Cyber Physical System Applications Verification and Validation

Teză destinată obţinerii
titlului ştiinţific de doctor inginer
la
Universitatea "Politehnica" din Timişoara
în domeniul CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI
de către

Ing. Mădălin Gavrilescu

Conducător ştiinţific: prof.dr.ing. Ionel Jian
Referenţi ştiinţifici: prof.dr.ing. Dumitru Burdescu
prof.dr.mat. Alexandru Cicortaş
prof.dr.ing. Ştefan Holban

Ziua susţinerii tezei: 29.11.2013

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Ingineria Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică şi Telecomunicaţii
8. Inginerie Industrială
9. Inginerie Mecanică
10. Ştiinţa Calculatoarelor
11. Ştiinţa şi Ingineria Materialelor
12. Ingineria sistemelor
13. Inginerie energetică
14. Calculatoare şi tehnologia informaţiei
15. Ingineria materialelor

Universitatea „Politehnica” din Timişoara a iniţiat seriile de mai sus în scopul diseminării expertizei, cunoştinţelor şi rezultatelor cercetărilor întreprinse în cadrul şcolii doctorale a universităţii. Seriile conţin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susţinute în universitate începând cu 1 octombrie 2006.

România, 300159 Timişoara, Bd. Republicii 9,
tel. 0256 403823, fax. 0256 403221
e-mail: editura@edipol.upt.ro

To my brother, Cătălin.
You represent so much for me!
.

Acknowledgement

This thesis is the result of my efforts over the last few years in the field of Cyber Physical System simulation. I would like to take this opportunity to express my gratitude to all those that supported me during this time.

I would like to thank my scientific research coordinators, Prof. Dr. Ionel Jian and Assoc. Prof. Dr. Dan Pescaru for their help during these years, for their advices and guidance, which have been of great use. I would also like to thank Assoc. Prof. Dr. Alex Doboli for his support in the first year of my research.

Special thanks go to my friend and PhD colleague, Gabriela Măgureanu, who has worked with me in all these years. Thank you for your commitment, for encouraging me when I needed it and also for putting up with all my moods. I appreciate you staying by my side for this entire period. In the end, it was all worth it.

A grateful thought goes to my friend Mădălina, who encouraged me to start the wonderful and important journey of academic research. I hope she enjoys the result.

I feel the words are not enough to detail how much I would like to thank my family, who was there for me regardless of how difficult I proved to be at times, for all their love and for supporting me in this pursuit. Thanks from my heart go to my brother Cătălin, my sister-in-law Anca, my mother and my father. This thesis would not have been possible without all of you.

Not lastly, I would like to thank my girlfriend Cristina, whose support and patience during the final stages of this PhD are appreciated.

Timişoara,
November 29-th, 2013

Mădălin GAVRILESCU

Gavrilescu, Mădălin

Cyber Physical System Applications Verification and Validation

Teze de doctorat ale UPT, Seria X, Nr. YY, Editura Politehnica, 200Z, 111 pagini, 36 figuri, 5 tabele.

ISSN:
ISSN-L:
ISBN:

Cuvinte cheie: sisteme cyber fizice, modele de simulare bazate pe dispizitive PSoC, verificare formala, arhitectura orientata pe modele

Rezumat,

Teza de doctorat prezintă un proces complet de testare si validare pentru aplicatii ale sistemelor cyber fizice. O astfel de abordare vine in sprijinul dezvoltatorilor de aplicaţii pentru astfel de sisteme.

Lucrarea demonstrează utilitatea unei astfel de abordări la nivel de validare prin studii de caz ce discută aplicații ale sistemelor cyber fizice din diverse domenii de activitate, cu nivel diferit de dificultate la nivel de cerințe de aplicație si de diferite dimensiuni.

Table of Contents

Abstract

Cyber physical systems (CPS) applications can be found in the last few years in various domains of activity and the interest given to this type of systems is growing worldwide. CPSs are massively distributed heterogeneous systems, linked through wired or wireless connections. They are characterized by both computational and physical processes and have an increasing economic and social potential. Using CPSs for designing and implementing applications in different domains of activity is a rather new approach for applications of sensor networks. Therefore, there are no general accepted solutions for the issues which can appear in the different stages of implementation.

The efforts of the research team produced an efficient, intuitive and easy to use high-level programming methodology for CPS applications. The implementation is based on Model Driven Architecture (MDA) approach, which provides a high-level perspective of specifying and deploying CPS applications. The innovation of this methodology is the goal-oriented perspective in specifying applications objectives and constraints. The research had two directions. The first research theme was the design of the CPS applications, starting from the requirements and using the defined UML artifacts. The second theme was testing and verification of the CPS applications. The goal was to validate the models before deployment on physical network. The second direction has constituted the research theme for the author of this thesis.

This thesis presents as novelty the complexity of the testing and validation process, the thorough verifications posed on both static and dynamic aspects in a CPS network, starting from component level and going until network level, the focus being on errors identification as well as errors handling mechanisms.

The author of this thesis used rigorous specification for identifying the lack of requirements in static properties of CPS applications, in units, nodes and network communication capabilities. The author also developed simulation models, which help reflect the possible dynamic aspects in the CPS networks. The testing and verification of the units used in the CPS application have formed the clock cycle level tests, integrating the units led to node level tests while the entire application's verification implied workflow tests, at network level. The author of this thesis has considered also the particular cases which can appear during a CPS network lifecycle, which are not seen as errors but require an appropriate handling.

Validation is a very important phase in developing the programming model for CPS applications. A tested and verified CPS network can be deployed on the physical environment with the certainty that the specifications errors and a large part of the behavioral errors have been already removed. This leads to the increased credibility for the programming methodology.

List of Abbreviations

API	Application Programming Interface
CIM	Computation Independent Model
CLV	Cross Left Valve
CPS	Cyber Physical System
CRV	Cross Right Valve
DM	Decision Module
DMA	Decision Module Area
DMP	Decision Module Perimeter
DMZ	Decision Module Zone
FSM	Finite State Machine
CG	Center of Gravity
LP	Linear Programming
NED	Network Description
NIC	Network Interface Cards
OCL	Object Constraints Language
OMNeT++	Objective Modular Network Testbed in C++
OOP	Object Oriented Programming
PIM	Platform Independent Model
PSoC	Programmable System-on-Chip
PSM	Platform Specific Model
PVS	Prototype Verification System
UML	Unified Modeling Language
TCC	Type Correctness Conditions

List of Figures

List of Tables

Chapter 1. Introduction

1.1 The Research Theme

Cyber physical systems (CPSs), as presented by Lee in [1], are massively distributed heterogeneous embedded systems, linked through wired and/or wireless connections. They integrate computational and physical processes, sensors, actuators and decision modules. Applications of CPSs have a great economic and social potential and can be found in various fields of activity. However, designing and developing networks composed of a large number of nodes is far from being an easy task due the challenges that programming massively distributed structures implies.

The current research aims to provide an efficient, intuitive and easy to use high-level programming model-based methodology for deploying CPS applications. The developer of such applications is not required to have deep knowledge of low-level programming skills; therefore the effort tends to be reduced on design and implementation level.

The first part of this research, *Visual modeling of Cyber Physical Systems,* has been presented by my research team colleague, Gabriela Magureanu, in [2].

The developed methodology is based on Object Management Group (OMG) Model Driven Architecture (MDA) approach [3]. It supports the design of various computational models and allows customization based on the application requirements. The novelty resides in the way the application requirements are specified, in a goal-oriented manner. The design approach addresses both hardware and software aspects of a CPS application and is based on Unified Modeling Language (UML) [4]. UML profiles allow specific type definitions for families of applications and are used for tailoring UML to application specific requirements [5].

MDA approach provides a high-level perspective of specifying and deploying CPS applications. It offers the possibility to design Platform Independent Models (PIMs) that can be translated into several Platform Specific Models (PSMs) [3]. Using MDA approach allows generating both simulation and application code. The simulation part for validating such applications represents an important step in the deployment process. The amount of generated code in the total code written for an application and also its precision depend on the accuracy and complexity of the transformation rules used.

The research of the author of this thesis aims to validate the UML models defined for the CPS applications. This implies testing and verification of the PIM for a specific CPS network. PIM validation includes verification at both hardware and software level, therefore both static and dynamic properties of the system. The tests are at unit level (clock cycle level tests), at node level and at network level (workflow tests).

PIM validation implies that the programming methodology is a functional solution and a CPS application deployed in the physical environment will have no errors on requirement level and a reduced number of errors regarding behavior.

1.2 Thesis Objectives

This thesis intends to accomplish the following:

- From a theoretical point of view, to define a methodology for testing and verifying CPS applications, at each level for the CPS design, for both static and dynamic aspects
  - To determine a rigorous specification of the static properties defined for units, nodes and network communication capabilities
  - To define simulation models for CPS applications, where the nodes are composed of PSoC devices and also optimization methodologies, for dynamic tests
  - Clock cycle level tests: testing and verification of the units used in CPS applications
  - Node level tests: Units integration and collaboration inside CPS nodes
  - Network level: workflow tests and CPS application validation
  - To define a handling manner of event-driven scenarios in simulating CPS applications: an event-driven simulation model and an event-driven programming model
  - To define a methodology for handling special cases that can appear during a CPS application lifecycle and also equipment failures

- From a practical point of view, to apply the validation techniques to CPS applications, in order to demonstrate the correctness and completeness of the testing and validation process
  - The CPS network are composed of sensors, actuators, communications units and decision nodes
  - The CPS network size can vary up to a large number of nodes
  - The CPS applications belong to different activity domains and can have various degrees of difficulty.

1.3 The Proposed Approach

A very important phase in the definition of the design and programming model for CPS application is the validation, which must be performed on the UML models corresponding to hardware and software specifications. This implies PIM validation before deployment on a PSM, the final objective being deployment on physical network without errors regarding CPS topology and with a reduced number of errors regarding application's behavior. The author of this thesis has developed methodologies to validate both UML models for both the hardware and software specifications of the CPS applications. Additionally, he defined handling manners for the special situations which can appear during the lifecycle of the CPS application, which are not errors, but also for equipment failure.

The UML models corresponding to the hardware representation of the CPS application, the network topology, can be verified by a rigorous specification at unit, node and network level. The opportunity of using this approach for validation of

static properties in CPS UML models is demonstrated considering some relevant case studies presented in literature. Structural models of CPS applications represent static information in a system design.

The UML models corresponding to the software representation of the CPS application, the network behavior, can be tested and verified in a simulation environment. To improve simulation duration and accuracy, simulation models for CPS networks composed of Programmable System-on-Chip (PSoC) devices are defined. They can be applied in case of different type of PSoC devices: devices running at the same clock frequency or different clock frequencies or devices having different clock offset. Also, a defined speed-up mechanism can be used for these simulation models, depending on the application's requirements.

Another part of the testing and validation process is constituted by the definition of the handling approach for of event-driven scenarios in simulating CPS applications. The author of this thesis proposes both an event-driven simulation model and an event-driven programming model to be customized starting from the application's specifications. This approach is suitable for a large range of CPS applications.

The author of this thesis also discusses particular situations which can appear during the lifecycle of a CPS application, regardless of the requirements. He uses specific case studies and scenarios to exemplify the handling methodology.

Starting from the specifications for a CPS application, one can use the methodology based on MDA approach and goal-oriented description of the application objectives to design and prepare the deployment to the physical network. While composing the UML models to describe the hardware and software specification for the CPS application plays an important role, it is of high importance to be able to validate the defined UML models. Following the testing and validation process defined by the author of this thesis, testing and verifying both network topology and network behavior ensures that the CPS design is correct and complete and the application behaves as expected. This implies that once the CPS application is validated to be deployed in the physical environment, it will have no errors regarding hardware requirements at unit, node or network level and it will behave as required and simulated.

1.4 Thesis Organization

The rest of the chapters in this thesis are organized as follows.

Chapter 2 reviews the visual modeling of CPSs, as summarized by Magureanu in [2], which includes an overview of this type of systems, the MDA approach used in the visual programming model for CPS applications developed inside the research team, the UML profiles and the goal-oriented approach in specifying CPS applications requirements and behavior. The same chapter includes an overview of the simulation frameworks described by literature for testing and verifying CPS applications. The last subchapter discussed modeling and validation approaches, starting from a rigorous specification of the requirements, the behavior and the objectives of CPS applications.

Chapter 3 presents the simulation models for CPS applications constructed on PSoC devices, developed by the author of this thesis using OMNeT++ simulation

framework. The simulation models are intended for different types of PSoC devices, which cover a large number of CPS topologies.

Chapter 4 presents specification and validation of static aspects regarding CPS applications which are described by UML models. The rigorous specification is exemplified in three different case studies, which underline several steps of the validation process.

Chapter 5 focuses on testing and verification of dynamic, behavioral aspects of CPS applications. The author of this thesis presents the handling manner of event-driven scenarios in simulating CPS applications. He discusses the specification of an event-driven simulation model and also an event-driven programming model used in the validation process.

Chapter 6 presents several situations which can appear during a CPS applications lifecycle, either application special cases or equipment failure and the approach to handle them using linear programming equations at specification level.

Chapter 7 concludes the thesis, presents the contributions, the publications where the author of the thesis is coauthor and presents future work perspectives.

Chapter 2. State of the Art

2.1 CPS Overview

CPSs integrate computation and physical processes and during this thesis are considered to be composed mainly of sensors, actuators, communication and control devices based on PSoC chips. The connections in CPS networks are made wired or wireless. Controlling such CPS networks by communication and command is a continuous challenge for researchers all over to world, in search of developing efficient technology for CPS.

2.1.1 CPS Modeling and Design

For embedded systems, the requirements for reliability and predictability were always higher than in the case of general-purpose computers because customers do not expect their car or TV to need a reboot [1]. The physical world is far from being predictable, therefore CPS networks must be characterized by robustness, as adaptation to unexpected environment conditions and system failures. As Lee stated in [1], in a CPS, the components at each layer of abstraction must be characterized by predictability and reliability, if this is a technologically feasible case. If this is not the case, the higher level of abstraction must compensate with robustness.

The organization into logical layers for the CPS network is the basic idea of the goal-oriented design and programming model intended for CPS applications, developed inside the research team. The main objective is to offer the designer of a CPS application the possibility to simply specify the requirements at the highest logical level, which is the network level. It is in care of the logic implemented in the decision nodes to translate the commands to the next logical level, until the physical nodes are also programmed.

Over the last years, a large number of applications have used CPSs for implementing the corresponding networks. Several examples of fields of activity for such systems are: intelligent traffic management, infrastructure management, critical infrastructure monitoring, healthcare, aerospace, energy consumption optimization in vehicles and buildings [6]. CPSs are characterized by sensing and actuation devices which are cheap, small and can therefore be deployed in a large number. This aspect makes CPSs suitable in applications where a high degree of precision is required. This is ensured by the large quantity of information that can be gathered from nodes. Malfunction of local nodes or even subsystems must not affect the entire CPS; therefore, this system must have a high degree of reliability. Large size networks are difficult to control as related problems are tackled together and not separately [7]. The control procedures in a CPS must understand and react

accordingly to quality failures caused by the application requirements. The required adjustments are far from being trivial, as the network contains a large number of parameters.

