Road Sensor, and Camera'').

The internal structure of the Crossroads block is represented by IBD. The diagram shows the flow ports, the port management allows continuous moving the direction of Controller System and the port of other parts (i.e. ``NorthandSouthLights and EastandWestLights''). Figure 4.7 shows the IBD diagram. In Figure4.8, we present the parametric diagram transmitter that illustrates the energy consumed by this element.

Moreover, the Figure 4.9 shows the SMD diagram. The timer for the crossfires is managed by a controller system that at each starts up. However, to initialize a clock that measures the duration of each crossfire color ``red (38 Sec.), yellow (5 Sec.), and green (33 Sec.)''.

5.4.1/ Combine SysML to SystemC

We will focus on how to perform and what are the parameters used to affect the implementation. The mapping methodology is used to create SystemC code from SysML diagrams. The Figures 5.4, 5.5 and 5.6 show the SysML2SystemC code.

5.4.2/ Simulation

After the SystemC code is successfully generated from the SysML representation of the Crossroads system (Figure 4.5, 4.7 and 4.9), the subsequent step is to simulate the generated SystemC design.

SystemC standard tracing facilities are based on VCD trace format. For communication, the action is recorded in the trace format. It is required that the exchange value be different from the current value of the thread or method, and the change in value has a duration of at least a delta-cycle. In Figure 5.7, we show the simulation results.

This simulation platform has been used to ensure that the packets are routed correctly to their destination under the Crossroads system. The simulation trace shows the state for each light as true or false values through the time. They can validate that no green light on North and South lights is turned on when there is also a green light on the East and West lights. It is highlighted by this timetable that the request packet and the response take different routing nodes at their destination.

The IDEA1 tool [Galos et al., 2013] is a discrete event simulator based on SystemC and C++ language. It is originally based on SCNSL existence in the IDEA1 simulation platform that can run simulations on heterogeneous sensor nodes that compose a network. We used

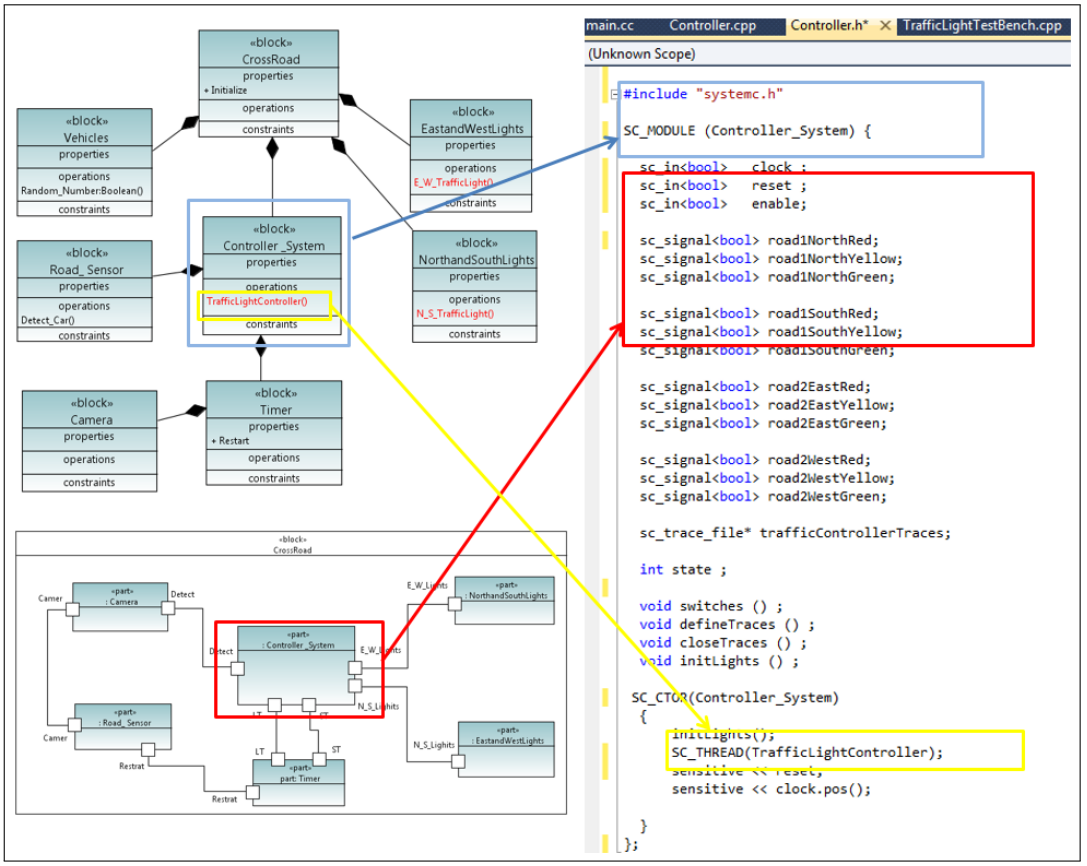

Figure 5.4: Code generation from BDD and IBD to SystemC

the IDEA1 tools to implementation WSN environment.
In Figure 5.8, we show that the behaviour WSN consists of four nodes and a wireless channel with control. That simulation of this test gives a VCD trace. The case study consist of four nodes, and one coordinator is deployed to compose a WAN network with a star topology.

All the nodes can directly communicate with the coordinator. It uses IEEE 802.15.4 slotted Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) algorithm to access the channel. The sensor nodes environment periodically get values every second and transmit data over the network. Each transmission includes two data bytes. Sensor nodes enter sleep mode as long as they can, and the coordinator is always awake.

We can monitor the coordinator nodes microcontroller, and radio frequency unit states Receive, Transmit, Active, Sleep CooMCUState stand for coordinator microcontroller state, Cooradiostate is the radio frequency coordinator state.

For classical nodes, states of microcontroller and radio frequency unit are also detailed with ``mcustate 0'' and ``radiostate 0'' for ``node 0'' and ``mcustate 3'' and ``radiostate 3'' for ``node 3''. The coordinator microcontroller is always Active. At time 410 ms, coordinator radio frequency unit sends a packet Transmit, node 0 radio frequency unit is in Receive mode, node3 is in power down mode 0.

Then, radiostate0 sends an acknowledgement Transmit, and then enters Sleep mode. As no more processing is required, the microcontroller of node 0 enters Sleep mode. Node

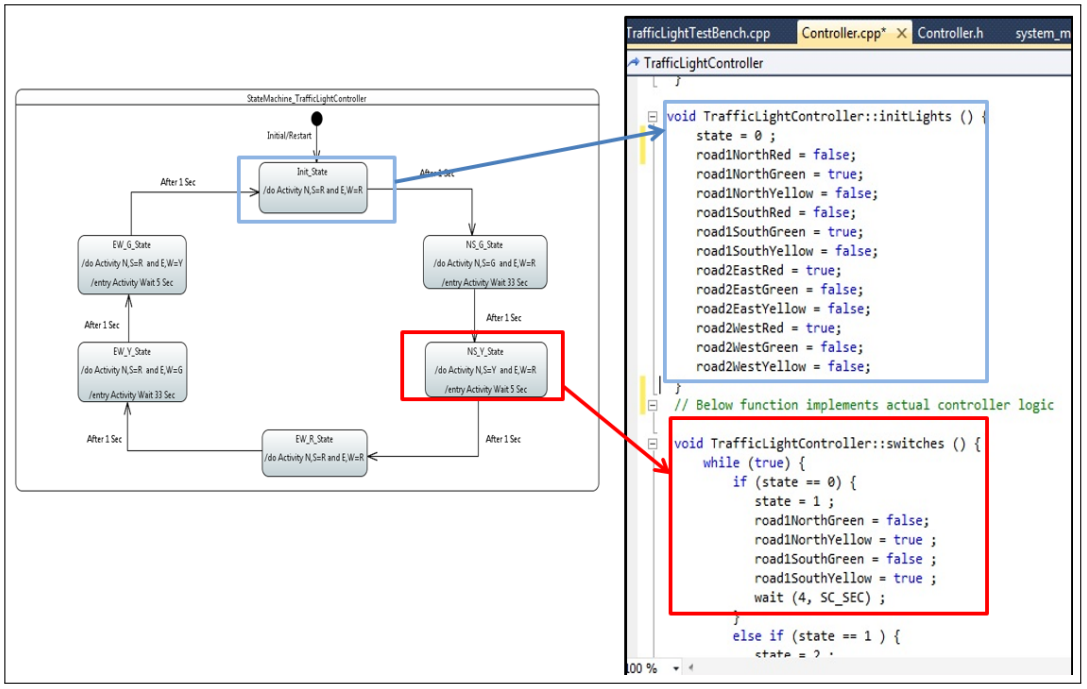

Figure 5.5: Code generation from SMD to SystemC

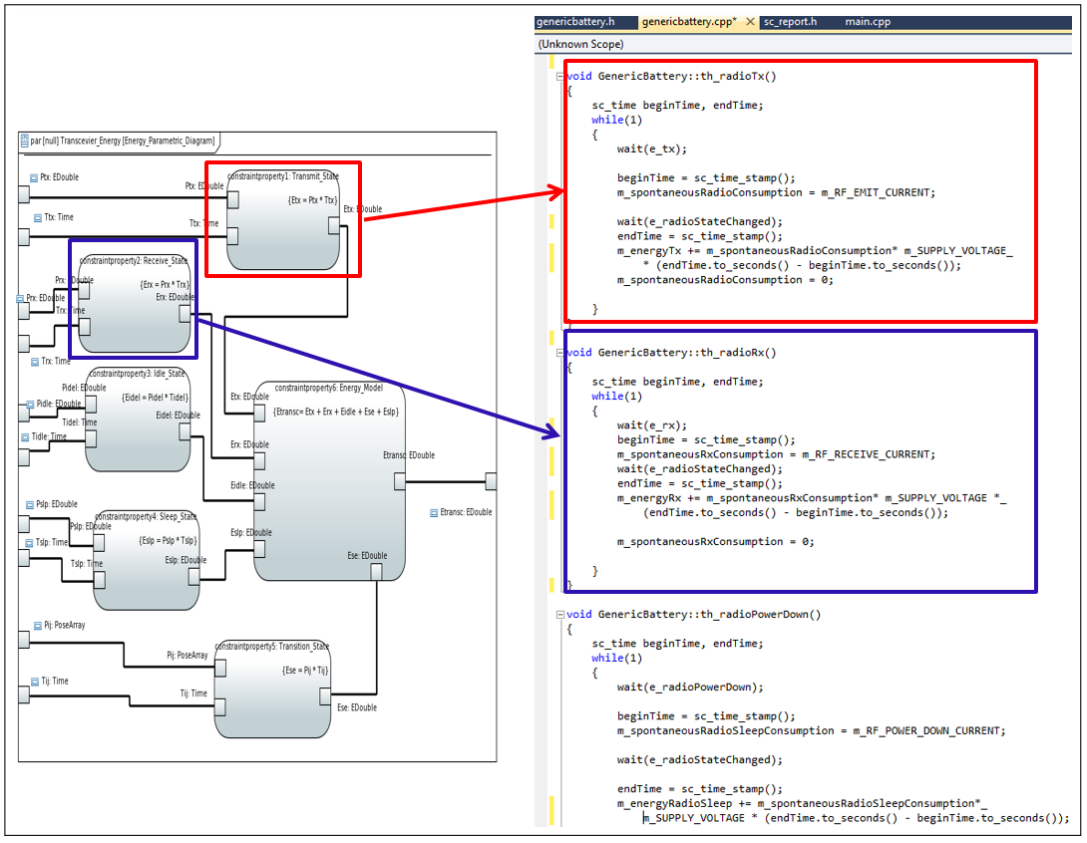

Figure 5.6: Code generation from PD to SystemC

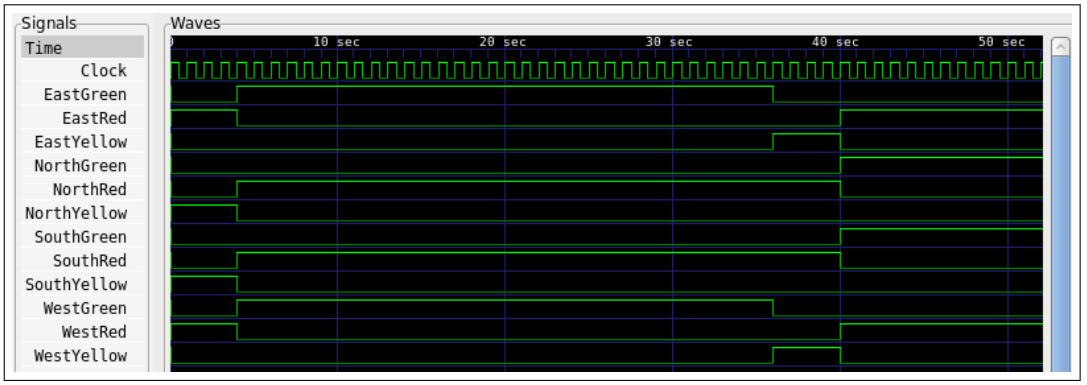

Figure 5.7: Timing chart showing the activity of a Crossroads simulation

3 wakes up at 420 ms. after a calibrating phase, the microcontroller is Active, radio frequency unit is in Receive mode.

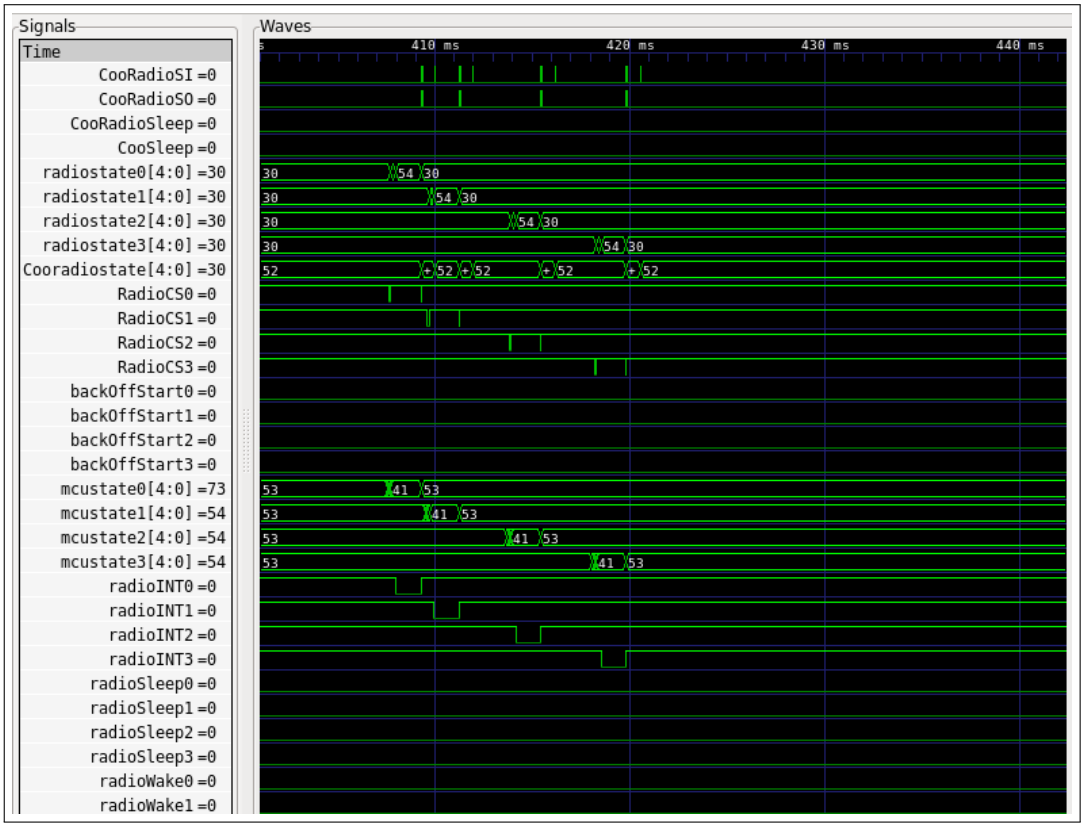

Figure 5.8: Graph from code simulation WSN behaviour

5.5/ Conclusion

In this chapter, we have presented our approach to simulate and validate complex systems from SysML models. Formal models of SysML blocks were acquired by applying transfor-

mation rules by ATL tool for mapping metamodel SysML diagrams with metamodel SystemC. We have proposed a model transformation from block definition diagram, internal block diagram, parametric diagram, and state machine diagram to SystemC model. Finally, we used the simulation in SystemC environments to verify SysML non-functional requirements from the requirement diagram to validate the designed system.