A suitable approach in controlling CPS networks is by logically layering the devices that compose such systems. Decisions at lower computational levels are influenced by decisions at higher computational levels and vice versa. The lower logical level, corresponding to the physical level, mainly addresses local constraints. The decisions at higher logical levels influence larger areas and at such level, the parameters for a certain model change much slower. At lower logical levels, the decisions can be taken using reactive models. At higher levels Task Graphs or Markov Decision Processes can be considered.

CPS design and programming is a rather new topic for researchers worldwide, by comparing it to subjects like sensor networks and embedded systems. Literature presents several attempts that are worth considering when proposing a new approach.

Lee discusses in [8] the open points in challenges in CPS design and presents the CPS aspect as rather an intersection than a reunion of cyber and physical issues [9]. His research and articles have been used as the starting point for several programming models for CPSs.

In [10], Tabuada discusses a decoupling within certain limitations of the physical characteristics of CPSs and the part that is available to the end user. He introduces notions on the topological abstractions of the physical devices, such as the notion of locality, with different meanings in the physical environment and for the CPS network based on sensors and actuators. Also, he discusses in-network computation, the information gathered from sensors and the commands to actuators to be managed within the network. The concepts presented in Tabuada's proposal are very similar to vision about CPS modeling inside the research team to which the author of this thesis belongs to. The separation into computational layers and the handling at the level of decision nodes, inside the CPS network, are similar aspects.

In [11], Gupta focuses on defining a programming support for location and time information in CPS applications. He presents the importance of a semantic support to use the physical location information and the validation of application's models against spatial and timing requirements.

In [12], Derler et al. discuss CPS modeling from perspectives of heterogeneity, concurrency and sensitivity to timing. The authors consider as CPS example a part of an aircraft vehicle, the fuel management system, for which they propose several solutions. The authors use the Ptolemy project, as described in [13], for the fuel system modeling and for simulating the flow of fuel between tanks.

In [14], Saeedloei and Gupta argue that the hybrid automata is a possible solution for specifying, designing and analyzing several types of systems, in particular CPSs. They model hybrid automata using logic programming with several extensions. The authors use the proposed framework for several CPS network examples, as the generalized railroad crossing problem and the reactor temperature control systems. The logical programming approach can be considered to have good evolving perspectives, as the authors have already presented the results in several related papers.

In [15], Liu discusses the adaptation of unified object model and classical programming techniques to CPS programming. Already defined software engineering technologies and tools, including UML, can be put together to CPS design. The strong points of Object Oriented Programming (OOP), encapsulation,

code reuse or customization can have a great impact in CPS design and programming.

CPS Proposal	Year of first publication	Summary
CPS support for location and time	2006	The author underlines the importance of location and time information in CPS semantics [11].
CPS position paper	2006	Decoupling into levels of abstraction. Decision management – taken inside network [10].
CPS position papers	2008	Defining CPS [1], challenges in CPS design. Used as starting point for many CPS researches.
Heterogeneity, concurrency and sensitivity to timing perspectives	2010	Ptolemy project is used as a solution for design, modeling and simulation of the CPS application [13].
Approach with hybrid automata	2011	The model hybrid automata is modeled using logic programming [14]. The approach is applied to several CPS networks.
OOP concepts extended for CPSs	2011	OOP defined concepts are applied in design and simulation of CPS applications [15].

Table 1 Summary of recent programming models for CPSs

2.1.2 Visual Modeling of CPSs

This subchapter entirely presents the state of the art in the research conducted by my team colleague Gabriela Magureanu and presented in her thesis called *Visual modeling of Cyber Physical Systems* [2]. It also represents a starting point of this thesis.

Fast implementing complete and robust applications of CPSs require an intuitive, easy to use and competitive design and programming model. The user of such a programming methodology can use already defined components, by specifying the components his network requires and the relationships between these components. The user does not have to take into consideration hardware requirements and limitations, as those are encapsulated in the components logic, as stated by Magureanu in [2]. The components are handled by the developers for the library of components. The user programs the CPS network, by simply specifying the goals of the application, the system logic being able to implement the desired goals.

The programming methodology implies well-defined steps. The two defined UML profiles cover hardware and software aspects at component, node and network level. The UML profiles form the Computation Independent Model (CIM) in MDA approach.

The UML profiles help defining the MDA Platform Independent Model (PIM). This model contains two types of diagrams. UML deployment diagrams are

customized using stereotypes from the UML hardware profile and are used to characterize the application topology, the types of nodes in the network and the connections and communication between these nodes, according to the application requirements for the hardware level. UML component diagrams describe network behavior, at different logical levels. These logical levels were defined to ease the understanding, designing and programming of the CPS network, as usually it contains a large number of nodes. The physical level corresponds to the lowest logical level. The network as a whole corresponds to the highest logical level. Application goals from higher logical levels are translated into goals at lower logical levels, while the goals at lower logical levels influence the ones from higher logical levels.

Once the PIM is defined, it needs to be tested and verified for validation before transforming it into a Platform Specific Model (PSM), characterized by user specifications and application requirements. Validation can be achieved using simulation models. Simulation as testing and verification methodology is recommended before deployment on hardware devices. Some of the major advantages of simulation are errors detection in early phases of development, as a result of testing, the possibility to obtain partial and/or total validation before deployment and obtaining a deterministic behavior for each component and node and for the entire network. A PIM is later translated into a network deployable PSM by using code specific transformations. The resulted code represents the final scope of using MDA approach.

A. The MDA Approach in CPS Applications

The methodology for designing CPS applications starting from the specifications is intended to be a visual one, based on UML models, in order to provide an intuitive solution for the developers for CPS applications. Two perspectives can be identified when discussing the MDA approach. The first one is the users' perspective, as they are able to use the artifacts already defined in the specialized library. Depending on applications requirements, the defined components can be used as they are or composed into more complex components, with further customizations regarding communication and links between components. The objectives for the CPS application are posed at the highest logical level and are translated into goals for lower logical levels. The influence is bidirectional between logical levels, as the accomplishment or the lack of accomplishment influences goals at higher computational levels.

The UML models corresponding to the hardware part contain definitions for the types of components, nodes, connections and the entire network topology, while the UML models corresponding to the software part define behavior for each of the hardware artifacts. The users can test and validate the UML models defined for both hardware and software specifications of UML networks. There are several methodologies for performing such testing. First, the models can be tested by generating models in a simulation environment. This type of testing and validation is discussed in more details in Chapter 4, with simulation models at low abstraction level. Then, the correctness and completeness of the UML models can be checked by verifying the Object Constraint Language (OCL) constraints defined and attached to the corresponding stereotypes. Also, a rigorous specification of UML models using

specification languages is a proper testing and validation method for static aspects identified in CPS applications. Complete specification and validation is discussed in greater details in Chapter 5.

When testing the UML models is proven unsuccessful, the user has the possibility to correct the high-level UML specifications. After several such steps, the specifications for the CPS applications will be validated. This allows the deployment of the network in the physical environment, with the high degree of confidence of the fact that hardware modeling errors and at least some behavioral errors have been solved. Deployment process implies loading application code, generated after UML models validation on the physical nodes.

The second perspective in which the MDA approach can be discussed is the developers of components perspective. The developers define component with the help of stereotypes defined in the two UML profiles for CPS applications. An UML profile corresponding to the hardware specification has been introduced by Magureanu et al. in [16] and is detailed in subchapter 2.1.2, C. The stereotypes contain tagged values and OCL constraints, which allow expressing customizations for several types of UML elements. The stereotypes help achieving a clear separation of devices and grouping into families of hardware devices. The work in [16] was continued by Magureanu et al. in [17], where customizations for wireless communication between nodes in a CPS network have been defined.

The second defined UML profile inside the research team provides stereotypes for software specification of CPS applications. This has been introduced by Gavrilescu et al. in [18] and detailed in [19]. The stereotypes in the UML software profile define the behavior for the hardware components and the goal-oriented approach. The stereotypes define the logical levels the network is tailored in and the connections of communication and control that each grouping has with another grouping, from the same logical level or from the upper or the lower logical level.

The defined MDA methodology covers different types of CPS applications and has been tested on three types of applications: the management of traffic lights in an intersection, the management of a gas distribution network and the fuel system management from an aircraft vehicle ([16], [17], [18], [19] and [20]).

Studies have indicated that using UML defined stereotypes for expressing the applications specifications help improving the overall understanding of the models in question. Such a study is detailed by Kurniarz et al. in [5]. This is a desired goal, as UML profiles usage helps in a better definition and understanding of the internal configuration of each node which is part of the network and the network as a whole.

The developers of the components define customizations for the possible types of nodes, for the network topology and other hardware units based on UML hardware profile. The developers also define behavior corresponding to each of these artifacts, using the UML software profile. These customized components, both at hardware and software level, are available for the users in order to define the CPS applications. The library components and functionalities cover general application requirements and constraints. Starting from the requirements, it is possible to design applications for CPSs, using different types of components, nodes and corresponding connections.

For a complex definition of the design methodology, a middleware to recognize the types of components has been defined. The middleware is intended to ensure the correct functionality for the hardware and software components of the CPS network. The custom hardware and software requirements are established by

the user, while the middleware correlated with library functions, customized according to the application's goals and requirements, is able to handle them.

The developers of the predefined components for the CPS applications also ensure the verification methodologies for the applications defined by the users. They provide the tools to check the OCL constraints defined in the UML models. Once the UML models are validated according to the used constraints, a code generator ensures the translation of the UML models into simulation models. After the application is validated in a simulation environment, the visual programming model is able to generate executable code, which can be deployed on the physical network.

The goal-oriented method proposed by the research team is based on MDA approach in the design of CPS applications. The steps which need to be accomplished by the developers of predefined CPS components and by the users of such components that define CPS applications can be mapped into the models identified in the MDA approach. MDA is a rigorous approach, suitable for creating specific models, in each development phase, for embedded applications, in general, and CPS applications, in particular [21].

Each of the models defined in the MDA approach has certain particularities and a well-defined role. In general, the CIM indicates a certain system in the environment where it will evolve, without going into details about the specific application implementation. In the MDA approach the research team has defined, the CIM holds the defined UML profiles, the stereotypes, tagged values and constraints. It is used to describe families of CPS applications. At hardware level, the CIM defines the types of components, the types of nodes, the types of networks connections. At software level, it defines the network variables, the internal node strategies, the network boundaries and requirement abstractions. The profiles defined by Magureanu et al. in [16], [17], [18] and [19] form the CIM in the proposed MDA approach.

The PIM is defined by the user of the MDA methodology using the UML stereotypes grouped in the CIM and the applications requirements. The UML models which form the PIM indicate the network topology, the expression of the functional and non-functional requirements, the types of nodes in the CPS network and the associated behavior. The PIM objective is to bring together the desired hardware and software functionalities, while handling them at an abstract level. The goal is to not take into consideration the hardware limitations. These will be handled at the PSM level, depending on the deployment environment. The PIM is composed of a hierarchy of deployment diagrams, corresponding to the hardware artifacts, and one of component diagrams, corresponding to the software artifacts. The diagrams are constructed starting from the network interconnections, going until each node and component is fully described, with respect to the applications requirements and the logical levels defined. In this phase, the stub application can be compiled in the design and programming level, at a static level, for detecting the lack of correlations between the applications goals and network topology or between goals themselves.

The static aspects in the PIM can be tested and validated in a rigorous manner, as presented in the next chapter. Also, it can be tested and validated through simulation models. This is a recommended verification methodology, before deploying the PIM into a PSM.

The construction of a PSM, especially a deployment on a physical network, without a proper verification of the PIM, is not recommended. The errors discovered directly on the physical network lead to increased costs and can also slow down the application. Repeatedly loading the application on the physical network, after the

errors discovered in a certain step are fixed, is more time consuming than properly testing the application at PIM level.

Designing CPSs for physical networks involves significant challenges, as several limitations can appear. At the same time, testing through simulation must be as realistic as possible for obtaining a high degree of confidence in the validation results, before deployment in a physical environment.

CPSs can consist of distributed devices, each of them being characterized by its own internal clock and being able to operate at a different clock frequency than other devices. Although CPS components come with clear specifications, the main issue in designing this type of systems for different applications resides in the management of complex interconnections. These interconnections contain dynamic aspects, which increases the difficulty of a complete design. Proposed techniques for concurrent programming of massively distributed embedded systems raise the same issue [22]. A recent solution for handling dynamic aspects of CPS is based on PSoC devices [23].

PSoC devices integrate reconfigurable analog and digital circuits, all of them managed by an embedded microcontroller. PSoC devices provide memory and programming circuits. This kind of architecture provides great flexibility, with a reduced number of components. A single PSoC device can integrate a large number of peripherals, saving in this way board space and designing spent time. Also, such a device provides low power consumption and reduces the overall cost for the system. The reconfiguration capabilities allow to the designer of the system to connect internal resources on the fly. This leads to using a small number of components for each specific task [24]. PSoC devices support a wide range of communication protocols.

Using PSoC technology for designing CPSs applications is helpful but does not solve all the problems. Research still needs to be made regarding a correct temporal semantic for all concurrent processes involved [25]. Synchronization mechanisms must be implemented at communication level, in order to achieve cooperation between devices. Some solutions regarding issues related to simulation of PSoC based CPSs are given in [26], [27], [28] and [29] and are detailed in Chapter 3.

B. Goal-Oriented Approach in CPS Applications

CPSs integrate computational and physical processes, are characterized by command, communication and control capabilities and, at logical level, are tailored into several subsystems. Each of these subsystems has assigned a specific task or objective and acts in the overall system independent and unaware of the existence of the other subsystems. This behavior determines one of the main issues in designing CPS applications. While the subsystems pose different objectives over the controlled devices, in certain cases even contradictory ones, all these must be accomplished.

The goal-oriented approach proposed by Wang et al in [7] and continued by Magureanu et al. in [30] handles the complexity of distributed applications in general, and of CPS applications in particular, by expressing the goals and constraints in a purely declarative way. Strategies define the algorithms required for processing the application goals. These strategies are grouped in specialized,

predefined libraries. The basic aspects of strategies remain the same, independent of the applications requirements, while the goals and constraints vary dynamically. The descriptions for the goals do not include the interaction mechanisms between distributed entities and the best interaction scheme is inferred on the fly, at application run. Therefore, the logical tailoring proposed in the goal-oriented approach in the research team is dynamically configurable.

The main entities in the goal-oriented approach were introduced in [7] and are called Decision Modules (DMs). For each logical layer, in each subsystem of the application, one dedicated DM node holds the defined application's logic. The DMs are composed of four main elements: inputs, outputs, goals and constraints. The physical data gathered from sensors in the application constitutes the inputs. These are a set of attributes, expressed in the form of equations and constraints (invariants), which are designed using Linear Programming (LP). Specific examples of using LP systems are presented in Chapter 6. The physical outputs generated by the application, when solving the inputs for the system and to which constraints are applied, form the DM outputs. The DM constrains are defined by the physical capability of the used platform (embedded nodes and communication infrastructure are included) and requirements of the application (such as timing constraints or precision). The goals are mathematical expressions which involve inputs and outputs and on which maximization or minimization functions must be applied.

The communication, command and controls between logical levels are maintained using DM nodes. As the outputs of one DM are inputs of another DM from the upper, the lower or the same logical level, the application's DMs form a hierarchical structure. At each logical level, different execution semantics are more suitable to be employed. The lowest logical level, corresponding to the physical level, is handled by reactive models like Finite State Machines (FSMs). At higher logical levels, models with less flexibility but with a more predictable performance are used. The execution models allow performing the transition from the behavior at physical node level until the one at application level. Top-down and bottom-up constraint transformations model the interactions between DMs at different logical levels.

Figure 1 expresses graphically the connections between logical levels and the corresponding goals defined for each of these levels. The physical node is maintained at perimeter level.

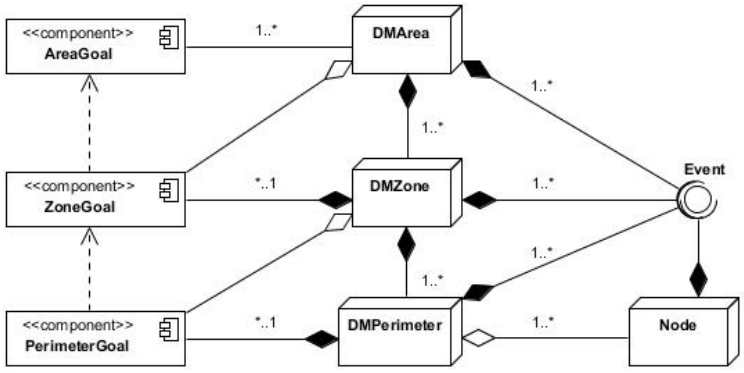

Figure 1 System DMs model [2]

A node can be contained in different subsystems at any given moment of time. This determines that, at each node level, several independent objectives need to be fulfilled.

In [20], Gavrilescu *et al.* have proposed a CPS network tailoring into several logical levels, depending on applications goals. This tailoring is applied to each of the subsystems that compose the CPS network. The goals logic is managed inside DM nodes. These nodes ensure communication and control inside the grouping they command and with the neighbor grouping, from the same, upper or lower logical level.

The boundaries for each subsystem are specified at the highest computational level. In goal-oriented terms, as defined in the research team, this represents the *Area* level. The applications goals at this level are described in a general accepted form. The goals at this level determined the ones at the lower logical levels, presented next. The business logic at *Area* level is kept in a *Decision Module Area* (DMA) node.

Zone is the next logical level and represents a subset boundary of the physical nodes located inside the *Area.* At this level, the goals are specialization over the internal managed devices of the inherited *Area* goals. At this level, the goals become more specific. The goals logic is hold in a *Decision Module Zone* (DMZ) node.

The computational level which corresponds to the physical level and brings together local logically coupled hardware devices is the *Perimeter* level. It contains sensors and actuators which act over the same environment "point". The goals at this level are clearly stated management aspects for the physical devices. The goals implementation implies well-defined execution blocks, which describe the co-working manner in particular cases. The business logic at *Area* level is kept in a *Decision Module Perimeter* (DMP) node where the most specific implementation for the objectives to be accomplished is provided.

In the defined goal-oriented methodology, the *Perimeters* are shared between the subsystems, as they correspond to the physical level. This implies that the goals of the subsystems are expressed as a goal union at *Perimeter* level, at any moment of time. The result of interpreting the goals at each logical level is represented by the set of commands which need to be achieved inside the grouping level. The result at each grouping, for each logical level, starting with the *Perimeter* level, obtained after executing the instructions, is made available to the upper logical layers. Therefore, in each subsystem, the goals will be adjusted.

In the case of *Zones*, depending on the application requirements, a *Zone* may be shared among subsystems. In such a case, the *Zone*'s goals are unified similar to the case of *Perimeters*. In this situation the managed *Perimeters* implied in fulfilling the goal are required to send the result only to this *Zone*. The *Zone* is part of all the cyber-subsystems, therefore it constructs feedbacks to all the *Areas* that contain the discussed *Zone*, to update their main objectives.

The current goal at subsystem level can be changed using two different approaches, the starting point being either at *Area* or at *Perimeter* level. In the first case, a DMA of a particular subsystem that monitors a set of sensors is involved, while in the second case the DM node which is responsible with monitoring a particular part of the environment triggers the modification.

For the first approach, when the goal change is triggered at *Area* level, all involved *Zones* must be identified. The next step is translating the new *Area* level goal into new specific goals for every DMZ. A similar procedure is used to translate new goals at *Perimeter* levels, for the *Perimeters* involved. In case inside a logical

grouping, the goal fails to be accomplished, the DM node corresponding to that grouping notifies the upper logical level. In such a case, the upper level assigns a new goal to the lower level, if possible, or cancels all the other goals already assigned to the other DMs in the same time with the one that failed to be fulfilled.

Regarding the second approach, each change in the sensed environment causes the involved node to trigger an event. The reaction of the upper grouping level can be to process the event without any other actions or to initiate a further notification to the higher level. In case the upper logical level reacts by consuming the received event without changing its internal goal, no feedback will be sent to the triggering DM node.

In the situation an upper level modifies the current goal, the event handling part impacts only the environment part managed by it. Modifications of goals at upper logical levels trigger modifications in the lower logical level goals.

C. UML Profiles for CPS Applications

High level-modeling of PSoC based CPS applications, which includes static and behavioral descriptions of distributed applications, can be achieved using UML models. The UML models can be defined for both hardware and software elements in embedded systems, in general, and CPS network, in particular. For the hardware specification, which starts from the network topology and becomes more specific, until node and component level, UML deployment diagrams are used. In case of software specification, which assigns behavior to each hardware defined part, UML component diagrams are of help.

To be able to customize the elements in each of these types of diagrams, according to different requirements for nodes and networks in CPS applications, Magureanu has defined in [16], [17], [18] and [19] two UML profiles.

C-1. UML Profile for Hardware Specification

The UML hardware profile contains stereotypes, tagged values and OCL constrains which are used when customizing hardware components, which instances are used in modeling the CPS application, at PIM level. Well defined stereotypes for specifying hardware components help achieving a clear separation and grouping between families of devices [16].

UML profile appliance is similar with the inheritance relationship, however there are some particularities when discussing UML profiles. The instance-of relationship is not transitive [31]. When a stereotype is applied to a certain element, the tagged values defined by the stereotype are applicable only to that element and not to instances of that element.

As mentioned earlier, UML deployment diagrams are used in modeling hardware aspects in distributed applications. Instance specifications elements are used in deployment diagrams and they can be customized using stereotypes that extend metaclass *Instance Specification.* The nodes are customized using stereotypes that extend metaclass *Node (Deployment)*. The relation between the *Node (Deployment)* stereotype applied to a node and *Instance Specification*

stereotype applied to an instance of that node is given by an OCL constraint. This constraint states that all instances must be customized using the corresponding *Instance Specification* stereotype, to which the naming convention applies. The *Instance Specification* stereotype will have *_Instance* suffix.

The stereotypes are grouped in several hierarchies, depending on their usability. The first level hierarchy of stereotypes which extend the *Node (Deployment)* metaclass is described visually in Figure 2.

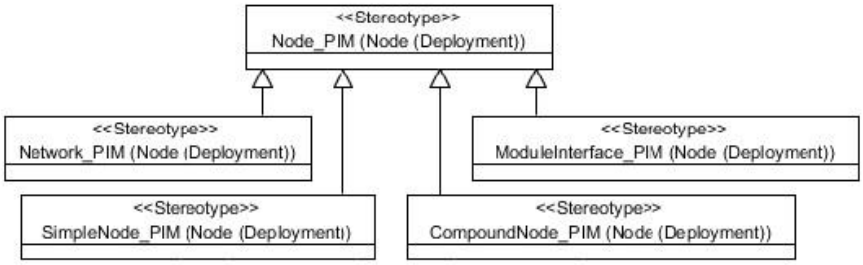

Figure 2 First level Node (Deployment) stereotypes [2]

The stereotypes contained in the first hierarchy level are as follows:

- *Node_PIM* is a stereotype which extends *Node (Deployment)* metaclass. It is an abstract stereotype and is the base for all stereotypes of *Node (Deployment)* in the UML hardware profile.
- *Network_PIM* stereotype represents a customization for each network, as main part of an application. The networks are either wired or wireless, depending on the type of communications between nodes, as components of the network.
- *SimpleNode_PIM* stereotype is used when customizing a unit in a network which contains only attributes and ports. This stereotype is used for grouping purposes, while more specialized stereotypes are used for customizing simple units in a network.
- *CompoundNode_PIM* stereotype is used for customizing a node in a network which also contains other units (except for attributes and ports), which can be simple or compound units.
- *ModuleInterface_PIM* stereotype is used in case compound nodes contain sub-modules that implement a certain interface, instead of being an instance of a certain simple or compound unit. This stereotype is the base for a hierarchy of possible module interfaces in the UML hardware profile.
- *Cypress_PredefinedUnit_HWST*, *BaseUnit_HWST*, *MIXIM_PredefinedUnit_HWST* are first level abstract stereotypes for simple units.
- *Bus_HWST* stereotype customizes a hardware component which acts as a regular data bus and eases communication between components in a distributed wired network.
- *PSOCUnit_HWST* stereotype has as base stereotype the *SimpleNode_PIM* stereotype. It defines the PSoC device contained by the hardware node.

Other categories of stereotypes defined in the UML profile for hardware specification are as follows: stereotypes for PSoC based CPSs, stereotypes for

wireless communications (developed using as starting point MiXiM project [32]), stereotypes defined for interfaces and stereotypes defined for compound modules.

OCL constraints are defined for the overall UML hardware profile, but can be applied to specific stereotypes. Constraints are identified in the UML profile using the name of the stereotypes with an indexed suffix.

C-2. UML Profile for Software Specification

The UML software profile presented below allows customization of the application behavior in accordance to the UML hardware profile, at network, node and unit level ([18] and [19]).

UML component diagrams are used to express the software requirements for distributed applications. A software component model is defined to correspond to each designed hardware deployment model.

The defined software stereotypes are grouped into several sets and have as base stereotype the *Component_PIM* stereotype, as shown in Figure 3. The presented groups of stereotypes provide a solution for specifying and modeling the software strategies defined for different types of units in CPS nodes and which are available in an external library.

For every stereotype presented above it is considered that at least one basic software implementation already exists in the external library. The PIM-PSM model identifies, binds and integrates all the customized software strategies and provides a customized CPS application, in accordance to users' requirements.

The first group is inherited from *BaseUnit_SWST* stereotypes and defines the software stereotypes for the units corresponding to the hardware already defined in the deployment diagrams. This group represents the first phase in customizing the business logic for each hardware unit.

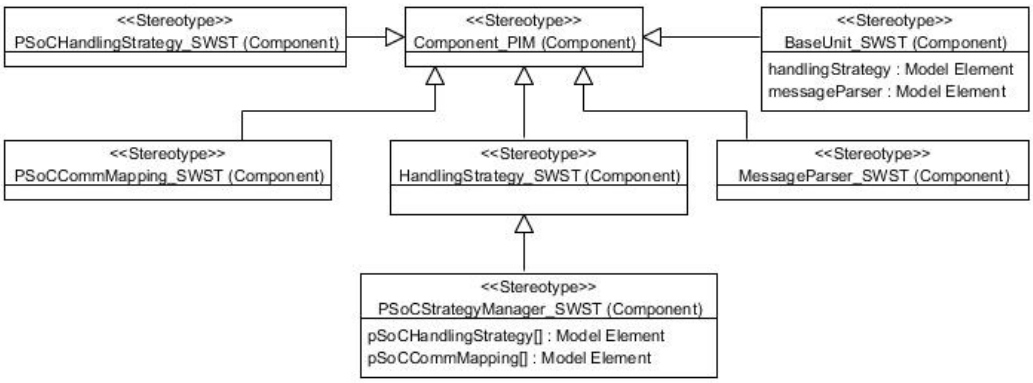

Figure 3 First level inheritance of the stereotypes composing the defined software profile [2]

The hierarchical group derived from the *Message_Parser_SWST* stereotype is intended for the communication part in an application.

The stereotypes inherited from *Handling_Strategy_SWST* stereotypes deal with the application business logic which has to be attached to the hardware unit.

The *PSoCStrategyManager_SWST* type stereotypes are intended to maintain the application software logic related to hardware PSoC units involved in the CPS nodes construction. They can be seen as software containers and have the constraint to host software artifacts inherited from the *PSoCHandlingStrategy_SWST* and *PSoCCommMapping_SWST* stereotypes, respectively.

The hierarchical group which has as base the *PSoCHandlingStrategy_SWST* stereotype holds the actual internal PSoC related business logic and offers the user the opportunity of customizing it.

The inheritance from *PSoCCommMapping_SWST* stereotype manages the message mapping and exchange between different hardware units managed by PSoC. This stereotype hierarchy offers the possibility of defining the state machine of the PSoC unit by taking into account all the possible actions the PSoC software application may take based on specified internal and external event mappings.

Other groups of stereotypes defined in the UML software profile contain stereotypes for software part definitions, stereotypes for message handling, stereotypes for customizing the strategy handling inside components, stereotypes specific for PSoC handling strategy and stereotypes related to communication.

Similar to OCL constraints defined for the UML hardware profile, the OCL constraints defined for the entire UML software profile can be applied to specific stereotypes. Constraints are identified in the UML software profile using the name of the stereotypes with an indexed suffix.

2.1.3 Summary

Considering the great number of applications which use nowadays CPS networks, an efficient, intuitive and easy to use design and programming model is desired for CPS applications. It is desired that even users without advanced knowledge about sensor networks design methodologies and low level programming skills to be able to specify applications for CPSs, in a high level of abstraction. The users can command, control and program CPS applications. At hardware level, they specify the network topology, the nodes, components and connections. At software level, they specify the goals, without taking into consideration the physical limitations of the environment.

The users can customize the CPS applications according to the requirements, by using predefined artifacts, grouped into two UML profiles, for hardware and software specification of CPS applications, respectively.

The novelty presented by this approach allows raising the level of abstraction at which applications must be handled by the users. This can be accomplished at first by tailoring the CPS network at logical level and then by specifying that application objectives only at the highest logical level. The application middleware is responsible in translating the goals from higher logical levels to lower logical levels, until their accomplishment at the physical level.

The methodology presented in this chapter uses MDA approach for CPS applications design and is described in more detail in [20]. The MDA characteristic models can be easily identified. The CIM groups the two UML profiles. These are used to customize UML deployment and component diagrams, corresponding to the network elements. The PIM static aspects are validated using rigorous specification, while for the low level and dynamic aspects, simulation models are recommended.

The code resulted once the PIM is fully tested and validated is the final scope of using MDA approach, as it reduces the total amount of code to be created when programming a CPS application.

2.2 Simulation Frameworks for CPS Applications

In the MDA approach, simulation environments play an important role as platforms for developing PSMs. The PSMs expressed using UML artifacts which describe applications for CPSs must be tested and eventually validated. This is possible for different platforms. An immediate approach would be deploying on a physical network. Along with the advantages of testing the application in the environment it will finally evolve in, there are also the disadvantages of directly deploying the network in the physical environment, without previous tests. This deployment is based exclusively on the code generated from UML models and is proved to be more costly and time consuming than using a method for testing the applications before deployment.

Testing applications for CPSs and tracking the execution steps after deployment is a difficult task because of the great number of messages exchanged between hardware devices. It is not very feasible to maintain and handle a detailed logging in case of distributed devices. PSoC devices usually have a reduced hardware display and offer us little information during testing. Additional hardware debugging devices are necessary for detecting errors at runtime. These devices are an expensive solution.