6

Comparison of Verification techniques of SystemC models

Contents

6.1/ Introduction

SystemC is an efficient system-level modelling language and simulation platform proposed to increase the abstraction level of embedded systems design. However, as the design of systems turns more complex, the exploration of design on a high abstraction level becomes more important than ever. In particular, they are the significant issue, the comparison between the techniques that can be used to provide verification for SystemC models. To ensure the correctness of systems designed with SystemC environment we need verification techniques, such as UPPAAL model-checking, within the SystemC designs.

We propose a verification approach guided by the requirement relationships, such as satisfy with the model elements, to verify non-functional requirements over SystemC models. We propose to adapt the approach proposed by [Pockrandt et al., 2012a]. We propose to apply verification over SystemC models using the UPPAAL model-checker. Therefore, we propose to convert SystemC models into UPPAAL models.

6.2/ Techniques for SystemC Verification

SystemC is a modeling language that can be used to describe embedded systems at different abstraction levels. However, simulation is not the only thing for which a SystemC model can be used. Tools that satisfy these needs must have a SystemC front-end that can retrieve the dynamically generated hierarchy and its behavior from the model. A SystemC

model must be chosen or developed. This section is dedicated to the presentation of the most used techniques for the verification of SystemC designs. All these techniques intend to help the system designer to develop correct SystemC designs. These techniques may be classified into three categories:

- Techniques based on SystemC library environment.
- Techniques that link SystemC library with other verification libraries.
- Techniques that translate SystemC models to other formalisms.

Given a system to design with SystemC and a specification of its requirements, in the Figure 6.1, we illustrate the techniques that may be used to provide a validation of its SystemC design.

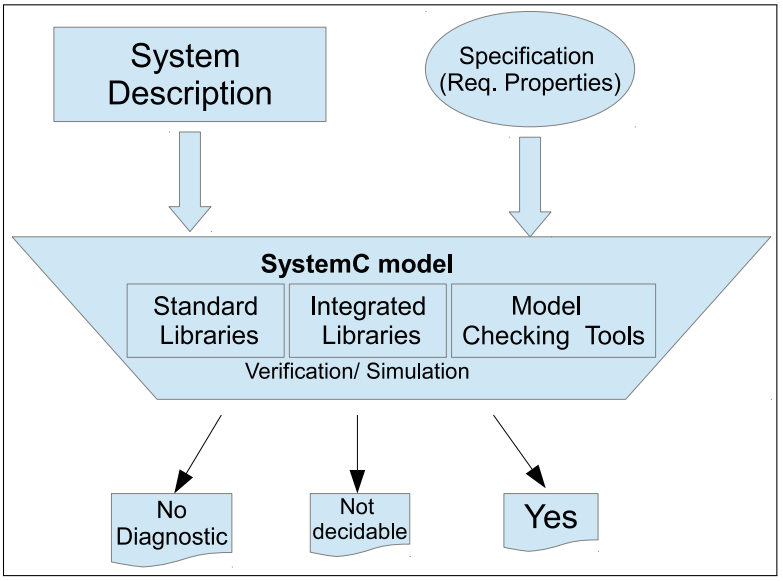

Figure 6.1: Overview techniques for SystemC verification

6.2.1/ Verification by SystemC Libraries

To provide verification mechanisms in SystemC, two libraries have been developed: the SystemC Verification Standard library and the Native SystemC Assertion library. In this section, we present how these libraries can be used to verify SystemC designs.

NSCa

The assertion temporal fundamentals in NSCa derived from SVA. In this, assertion level evaluations take place on every clock. By application, NSCa can build simplistic to complex property sequence to verification when executing interfaces and/or protocol-level checks in several lines required in SystemC environment. NBridge-SVA, a bidirectional NSCa to SVA, is packaged including NSCa to provide a bridge between NSCa and SVA. The Figure 6.2 illustrates the comprehensive transaction cycle in a system level based design.

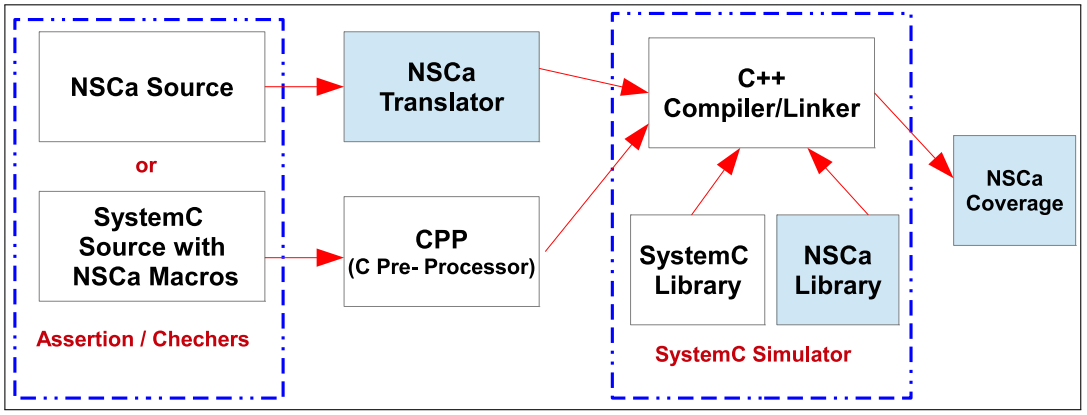

Figure 6.2: Assertion flow for NSCa

SCV library

The SystemC Verification Standard (SCV) library provides a common set of APIs for transaction-based verification, constrained randomization, weighted randomization, and exception handling. As a result of this, the random data types can be used in variable recording, transaction recording, constraints, randomization, and other functions. By transaction is based verification from transaction recording. The transaction recording ability in the verification standard allows to capture transaction level activities during simulation. By a callback mechanism, these activities can be monitored by another SystemC module at runtime, or can be used recorded into a database for visualization, debugging, and post simulation analysis.

Furthermore, randomization allows a large number of stimuli to be generated with less manual effort than directed checking. To improve service coverage and to focus on specific appearances of the design, constraints or weights typically are used in the randomization. While many real test benches may use ``rand ()'' from the C library to create a random integer. The verification standard supports randomization of several data type through the use of the data introspection facility. Data objects of temporary data types can be randomized during the application of ``scv_smart_ptr''. For example, a random value for an ``sc_uint $< 8 >$'', ``sc_uint $< 12 >$'' can be generated using the code shown in Listing 6.1.

The SCV also provides a simple database to store and investigate verification results and gives simple SystemC to VHDL simulator linkage mechanism. The compatible mechanism for errors is processing and debugging error detection mechanism.

SCNSL

SystemC Network Simulation Library is interposed between SystemC and standard C++ libraries and the classes that the developers will realize.
The SystemC libraries and SCNSL starting from the system specifications. The primary model can be described at different levels of abstractions as the RTL and the TLM levels or with combinations of parts of different levels. On this initial model it can be built the verification systems that will establish the validity of the final product. From this initial model, the process continues to improve the system, developing the final product in all three domains at once. Usually SCNSL is a functional tool, both for flexibility

and performances. SCNSL demonstrates a high perspective for accurate system-level simulation of WSN systems, and its architecture and language are well suited. SCNSL models include nodes and network separately.

Listing 6.1: simple random mechanism

```
#include <scv.h>
#define RND_SEED 1
class packet_t
{
public :
sc_uint<8> addr;
sc_uint<12> data;
unsigned payload[2];
}

SCV_EXTENSIONS(packet_t)
{
public:
scv_extensions< sc_uint<8> > addr;
scv_extensions< sc_uint<12> > data;
scv_extensions< unsigned [2] > payload;
....
}

int sc_main (int argc, char* argv[])
{
scv_smart_ptr<packet_t> pkt_p("packet");
scv_shared_ptr_
<scv_random>_rand_p(new scv_random("gen", RND_SEED));
pkt_p->set_random(rand_p);
cout << "Packet Pre Random: " << endl;
....
cout << "Packet Post Random: " << endl;
pkt_p->print();
return 0;
}
```

6.2.2/ Verification by Libraries Integrated to SystemC

The SystemC libraries developed to provide a verification are limited and do not allow the verification of temporal properties. At this issue, several techniques have been proposed, like ``CRAVE, CHIMP, and ISIS''. They have been developing some verification libraries built on the top of SystemC libraries for enabling the verification of temporal properties. In the section, we present one of these libraries and its applications to verify SystemC designs.

SVM

The SVM Library improves the OVM/UVM for SystemC, by adding features based on the OVM for SystemVerilog version 2.2.1. Additionally, the figure 6.3 show the structure of the

SVM library packages, which integrate libraries to provide Assertion, Randomization/-Constraints and Coverage, to support advanced RTL/TLM for SystemC. The outline details of the functional coverage implementation of the SVM as a SystemC library, which is based in the following areas covered in SVM.

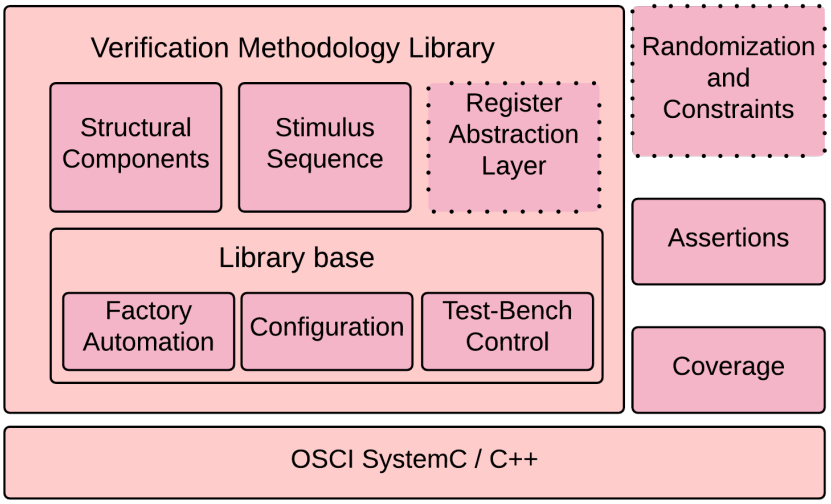

Figure 6.3: Structure of SVM libaray

6.2.3/ Verification through Model-checking Tools

Model-checking tools are well-known techniques that allow verification of systems. Employing this technique to provide a verification of SystemC designs is the subject of many pieces of research. At this issue, several techniques has been proposed, like ``SPIN, SCIVER, SYSFIER, KRATOS, SDSS,and CADP'' We will explain in the next section, the method proposed to verify SystemC designs using projects and tools ``UPPAAL''.

6.3/ UPPAAL and TCTL

The model checker Uppaal based on the theory of timed automata was presented in [Soliman et al., 2012]. It is a combined tool environment for modeling, validation, and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc...). Uppaal model comprises three parts: ``global declarations'', ``parameterized timed automata'' and a ``system declaration''. The global declarations segment has global variables, constants, channels and clocks. The timed automata templates describe timed automata that can be instantiated with different parameters to model similar process. In the system declaration, the templates are instantiated, and the system to be composed is given as a list of timed automata.

In UPPAAL, validation is done by graphical simulation and verification is done by automatic model-checking. In the simulation, the modeled system is executed interactively

and is observed whether the system satisfies the expected behavior or not. UPPAAL uses finite state automata extended with clock and data variables.

Two main parts of UPPAAL are the graphical user interface and a model checker engine. Graphical user interface is performed in Java language and is executed on the user end. Model checker engine is advanced C++ and is also executed on the user workstation, but UPPAAL offers the flexibility of running the engine on a separate machine that is more powerful and can be referred as the server.
Model checker engine applies on-the-fly searching technique in combination with the symbolic technique that makes the verification problem reduced to the problem of solving simple constraints system. UPPAAL can also generate a problem trace automatically, that can be used to diagnose the problem and can also be used to explain why a property is or is not satisfied by the described system.

UPPAAL verification engine supports requirements based on the Computation Tree Logic (CTL) [Friesen, 2011], manually support the implementation of formal verification techniques, the changing requirements formalized are into Time Computation Tree Logic (TCTL) properties [Alur et al., 1990]. TCTL is an extension of CTL, which allows analyzing several possible states of a system. The requirement that overtaking must not last longer than two-time units can be expressed by referencing the clock ``c'' within the TCTL formula $AG\neg$ (overtaking $\wedge$ c > 2).

TCTL formulas Φ are defined by the following grammar:
$\Phi ::= p \mid x + c \leq y \mid y + d \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid E\Phi_1 U\Phi_2 \mid A\Phi_1 U\Phi_2 \mid z.\Phi$ for

- proposition $p \in P$.
- clocks x, y $\in C$.
- specification clocks z $\in C_\Phi \subseteq C$.
- non-negative integer constants c, d $\in$ N.

The formalization of requirements in Uppaal is based on TCTL formulas. The temporal operators G and F correspond to the UPPAAL operators [] and $\langle\rangle$. As a restriction of the original TCTL syntax. UPPAAL does not support nested path quantifiers, the only exception is UPPAAL additinal $--{>}$ operator. A formula $p --{>} q$ coorresponds to the formula $A[]$ (p imply A<> q).

All nested formulas are restricted to so-called state properties, consisting of time and data constraints, as well as activity constraints on certain locations. Time constraints are expressible by a direct reference to the value of a clock in the underlying network of timed automata. A clock local to a certain process is referenced by the statement process.clock. UPPAAL does not support additional specification clocks.

The activity of a certain location for a process is expressible by the statement process.location. Besides references to clocks, UPPAAL also enables its data variables to be part of state properties, whereas a local variable is referenced by process.variable. As a specification feature, the deadlock statement allows for the specification of progress execution requirements.

6.4/ Transformation of SystemC Model for UPPAAL Verification

To provide a formal verification of SystemC designs, we use existing tools to transform, by a sequence of refinement steps, a SystemC design into an Uppaal automata. Our objective is to analyze a SystemC design with model-checking techniques. As shown in Figure 6.4, the tool STATE (SystemC to Timed Automata Transformation Engine) [Pockrandt et al., 2012b], is used to transform an abstract SystemC design into a Uppaal model and the Uppaal model-checker is used to check properties expressed as temporal logic formulas.