Simulation for applications for CPSs, for a proper testing and validation, is a more suitable approach and a less expensive one than directly deployment on the physical network. This is where simulation environments, their characteristics and the models developed using specific simulators intervene.

The required time for validating an application and also the material effort is reduced in case of simulation than in case of directly deploying the network on physical environment. Loading the application on physical devices, correcting some errors and then repeating the entire process is more time consuming than running simulation software. Simulation before deployment increases the robustness of the application, as it allows the user to detect bottlenecks before deploying the network. A validated simulation process ensures a deterministic behavior for each hardware node and for the entire network. Simulation before deployment also allows the evaluation of the performances of the application, using relevant simulation cases over CPS networks, being able to provide results even for large sized distributed networks.

The concepts on the importance of testing and validation using simulation can also be found in [26], [27], [28], [29] and [33]. These have constituted the basis for developing the simulation models detailed in Chapter 3.

2.2.1 Simulation Environments for Sensor Networks – Overview

The simulation environments take into consideration, into the simulation models they propose, several aspects regarding simulation of sensor networks and, in particular, distributed networks. Some of the most known approaches will be detailed next, underlining their characteristics, both the strong points and the drawbacks. These characteristics, especially the weak points, are taken into consideration when deciding for a suitable simulation environment for testing and validating the MDA approach in designing applications of CPSs [33].

ATEMU [34] is a fine grained sensor network simulator, intended to bridge the gap between sensor network simulations and sensor network deployments. The authors adopt a hybrid strategy: the operation of individual sensor nodes is emulated in an instruction by instruction manner, while the interactions between nodes using wireless transmissions are simulated in a realistic manner. ATEMU has the ability to perform extremely low-level emulation of the hardware for sensor nodes. ATEMU uses a cycle-by-cycle implementation strategy for each node and device, as part of the network. Loading and initialization the network, for each component in the network is followed by modifications of the clock. The clock is increased by one clock cycle every round to ensure that nodes, the internal devices and the radio communication are accurately synchronized. Synchronization is a very important issue as a great part of the microcontroller program's execution is dependent on the timing and behavior of the component devices. A strong point in favor of using ATEMU is the fact that this tool has the ability to simulate a heterogeneous sensor network. A major drawback for ATEMU simulator is the fact that it cannot simulate the synchronization of two devices running at different clock frequencies. This is exactly the case met when using CPSs composed on PSoC devices. The accuracy of simulation and the emulation possibilities offered by ATEMU ensure that in case on using actual hardware, the software will already have undergone complex testing and debugging on an accurate platform. However, this characteristic can only be used in case of devices running at the same clock frequency.

TOSSIM [35] is presented as a simulator for TinyOS [36] wireless sensor networks. Therefore, the primary goal stated for this simulator is to provide a high fidelity simulation of TinyOS applications. The simulator is focused on simulating TinyOS and its execution, rather than simulating the actual physical world. A strong point in favor of TOSSIM is that it can capture network fidelity in case of large size networks, scaling to thousands of nodes, and a wide range of network interactions. TOSSIM has proven his functionality when the authors discovered several bugs in the older and more widely used TinyOS. These discovered bugs also come in favor of simulation before deployment in the physical environment for sensor networks. TOSSIM can be used to understand the causes of different behaviors in the real world, but it does not capture the entire behavior, therefore is not suitable for absolute evaluations. As a comparison, ATEMU is 30 times slower than TOSSIM.

Avrora [37] a cycle-accurate instruction level sensor network simulator which scales to networks of up to 10,000 nodes and performs as much as 20 times faster than previous simulators with equivalent accuracy, handling as many as 25

nodes in real-time. This simulator is intended for the AVR microcontroller and for nodes of sensor networks built on the AVR microcontroller. The authors show how an event queue can enable efficient instruction-level simulation of microcontroller programs and allow the hidden parallelism in fine grained sensor network simulations to be extracted. Avrora scales better than ATEMU and approaches the performance of TOSSIM simulator and preserves cycle accuracy. ATEMU and TOSSIM are implemented in C programming language. Avrora uses Java as implementation programming language, which helps improving flexibility and portability. ATEMU and Avrora have strong points in favor of using them as simulators for the distributed networks in question. These simulators have language and operating system independence by simulating machine code, unlike TOSSIM which can simulate only TinyOS programs. As a weak point for Avrora, its simulation possibilities are limited to simulating a network of nodes, running the actual microcontroller programs and running accurate simulations of the devices and radio communication. This simulator cannot be used for running software simulations.

In search for the simulator environment suitable for testing and validating the high level UML models created MDA approach, there have been considered aspects of friendly interfaces and intuitive usage for the tools. The simulation environments presented above do not provide such a rich user interface which would be helpful to the user in defining the simulation model.

Ns-2 [38] is a discrete event simulator, well-known in academic research. This simulator has facilities for describing network topology, network protocols, routing algorithms and generation of communication traffic. It also allows the incorporation of protocols (TCP), routing algorithms and traffic generations defined by users, over wired and wireless networks. However, Ns-2 has some major disadvantages that make him unsuitable for testing and validating through simulation the applications for CPS networks. Ns-2 is intended to be an object-oriented simulator. This characteristic is sometimes a drawback as it introduces unnecessary dependencies between modules. These interdependencies make the incorporation of new protocol models very difficult.

Other simulator environments, such as C++SIM [39], JavaSim [40] and GloMoSim [41] made attempts in addressing the problems raised by Ns-2 and left unsolved. JavaSim and C++SIM are similar tools, both providing discrete event process-based simulation. C++SIM is a programming tool for simulation of discrete processes. It is an extension of C language obtained by including SIMULA-like possibilities using C macros and functions. The user writes the program in the C language and calls the library functions and macros. With JavaSim, the researchers have tried to build a component-based architecture. The penalty in case of using Java as the implementation programming language was the increase for simulation time.

GloMoSim has been intended for extensibility and composability. The researchers were interested in parallel simulation of wireless networks. GloMoSim was designed as a set of library modules, each simulating a specific wireless communication protocol from the protocol stack. This simulation tool was implemented on computers that had both shared and distributed memory and can consider different protocols for the simulation models. A strong point in favor of GloMoSim is the visually configuration of the network, and intuitively selecting the synchronization algorithms. This simulation environment is at most comparable with Ns-2 in terms of design and extensibility.

TSync [42] is a lightweight bidirectional time synchronization service for wireless sensor networks. TSync's service offers a push mechanism for accurate and

low overhead global time synchronization and a pull mechanism for on-demand synchronization by individual sensor nodes. Both mechanisms can be flexibly parameterized in order to meet the time synchronization terms for specific applications. The performance for TSync is improved by multi-channel enhancements. The communication protocol takes into consideration the message propagation delay and the local clock offset. This service can be adapted to wireless sensor networks that do not support multiple channels. The drawback is given by the decrease of accuracy due to increased packed collision.

Sinalgo [43] is a simulation framework for testing and validating network algorithms, which focuses on the verification of network algorithms and abstracts the underlying layers. The prototypes for the network are described in Java, which implies an increased speed compared to using hardware specific languages. Sinalgo was also intended for easy extensibility, for which it offers a set of extension point. Sinalgo network simulator was used in testing a completely distributed time synchronization protocol. This protocol was able to take into consideration the clock screw and the clock offset. Simulations were made for obtaining a correct view on how the algorithm performs on large scale networks. Such large scale networks are more difficult to implement physical, therefore simulation is the first required step. The results of this study were detailed in paper [44]. For simulating sensor nodes, the hardware clock of node was modeled in software. The simulation tests were performed using random clock drift values for nodes, at network start-up. It was considered that all nodes start at the same time. Such a situation is an unrealistic assumption for physical network composed of different devices, where delays at start-up are on regular basis. The simulation tests were run to indicate the neighbor simulation error, in absence of the details for the simulation model. The studies conducted to the conclusion that the synchronization errors between neighbors are increasing with the size of the network.

SystemC [45] is a commonly used solution for simulating applications for embedded systems. SystemC is a system level modeling language which allows simulation at different levels of abstraction. Several approaches involving SystemC at simulation level were already presented in [46], [47], [48] and [49]. However, SystemC presents also some disadvantages related to embedded distributed systems. Paper [50] presents some of them. SystemC was designed for single host simulation. All threads are created in the same simulation process assigned by the operating system. This means that the created simulation threads run concurrently with the other threads in the system, such as operating system threads or threads for other applications. The priority of the thread created for simulation determines synchronization and communication between modules. Therefore synchronization and communication depend directly in the processor's speed. This implies a major drawback for using SystemC in simulation distributed embedded systems: the duration of the simulation cannot be estimated. At most, an improvement can be made regarding the duration of the simulation. This can be eventually controlled by increasing the priority of the simulation thread but this implies another programming effort. Some speed-up solutions were proposed, like multicore machines and ones using geographically distributed systems [50], [51]. Although the improvement proposed works well, the lack of performance in case of simulated distributed applications on single thread simulation kernel is more obvious in case simulated applications become more complex. Another drawback of SystemC, as presented in [52], is that it does not offer support for upper layers, including network communication. The major drawback for SystemC that imposed the search

for another simulation environment was the lack of support offered by SystemC for simulating distributed applications.

Cypress Semiconductor Corporation products [53] are a solution when considering simulation environments, because of the variety of devices offered. Cypress is a leading high-tech company which designs and produces also PSoC devices. The interfacing between functionality of PSoC devices and user-deployed applications is maintained through a component based middleware layer. This interfacing is based on several Application Programming Interfaces (APIs). These APIs offer control for different internal PSoC hardware components like: clock units, power supply management, interrupts, cache memories, pins for connection of external hardware modules to the device, internal access registers, flash an EEPROM memories management, watchdog timers, system boot loader and system startup.

PSoC Creator IDE is intended to offer a fully functional hardware-software co-design development environment for the PSoC devices in case of embedded systems applications. This IDE gives the users the possibility to construct more complex devices by connecting to the main PSoC device a wide range of analog and digital configurable external components. A main feature for PSoC Creator is that it offers support for loading the users' applications on the physical PSoC device. Components binding and mapping from simulation to physical node is also provided. This IDE covers instruction level and clock accurate simulation for the defined node. PSoC Creator provides also an intuitive interface.

PSoC Creator provides accurate node level simulation in case of PSoC nodes, even more complex ones, composed of multiple analog and digital components. However, this is not the case for networks composed of distributed devices. PSoC Creator does not offer simulation features for applications using distributed embedded systems, at network level.

A complex and useful survey regarding the mechanisms required for clock synchronization is presented in [54]. The authors perform a rich classification of clock synchronization mechanisms. Each category has associated a representative communication protocol. However, the authors cannot propose a simulation platform able to support simulation for all existing clock synchronization protocols. Simulation models that took into consideration the clock phase shift between nodes were mostly designed for simulating a particular communication protocol rather than a general proposed model.

2.2.2 OMNeT++ Simulation Environment

The author of this thesis has chosen OMNeT++ as a simulation environment and the reasons that led to this decision are detailed next. OMNeT++ [55] is an object-oriented modular discrete event network simulation framework, which allows network simulation on a large scale. At the same time, the OMNeT++ simulator overcomes some of the issues that make SystemC unsuitable for simulating distributed embedded applications. OMNeT++ allows embedding simulations into larger applications and offers support for parallel simulation, by using the characteristics of modularity and extensibility. A strong point in favor of OMNeT++ it the fact that it handles each event in sequence and maintains its own virtual clock, independent of the processor's clock [56]. This was not the case with SystemC. Simulation time results are obtained in terms of virtual OMNeT++ clock, at

successive runs. The virtual system clock is updated only at the end of all tasks associated to the events to be handled at the current system time. This property allows the elimination of the problems related to real-time synchronization constraints, for complex systems simulations.

OMNeT++ also facilitates the development for models of the applications and analysis of the obtained results. Visualizing and debugging the simulation models is helpful in both specifying the applications and also reducing debugging time. Communication between modules of the application is made through messages. OMNeT++ provides a library for multithreading applications and also provides support for scheduling, sending and receiving messages.

Studies have shown that simulations performed using OMNeT++ are executed at least an order of magnitude faster than the ones performed using Ns-2 [57]. At the same time, OMNeT++ makes more efficient use of the available memory.

Unlike SystemC, OMNeT++ does not offer support for instruction level simulation. There have been made some attempts to unify the benefits of SystemC and OMNeT++ simulators [52]. However, a conclusion that can be drawn is that developers must choose the simulation environment according to the specifications and behavior for the applications in question. Considering applications of CPSs constructed on PSoC devices, which need to be simulated before deployment, OMNeT++ simulator is a more suited solution.

As a characteristic of OMNeT++, this environment allows separation of simulation into two major aspects: model topology and model behavior. Network Description (NED) files are used to illustrate the network topology and the internal configuration of the components for the different types of nodes in the network. C++ files capture the behavior for the network and for the components in the network.

Table 2 presents some similarities between the NED language and the SystemC language. The hardware semantics presented indicate that both languages are appropriate in specifying the components and the interconnections at network level. OMNeT++ ensures an intuitive visualization of the simulated models of applications and also of the interconnections between nodes in the simulated network.

Type of unit	**OMNeT++ NED**	**SystemC**
Module	simple, module, network	sc_module
Gate	in, out, inout	sc_in, sc_out, sc_inout
Signal	signal	sc_signal

Table 2 Semantic correspondence between OMNeT++ NED language and SystemC [16]

The NED language [55] is used in OMNeT++ simulation environment to describe the network topology and the hardware configuration for the components in the network. Several features of the NED language make it suitable for describing massively distributed applications. The NED structures are hierarchically built: complex modules can be composed of simple modules, therefore acting as compound modules. The NED language is component-based: simple and compound modules are reusable which allows component libraries. Modules and channels can be sub-classed in NED files. Derived modules and channels can add parameters and gates, in case of simple components. In case of compound components, derived modules and channels can add sub-modules and connections. The NED language

has defined a Java-like package structure. Channels and modules can be defined as inner types, when used locally, to reduce namespace pollution. Module of channel types, parameters, gates and sub-modules can be enriched by metadata annotations.

Channels are similar to modules. At the same time, predefined channels cover a wide range of the specifications for the requirements of the applications. Channels encapsulate behavior and parameters which are associated with the connections.

Simple and compound modules contain parameters and gates. Compound modules also contain sub-modules and connections between the sub-modules. Parameters can be used on hardware specification for the network topology and for the components of the network. Also, parameters can be used on software specification, to supply input for the C++ code that implements the behavior for modules and channels.

The connection points between modules are the gates. NED language defines three types of gates: input, output and input-output gates. Each module contains specifications for connecting the sub-modules between them and with the container module.

The characteristics of OMNeT++ environment made it a suitable choice for testing and validation of the goal-oriented model based on MDA.

2.2.3 Simulation Frameworks Based on OMNeT++

OMNeT++ can be used in simulating different types of network, in a generic manner. For solving specific problems, the researchers have created several simulation frameworks based on OMNeT++.

Mobility Framework [58] is a framework for OMNeT++ able to support simulations of mobile networks. This framework has three main components. It provides a set of independent modules that implement node mobility, dynamic connection management and a wireless channel model. It includes a library of standard modules for common protocols. It suggests design guidelines for simulation models of wireless devices. However, more sophisticated mobility architecture modules are required in order to reflect real geographical/topological settings and also more complex channel models.