Verification results express satisfaction or no satisfaction of properties. If a property is not satisfied, the Uppaal model checker additionally generates a feedback, which can be used for debugging purposes. The feed-back can also be visualized and animated in the Uppaal tool to understand where the problem appears.

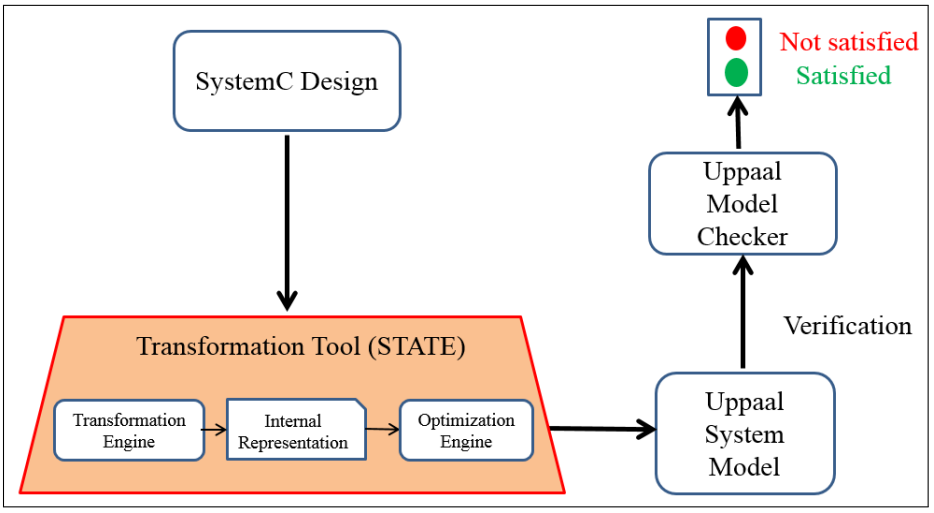

Figure 6.4: Methodology Model Transformation SystemC to Uppaal

We will focus on how to perform and what are the parameters used to affect the implementation. Also, has been defining the requirement and structure SystemC model to generate a UPPAAL environment. The mapping methodology uses the STATE tools to create UPPAAL code from SystemC diagrams. Figures 6.5 shows the SystemC2UPPAAL code.

The approach in STATE takes a SystemC design as input and generate a corresponding Uppaal model as output. As a front-end for SystemC, we used the Karlsruhe SystemC Parser (KaSCPar). The KaSCPar parses a given SystemC design and produces an Abstract Syntax Tree (AST) in XML. The AST in XML serves as input for STATE, which creates an Uppaal model that is also in XML format and that can be utilized as input for the Uppaal environment. In STATE, the transformation of a given SystemC design is performed in two phases. First, the transformation engine constructs an Uppaal model from the given AST of the SystemC design. Second, the optimization engine performs several optimizations on that. When, a UPPAAL model is written and can be used as input for the UPPAAL model.

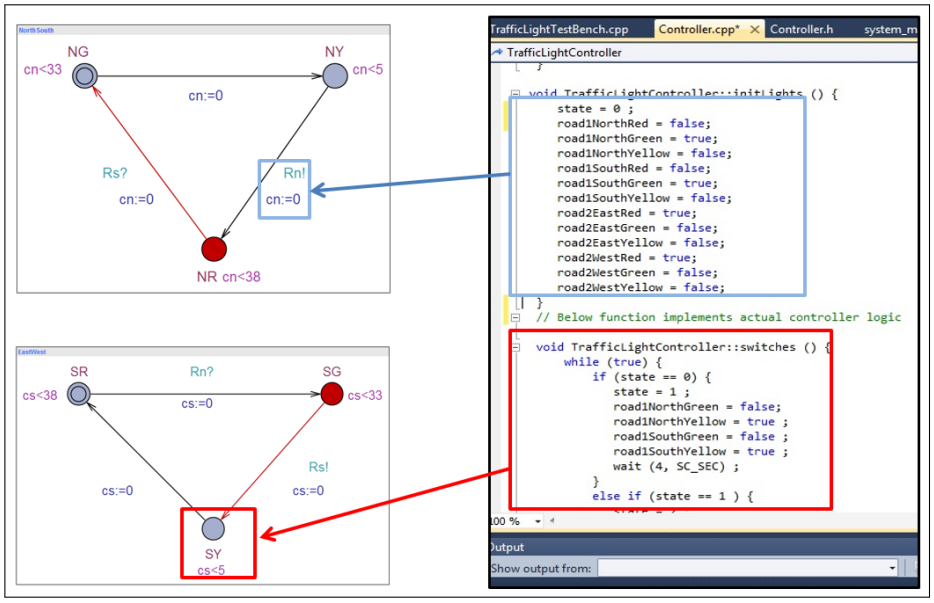

Figure 6.5: Code generation SystemC to Uppaal

6.5/ Illustration on the case study

The general process of our approach consists of several stages, modeling with SystemC environment that will be the source of transformation models. First, we transform the requirements properties of the Crossroads system to CTL, manually support the implementation of formal verification techniques, the dynamic requirements are formalized into TCTL properties.

To verify the requirements of the Crossroads system, the verification of SystemC functional requirements by translating SystemC model into formal models. Requirements are expressed into temporal logic properties expressed in TCTL. Primary, we verify that the system is deadlock free. We express this property in TCTL by the formula ``AG not deadlock''. Then, we verify time properties. As example, we verify that both ``NorthandSouth Lights'' and ``EastandWest Lights'' cannot stay in yellow color more than 5 seconds, which is expressed in TCTL as:

- AG (NorthSouth.NY imply NorthSouth.cn<5).
- AG (EastWest.SY imply EastWest.cs<5)).

Then, we verify that both NorthandSouth Lights and EastandWest Lights cannot stay in red color more than 38 seconds, which we express in TCTL as:

- EG(NorthSouth.cn>38).
- EG(EastWest.c>38).

Figure 6.6, we present Uppaal environment, where Uppaal automata, properties and their verification results are shown.

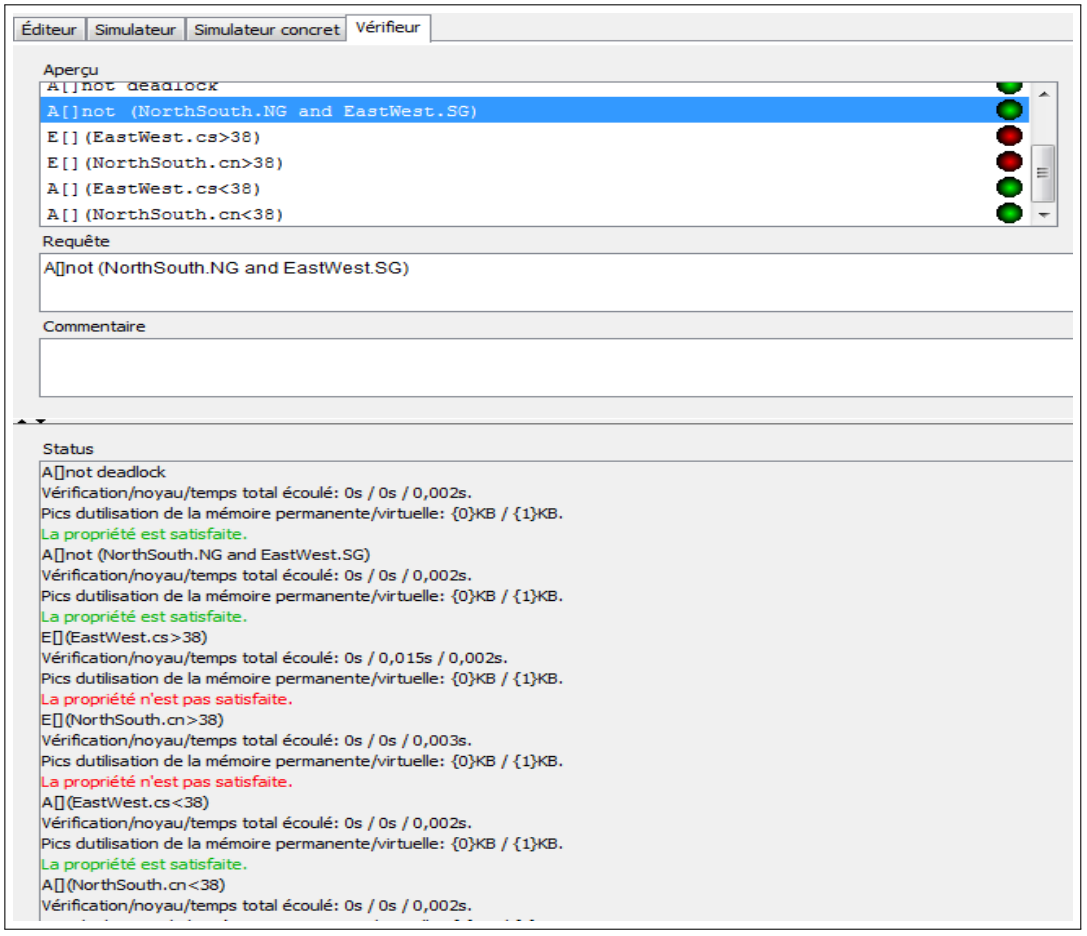

Figure 6.6: Verification in Uppaal environment

6.6/ Classification of Verification in SystemC

Studies and associated tools presented in this chapter show that providing verification for SystemC has an increasingly importance. They intend to introduce a verification mechanism to assess the correctness of SystemC designs. As shown in Figure 6.7, we can group the proposed techniques into three classes.

We find in the first class the SystemC verification libraries. Specifically, the SystemC verification standard library, which constrained randomization, weighted randomization and transaction monitoring. It allows a large number of stimulus to be generated with limited manual resolution than directed checking. The Native SystemC assertion library that is an assertion path coverage and assertion activation coverage, as well as primitives to construct assertions for transaction-level models at higher levels of abstraction. The provided timed/untimed properties, transactional and bus cycle accurate.

The second class provides the verification of SystemC design through external libraries which use SCV. The tactics of this class enhance the library for SystemC, integrate libraries to provide Assertion, Randomization/Constraints and Coverage, to support advanced RTL/TLM for SystemC. As examples, ISIS enabled the runtime ABV of SystemC TLM virtual platforms ``untimed or timed, clocked, or unclocked'', for verifying behaviour

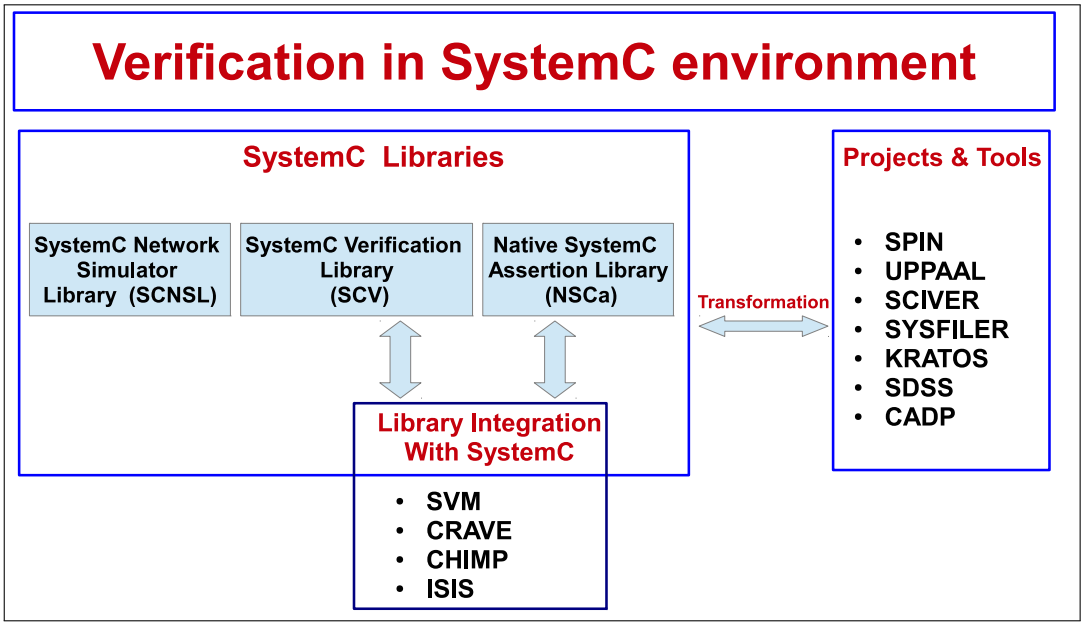

Figure 6.7: Classification SystemC Verification

and requirements. The technique has also been improved to support the PSL modeling layer that enables the use of ``global'' auxiliary variables in assertions.

As another example, CRAVE is a new library for constrained random stimuli generation, provided with SCV in new API, dynamic constraints and improved in line usability. The advantages include dynamic constraint specification and management, enhanced usability and much faster constraint-solving based on a portfolio approach. As another example ISIS, it inputs PSL assertions and performs the automatic construction of TLM-oriented SystemC monitors, built compositionally from primitive components.

Moreover, CHIMP includes automatic generation of monitors from temporal assertions, automatic instrumentation. The monitors that are generated from a trace property can have three possible outcomes, pass, fail, and undetermined, an assertion states a property about the entire execution trace of the MUV. Automatically the monitors are linked to the designers, this instrumented design is compiled using the SystemC library of primitive monitors.

The SystemC simulator can then be run on this combination of modules. The monitors inform about the satisfaction of the properties during simulation, the runtime ABV of SystemC TLM virtual platforms untimed or timed, clocked or unclocked.

The approaches of the third class are based on translating SystemC design into other formalisms, such as Model-Checking. In these approaches, properties to verify are described using LTL, CTL, and beneficial approach for developing reliability system designs. Here we summarize the systems discussed above.

As examples, KRATOS is a software model checker for sequential and threaded C programs. Kratos verifies safety properties in the form of program assertion, provides two different analyses for verifying SystemC designs: ``sequential'' and ``concurrent analyses''. The sequential analysis, based on the lazy predicate abstraction, verifies the C program resulting from the sequential of the SystemC design. The concurrent analysis, based

on the novel ESST algorithm, combines explicit state techniques with lazy predicate abstraction to verify threaded C program that models a SystemC design.

As another example SDSS, is the formulation interpreting SystemC designs with timed language constructs as Kripke structures. With this formulation, apply symbolic model-checking to the Kripke structure BM defined in the previous section.

Table 6.1 provides the feature comparison between these techniques. We classified the existing approaches into three classes: SystemC Verification Standard, combine with other library verification and transaction SystemC design to other formal language dependence of some rule mapping and methodology. To use tools to assertion and verification properties system.

Table 6.1: Comparison of SystemC with all techniques

Libraries-Tools	Assertion	Model-checking	Randomize-Constraint
NSCa	Yes	No	Yes
SCV	Yes	No	Yes
SVM	Yes	No	Yes
CRAVE	No	No	Yes
SPIN	Yes	Yes	No
UPPAAL	Yes	Yes	No
ISIS	Yes	No	No
SCIVER	Yes	Yes	No
SYSFIER	No	Yes	No
KRATOS	Yes	Yes	No
CHIMP	Yes	No	No
SDSS	No	Yes	Yes
CADP	No	Yes	Yes

We think that the verification of SystemC designs by SystemC libraries are not yet mature and need more development to support the verification of temporal properties. For that, the approaches based on translating SystemC design into other formal languages is more efficient.