Another particular implementation using OMNeT++ is Castalia framework [59]. This framework was intended for simulations of wireless sensor networks. The Castalia framework contains specific mechanisms such as wireless channel modeling, used for the estimation of the average path loss between two nodes in the network. Characteristic for Castalia is the time variability regarding radio transmission. Realistic channel modeling is not met in case of large size networks; however, mobility can be handled efficiently, depending on the cell size.

OverSim [60] is a flexible overlay network simulation framework based on OMNeT++. OverSim facilitates the implementation of new overlay protocols. OverSim includes several structured and unstructured peer-to-peer protocols and provides several functions such as overlay message handling, visualization and a generic lookup mechanism. The implementations for these protocols can be used for both simulation and real networks. With OverSim, applications using networks with

100,000 nodes can be simulated in a reasonable amount of time. OverSim also provides an efficient event scheduler and strong interface support.

MiXiM [32] is an OMNeT++ based modeling framework created for mobile and fixed wireless networks. MiXiM provides detailed models of radio wave propagation, interference estimation, radio transceiver power consumption and wireless MAC protocols. MiXiM project is suitable as communication model for wireless sensor networks. In MiXiM, a network contains parameters and sub-modules. The sub-modules are described in separate NED files. The parameters for the entire network are grouped in an *omnetpp.ini* file, which contains also the parameters for the simple and compound modules. This method allows a global view for all parameters for a network and contained nodes, customized by the user.

A base network contains as parameters information about the coordinates of the area the nodes are in (expressed in meters) and the total number of hosts in the network. The network contains as sub-module a *ConnectionManager* which is a central module that coordinates the connections between all nodes and handles dynamic gate creation. *ConnectionManager* periodically communicates with the mobility module and *ChannelAccess*. The network contains as sub-module a *BaseWorldUtility* module which provides utility methods and information used by the entire network as well as simulation wide black board functionality.

A *BaseNode* is a compound module which contains parameters, gates, sub-modules and connections. The basic node contains a *BaseUtility* module which is a basic utility module for a host. It mainly provides blackboard like information exchange between the other modules of a host. The basic node implements *IBaseMobility*, which is an interface for mobility modules, *IBaseApplLayer*, which is an interface for application layer modules, *IBaseNetwLayer*, which is an interface for network layer modules. The basic node contains a *BaseNic* module which implements a CSMA network interface card using the *CSMAMacLayer* module.

The node describes the connections between components: the network interface cards (NIC) layer component, the network layer component, the application layer component and the exterior of the node. *BaseNic* is the network interface card module, which contains gates, sub-modules and connections. *BaseNic* module contains an instance of a simple module which implements the CSMA MAC protocol and an instance of a simple module, *PhyLayer*, which knows how to initiate the *Deciders* and *Analogue Models* from the modules directory. The information presented here is only a short description of the MiXiM modules and facilities. More details about MiXiM are available online.

The INET framework [61] is an open-source communication networks simulation package for the OMNeT++ simulation environment. This framework contains models for several wired and wireless networking protocols.

While MiXiM is focused on providing very detailed models of wireless network interface cards, it does not provide very sophisticated upper layers like network layers or application layers. These upper layers are exactly the ones addressed by the INET framework. These characteristics led to developing Mixnet [62], which means combining the strong points of MiXiM and INET into using the wireless NICs from MiXiM together with the upper network stack from INET.

2.2.4 Summary

The table presents briefly the strong and weak points identified for the studied simulation environments.

Project name	Year of first publication	Strong points	Weak points
Ns-2	1989	It is a well-known simulator, widely used in academic research.	It is an object-oriented designed simulator, which introduces sometimes unnecessary interdependency between modules. The extension with new protocol models is very difficult.
C++Sim	1990	It is a discrete event process-based simulation similar to SIMULA's simulation class and libraries.	
JavaSim	1997	It tries to build a component-oriented architecture.	The penalty paid for using Java implementation language is the increase of simulation time.
GloMoSim	1998	It is focused on parallel simulation.	It is not superior to Ns-2 in terms of design and extensibility.
OMNeT++	1999	It provides support for sensing, scheduling and receiving messages. It handles each event in sequence and maintains its own virtual system clock. It ensures accurate traceability and focusing on the behavior of each device. It provides a library for multithreading applications.	It does not offer support for instruction level simulation, as SystemC does.
SystemC	1999	It is a system level modeling language. It allows simulation at different levels of abstraction. It is widely used for simulating embedded systems	The threads are created in the same simulations process assigned by the operating system. Synchronization and communication between modules depends on the thread's priority and depends

		applications.	directly on the processor's speed. This implies that the duration of the simulation cannot be estimated and controlled. On sole-thread simulation kernel the lack of performance is aggravated when the simulated applications gain complexity.
TOSSIM	2003	It provides a high fidelity simulation of TinyOS applications.	It does not capture the causes of behavior in the real world and should not be used for absolute simulations.
ATEMU	2004	The internal devices for nodes in the network and the radio communication are correctly synchronized.	It cannot simulate synchronization of two devices that operate at different clock frequencies.
INET	2004	It contains models for several wired and wireless networking protocols.	It does not provide support for complex communication layers.
TSync	2004	It is a lightweight bidirectional time synchronization service for wireless sensor networks. The communication protocol takes care about message propagation delay and the local clock offset.	The design requires mutual trust between all nodes and therefore is vulnerable to malicious nodes.
Avrora	2005	It simulates a network of nodes, runs the actual microcontroller programs, and runs accurate simulations of the devices and the radio communication.	It cannot be used for running simulation models of the software.
Cypress PSoC Creator	2005	The IDE is intended to offer a fully functional hardware-software co-design development environment for the PSoC devices in embedded system applications. It coves instruction level and clock accurate simulation of the defined node.	It does not offer simulation features at network level for the distributed embedded applications.
Castalia	2006	It contains dedicated mechanisms for simulation of wireless sensor networks.	The implementation of path loss is completely left as responsibility of the developer.
Mobility	2007	The core framework	The lack of more complex

Framework		implements the support for node mobility, dynamic connection management and a wireless channel model. It provides basic modules that can be derived in order to implement own modules.	mobility architecture modules to reflect topological settings and more complex channel modules.
OverSim	2007	It is based on OMNeT++. It provides a generic lookup mechanism for structured peer-to-peer networks and an RPC interface.	There is currently no support to import models of topology generators.
Sinalgo	2007	It is used for simulating a completely distributed time synchronization protocol, able to take care for the clock screw and clock offset.	Simulations were performed considering that all nodes start at the same time. This is an unrealistic assumption for physical network composed of different devices.
MiXiM	2008	It provides detailed models of wireless network interface cards.	It does not provide support for complex upper layers.
Mixnet	2010	It uses: MiXiM NICs together with INET's IP stack; INET application layers; MiXiM's mobility modules; INET's non-wireless NICs; both blackboard modules.	It cannot use INET's wireless related modules. Information cannot be exchanged between the blackboard modules of MiXiM and INET.

Table 3 Simulation environments for sensor networks

2.3 CPS Verification Methodologies

Modeling is a key aspect of a good design for both embedded and distributed systems. As CPSs can be seen as embedded distributed systems, various modeling techniques were proposed.

Object-oriented modeling is wide spread in software design and it could be successfully adapted to CPS structural design. The main issue in this case regards the informal nature of most object-oriented models. A solution to this problem was presented in details by D. B. Aredo et al. in [63] and by R. France et al. in [64]. According to them, there are three approaches to define object-oriented modeling notations: supplemental, object-oriented extended notations and methods integration. In supplemental approach, rigorous constructs replace parts of the model expressed in informal object-oriented notations. In the object-oriented extended notation, existing notations are extended by object-oriented mechanisms, therefore becoming more compatible with object-oriented notations. The advantage of using such approach is given by the fact that rigorous systems are obtained. The

methods integration approach is a more suitable solution that makes informal object-oriented modeling concepts more precise by integrating them with proper specification techniques. This approach is the most commonly used in the standardization of object-oriented modeling notations. Therefore it is appropriate for analysis tools development, since the first two concepts force the user to be in contact with a great amount of complex mathematical artifacts. In this case, the designers can directly manipulate graphical artifacts in the object-oriented models and they do not require strong rigorous background.

A deep analysis was presented in [65]. It provided a complex view on the verification tools and methods that can be used for UML based development systems. Additionally, the authors investigated the different levels of system design in which a certain method is more appropriate.

In OCL, the expressions can be undefined. The available OMG documentation ([66]) does not specify how the OCL checker tools should deal with the undefined queries from the expressions (Hamie *et al*. in [67]). In most of the cases, the OCL tools denote nothing in the model verification process.

2.3.1 Z Language

Significant research on specifying UML models proposes Z language for system specification. Z is a specification language based on standard mathematical notation [68]. An empirical study regarding the requirements specifications for an information system can be found in [69]. Both natural description in UML and the corresponding description using Z language are discussed for a particular system. The author has formulated correspondence rules between modeling elements in UML and Z language. He concludes that expressing the system requirements in both rigorous and informal way leads to building a robust documentation regarding what is expected from the system.

The Z language maintains a clear distinction between logical operators and expressions. The logical expressions are not treated as expressions within the language, therefore their truth values are unknown if they involve undefined expressions.

A hybrid solution by merging both advantages of OCL and Z languages was described by Roe et al. in [70]. It provided a mapping from an integrated model of UML model and OCL specifications into a Z specification. A similar mapping is used in Chapter 5 to translate the OCL constraints of the used stereotypes from the UML designed models.

Reference [71] presents a meta-modular framework created on two levels using both UML and Z language. At the meta-level, the modular semantic interpretation of UML diagrams is expressed using templates and generics. At the instantiation level, the UML models are translated into Z specifications by instantiating the semantic interpretations of the corresponding meta-level.

A complex approach was discussed in [72]. It was based on a tool called RoZ, which helped automatically transforming the UML class diagrams and specific annotations to Z specifications. The tool allows the generation of specifications for elementary operations on classes and the generation of proof obligations to validate operation guards. It allows automatic generation of Z schemas for the UML class,

object and deployment diagrams along with the elementary operations. The author of this thesis has used RoZ tool as support for developing specifications.

The research in this direction was continued and presented in three new references ([73], [74] and [75]). In [73], the authors were focus on rigorous and informal specifications and the role of the RoZ tool, while in [74] a method for validating UML models was introduced. To achieve this goal, they have proposed translation of the UML models into Z and Lustre ([76]) specifications. This approach allows the user of a theorem prover and a test generator to validate the initial models. The research was further developed in [75], where authors showed how annotated class diagrams can be adapted to support animation. To validate Z theorems, several tools were proposed in literature. The Z/EVES theorem prover, proposed by Saaltink in [77] is used in the described study.

In [78], it is proposed a specification of the primary UML constructs used to build class structures based on Z language. Here, a Z state schema represents the static aspect of a class. The attributes and object identifiers of class instances are represented by state variables. Class invariants are specified in the predicate part of a Z schema. Starting from the Z specification of the UML class constructs, the author of this thesis has developed similar rules for specifying the constructs of UML state machine and deployment diagrams.

Along with the classification of approaches for object-oriented modeling concepts, in [64] the authors have defined the abstract syntax of a subset of the UML static model notation. They have also defined an appropriate semantic domain for its components. A meaning function completes the previously defined syntax. The Z notation is then used for analyzing the UML models.

2.3.2 Prototype Verification System

As an alternative to Z language, many researchers have used PVS ([79]) in order to define and analyze UML models. PVS consists in a specification language, a type checker, a model checker and a theorem prover. The PVS language has the ability to directly support reasoning about a large number of traces. According to D. B. Aredo *et al.*, this is a strong reason to use PVS tools for specify of the UML class structure [63]. The authors have defined a general approach and then the specification for UML interfaces, classes, associations, generalizations and aggregations. The author of this thesis has extended the specification of UML class diagrams to other types of diagrams used in CPS applications design. These diagrams were translated in both Z and PVS specifications.

Another work that explores opportunity of using the PVS tools for checking the correctness and completeness of UML models is presented in [80]. Here, the author proposes semantic definition for UML statecharts using the PVS specification language. The author has developed a general framework for translating UML statecharts diagrams into PVS specifications. The advantage of using such translation resides in the capability to produce precise and analyzable specifications and to use the rigorous reasoning provided by the PVS tools.

In [81], the author presents semantics of UML sequence diagrams using PVS language as an underlying semantic foundation. The basic concepts of sequence diagrams must be specified at first: events, messages, objects and traces of events.

Next, the semantics of the sequence diagrams are modeled by a PVS theory that composes the constituents.

Papers [63], [80] and [81] are part of a larger project that has defined semantics of UML models in PVS, for UML classes, statecharts, sequence and state machines diagrams.

An interesting research is presented in [82]. The authors translate Java classes into high order, classical logic of PVS tools. PVS language is used as back-end to the Logic of Object-Oriented Programming (LOOP) tool. This tool provides a logical semantics for JAVA. Some elementary properties in JAVA programs can therefore be proved using PVS tools.

A prototype tool that analyzes the syntax and semantics of OCL constraints from an UML model was presented in [83]. Translating the elements of the UML model, including the OCL constraints, into PVS specification language follows this analysis. The prototype tool defines semantics for UML and OCL and enables the verification of models. It solves also the translation of three-valued logic from OCL into two-valued logic in PVS. This research is useful as the author of this thesis uses OCL constrains for modeling the behavior and restrictions of CPSs.

Based on these considerations, the Z language was chosen to specify the CPS UML models. There are two important reasons for this choice. First, the Z language is based on rigorous mathematical constructs. The second reason is the similarity between Z and OCL, since OCL is integrated into UML models.

Additionally, PVS tools are used for methods integration as shown in Chapter 5. From PVS the type checker and the theorem prover were found very useful. As analyzing a single example does not offer enough information for defining general guiding rules for rigorous specification, the case studies will be discussed in Chapter 5.

2.3.3 Summary

CPS UML design can be improved with the help of OCL constraints; however the models cannot be verified for correctness and completeness. Specification for UML models can be made with rigorous constructs, specified in different languages. The objective is to verify the system before deployment.

Z has the advantages of providing a rigorous mathematical specification. There are no general accepted tools to verify Z specifications, therefore Z language is used to specify examples of reduced complexity.

PVS is considered an alternative to the lack of verification tools for Z. The advantages of PVS specifications are visible in case of large-scale CPS models. In this perspective, Z involves a strong mathematical background, PVS being closer to programming languages and object-oriented design.

Chapter 3. Simulation Models for PSoC Based CPS Applications

The author presents in this chapter an extension over the OMNeT++ framework in form of simulation models for CPS applications. The author considers in this thesis CPSs which consist of different distributed PSoC devices, each of them being characterized by its own internal clock. Therefore, CPSs can contain different interconnected PSoC devices operating at different clock frequencies. In order to achieve cooperation between devices, synchronization mechanisms are required to be implemented at communication level. Testing applications written for CPSs is not an easy task. Tracking the execution after deployment is very difficult because the large number of messages exchanged between devices cannot be followed with accuracy. The screen provided by PSoC devices for displaying debugging purpose is limited, therefore it is straightforward that a detailed logging is not possible. Usings external debugging devices is an expensive solution. Moreover, it is not always feasible due to systems dimension.