6.7/ Conclusion

We have illustrated in this chapter our proposal for verifying complex systems designed in SystemC environment, we have many techniques of verification approaches used in SystemC models. The practicability of our verification approach was illustrated by a case study that describes a traffic light system. We have specified the system using SystemC environment. Then, from these specifications, a SystemC model with different formulas and methodologies is used to achieve verification and validation of requirement specification, which is vital for a successful software development project. We classified the existing approaches into three groups of classification. SCV standard is amalgamate with other library verification and performance SystemC design to another formal language. Which dependence from rule and methodology is used to assertion and verification function properties of the complex systems.

The practicability of our verification approach was illustrated by a case study that describe

a traffic light system. We have specified the system using SystemC models. Then, the UPPAL model-checker was used to achieve the verification.

We think that verification and validation of SystemC designs through SystemC verification standard libraries is not yet mature and need more development to support the verification of temporal properties. For that, the approaches based on translating SystemC design into other formal languages is more efficient.

7

Transformation of SysML Specification into Promela-SPIN

Contents

7.1/ Introduction

Ensuring the correction of heterogeneous and complex systems is an essential stage in the process of engineering systems. In this chapter, we suggest an approach to verify and validate complex systems specified by SysML language. We transform SysML specifications into Promela models to validate the designed systems by model-checking SPIN [Holzmann, 1997]. The requirements properties are transformed to Linear Temporal Logic (LTL) formulae verified by SPIN environment. In Figure 7.1, we show the position of the contribution presented in this chapter, regarding the contributions of this thesis.

We propose a verification approach guided by the requirement relationships, such as ``verify'', with the model elements, to verify functional requirements with state machine diagrams. To using an abstraction that separates the system down into smaller state machines can perform it simpler to understand, and can develop our ability to use the advantage of similarities between the different methods. The state machine diagram is converted into Promela model. The proposed translation is implemented in ATL. From the SysML requirements, we extract LTL properties, which we verify by the SPIN model-checker.

7.2/ Approach

One purpose of modeling is to enable the analysis, verification and validation of systems. Models may describe requirements, structure and behaviors of a designed system. They

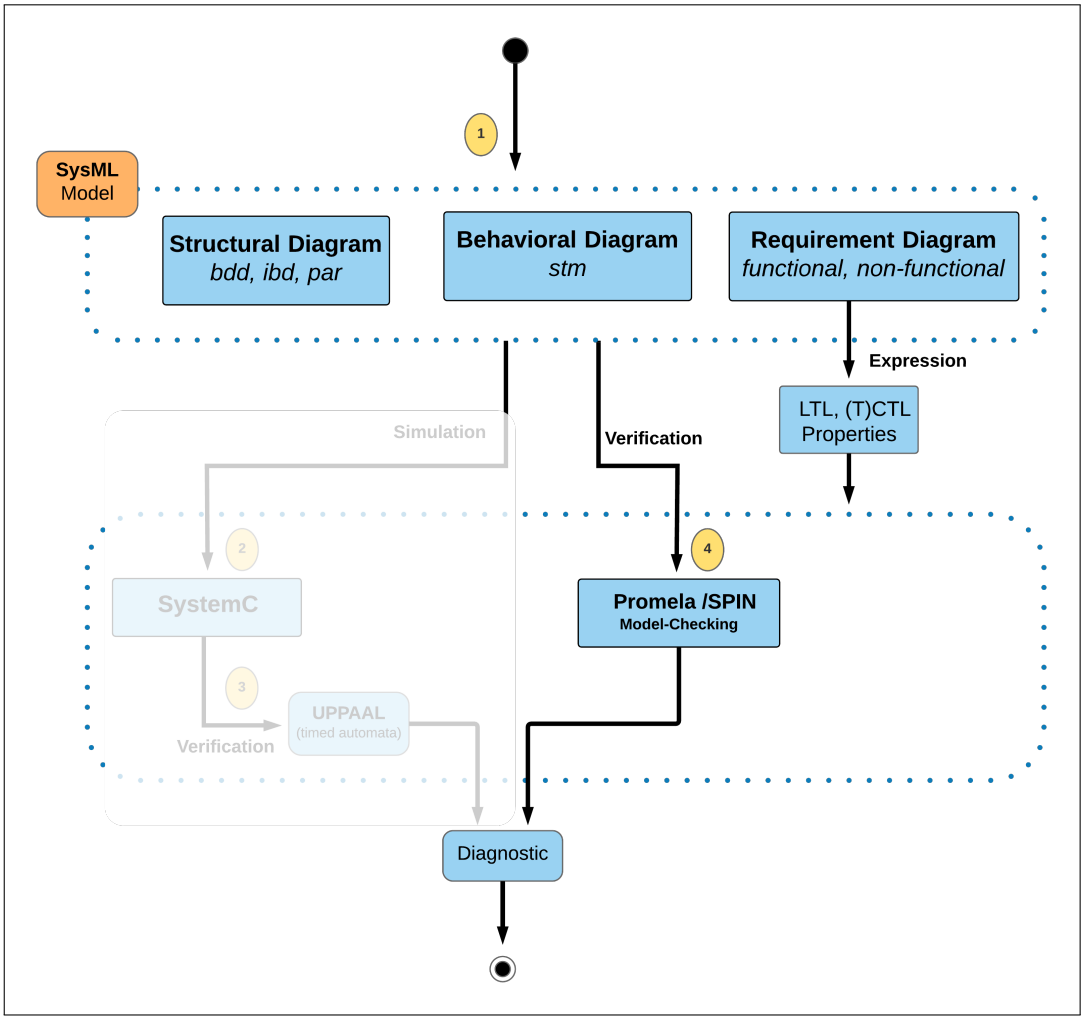

Figure 7.1: Methodology flow for approach

may be used to validate the characteristics of some part or the designed system, e.g. its functionality or its performances. In this section, we define a group of activities related to our approach to specify, verify and validate formal SysML specifications.

In this approach, SysML state machine diagrams are converted into Promela models, and then used to verify the functional requirements of the system. These requirements are expressed as temporal logic formulas and verified using the SPIN model-checker.

Figure 7.2 summarizes the main steps of the proposed approach. First, we create the SysML diagrams ``RD, BDD, and SMD'', to specify the system requirements, structure, and behavior. Second based on MDE, we develop a transformation technique to map SMD into Promela. Then, we describe the SysML requirements as a result of LTL temporal logic. Therefore, we analyze, and verify the requirements using the SPIN model-checker tool. Our verification approach will be subject to the ``verifying'' relationships between requirements, blocks and state machine diagrams.

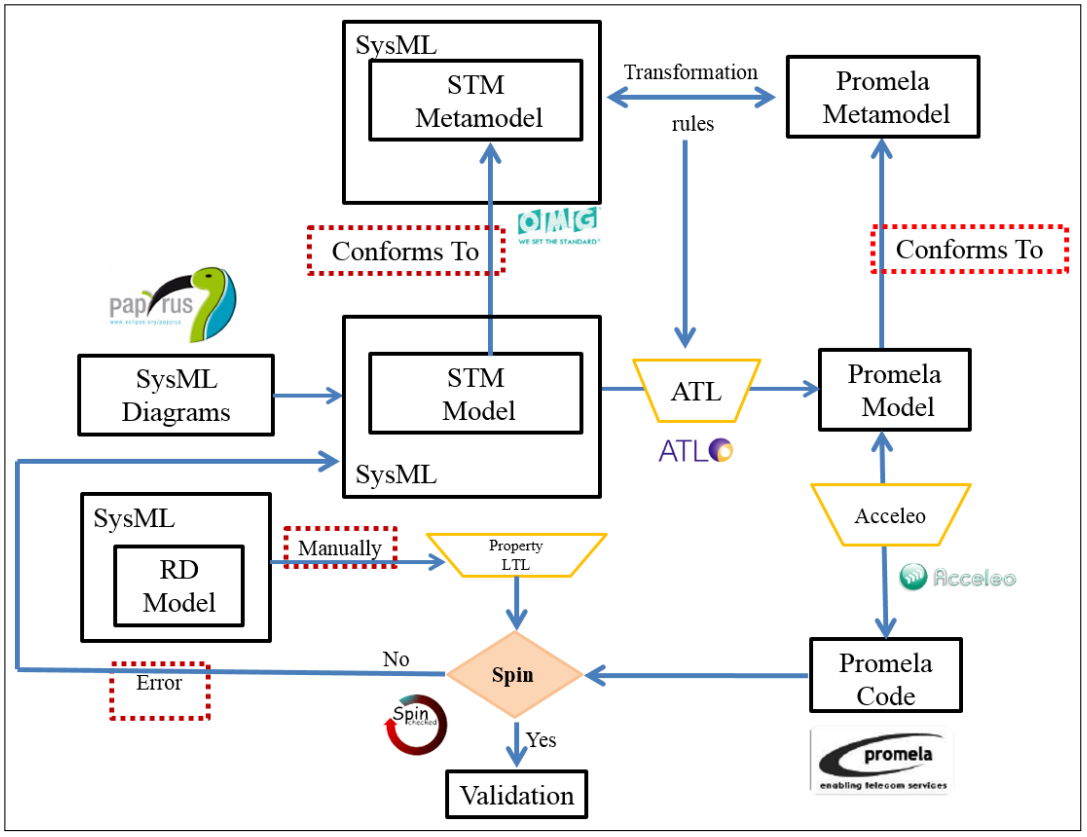

Figure 7.2: Methodology flow for approach

7.3/ From SysML to Promela

In this section, we propose a transformation technique according to MDE to map SMD into Promela.
We focus on defining how and what are the transformed SMD elements to formalize behavioral aspects and to preserve the structural representation of the system. After that, we define the mapping rules used to create a Promela model from the SysML SMD.

7.3.1/ Promela MetaModel

The source metamodel represents the SysML metamodel, and the target metamodel will represent the Promela metamodel [McUmber et al., 2001]. Both are carried out under the metamodel formalized of EMF. The different stages of implementation are shown in the Figure 7.3.

7.3.2/ Transformation Process

The general process of our approach consists of several stages, beginning modification with SysML diagrams that will be the source of transformation models. In this work, we consider the conversion of two diagrams: RD and SMD. First, we describe the RD to LTL

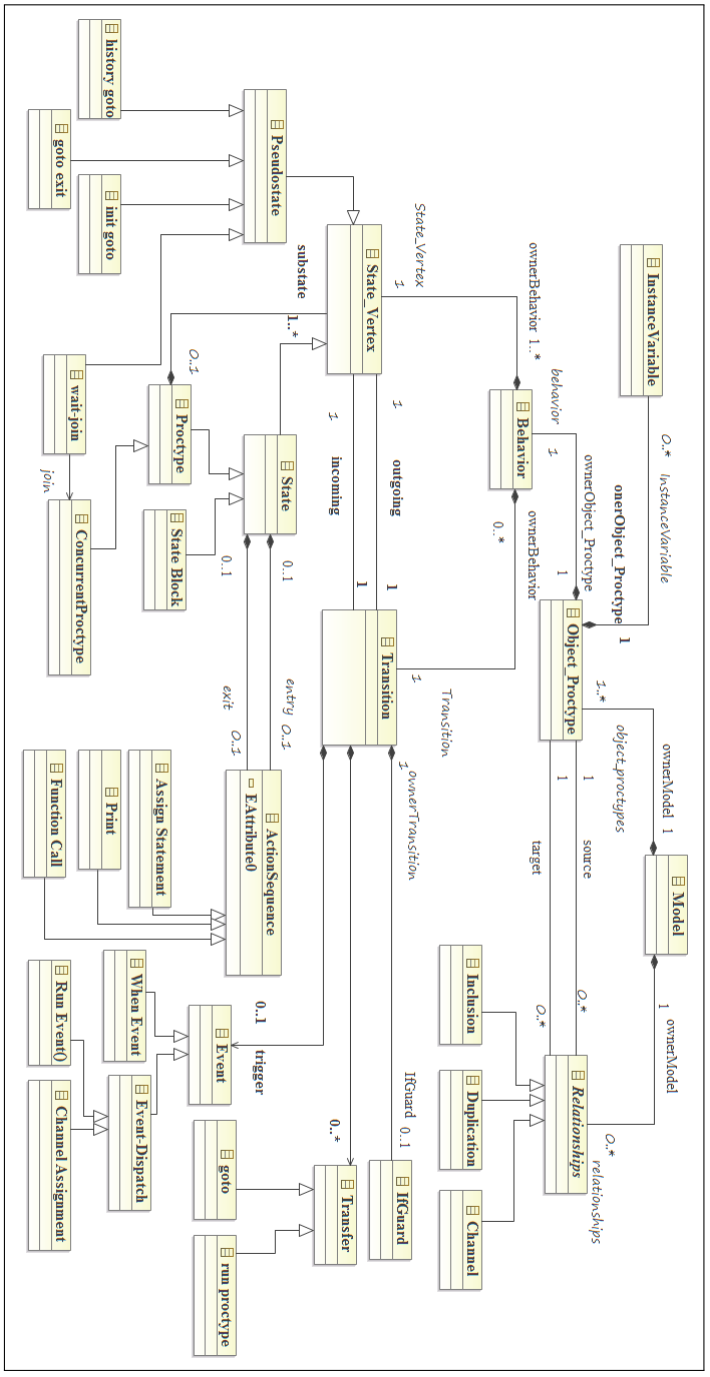

Figure 7.3: Metamodel of Promela

manually. Second, based on model transformation ``Model2Model'', and by exploiting the ATL language, we transform SysML models to Promela models.

The application of this methodology with ATL is based on the definition of the source and target metamodel, and description of transformation. Finally, the definition of the source model that conforms to source metamodel. ATL is a transformation language based on the Eclipse framework. ATL provides three kinds of rules namely matched rule, called rule, and lazy rule.

The ATL matched rules allow to specify which source model element must be matched, the number and the type of the generated model elements and the way these target model elements must be initialized from the matched source elements. Inverse to matched rules called rules enables to generate target model element from imperative code. This kind of rule must be called from an ATL imperative block. Also, a called rule can accept parameters. Finally, lazy rules can be called from matched rules. ATL allows to write methods with parameters and return type. These ATL functions are called helpers. They make it possible to define factorized ATL code. The rule of ATL are obligatory elements from to syntax:

- A pattern on the source model ``from'' with a possible constraint.
- One or more grounds of the target model ``to'' that explain how target elements are initialized from the corresponding source element.

7.3.3/ SysML To Promela Transformation

The metamodel UML and SysML have to satisfy the subset for animation described in Chapter 5. This metamodel depicts all the information that concern classes, instances, state-machines, associations, links, and behaviours. Table 7.1 summarizes the mapping between SysML elements and out pivot metamodel. Although creating a target item from source part, ATL retains a traceability link between the two elements. This link is used to initialize a target item in the ``to'' match as seen in Listing 7.1. The ATL code shows an example of the rules used in the ATL model to transform the SysML SMD to Promela description.

7.3.4/ Mapping Rules for the Transformation

SysML diagrams are created using the Papyrus [Lanusse et al., 2009] tool, to combine SysML and Promela. The commencement of a system description given as the input conditions of extending the model with the SMD is to introduce and transform behavioral information about the system. The target language, in this case, Promela are different possible translations of the considered semantics with the behavioral concepts of Promela [McUmber et al., 2001]. The following translation rules are chosen:

Listing 7.1: Rule SysMl model to Promal model

```
-- @nsURI MMSysML=http://www.eclipse.org/papyrus/0.7.0/SysML
-- @nsURI MMUML=http://www.eclipse.org/uml2/3.0.0/UML
module SMDtoPromela;
create OUT : MMPromela from IN:MMSysML, IN1: MMUML;

helper context MMUML!NamedElement def: getModel()_
 : MMUML!Model =
        if self.owner.oclIsTypeOf(MMUML!Model) then
        self.owner
else
        self.owner.getModel()
endif;

rule Model2Model {
 from
        UML_Model : MMUML!Model
 to
        Promela_Model : MMPromela!Model()
 }

rule Class2ObjectProctype {
        from
                UML_Class: MMUML!Class
        to
                Promela_ObjectProctype: MMPromela!Object
                _Proctype(
                        ownerModel <- UML_Class.getModel()
        )
}

abstract rule State2State {
        from
                UML_State : MMUML!State
        to
                promela_State : MMPromela!State (
        )
}
......

rule CompositeState2ProcType extends State2State {
from
        UML_State:MMUML!State(UML_State.isComposite)
to
        promela_ProcType: MMPromela!Proctype(
        )
}
```

- A SMD is mapped into Promela.
- A Promela Proctype is used to allow parallel states activation semantics.
- A boolean signal represents the activation of each state, more than one state can be active at the same time.

- A Promela Proctype are sensitive each one to another by notification to represent every event trigger that can fire transition from the associated state.

- In Promela Proctype, condition statements are utilized to represent the guards used in SMD.

By applying these rules, the basic mappings of SMD elements into Promela specifications are as defined in Table 7.1.

Table 7.1: Mapping between SMD and Promela

SysML/SMD	Promela/Spin
Object	Proctype
Instance Variable	Variable
Association	Channel
Generalization	Duplicated Proctype
State	State Block
Composite State	Proctype
Concurrent Composite State	Concurrent Proctype
Transition	Messages/ Assignments

7.3.5/ The Promela Model

The syntax of Promela is C-like. This section describes the Promela syntax that is involved in the translation. The main task of the Promela interpreter is to maintain the environment. The environment is the sum of all variables, proctypes, and channels and characterizes system states. Equal environments are equivalent to same states. The model checker uses the interpreter to build up the state space.

Therefore, it repeatedly makes calls to the interpreter. Each call has as input the current environment and delivers as output all executable transitions plus the subsequent environments. In a deterministic model, there is only one possible computation. Therefore, for a concurrent or non-deterministic model, checking all possible computations involves performing the program and backtracking all over the selection of the next statement to execute. One of the ways that SPIN archives efficiency is by generating an optimized model called a verifier for each Promela model. The structure verifies in Promela environment as:

- Generate the verifier from the Promela source code.

- Verifier is a program written in C structure.

- Compile the verifier relating environment C compiler.

- Execute to verify a result of the accomplishment of the verifier is a report that all computations are correct or else that some computation contains an error.

7.3.6/ Conversion Promela model to Promela Code

The last step of the proposed MDA approach concerns the Promela code generation using the Acceleo technology which is an open source code generator from the Eclipse Foundation. It executes the MDA approach to develop the application from EMF based models. The Acceleo language is an application of the MOF Models to Text Transformation (MOFM2T).

Once this definition is done, we can implement the code generator in our previous example. We have the metamodel and model of Promela for code generation, and we need to create an Acceleo project and configure the workflow necessary to automatically generate Promela code from the Promela model. Hence, we have defined Acceleo templates to match Promela model elements with the Promela syntax. In the first line of Acceleo code, we are importing the metamodel so that the generator knows the structure of our model. The important concept within Acceleo is called ``Template'', it is the smallest unit identified in a template file.

7.4/ Verification using the SPIN Tool

The SPIN tool for checking the satisfaction of the extracted LTL properties in the Promela model derived from the state machine diagrams. Our approach was organized by the ``verify'' relationships between requirements, blocks and state machine diagrams. Verification in SPIN is a three-step process Figure 7.4.

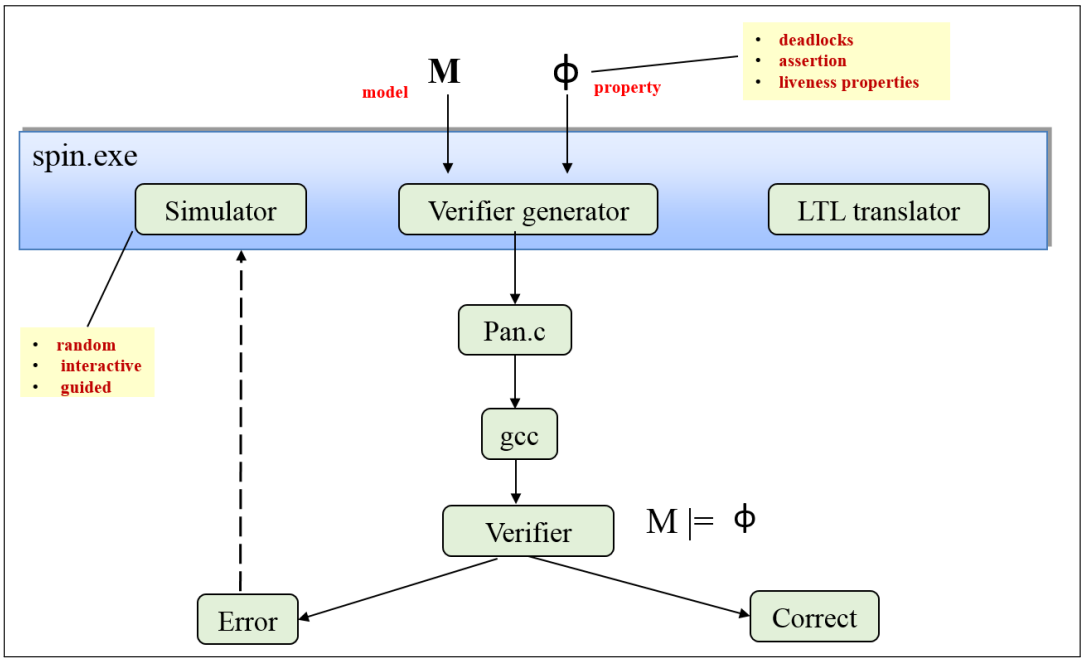

Figure 7.4: The architecture of SPIN

7.4.1/ LTL Model Checking

Linear Temporal Logic (LTL) is a linear-time logic that provides operators for describing events along a single computation path. They offer the possibility to describe events that will eventually become true, a fact that cannot be expressed in first-order logic. LTL formulas are inductively defined as follows:

- Every $p_i \in AP$ (atomic propsitions) is an LTL formula.
- LTL is complete under the boolean operations $\neg$, $\longrightarrow$, $\wedge$, and $\vee$.
- If φ_1 is an LTL formula, then X_φ, G_φ and F_φ are LTL formulas.
- If φ_1 and φ_2 are LTL formulas, then $\varphi_1 \cup \varphi_2$ and $\varphi_1 \, R \, \varphi_2$ are LTL formulas.

An important step in model-checking is to phrase the properties that should be verified at a model. Linear temporal logics are often the first choice, because they can express two main types of properties easily ``safety'' properties and ``liveness'' properties. Safety properties state that something bad never happens (G$\neg$ φ) and liveness properties state that something good keeps happening (G($\varphi_1 \longrightarrow F \, \varphi_2$)). Therefore we open the ``Check LTL Formula'' option and type in our LTL formula. Note that Promela expressions have to be put into braces {.....}.

For example, we would like to check the safety property is:

$$\text{G}(\neg \{\text{turn==C}\}).$$

7.4.2/ Verification

To verify the requirements of the Crossroads system, the verification of SysML functional requirements by translating SysML diagrams into formal models. So, we consider the SysML requirement diagram, the block definition diagram. Our verification approach is guided by the requirement relationships, such as ``verify'', with the model elements, to verify functional requirements on state machine diagrams.

7.5/ Illustration on the case study

To represent the behavior of the ``Controller_System'' block by SMD ``StateMachine_Light''. The diagram shows the state of traffic light colors as managed by the ``TrafficLightController'' operation. From the default state, the system changes to the ``Init_State'' state. The light colors can be in one of six states: ``Init_State'', ``NS_G_State'', ``NS_Y_State'', ``EW_R_State'', ``EW_Y_State'' and ``EW_G_State''. It can respond to the events north with south lights and east with west lights.

Note that not all events are valid in all states; for example, if a light is ``NS_G_State'', you cannot change it until you ``NS_Y_State'' it. Also, notice that a state transition can have a guard condition attached, so the system changes from one state to another according the transition conditions (guards).

Default is the next state if no transition condition is satisfied as shown in state ``Init_State''. If the state has any unconditional transition line, then assigning default state to next state is omitted as shown in case ``Init_State'', we see that the next state conditions appear in the generated Promela code according to the assigned priority. Consider the following situations where:

``NS'' and``WE'' are two inputs for a state machine.

``if (NS==1) and (WE==0) NextState = NS_G_State''
``else if (NS==1) NextState = NS_Y_State''

If both ``NS and EW are 0'', then NextState is dependent on the order of the appearance of the conditions in the code. Figure 7.5 shows the SMD diagram.

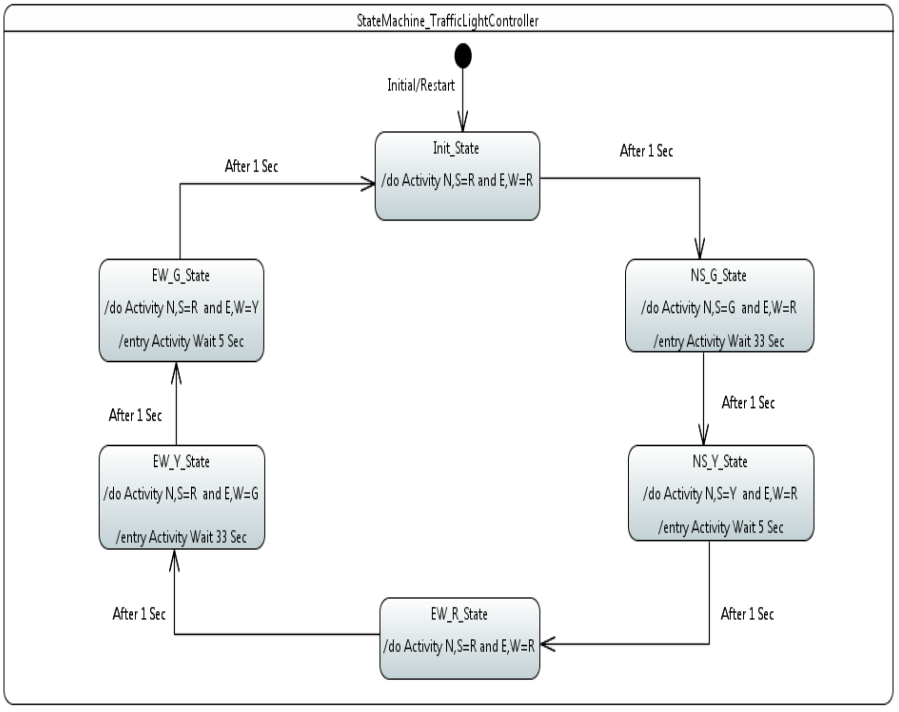

Figure 7.5: SMD of Controller System block

7.5.1/ Combine SysML to Promela

We will focus on how to perform and what are the parameters used to affect the implementation. As well as after has been defining the requirement, structure, and behaviour modeling to generate a Promela environment. The mapping methodology is used to create Promela code from SysML diagrams. Figures 7.6 demonstrate the SysML2SystemC code.

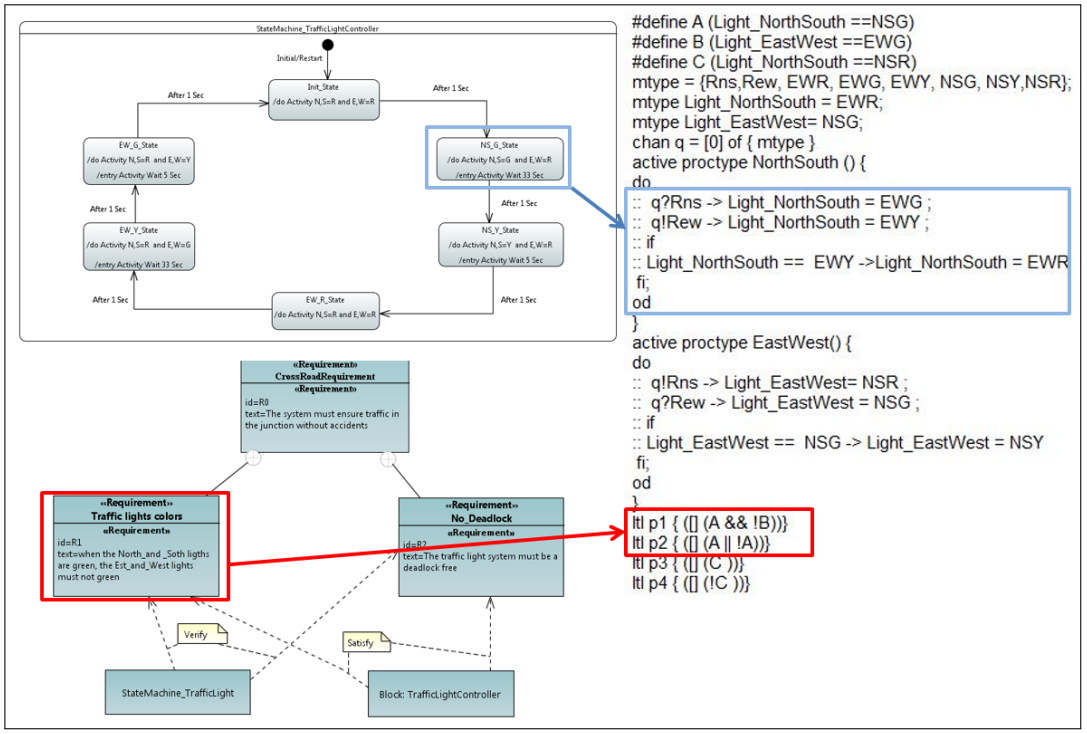

Figure 7.6: Code generation from RD and SMD to Promela

7.5.2/ Functional Requirements

There are two major properties we can prove about programs: safety and liveness. They are also referred to as invariance and eventuality, respectively. Defining between the two types of properties is helpful because different techniques can be used to prove each type. For example, the understanding advantage of the structure of safety properties can be used to optimize assume-guarantee reasoning. That appears a safety property holds involves an invariance argument while liveness properties require a model of justifying.

After defining the flags to track the execution state of the system, LTL properties can be written as boolean expressions over the flags. In our approach, we propose to transform SysML requirements to LTL properties by respecting this formalism with flags.