A more suitable and less expensive solution is deploying and testing the application over a simulator. Relevant simulations are indicated to be performed before deploying the application, for debugging and also for performance evaluation. To achieve these goals, the simulation must be as realistic as possible and therefore to capture the correct system behavior.

An accurate simulation of the middleware layer is essential, to achieve a realistic simulation of each PSoC device [27]. Each system function implemented by the middleware requires a number of clock cycles. These clock cycles are executed by the middleware and cannot be interrupted from the application. Therefore, the clock cycles spent by middleware for processing each system function called by the application must be considered by the simulation models. A self-message mechanism can be used in OMNeT++ to simulate the time spent for performing a system function call in a PSoC node [26]. The author of this thesis makes a clear distinction between PSoC devices running at the same and at different clock frequencies. The next subchapters analyze these two possible cases.

The nodes cannot be powered up all at the same time in a distributed network [27]. This leads to different random phase shift values on each node. This is an important aspect for time aware distributed networks. The clock offset of each node increases the arrival time of the sent message on a multi-hop transmission. Phase shift may also lead to the loss of messages on a node. The results have low accuracy if the simulation describes only ideal conditions, in absence of phase shift support. A real simulation must consider all these aspects before starting the development. It allows phase shift aware simulation of distributed applications. The third subchapter concentrates on these issues and proposes a proper simulation model that covers these aspects.

The fourth subchapter introduces a speed-up mechanism for the presented models.

3.1 Interconnection of PSoC Devices with the Same Clock Frequency

The case study consists of a CPS composed of PSoC devices running at the same clock frequency [26]. At a certain moment of time, PSoC devices can have different values of their internal clocks. A possible scenario implies starting all PSoCs sequentially. The simulation model considers the internal clock value of each device and updates it accordingly. As shown in Figure 4, the clock and the scheduling module are implemented separately from the *Application* module. The consistency of the ticking mechanism can be ensured as long as the clock frequency of each PSoC device is the same as the frequency of simulation's virtual time. The *Internal Clock* module simulates the implementation of a physical clock unit from a PSoC device. At the initialization, the initial clock value can be expressed for each PSoC. To initiate a correct run, an internal tick event is scheduled during initialization step. At simulation's start, the internal clock of each node is synchronized with the system's virtual clock.

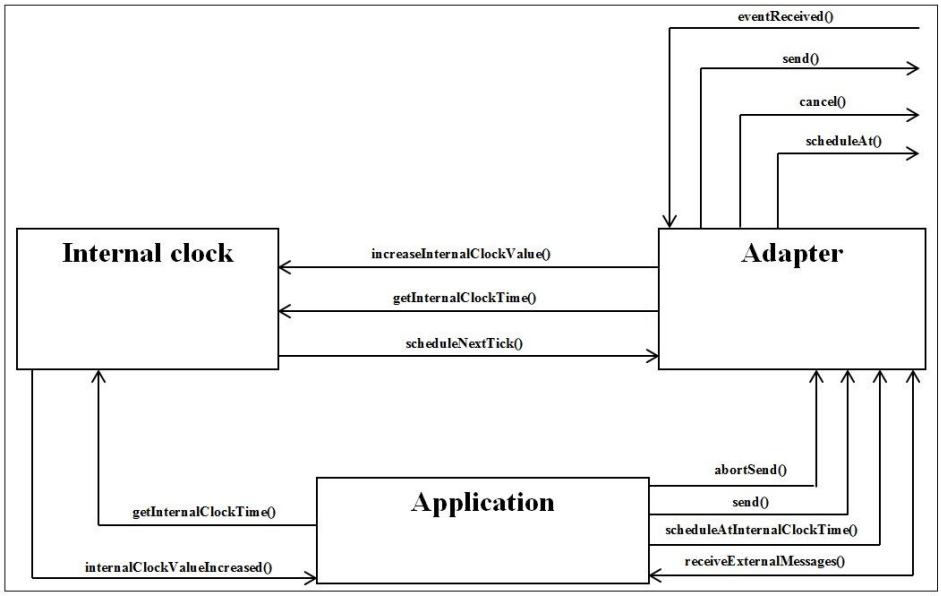

Figure 4 Node model for simulating PSoC devices with the same clock frequency

The *Adapter* module is the interface with the OMNeT++ simulator. It receives all the events that have the node as destination. This module provides a clear distinction between the receivers of the event and it parses the events accordingly. The *Adapter* module is responsible for adjusting the node's time to OMNeT++'s virtual time. It also sends and schedules the events. Scheduling an event implies determining the virtual OMNeT++ time. For determining the OMNeT++ time, the *Adapter* identifies the difference between the current hardware time and the desired hardware time. A mechanism for cancelling already scheduled events is also available.

The ticking mechanism functions as follows. At simulator's runtime, the *Adapter* receives the internal tick event scheduled in an initialization step. It is then

forwarded to the *Internal Clock* module. This module increases its internal clock value and schedules the next tick event. Next, the *Internal Clock* announces the *Application* about clock value change and, if necessary, the *Application* will read the node's hardware clock value.

3.2 Interconnection of PSoC devices with Different Clock Frequencies

The simulation model in case of different clock frequencies raises more complex issues [26]. Clock cycle level synchronization for different devices can be achieved using the greatest common divisor of the corresponding clock values. The initial assumption is that the virtual clock frequency has the same value as the greatest common divisor. This implies that for every PSoC device, a fixed number of virtual clock iterations must be consumed in order to increase its internal clock value. The *sampledValue* constant value is calculated for each PSoC node as the result of dividing its internal clock frequency to the greatest common divisor, calculated previously. System virtual clock frequency must be set to the greatest common divisor value, in order to achieve synchronization at clock cycle level for a CPS containing PSoC devices at different clock frequencies.

Figure 5 describes the node model for simulating PSoC devices that use different clock frequencies. The *Virtual Time Sampler* module is a major improvement for this scheme compared to the one presented in Figure 4. This module implements the connection between the *Adapter* and the *Internal Clock* of a PSoC device.

At system initialization, the *Virtual Time Sampler* module is responsible with the storage of the greatest common divisor. Also, the *Virtual Time Sampler* takes in charge the mechanism for scheduling internal clock ticks. The *Adapter* module announces the *Sampler* module that the virtual system time was already increased. The increase of the internal clock is performed every time the *Virtual Time Sampler* module receives a number of ticks equal to the constant value stored at initialization. When the tick occurs, it announces the *Internal Clock* module. The *Internal Clock* increases its value and announces the *Application* module.

In case an event must be scheduled, the scheduling mechanism must consider the internal clock time and the *sampledValue* constant for computing the simulation clock time. The formula used by the *Adapter* module for computation of resulting virtual clock time is presented in (1):

$$\Delta clk_v = \Delta clk_{HW\_x} \cdot k_{HW\_x} \quad (1)$$

The Δclk_v represents the difference between current virtual system time and the future virtual system time, the Δclk_{HW_x} represents the difference between current internal clock time and the future internal clock time with respect to that node, whereas k_{HW_x} represents its calculated *sampledValue*.

The rescheduling mechanism uses (1) when computing the necessary future simulation time. The *Application* module does the computation of the next value for internal clock time by evaluating the required execution time of the current system function call.

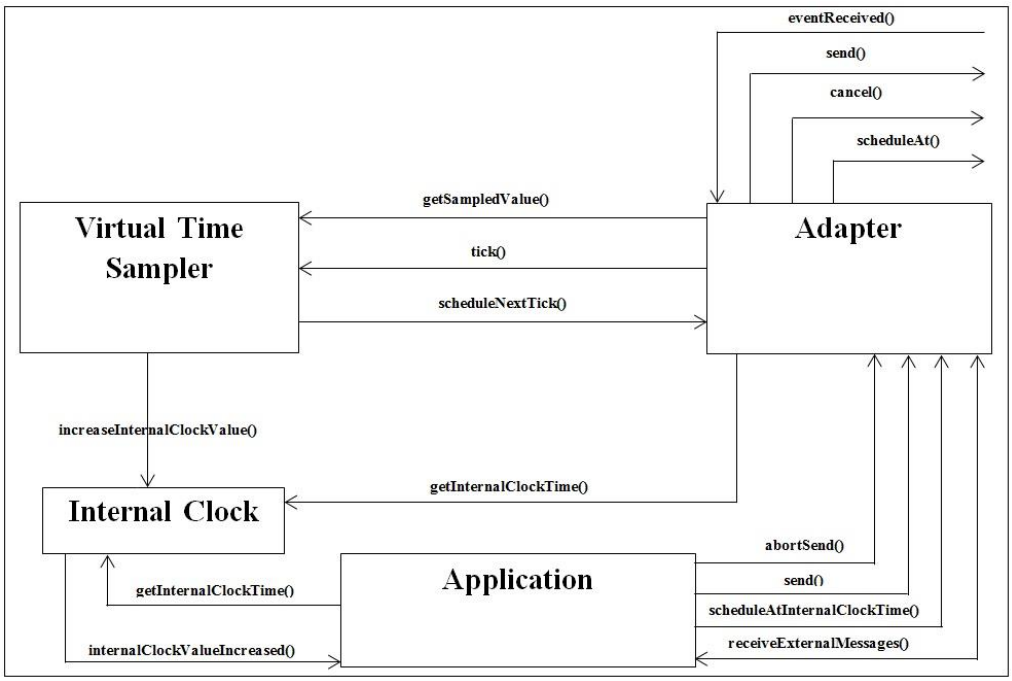

Figure 5 Node model for simulating PSoC devices with different clock frequencies

For exemplification, a possible rescheduling example that involves two nodes is detailed next. A correct illustration implies using numerical values. The internal clock frequency of the first node is set to 70 MHz and the frequency for the second one is set to 30 MHz. The greatest common divisor is set to 10 MHz.

Figure 6 captures the moment when the first node schedules an event for the second node, at its next internal clock time. The N_t represents the timeline of the system virtual clock having the frequency set to 10 MHz. The N_1 is the timeline for the node having the internal clock frequency set to 70 MHz. The N_2 is the timeline for the node having the internal clock frequency set to 30 MHz.

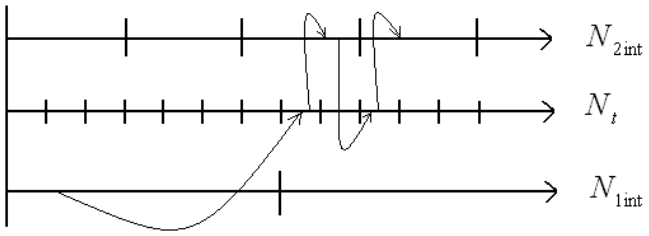

Figure 6 The scheduling scheme for different clock frequencies [26]

In Figure 6 there can be seen that node N_t receives the event at current virtual system time increased with 7 ticks. This is the result of applying formula (1). At the moment when node N_2 receives the message, it is busy for 1 internal clock

cycle. Therefore, it will reschedule the message for 3 internal clock ticks after receiving the message. Finally, node N_t will handle the message after 9 clock ticks of virtual system clock.

3.3 Simulating Devices Having Different Phase Shift (Clock Offset)

In simulation models for CPS networks, higher time accuracy requires taking into consideration the part of the time period of each node, already passed before powering up the entire network.

There is a possibility that at the end of start-up, some nodes in the network can have a delay greater than the clock cycles of those nodes. This implies that at end of start-up, some nodes might have their internal clock value set to a number greater than zero. Such nodes can be in the middle of their next clock cycle.

This situation is presented in Figure 7. It shows three different nodes (*N1*, *N2* and N3), each of them having their own internal frequency. The vertical line marked with t_0 is considered the network's end of start-up. At t_0, the internal clock time of each node *N1*, *N2* and *N3* is already ticking in the middle of the first cycle named T_1, T_2, and T_3, respectively. The passing time for each node at network's time t_0 is considered as being $Diff_1$, $Diff_2$, and $Diff_3$. In order to achieve the increase of their internal clock by plus one, the remaining time for each node after t_0 is considered to be t_1, t_2 and t_3, respectively [27].

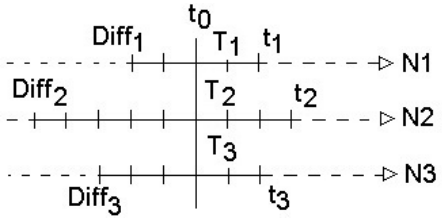

Figure 7 Simulation start-up with nodes having different phase shift [27]

Node *N1* has the highest frequency from all the three nodes. It could already achieve a full period, before $Diff_1$, but for clarity this is not shown in Figure 7.

Mathematical correlations performed before describing the simulation model try to offer a solution to the already described situation. The problem is raised by the exact determination of the duration of simulation time period, in order to correlate it correctly with the period passed for each node. The greatest common divisor, computed through equation (2), is used for solving this.

$$\gcd(Diff_1, Diff_2, ..., Diff_n, t_1, t_2, ..., t_n), \quad (2)$$

The function $\gcd$ computes the greatest common divisor of the terms from parenthesis. The set $Diff_1, Diff_2, ..., Diff_n$ expresses the part of the period already passed for each node at the end of the network's start-up. The set $t_1, t_2, ..., t_n$ represents the required part of the period to pass for each node from the end of the

network's start-up, in order to increase the internal clock value of that node with one [27].

The greatest common divisor will represent the value of the period of OMNeT++ virtual time. The usage of greatest common divisor ensures a correct update of the internal clock of a node, exactly on its rising edge. The demonstration, as stated in [27], is presented below.

$$Diff_{x,\,x=1..n} = m_{Diff_{x,\,x=1..n}} * \gcd \quad (3)$$

$$t_{x,x=1..n} = m_{t_{x,x=1..n}} * \gcd \quad (4)$$

In (3), the $Diff_{x,x=1..n}$ represents the part of the period already passed for node *x* at the end of the network's start-up. The $m_{Diff_{x,x=1..n}}$ represents the number with which the greatest common divisor must be multiplied in order to achieve, for node *x,* the value $Diff_{x,x=1..n}$. The $t_{x,x=1..n}$ represents the part of the period required to pass for node *x* from the end of the network's start-up, in order to increase the internal clock value of that node with one. The $m_{t_{x,x=1..n}}$ represents the number with which the greatest common divisor must be multiplied to achieve the value of $t_{x,x=1..n}$, for node *x*.

The period $T_{x,x=1..n}$ represents the period of node *x* required by the internal clock frequency of the node and is computed using (5).

$$T_{x,x=1..n} = t_{x,x=1..n} + Diff_{x,x=1..n} \quad (5)$$

Replacing the terms in (5) with (3) and (4), equation (6) will be obtained.

$$T_{x,x=1..n} = (m_{Diff_{x,x=1..n}} + m_{t_{x,x=1..n}}) * \gcd \quad (6)$$

Equation (6) states that the greatest common divisor already determined for all the nodes using (2) can be multiplied by a number for each node from the network in such manner that the exact value of the period of that node can be achieved [27].

The multiplier is obtained from (6) and it is presented below in (7).

$$m_{x,x=1..n} = m_{Diff_{x,x=1..n}} + m_{t_{x,x=1..n}} \quad (7)$$

The $m_{t_{x,x=1..n}}$ represents the number used to achieve the value of the period $T_{x,x=1..n}$ for the node *x*, by multiplying with the greatest common divisor.

In terms of simulation, if using (2) in determining the greatest common divisor, (6) and (7) represent a guarantee that a correct update of the internal clock time of nodes will be performed.

The model for simulating the distributed devices running at different frequency and having different phase shift is presented in Figure 8 and detailed in [27].

The *Shift Manager* module implements the connection between the *Adapter* and the *Internal Clock* of such device. This module is responsible with the storage of the greatest common divisor.

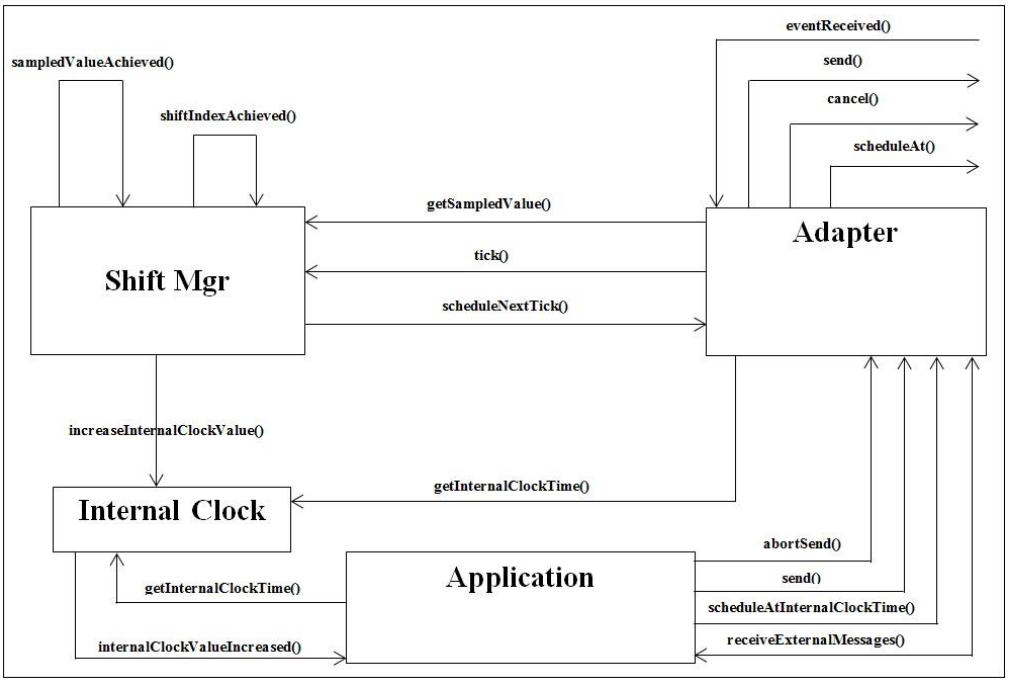

Figure 8 Node model for simulating PSoC devices with different phase shifts

At system's start-up, the constant values $m_{Diff_{x,x=1..n}}$ and $m_{t_{x,x=1..n}}$ are stored for each node. They represent the number of ticks already received from simulation's virtual clock at simulation's run-up, respectively the number of ticks necessary for *Shift Manager* module in order to notify the *Internal Clock* module to increase its internal clock value.

They are called *shiftThreshold* and *sampledThreshold* of the node *x*. An internal variable called *sampledValue* is used to count the number of ticks already received from the virtual clock of OMNeT++' simulator. The *sampledValue* is initialized with the constant value *shiftThreshold*.

At the end of initialization, the *Shift Manager* module calls *scheduleNextTick()* method. This implies that the *Adapter* module constructs an internal self-message and schedules it, by using the OMNeT++'s scheduling mechanism, to take place at the next OMNeT++'s virtual time. The existence of the scheduled self-message ensures that the *Shift Manager* module will be notified by the *Adapter* module about the increase of the simulator's virtual time.

Every time the OMNeT++ simulator increases its internal value, it will announce the *Adapter* module. The *Adapter* module makes a clear separation between notifications received from the simulator regarding the increase of its virtual clock time and external messages, which are triggered by other nodes.

The *Adapter* module notifies the *Shift Manager* module about the increase of the virtual simulator's clock value. The *Shift Manager* module will update the *sampledValue* by increasing it. Then, it will schedule a self-message for another notification about the increase of virtual clock time, by calling again the

scheduleNextTick() method. Next, it will check the *sampledValue* against the constant values *shiftThreshold* and *sampledThreshold*. If the value of *shiftThreshold* is achieved, the *Shift Manager* will update an internal flag (*shiftFlag*), in order to notify that the *sampledThreshold* value was already achieved and will reset the *sampledValue*. The *shiftFlag* is used also for debugging purpose, to know the situation in which the *sampledValue* is found at a given moment of time. If the value of *sampledThreshold* is achieved, it will notify the *Internal Clock* module to increase the stored internal clock value. After the call to the *Internal Clock* module's method, the *Shift Manager* will clear the *shiftFlag* and will reset the variable *sampledValue* [27].

Each time the *Internal Clock* module receives a command for increasing the stored *Internal Clock* value, it will update the *Internal Clock* value. Then it will announce the *Application* module about this modification.

Sending, scheduling, rescheduling and aborting a message is done in the same way as described in case of model for interconnecting distributed devices running at different clock frequencies [98]. The difference comes from the fact that the sampled value returned to *Adapter* module is calculated based on the value of *sampledValue* variable and based on the *shiftFlag*.

When an event must be scheduled, the scheduling mechanism must take into consideration the internal clock time, *sampledThreshold* and *shiftThreshold* values for computing the simulation clock time. The *Application* module computes the next value for internal clock time, by evaluating the necessary execution time of the current system function call and forwards the request to Adapter module. The *Adapter* module requests the values of *sampledThreshold* and *shiftThreshold* from *Shift Manager* module.

The formula used by the *Adapter* module for computation of the resulting virtual clock time in case of scheduling of a message is presented in (8).

$$\Delta clk_v = \Delta clk_{HW_x} \cdot m_x \quad (8)$$

The Δclk_v represents the resulting difference between current virtual system time and the future virtual system time. The $\Delta clk_{HW\,x}$ represents the difference between current internal clock time and the future internal clock time for node *x*. The m_x is the number with which the greatest common divisor must be multiplied to achieve the value of T_x, for node *x,* calculated using (7).

In case of requests arrived from the application for scheduling a message, it is considered that the schedules are possible only at the times at which the *shiftFlag* and the *sampledValue* of that node become zero, which means on the "rising edge" of the new internal clock value. A message cannot be scheduled inside a period of that node.

The formula used by the *Adapter* module for computation the resulting virtual clock time in case of rescheduling a message is presented in (9).

$$\Delta clk_v = T_x - s_x + 1 \quad (9)$$

The T_x represents the period of node *x*. The s_x is the current sampled value calculated by the formula (10).

$$s_x = sampledValue_x + shiftFlag_x \times m_{Diff_x} \quad (10)$$

The $sampledValue_x$ represents the number of ticks of OMNeT++ virtual clock received by the node *x*, after the last update of the node's *x* internal clock. The $shiftFlag_x$ represents the flag set on node *x* if the *sampledValue* has already achieved the constant value *shiftThreshold* of that node. Here, the m_{Diff_x} represents the constant value *shiftThreshold* in the node *x*.

The rescheduling mechanism uses (9) when computing the required future simulation time. Such a mechanism is required as there is a common situation in which a node receives an external message in the middle of the current period. It is necessary to reschedule for that node the already received message at the beginning of the next period.

The below example, which involves 3 nodes, will better clarify the scheduling and rescheduling mechanisms performed by the described model [27].

The internal clock frequency of the first node is set to 40 MHz, the frequency of the second node is set to 80 MHz and the frequency for the third one is set to 50 MHz. At simulation' start-up, it is considered that half of period for the first node, 5/8 of period for the second node and 3/5 of period for the third node have already passed. The remaining time in order to complete the period is as follows: half period for the first node, 3/8 of period for the second node and 2/5 of period of the third node. Therefore, the greatest common divisor is set to 10 MHz.

Figure 9 presents three distinct situations in which two nodes communicate.

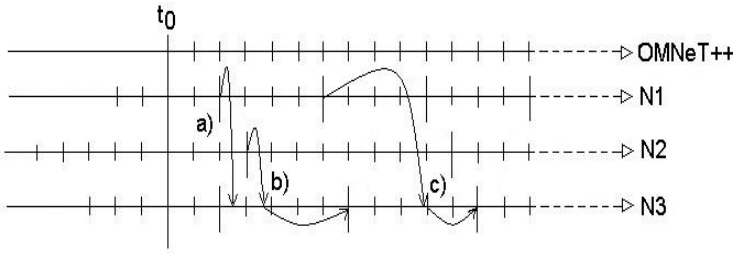

Figure 9 The scheduling scheme for nodes having different clock frequencies and different phase shifts [27]

When node's *N1* first update of its internal clock occurs, the node *N1* tries to send a message to node *N3*. The *Adapter* module of node *N3* receives the message and verifies the *shiftFlag* and the *sampledValue* from *Shift Manager* module. As node *N3* has already updated its internal clock value, the *shiftFlag* and the *sampledValue* are found set to zero. This allows the *Adapter* module to notify the *Application* module of node *N3* about the new message arriving. The *Application* module of node *N3* is not processing anything for the moment, therefore starts processing the arrived message.

After first internal clock update, the node *N2* decides to send an external message to node *N3*. The *Adapter* module of node *N3* receives the message and verifies the *shiftFlag* and the *sampledValue* from *Shift Manager* module. Although the *shiftFlag* is set to zero, the *sampledValue* is not equal to zero, and the *Adapter* module decides to reschedule the message in such manner that it will arrive at the time when a new internal clock update will be performed on node *N3*. After applying equation (10), the *Adapter* module of node *N3* decides to reschedule the message

after the last three parts of the remaining period and to receive the message exactly at the time when a new internal clock update takes place.

At node's *N1* second update of its internal clock, the node *N1* tries to schedule a new message to node *N3*, after one internal clock cycle. When the internal clock cycle of node *N1* is updated again, OMNeT++ sends the message to node *N3*. The *Adapter* module of node *N3* receives the message and verifies the *shiftFlag* and the *sampledValue* from *Shift Manager* module.

Although the *shiftFlag* is set to zero, the *sampledValue* is not equal to zero and the *Adapter* module decides to reschedule the message in such manner that it will arrive at the time when a new internal clock update will be performed on node *N3*. After applying equation (10), the *Adapter* module of *N3* decides to reschedule the message after the last two parts of the remaining period and to receive the message exactly at the time when a new internal clock update takes place.

3.4 Speeding Up Simulation Models

All three previous models present situations in which there were included also simulations of system function calls. These functions are made available through middleware implementation to the simulated application. A system function call requires uninterrupted execution and lasts a specific number of clocks. The duration of each system function is known before deploying the application and can be simulated using one of the three models presented above [28].

One requirement for using one of the three presented models is to implement also the simulation of the middleware, eventually separated from the simulation of the deployed application, on *Application* module side. This separation is to be performed by the developer of the simulation. To perform a realistic simulation of a CPS network, it is required for the developer to know the lasting time of each system function. The simulation of a system function requires no other activity to be performed on the node, as long as a system function is simulated [28].

This requirement may lead to situations in which, at a given moment of time, all the nodes are blocked in simulating middleware, as presented in Figure 9. In the presented situation, no node will generate any event. Each node must simulate the time required by the middleware for processing the called system function. In Figure 9, it can be seen that during the middle period shown in OMNeT++ timeline, the nodes *N1*, *N2* and *N3* have already scheduled events and wait for time to pass. The nodes will remain in idle state for the entire middle period. After several tests on different applications, it was observed that the presented situation is met very frequent.

A speed-up simulation time mechanism can be implemented as, in the presented case, the models must simulate only the time passing in each node. A mechanism which is able to speed-up the simulation in typical situations such as the one presented above has the major advantage of reducing the real time of simulation spent on the workstation on which the simulation runs [28].

For achieving this, it is necessary first to remove the correlation between the OMNeT++ simulation time and the absolute time of the network. In all three models, the *Adapter* module must not be connected through an intermediate module to OMNeT++ simulator, and not directly.

This new module should be controlled directly by the virtual time increase of the OMNeT++'s simulation. The module should be capable to translate correctly an OMNeT++ virtual time increase into network time increase, based on the minimal network time required to pass for simulating the middleware for a node. In order to perform a correct translation, the instance of this module should be the same for all the nodes. Therefore, the new introduced module will be named next as *Platform* module.

In a simulation implemented in such manner, four different timing notions will be distinguished, each of them being managed by its dedicated unit [28]:

- Workstation real time, managed by the operating system on which the simulation runs
- OMNeT++ virtual simulation time, managed by the OMNeT++ simulator
- Network time, managed by the *Platform* module
- Internal clock time of node, managed by the *Internal Clock* module of each node.

A clear separation is therefore performed in terms of timing. This will speed-up and increase the control of the simulation.

The interface with *Platform* module requires implementation of a new function on *Adapter* side, called *increaseNetworkTimeTo(),* and the call of new functions implemented on *Platform* module. The architectural view of a platform model is presented in Figure 10.

At simulation's start-up, for each node to be notified about the increase of the network time triggered by the *Platform* module, it must register itself as network time increase listener. The registration of each node can be done using *setTickListener()* method of the *Platform* module interface. At the time a node decides not to be updated anymore about the network time increase, it can call *unsetTickListener()* method in order to announce the *Platform* module about its request.

In order to simulate a system function call, a node can call the *scheduleMsg()* method. This method has the role to notify the *Platform* module that a node requires an amount of time to pass in order to perform a specific task. Until then, the node it will be idle. The *Platform* module will create a priorities queue and will order the node's request in the queue by the requested network time. The requested time must be translated by each node , before calling *scheduleMsg()* function, from its internal clock requested time into requested network time. This can be done internally by the *Adapter* module of each node, by applying the formula corresponding to its model. The *Platform* is notified when OMNeT++ simulation time increases. It takes the first request from the queue and notifies all the nodes about the increase of network time with a value equal to the requested network time found in the first position in the queue. By acting like this, all the nodes update their internal time values at the same time, in a controlled manner. The *Platform* sends the scheduled message to the node which initiated the request, after finishing the update of the *Internal Clock* module of each node from the network. It removes it from the queue and schedules to be notified about the next OMNeT++ virtual time increase.

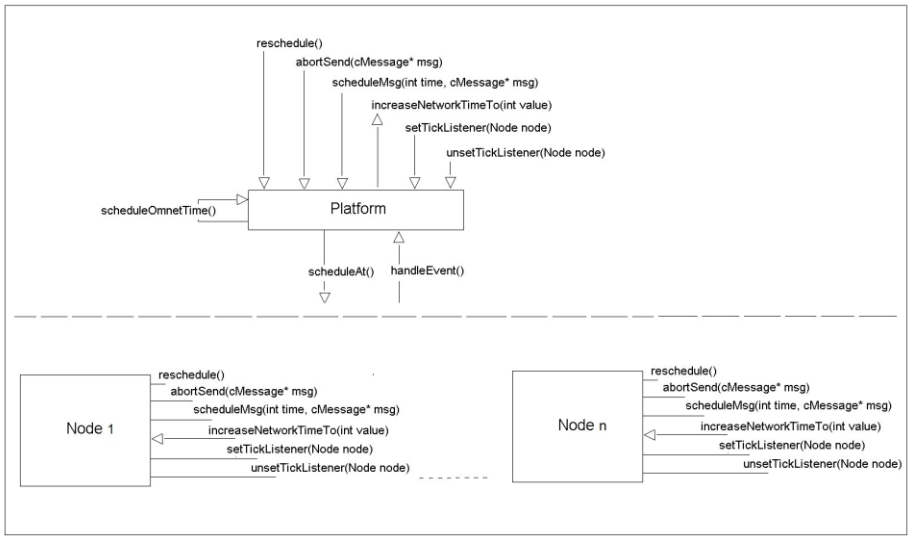

Figure 10 Platform model for simulating CPS networks

If a node decides to cancel a scheduled task at a given moment, it can call the *abortSend()* message and the *Platform* module will remove it from its internal queue. If a node decides to modify an already requested time at a given moment, it calls the *reschedule()* method. The *Platform* module will search for the request in queue and will update it with the new requested network time. When a node decides to send a message to another node, it will call the *send()* method from OMNeT++ simulator. The message will be sent to that node to be later processed [28].