Furthermore, in Figure 7.7 we detail the safety requirements of the crossroads. For example, the requirement (Id=R1) expresses that the traffic lights on both roads that form the junction are different all the time. The constraint that represents the requirement is considered like an invariant of the block ``Controller_System''. To show how the requirements of the ``CrossRoad'' system are presented in an RD. The RD that contains three requirements (``CrossRoadRequirement''), a test case (``StateMachine_Light'') represented as SMD and the block (``Controller_System'') which represent the traffic light system. In this diagram, we show that the requirement (``CrossRoadRequirement'') is composed of the two (``Traffic lights colors'') and (``No_Deadlock''). The state of ID: R1 and R2 are satisfied by the block (``Controller_System'') and verified by the state machine (``StateMachine_Light'').

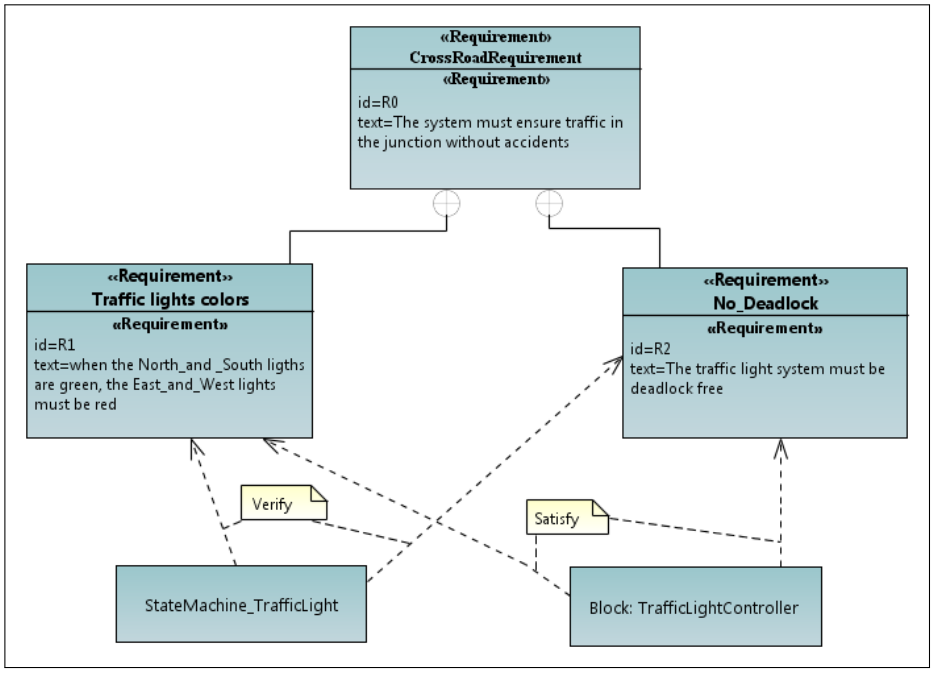

Figure 7.7: Crossroads System Requirement

7.5.3/ Verification of LTL properties

To verify the requirements of the Crossroads system, the verification of SysML functional requirements by translating SysML diagrams into formal models. Requirements are expressed into temporal logic properties expressed in LTL. First, we verify that the system is deadlock free. We express this property by the formula: ``[]! deadlock''. Others, we verify temporal properties. As example, we guarantee that both North with South Lights and East with West Lights can not stay in yellow color, which is expressed in LTL as:

- $[](((Light\_NorthSoth==NSG))\&\&(!((Light\_EastWest == EWG))).$
- $[](((Light\_NorthSoth==NSG))||(!((Light\_EastWest == EWG))).$

Then, we verify that both NorthandSouth Lights and EastandWest Lights cannot stay in red color. This is expressed in LTL as:

- $[]((Light\_NorthSoth==NSR)).$
- $[](!(Light\_EastWest==EWR)).$

In Figure 7.8, we present the SPIN environment and the obtained diagnostic for the Promela model properties and their verification results.

7.6/ Conclusion

We have illustrated in this chapter our proposed methodology to verify and validate complex systems designed with SysML functional requirements over SysML blocks. To solve

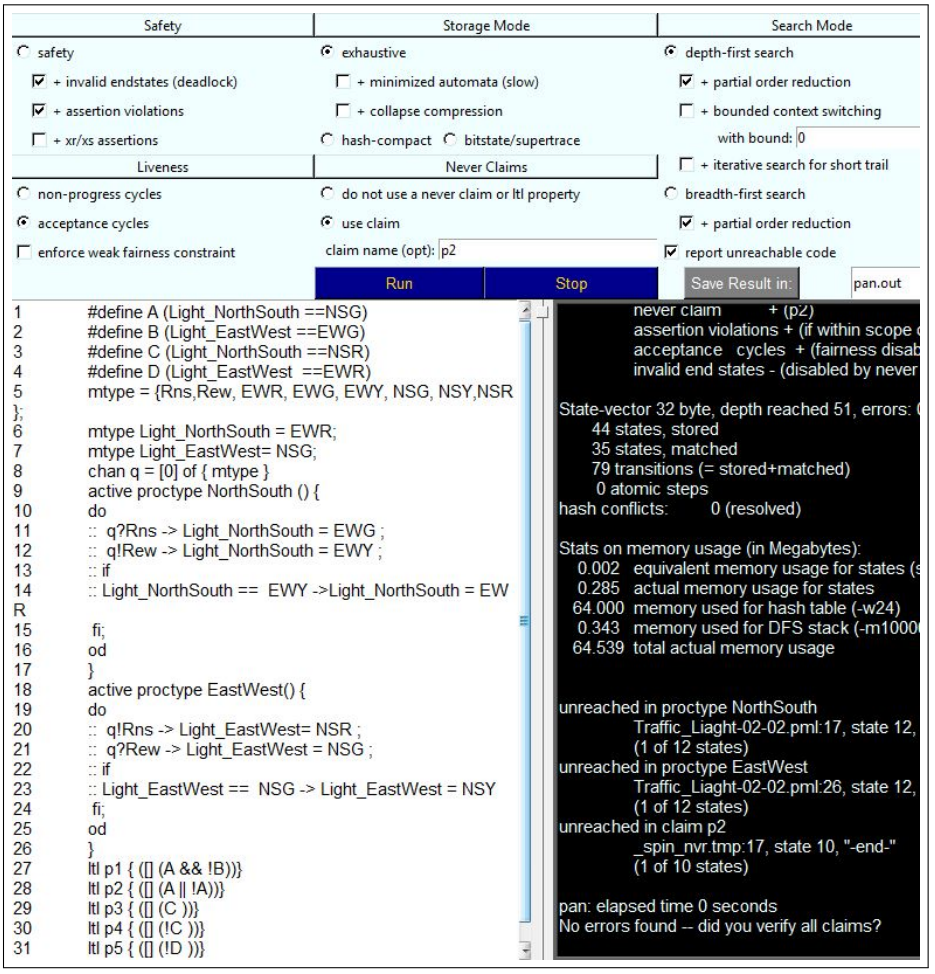

Figure 7.8: verification results

this issue, we have converted requirements into formal properties using linear temporal logic. LTL properties were then verified by a formal model of a SysML block. Formal models of SysML blocks were obtained by applying the approach of rule mapping to transform SysML state machine diagrams into Promela descriptions. Finally, we used the SPIN model checker tools to verify LTL properties over the Promela descriptions.

IV

Conclusions and Future works

8

Conclusion and Perspectives

We have presented in this thesis an approach to jointly use simulation and model-checking for verification and validation purposes. These techniques are combined to raise confidence during the design of complex systems.

In this approach, we propose a methodology based on SysML models combined with SystemC and Promela/SPIN, to specify and validate a complex systems. This approach is based on Model Driven Engineering techniques to firstly translate SysML models to SystemC with the aim of simulation and to map SysML behavioral diagrams to Promela/SPIN in order to verify some temporal properties extracted from the requirements.

Within this conclusion, we summarize the obtained results and we present some discussions about our work. We address in the Figure 8.1 the proposed approach and the future work.

Therefore, there are two challenges that a designer must solve:

- How to integrate formal aspects within SysML specifications to build reliable systems, for both specifications and requirements?
- How to combine simulation and verification approaches to validate complex systems described by SysML models? How to ensure that the functional and the non-functional requirements are satisfied by the system under design?

8.1/ Main Contributions

In this dissertation we have focused on the use of SysML and SystemC languages as alternatives to specify, simulate and verify complex systems. SysML is popular and allows the modeling of software and hardware components with a high level of abstraction by ignoring the details of the implementation.

Combining verification and simulation techniques to verify requirements in reliable applications is very important. To achieve this goal, we have proposed an approach that take into account functional and non-functional requirements. Functional properties are verified with model-checkers and non-functional properties are simulated.

Specifically, our purposes consist in using particular SysML diagrams and elements to describe different aspects of the specification from functional or non-functional requirements.

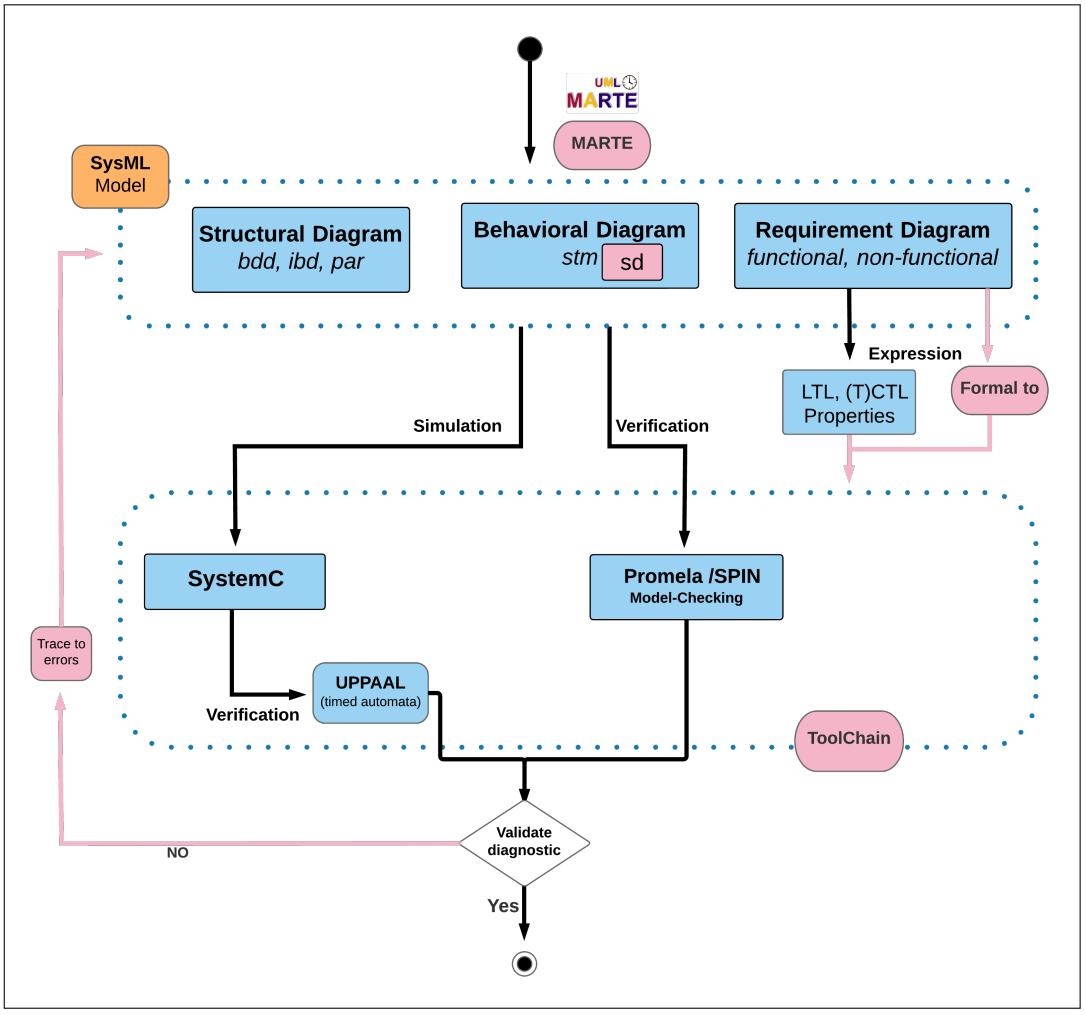

Figure 8.1: Our perspectives

To demonstrate the feasibility of the approach, a traffic light system is taken as a reference case study and is used for simulation and verification of properties.

In this thesis, we have presented three contributions. The first one is the starting point of the two others contributions:

- Using Model-driven engineering for transformations. We have used Model Driven engineering approaches to translate SysML diagrams to SystemC executable specifications. Globally, the work describes a transformation from SysML structure diagrams (Block Definition Diagram, Internal Block Diagram, and Parametric Diagram) and behavior diagrams (State machine diagram) to SystemC model. These translations are based on metamodels rules and implemented within the Eclipse framework. We decided to use the Papyrus modeling environment to design SysML models. Then we have used ATL and Acceleo to translate SysML models into SystemC. SystemC code is generated automatically and is used for simulation. We have illustrated the practicability of our approach with a case study about a traffic-light system.

- Verification of functional properties. In a first time, we have used the STATE tool to transform a SystemC module as an input for the Uppaal model-checker in order to verify timed automata properties. The obtained Uppaal automata seems to be not usable for our purposes. Also, in a second time, we have proposed an approach to verify SysML functional requirements expressed as LTL properties over state machine diagrams. As SysML is a semi-formal language, we have proposed a translation of state machine diagrams into Promela model. This translation is based on model transformation and is implemented in Eclipse platform using Papyrus, ATL and Acceleo.

  From the SysML requirements diagram, we have extracted LTL properties. Then, we have used the SPIN tool for checking the satisfaction of the extracted LTL properties in the Promela model. Our verification approach is guided by the ``satisfy'' and the ``verify'' relationships between requirements, blocks and state machine diagrams.

8.2/ Future work

The work of this thesis targeted the main issue: how to integrate formal specifications during the design of complex systems using SysML specifications? Requirements diagrams describe functional and non-functional properties of the system. Our contributions addressed some solutions and a feasible approach to build reliable SysML is proposed. To complete these results, we have identified some perspectives and future work.

- Combining SysML and MARTE for timed non functional properties.

  In reliable systems, it is crucial to specify a system in accord with functional and non functional requirements. To achieve this goal, we have proposed an approach to specify the system architecture with the SysML requirements diagram for untimed requirements. So, as a future work, it is interesting to address timed properties using the UML/MARTE profile with SysML.

- Extending the approach using Sequences Diagrams for verification.

  In our approach we have specified the system using SysML requirements, block definition and state machine diagrams. From these specifications, a Promela model with LTL formulas is derived. It would be interesting to investigate other diagrams like the sequence diagram to generate Promela model. Indeed, it would be possible to specify synchronous messages between components in order to verify communication protocol, for instance, between the components of a complex system. In this case we may use the SPIN model-checker to verify some properties over these protocols.

  In addition, other languages may be envisaged to describe temporal properties. Indeed, it would be interesting to do a strong survey about existing temporal languages. Which permits to mathematically describe temporal properties.

- Investigating results feedback to SysML specifications.

  As depicted on Figure 8.1, we plan to investigate on reverse engineering or back engineering techniques. Indeed, it would be useful to feedback results from simulation or model-checkers into the SysML specifications. It would be a strong asset for both

the traceability of the requirements and their maturity. In addition, it would be possible to feedback errors or bugs that are detected during simulation and verification. The main goal is to assess the validity of the requirements and of the model at the soonest in the verification and validation process. With this approach, the designer can iterate the process until the specifications are correct.

- A (semi-)automatic tooled process for simulation and verification.

  We plan to develop a tooled process that supports the automatic simulation and model-checking activities. This tooled appraoch may be composed of tools that enable the following processes:

  1. Automatic translation of SysML diagrams into Promela language and automatic verification of LTL properties with the SPIN model-checker.
  2. Automatic translation of SysML diagrams into SystemC language and automatic verification of properties using the UPPAAL model-checker.
  3. Automatic translation of informal requirements (text) to formal properties. For instance, it would be possible to use pattern recognition to translate informal sentences into LTL or CTL.

  Finally, we plan to perform extensive experiments on industrial case study to evaluate the scalability of the proposed approach.

Bibliography

[Abdulhameed et al., 2014a] Abdulhameed, A., Hammad, A., Mountassir, H., et Tatibouet, B. (2014a). An approach based on SysML and SystemC to simulate complex systems. In Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on, pages 555--560. IEEE.

[Abdulhameed et al., 2014b] Abdulhameed, A., Hammad, A., Mountassir, H., et Tatibouët, B. (2014b). An approach combining simulation and verification for sysml using systemc and uppaal. In CAL 2014, 8ème conférence francophone sur les architectures logicielles, page 9 pages, Paris, France.

[Abdulhameed et al., 2015] Abdulhameed, A., Hammad, A., Mountassir, H., et Tatibouet, B. (2015). An approach to verify SysML functional requirements using Promela/SPIN. In Programming and Systems (ISPS), 2015 12th International Symposium on, pages 1--9. IEEE.

[Abrial et al., 1998] Abrial, J.-R., et Mussat, L. (1998). Introducing dynamic constraints in B. In B'98: Recent Advances in the Development and Use of the B Method, pages 83--128. Springer.

[Adrion et al., 1982] Adrion, W. R., Branstad, M. A., et Cherniavsky, J. C. (1982). Validation, verification, and testing of computer software. ACM Computing Surveys (CSUR), 14(2):159--192.

[Al Obisat, 2012] Al Obisat, F. M. (2012). Using Spin Model Checker for Learning the Semantics of UML Models. International Journal of Computer Science and Telecommunications, 3:7--10.

[AlRawashdeh et al., 2014] AlRawashdeh, H., Idris, S., et Zin, A. M. (2014). Using Model Checking Approach for Grading the Semantics of UML Models. pages 122--129.

[Alur et al., 1990] Alur, R., Courcoubetis, C., et Dill, D. (1990). Model-checking for real-time systems. In Logic in Computer Science, 1990. LICS'90, Proceedings., Fifth Annual IEEE Symposium on e, pages 414--425. IEEE.

[Ando et al., 2013] Ando, T., Yatsu, H., Kong, W., Hisazumi, K., et Fukuda, A. (2013). Formalization and Model Checking of SysML State Machine Diagrams by CSP#. In Computational Science and Its Applications--ICCSA 2013, pages 114--127. Springer.

[Antonis et al., 2008] Antonis, K., et Voros, N. S. (2008). System level design of telecom systems using formal model refinement: Applying the B method/language in practice. Journal of Systems Architecture, 54(1):287--304.

[Apvrille et al., 2013] Apvrille, L., et de Saqui-Sannes, P. (2013). Static analysis techniques to verify mutual exclusion situations within sysml models. In SDL 2013: Model-Driven Dependability Engineering, pages 91--106. Springer.

[Aynsley, 2006] Aynsley, D. J. (2006). Ieee standard systemc language reference manual. IEEE Computer Society Std, 1666.

[Barnat et al., 2013] Barnat, J., Brim, L., Havel, V., Havlíček, J., Kriho, J., Lenčo, M., Ročkai, P., Štill, V., et Weiser, J. (2013). DiVinE 3.0--an explicit-state model checker for multi-threaded C & C++ programs. In Computer Aided Verification, pages 863--868. Springer.

[Beato et al., 2004] Beato, M. E., Barrio-Solórzano, M., Cuesta, C. E., et de la Fuente, P. (2004). UML automatic verification tool (TABU). SAVCBS 2004 Specification and Verification of Component-Based Systems, page 106.

[Behrmann et al., 2004] Behrmann, G., David, A., et Larsen, K. G. (2004). A Tutorial on Uppaal. In Bernardo, M., et Corradini, F., editors, Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004, number 3185 in LNCS, pages 200--236. Springer--Verlag.

[Benini et al., 2003] Benini, L., Bertozzi, D., Bruni, D., Drago, N., Fummi, F., et Poncino, M. (2003). SystemC cosimulation and emulation of multiprocessor SoC designs. Computer, 36(4):53--59.

[Berrani et al., 2013] Berrani, S., Hammad, A., et Mountassir, H. (2013). Mapping SysML to modelica to validate wireless sensor networks non-functional requirements. In Programming and Systems (ISPS), 2013 11th International Symposium on, pages 177--186. IEEE.

[Bézivin et al., 2003] Bézivin, J., Dupé, G., Jouault, F., Pitette, G., et Rougui, J. E. (2003). First experiments with the ATL model transformation language: Transforming XSLT into XQuery. In 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture, volume 37.

[Bhaduri et al., 2004] Bhaduri, P., et Ramesh, S. (2004). Model checking of statechart models: Survey and research directions. arXiv preprint cs/0407038.

[Blanc et al., 2010] Blanc, N., et Kroening, D. (2010). Race analysis for SystemC using model checking. ACM Transactions on Design Automation of Electronic Systems (TODAES), 15(3):21.

[Blanchard et al., 1990] Blanchard, B. S., Fabrycky, W. J., et Fabrycky, W. J. (1990). Systems engineering and analysis, volume 4. Prentice Hall Englewood Cliffs, New Jersey.

[Bodeveix et al., 2005] Bodeveix, J., Filali, M., Lawall, J. L., et Muller, G. (2005). Formal Methods Meet Domain Specific Languages. In Integrated Formal Methods, 5th International Conference, IFM 2005, Eindhoven, The Netherlands, November 29 - December 2, 2005, Proceedings, pages 187--206.

[Bombino et al., 2012] Bombino, M., et Scandurra, P. (2012). A model-driven co-simulation environment for heterogeneous systems. International Journal on Software Tools for Technology Transfer, pages 1--12.

[Bonanome, 2001] Bonanome, G. (2001). Hardware description languages compared: Verilog and SystemC. Department of Computer Science, Columbia University.

[Bousse et al., 2012] Bousse, E., Mentré, D., Combemale, B., Baudry, B., et Katsuragi, T. (2012). Aligning SysML with the B method to provide V&V for systems engineering. In Proceedings of the Workshop on Model-Driven Engineering, Verification and Validation, pages 11--16. ACM.

[Boutekkouk, 2010] Boutekkouk, F. (2010). Automatic SystemC code generation from UML models at early stages of systems on chip design. International Journal of Computer Applications, 8(6):10--17.

[Bustan et al., 2012] Bustan, D., Korchemny, D., Seligman, E., et Yang, J. (2012). SystemVerilog assertions: Past, present, and future SVA standardization experience. Design & Test of Computers, IEEE, 29(2):23--31.

[Cai et al., 2003] Cai, L., et Gajski, D. (2003). Transaction level modeling: an overview. In Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pages 19--24. ACM.

[Calazans et al., 2003] Calazans, N., Moreno, E., Hessel, F., Rosa, V., Moraes, F., et Carara, E. (2003). From VHDL register transfer level to SystemC transaction level modeling: a comparative case study. In Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on, pages 355--360. IEEE.

[Cao et al., 2011] Cao, Y., Liu, Y., et Paredis, C. J. (2011). System-level model integration of design and simulation for mechatronic systems based on SysML. Mechatronics, 21(6):1063--1075.

[Cavada et al., 2014] Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., et Tonetta, S. (2014). The NUXMV symbolic model checker. In Computer Aided Verification, pages 334--342. Springer.

[Chou et al., 2012] Chou, C.-N., Ho, Y.-S., Hsieh, C., et Huang, C.-Y. (2012). Symbolic model checking on SystemC designs. In Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE, pages 327--333. IEEE.

[Cimatti et al., 2000] Cimatti, A., Clarke, E., Giunchiglia, F., et Roveri, M. (2000). NuSMV: a new symbolic model checker. International Journal on Software Tools for Technology Transfer, 2(4):410--425.

[Clarke et al., 1986] Clarke, E. M., Emerson, E. A., et Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems (TOPLAS), 8(2):244--263.

[Clarke et al., 1999] Clarke, E. M., Grumberg, O., et Peled, D. (1999). Model checking. MIT press.

[Czarnecki et al., 2006] Czarnecki, K., et Helsen, S. (2006). Feature-based survey of model transformation approaches. IBM Systems Journal, 45(3):621--645.

[Debbabi et al., 2010] Debbabi, M., Hassaine, F., Jarraya, Y., Soeanu, A., et Alawneh, L. (2010). Verification and Validation in Systems Engineering: Assessing UML/SysML Design Models. Springer Science & Business Media.

[dos Santos Soares et al., 2011] dos Santos Soares, M., Vrancken, J., et Verbraeck, A. (2011). User requirements modeling and analysis of software-intensive systems. Journal of Systems and Software, 84(2):328--339.

[Espinoza et al., 2009] Espinoza, H., Cancila, D., Selic, B., et Gérard, S. (2009). Challenges in combining SysML and MARTE for model-based design of embedded systems. In Model Driven Architecture-Foundations and Applications, pages 98--113. Springer.

[Fernandez et al., 1996] Fernandez, J.-C., Garavel, H., Kerbrat, A., Mounier, L., Mateescu, R., et Sighireanu, M. (1996). CADP a protocol validation and verification toolbox. In Computer Aided Verification, pages 437--440. Springer.

[Ferro et al., 2010] Ferro, L., et Pierre, L. (2010). ISIS: Runtime verification of TLM platforms. In Advances in Design Methods from Modeling Languages for Embedded Systems and SoC's, pages 213--226. Springer.

[Foster et al., 2005] Foster, H., Marschner, E., et Wolfsthal, Y. (2005). IEEE 1850 PSL: The next generation. In DVCon'05: Design and Verification Conference and exhibition. Citeseer.

[Friedenthal et al., 2008] Friedenthal, S., Moore, A., et Steiner, R. (2008). OMG Systems Modeling Language (OMG SysML™) Tutorial. In INCOSE International Symposium, volume 18, pages 1731--1862. Wiley Online Library.

[Friesen, 2011] Friesen, A. (2011). On Challenges in Enterprise Systems Management and Engineering for the Networked Enterprise of the Future. In Enterprise Interoperability, pages 1--2. Springer.

[Fritzson et al., 1998] Fritzson, P., et Engelson, V. (1998). Modelica—a unified object-oriented language for system modeling and simulation. In ECOOP'98—Object-Oriented Programming, pages 67--90. Springer.

[Fummi et al., 2008] Fummi, F., Quaglia, D., et Stefanni, F. (2008). A SystemC-based framework for modeling and simulation of networked embedded systems. In Specification, Verification and Design Languages, 2008. FDL 2008. Forum on, pages 49--54. IEEE.

[Galos et al., 2013] Galos, M., Mieyeville, F., Navarro, D., et O'Connor, I. (2013). SystemC fine-grained HW--SW fully heterogeneous WSN simulation and UML metamodel behavioural extraction. Analog Integrated Circuits and Signal Processing, 77(2):123--133.

[Gardner et al., 2003] Gardner, T., Griffin, C., Koehler, J., et Hauser, R. (2003). A review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards the final Standard. In MetaModelling for MDA Workshop, volume 13, page 41. Citeseer.

[Glasser, 2009] Glasser, M. (2009). Open Verification Methodology Cookbook. Springer Publishing Company, Incorporated, 1st edition.

[Gnesi et al., 2002] Gnesi, S., Latella, D., et Massink, M. (2002). Modular semantics for a UML statechart diagrams kernel and its extension to multicharts and branching time model-checking. The Journal of Logic and Algebraic Programming, 51(1):43--75.

[Goldsby et al., 2006] Goldsby, H., Cheng, B. H., Konrad, S., et Kamdoum, S. (2006). A visualization framework for the modeling and formal analysis of high assurance systems. In Model Driven Engineering Languages and Systems, pages 707--721. Springer.

[Große et al., 2007] Große, D., Ebendt, R., et Drechsler, R. (2007). Improvements for constraint solving in the SystemC verification library. In Proceedings of the 17th ACM Great Lakes symposium on VLSI, pages 493--496. ACM.

[Große et al., 2010] Große, D., Le, H. M., et Drechsler, R. (2010). Proving transaction and system-level properties of untimed SystemC TLM designs. In Formal Methods and Models for Codesign (MEMOCODE), 2010 8th IEEE/ACM International Conference on, pages 113--122. IEEE.

[Gunter et al., 2000] Gunter, C., Gunter, E. L., Jackson, M., Zave, P., et others (2000). A reference model for requirements and specifications. In Requirements Engineering, 2000. Proceedings. 4th International Conference on, page 189. IEEE.

[Haedicke et al., 2012] Haedicke, F., Le, H. M., Große, D., et Drechsler, R. (2012). CRAVE: An advanced constrained random verification environment for SystemC. In System on Chip (SoC), 2012 International Symposium on, pages 1--7. IEEE.

[Hai-yan et al., 2001] Hai-yan, C., Wei, D., Ji, W., et Huo-wang, C. (2001). Verify UML statecharts with SMV. Wuhan University Journal of Natural Sciences, 6(1-2):183--190.

[Hammad et al., 2002] Hammad, A., Tatibouët, B., Voisinet, J.-C., et Weiping, W. (2002). From a B specification to UML statechart diagrams. In Formal Methods and Software Engineering, pages 511--522. Springer.

[Hause et al., 2010] Hause, M., Stuart, A., Richards, D., et Holt, J. (2010). Testing safety critical systems with SysML/UML. In Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on, pages 325--330. IEEE.

[Havelund et al., 1997] Havelund, K., Skou, A., Larsen, K. G., et Lund, K. (1997). Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE, pages 2--13. IEEE.

[Holzmann, 1997] Holzmann, G. J. (1997). The model checker SPIN. IEEE Transactions on software engineering, (5):279--295.

[Honour, 1998] Honour, E. C. (1998). INCOSE: history of the International Council on Systems Engineering. Systems Engineering, 1(1):4--13.

[Houcque, 2005] Houcque, D. (2005). Introduction to matlab for engineering students. Northwestern University.

[Jackson, 1985] Jackson, M. (1985). Developing ada programs using the Vienna development method (VDM). Software: Practice and Experience, 15(3):305--318.

[Jain et al., 2012] Jain, V., Kumar, A., et Panda, P. (2012). Exploiting UML based validation for compliance checking of TLM 2 based models. Design Automation for Embedded Systems, 16(2):93--113.

[Kasuya et al., 2007] Kasuya, A., et Tesfaye, T. (2007). Verification Methodologies in a TLM-to-RTL Design Flow. In Proceedings of the 44th Annual Design Automation Conference, DAC '07, pages 199--204.

[Kelton et al., 2000] Kelton, W. D., et Law, A. M. (2000). Simulation modeling and analysis. McGraw Hill Boston.

[Kent, 2002] Kent, S. (2002). Model driven engineering. In Integrated formal methods, pages 286--298. Springer.

[Kleijnen, 1995] Kleijnen, J. P. (1995). Verification and validation of simulation models. European Journal of Operational Research, 82(1):145--162.

[Knapp, 2002] Knapp, A., M. S. (2002). Model checking and code generation for UML state machines and collaborations. In Proceedings of 5th Workshop on Tools for System Design and Verification Report 2002-11, Reisensburg, Germany, Institut fur Informatik, Universitat Augsburg.

[Kossiakoff et al., 2011] Kossiakoff, A., Sweet, W. N., Seymour, S., et Biemer, S. M. (2011). Systems engineering principles and practice, volume 83. John Wiley & Sons.

[Kwon, 2000] Kwon, G. (2000). Rewrite rules and operational semantics for model checking UML statecharts. In UML 2000—The Unified Modeling Language, pages 528--540. Springer.

[Laleau et al., 2010] Laleau, R., Semmak, F., Matoussi, A., Petit, D., Hammad, A., et Tatibouet, B. (2010). A first attempt to combine SysML requirements diagrams and B. Innovations in Systems and Software Engineering, 6(1-2):47--54.

[Lanusse et al., 2009] Lanusse, A., Tanguy, Y., Espinoza, H., Mraidha, C., Gerard, S., Tessier, P., Schnekenburger, R., Dubois, H., et Terrier, F. (2009). Papyrus uml: an open source toolset for mda. In Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009), pages 1--4. Citeseer.

[Larsen et al., 1997] Larsen, K. G., Pettersson, P., et Yi, W. (1997). UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer (STTT), 1(1):134--152.

[Latella et al., 1999] Latella, D., Majzik, I., et Massink, M. (1999). Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker. Formal Aspects of Computing, 11(6):637--664.

[Lazăr et al., 2010] Lazăr, C., Lazăr, I., Pârv, B., Motogna, S., et Czibula, I. (2010). Tool Support for fUML Models. Int. J. of Computers, Communications & Control, 5(5):775--782.

[Leuschel et al., 2003] Leuschel, M., et Butler, M. (2003). ProB: A model checker for B. In FME 2003: Formal Methods, pages 855--874. Springer.

[Leuschel et al., 2001] Leuschel, M., Currie, A., et Massart, T. (2001). How to make fdr spin ltl model checking of csp by refinement. In FME 2001: Formal Methods for Increasing Software Productivity, pages 99--118. Springer.

[Ma et al., 2014] Ma, X., Rinast, J., Schupp, S., Gollmann, D., Turau, V., Kwiatkowska, M., Mangharam, R., et Weyer, C. (2014). Evaluating on-line model checking in uppaal-smc using a laser tracheotomy case study. In MCPS, pages 100--112.

[Machida et al., 2011] Machida, F., Andrade, E., Kim, D. S., et Trivedi, K. S. (2011). Candy: Component-based availability modeling framework for cloud service management using sysml. In Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on, pages 209--218. IEEE.

[Mason et al., 2004] Mason, P., Cosh, K., et Vihakapirom, P. (2004). On structuring formal, semi-formal and informal data to support traceability in systems engineering environments. In Proceedings of the thirteenth ACM international conference on Information and knowledge management, pages 642--651. ACM.

[McUmber et al., 2001] McUmber, W. E., et Cheng, B. H. (2001). A general framework for formalizing UML with formal languages. In Proceedings of the 23rd international conference on Software engineering, pages 433--442. IEEE Computer Society.

[Mello et al., 2010] Mello, A., Maia, I., Greiner, A., et Pecheux, F. (2010). Parallel simulation of SystemC TLM 2.0 compliant MPSoC on SMP workstations. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010, pages 606--609. IEEE.

[Mikk et al., 1998] Mikk, E., Lakhnech, Y., Siegel, M., et Holzmann, G. J. (1998). Implementing statecharts in PROMELA/SPIN. In Industrial Strength Formal Specification Techniques, 1998. Proceedings. 2nd IEEE Workshop on, pages 90--101. IEEE.

[Miyamoto et al., 2012] Miyamoto, N., et Wasaki, K. (2012). An Integrated Design and Verification Environment Handling the Transformation from Upstream Design to the Model Checking Process. International Journal of Advancements in Computing Technology, 4(14).

[MKuster et al., 2012] MKuster, A. V., Burger, A., et Rosenstiel, O. B. W. (2012). Meta-Modelling the SystemC Standard for Component-based Embedded System Design. In Proceedings of the 1st International Workshop on Metamodelling and Code Generation for Embedded Systems (MeCoES), S, pages 35--40.

[Mueller, 2013] Mueller, W. (2013). How do they apply in embedded system design? UML-B Specification for Proven Embedded Systems Design, page 1.

[Mura et al., 2008] Mura, M., Panda, A., et Prevostini, M. (2008). Executable models and verification from marte and sysml: a comparative study of code generation capabilities. In Proceedings of MARTE Workshop (DATE08), Munich, Germany.

[Nikiforova et al., 2012] Nikiforova, O., Pavlova, N., Gusarovs, K., Gorbiks, O., Vorotilovs, J., Zaharovs, A., Umanovskis, D., Sejans, J., et others (2012). Development of the Tool for Transformation of the Two-Hemisphere Model to the UML Class Diagram: Technical Solutions and Lessons Learned. In Proceedings of the 5th International Scientific Conference „Applied Information and Communication Technology, pages 11--19.

[Ober, 2004] Ober, I., G. S. O. I. (2004). Validation of UML models via a mapping to communicating extended timed automata. In Proceedings of 11th International SPIN Workshop on Model Checking of Software, 2004. Volume LNCS 2989.

[Oberkampf et al., 2010] Oberkampf, W. L., et Roy, C. J. (2010). Verification and validation in scientific computing. Cambridge University Press.

[Odey et al., 2012] Odey, A. J., et Li, D. (2012). Low power transceiver design parameters for wireless sensor networks.

[Oliveira et al., 2012] Oliveira, M. F., Kuznik, C., Le, H. M., Große, D., Haedicke, F., Mueller, W., Drechsler, R., Ecker, W., et Esen, V. (2012). The system verification methodology for advanced TLM verification. In Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pages 313--322. ACM.

[OMG, 2008] OMG (2008). Mof model to text transformation language.

[OMG, 2012] OMG (2012). OMG Systems Modeling Language (OMG SysML™) Version 1.3.

[OMG,] OMG, M. Guide, Version 1.0. 1, 2003. Object Management Group, page 62.

[Othman et al., 2013] Othman, S., Beydoun, G., Clarke, R., et Opper, S. (2013). DM model transformations framework. In 24th Australasian Conference on Information Systems (ACIS), pages 1--12. RMIT University.

[Pedroza et al., 2011] Pedroza, G., Apvrille, L., et Knorreck, D. (2011). Avatar: A sysml environment for the formal verification of safety and security properties. In New Technologies of Distributed Systems (NOTERE), 2011 11th Annual International Conference on, pages 1--10. IEEE.

[Piccolboni et al., 2014] Piccolboni, L., et Pravadelli, G. (2014). Simplified stimuli generation for scenario and assertion based verification. In Test Workshop-LATW, 2014 15th Latin American, pages 1--6. IEEE.

[Plat et al., 1992] Plat, N., et Larsen, P. G. (1992). An overview of the ISO/VDM-SL standard. ACM Sigplan Notices, 27(8):76--82.

[Pnueli, 1977] Pnueli, A. (1977). The temporal logic of programs. In Foundations of Computer Science, 1977., 18th Annual Symposium on, pages 46--57. IEEE.

[Pockrandt et al., 2012a] Pockrandt, M., Herber, P., Gross, H., et Glesner, S. (2012a). Optimized Transformation and Verification of SystemC Methods. Electronic Communications of the EASST, 53.

[Pockrandt et al., 2012b] Pockrandt, M., Herber, P., Gross, H., et Glesner, S. (2012b). Optimized Transformation and Verification of SystemC Methods.

[Prevostini et al., 2007] Prevostini, M., et Zamsa, E. (2007). Sysml profile for soc design and systemc transformation. ALaRI, Faculty of Informatics University of Lugano via G. Buffi, 13(5).

[Razavi et al., 2011] Razavi, N., Behjati, R., Sabouri, H., Khamespanah, E., Shali, A., et Sirjani, M. (2011). Sysfier: Actor-based Formal Verification of SystemC. ACM Trans. Embed. Comput. Syst., 10(2):19:1--19:35.

[Riccobene et al., 2012] Riccobene, E., et Scandurra, P. (2012). Integrating the SysML and the SystemC-UML profiles in a model-driven embedded system design flow. Design Automation for Embedded Systems, pages 1--39.

[Riccobene et al., 2009] Riccobene, E., Scandurra, P., Bocchio, S., Rosti, A., Lavazza, L., et Mantellini, L. (2009). Systemc/c-based model-driven design for embedded systems. ACM Transactions on Embedded Computing Systems (TECS), 8(4):30.

[Rowson et al., 1994] Rowson, J., et others (1994). Hardware/software co-simulation. In Design Automation, 1994. 31st Conference on, pages 439--440. IEEE.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., et Booch, G. (1999). The uml reference manual. New York: Addison-Wesley, 1:999.

[Schinz, 2004] Schinz, I., T. T. M. C. W. B. (2004). The Rhapsody UML verification environment. In Proceedings of SEFM, IEEE Computer Society (2004), pages 174--183.

[Scholtz et al., 2013] Scholtz, B., Calitz, A., et Snyman, I. (2013). The usability of collaborative tools: application to business process modelling. In Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference, pages 347--358. ACM.

[Shahdad, 1986] Shahdad, M. (1986). An overview of VHDL language and technology. In Proceedings of the 23rd ACM/IEEE Design Automation Conference, pages 320--326. IEEE Press.

[Snook et al., 2006] Snook, C., et Butler, M. (2006). UML-B: Formal modeling and design aided by UML. ACM Transactions on Software Engineering and Methodology (TOSEM), 15(1):92--122.

[Soley et al., 2000] Soley, R., et others (2000). Model driven architecture. OMG white paper, 308(308):5.

[Soliman et al., 2012] Soliman, D., Thramboulidis, K., et Frey, G. (2012). Transformation of Function Block Diagrams to UPPAAL timed automata for the verification of safety applications. Annual Reviews in Control, 36(2):338--345.

[Specification, 2008] Specification, O. A. (2008). MOF Model to Text Transformation Language.

[Sreemani et al., 1996] Sreemani, T., et Atlee, J. M. (1996). Feasibility of model checking software requirements: A case study. In Computer Assurance, 1996. COMPASS'96, Systems Integrity. Software Safety. Process Security. Proceedings of the Eleventh Annual Conference on, pages 77--88. IEEE.

[Tabakov et al., 2012] Tabakov, D., Rozier, K. Y., et Vardi, M. Y. (2012). Optimized temporal monitors for SystemC. Formal Methods in System Design, 41(3):236--268.

[Vanderperren et al., 2012] Vanderperren, Y., Mueller, W., He, D., Mischkalla, F., et Dehaene, W. (2012). Extending UML for Electronic Systems Design: A Code Generation Perspective. In Design Technology for Heterogeneous Embedded Systems, pages 13--39. Springer.

[Vieira et al., 2014] Vieira, A., et Ramalho, F. (2014). Metrics to Measure the Change Impact in ATL Model Transformations. In Product-Focused Software Process Improvement, pages 254--268. Springer.

[Windisch et al., 2013] Windisch, A., Monjau, D., Schneider, T., Mades, J., Glesner, M., et Ecker, W. (2013). A VHDL-Centric Mixed-Language Simulation. System-on-Chip Methodologies & Design Languages, page 37.

[Wymore, 1993] Wymore, A. W. (1993). Model-based systems engineering, volume 3. CRC press.

[Zervoudakis et al., 2013] Zervoudakis, F., Rosenblum, D. S., Elbaum, S., et Finkelstein, A. (2013). Cascading verification: an integrated method for domain-specific model checking. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pages 400--410. ACM.

List of Figures

List of Tables

Document generated with LaTeX and:
the LaTeX style for PhD Thesis created by S. Galland --- http://www.multiagent.fr/ThesisStyle
the tex-upmethodology package suite --- http://www.arakhne.org/tex-upmethodology/

Abstract:

Heterogeneous Systems are complex and become very critical. These systems integrate software and hardware components with intensive interaction between them. In this context, there is a strong necessity to develop methodologies and techniques to specify and validate these systems.
In engineering, the requirements are the expression of needs on what a particular product or a service should be or to make. They are used most of the time in a formal sense in the systems engineering. In this kind of systems, several types of requirements are present: non-functional requirements such as the performance and the reliability and functional requirements such as the liveliness. To validate these requirements of a system, an environment to simulate and to check the properties is essential.

In our work, we propose a methodology based on SysML combined with SystemC and Promela/SPIN to specify and validate complex systems. This approach is based on Model Driven Engineering techniques to firstly translate SysML models to systemC with the aim of simulation and to map SysML behavioral diagrams to Promela/SPIN in order to verify temporal properties extracted from the requirements. The approach is experimented on case studies to demonstrate its feasibility.

Keywords: SysML specifications, Block diagrams, Model Driven Engineering MDE, SystemC, Simulation, Verification, Requirements, LTL properties, Model-Checking, Promela/SPIN, STATE tool, UPPAAL.

Résumé :

De nombreux systèmes hétérogènes sont complexes et critiques. Ces systèmes intègrent du logiciel et des composants matériels avec des interactions fortes entre ces composants. Dans ce contexte, il est devenu absolument nécessaire de développer des méthodologies et des techniques pour spécifier et valider ces systèmes.
Dans l'ingénierie des systèmes, les exigences sont l'expression des besoins qu'un produit spécifique ou un service doit réaliser. Elles sont définies formellement à de nombreuses occasions dans l'ingénierie des systèmes complexes. Dans ce type de système, deux catégories d'exigence sont présentes : les exigences non-fonctionnelles telles que la performance et la fiabilité, les exigences fonctionnelles telles que la vivacité. Pour valider ces exigences, un environnement permettant de simuler et vérifier ces propriétés est essentiel.
Dans notre travail, nous proposons une méthodologie basée sur SysML et combinée avec SystemC et Promela/SPIN pour spécifier et valider des systèmes complexes. Cette approche est basée sur l'ingénierie dirigée par les modèles pour premièrement traduire des modèles SysML en SystemC afin de réaliser des simulations et deuxièmement traduire des diagrammes d'état SysML en Promela/SPIN afin de vérifier des propriétés temporelles extraites des exigences. Cette approche est expérimentée sur une étude de cas pour démontrer sa faisabilité.

Mots-clés : Spécifications SysML, Diagrammes de block, Ingénierie Dirigée par les Modéles IDM, SystemC, Simulation, Vérification, Exigences, Propriétés LTL, Model-Checking, Promela/SPIN, STATE tool, UPPAAL.

SPIM

■ École doctorale SPIM 16 route de Gray F - 25030 Besançon cedex
■ tél. +33 (0)3 81 66 66 02 ■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr

UFC
UNIVERSITÉ
DE FRANCHE-COMTÉ