3.5 Experimental Results

In this subchapter, the author of this thesis considers several sets of tests, all of them involving a grid based architecture for CPS networks [26]. In such a network, each node composes its own message. Then, it sends it in the network at simulation's start-up. The simulation is finalized when each node receives the messages sent from all other nodes in the network. This network used for simulation consists of 9, 25, 49 and 81 PSoC devices, respectively. A layout example of the simulation with 9 PSoC devices can be seen in Figure 11. The physical CPS network has the size and configuration of the one presented in the figure. It consists of 9 Cypress PSoC CY3210 devices linked by wireless connections.

For the first set of tests, the network has been simulated using PSoC devices set to the same clock frequencies. Each node has the architecture similar to the one presented in Figure 4. The network configurations have been further kept, but the nodes were implemented using architecture similar to the one in Figure 5. The CPS network has contained four types of nodes. Each one had a different clock frequency set at 10 MHz, 20 MHz, 30 MHz and 40 MHz, respectively. Therefore, the greatest common divisor was computed at 10 MHz. For a node with its internal clock frequency set to 10 MHz, one virtual clock cycle should be passed in order to tick

one time. In case of a node with the clock frequency set to 20 MHz, 30 MHz or 40 MHz, two, three or four virtual clock cycles should be passed, respectively, to be able to simulate their frequencies [26].

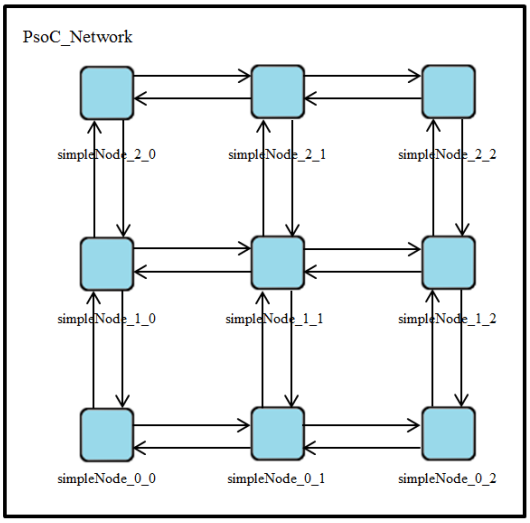

Figure 11 Simulation of a grid CPS architecture in OMNeT++

Figure 12 presents the simulation times obtained in case of CPS networks consisting of 9, 25, 49 and 81 PSoC device, respectively [26].

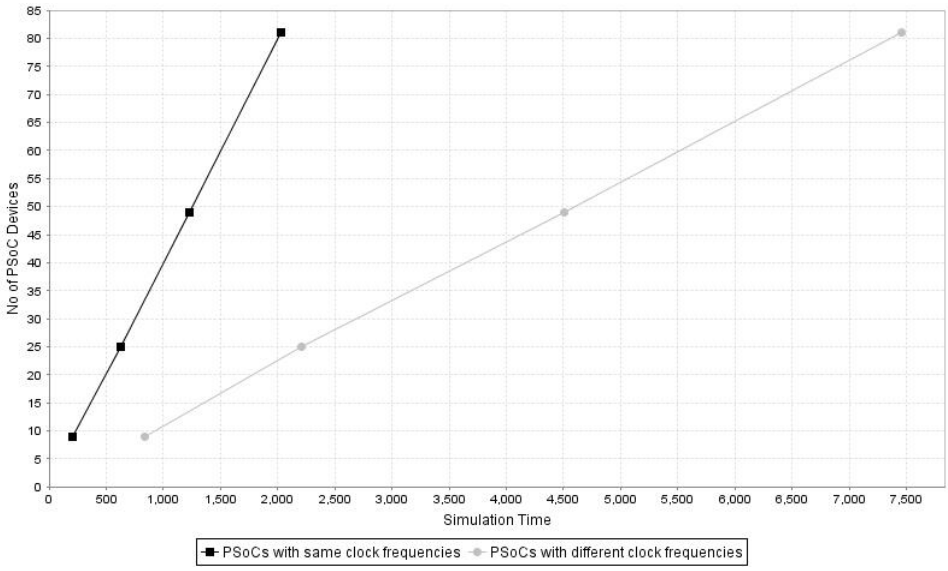

Figure 12 Simulation time for different CPSs [26]

The results obtained when the PSoC devices have the same clock frequency are represented in the thick line. The results obtained when the PSoC devices were set to four different clock frequencies are represented in the thin grey line. For a CPS with PSoC devices set at the same frequency, the simulation time increases proportionally with the size of the network.

The explanation for this is that the simulation ends when each node receives all the messages sent by the other nodes from the network. The size of the network increase determines the increase of the number of messages expected on each node. A similar situation was obtained when the CPS consisting of PSoC devices with different clock frequencies was simulated. However, comparing the results in the last two described tests, there can be seen that the time is significantly greater in case of simulating different clock frequencies. The explanation for this is that in a PSoC device that runs at a different clock frequency then the virtual clock frequency of the simulator, its clock frequency must be also simulated. This is a time consuming process.

The goal for the second set of tests was to determine how the number of different PSoC clock frequencies influences the simulation time necessary for a CPS consisting of 81 PSoC devices. First, the CPS network is set to contain PSoC devices with only two possible frequencies. One frequency was set to 10 MHz and the other one was set to 20 MHz. The next three tests were performed on the same CPS network that contains PSoC devices with internal clock frequency chosen from a set of 4, 6 or 8 possible frequencies. The frequencies were obtained using uniform 10 MHz increments.

Figure 13 shows the simulation time results obtained for a CPS with its network dimension set to 81 nodes. The clock frequency of each PSoC device was set to one of 2, 4, 6 and 8 different clock frequencies. As expected, if the number of possible clock frequencies a PSoC device may have is increased, the simulation time will increase exponentially. However, for the PSoC devices, a CPS network does not usually contain more than four possible clock frequencies [26].

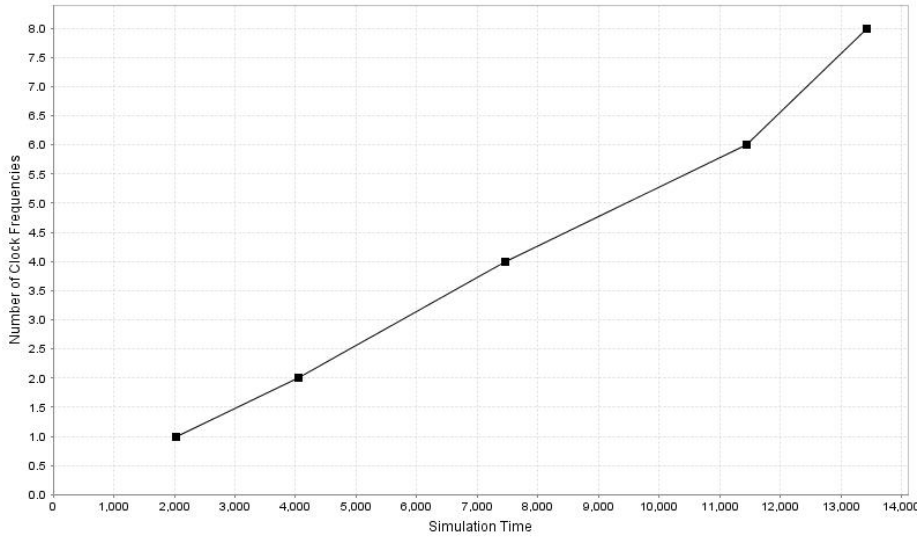

Figure 13 Simulation time for CPS with 81 PSoCs using different clock frequencies [26]

The next tests were based on simulating a simple grid-based CPS application constructed of nodes responsible with periodical measurement and control of the temperature for an environment [27]. The simulation has ended on a node when the node has already ended the sensing task for the third time and there is no message to be forwarded to other nodes or to target point.

The third set of tests has compared the phase shift model with the model for simulating distributed devices running at different clock frequencies. The author of this thesis has set for both models two families of distributed devices, each family having the internal clock frequency set to 50 MHz and 100 MHz, respectively. Based on the results from Figure 13, he has decided to use only two device families in these tests. The network size was set to 9, 16, 25 and 49 nodes, respectively. The greatest common divisor for the model for simulating nodes running at different clock frequencies is computed to 0.01, and the internal multiplier used for translating the tick period of each network clock equal to 2 for devices running at 50 MHz, respectively equal to 1 for devices running at 100 MHz [27].

In case of the phase shift model, the clock offset at network's start-up was considered having randomly one of the values 0.01 μsec or 0.005 μsec for each node. In this case, the greatest common divisor is equal to 0.005, and the internal multiplier used for translating the tick period of each network clock equal to 4 for devices running at 50 MHz, and equal to 2 for devices running at 100 MHz, respectively.

Figure 14 displays the simulation time requirements of the phase shift model compared to the different clock frequencies model. The clock-offset value of each node at the end of network's start-up is additionally taken into consideration for calculating the greatest common divisor. Decreasing the greatest common divisor leads to the increase of the internal multiplier of each node. This leads to the increase of the expected number of network periods to pass in order to update the internal clock value. For both models can be observed that the simulation time increases with the dimension of the network [27].

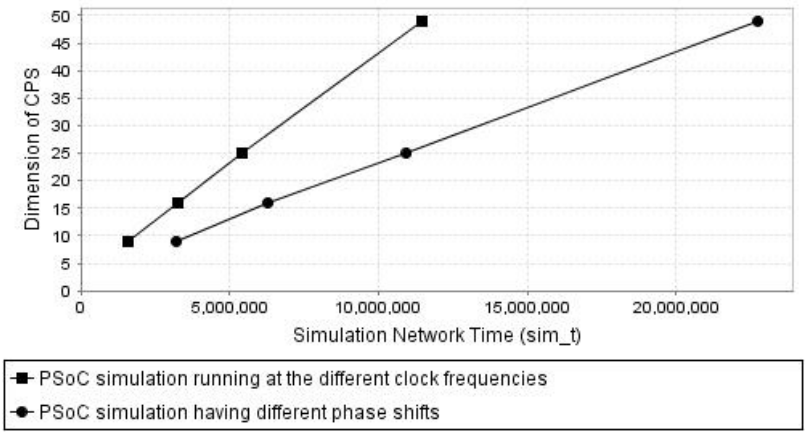

Figure 14 CPS simulation efficiency with and without different phase shifts [27]

Another simulation is on a simple grid-based application in which, the node located at position (0,0) exchanges three consecutive messages with the node located at the right-up corner of the network [27]. It is considered that the other

nodes are in idle state and therefore not performing any particular task. This simplification has been considered in order to exclude the influence of the results with particular situations in which a node may be in. The application has run on different CPS's having the network dimension equal to 9, 16, 25 and 49 nodes, respectively.

In the first set of tests, the CPS network consisted of two families of nodes running at 50 MHz and 100 MHz. No clock offset was defined. In the second set of tests the same families of nodes were considered. The nodes could then choose an internal phase shift value of 0 μsec, 0.01 μsec, 0.005 μsec, 0.0025 μsec, or 0.00125 μsec.

The results presented in Figure 15 demonstrate that the time required for the messages to arrive to nodes is increased in case of simulating the clock offset of each node. The delay can easily increase to significant values if a node implied in the construction of the communication channel is busy with a particular task. The delay becomes very important in case of simulating time-constrained distributed applications. These results show the importance of simulating the phase shift in a distributed network, where there is no synchronization protocol between nodes [27].

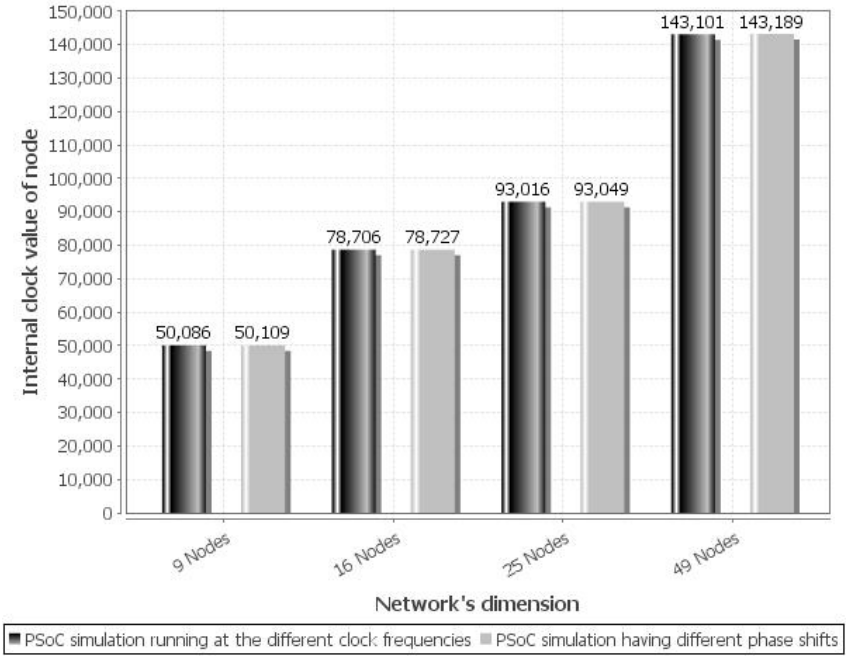

Figure 15 Results for simulating three message exchanges between nodes located in the corners of a CPS [27]

The next case considered is a CPS application which has run over a network consisting of devices running at the same frequency and which have had the network size set to 9, 16, 25 and 49 nodes, respectively.

Table 4 summarizes the results of this case study, compared to the speed-up model implemented on the same networks. As expected, the time for simulating the application decreases for the speed-up model and the difference grows exponentially with the size of the network. The explanation for this is that the

speed-up model does not require the scheduling of the tick messages, but only the useful ones.

Virtual simulation time compared to network size	9 nodes	16 nodes	25 nodes	49 nodes
Without speed-up model	1038089	1804374	3261379	7481212
Using speed-up model	4457	11194	25804	78242

Table 4 Simulation of distributed networks having devices running at the same clock frequency with and without the speed-up model support. The values are expressed in terms of OMNeT++ virtual clock cycles [28]

3.6 Summary

The author of this thesis concludes that the presented models are suitable for testing and validating with clock cycle level accuracy of dynamic aspects in massively distributed embedded networks (the units, nodes and the entire network). CPSs are representative for such networks.

The first model allows testing CPSs composed of PSoCs running at the same clock frequency. This model allows considering also the simulation time spent for each call of a middleware function to complete.

The second model allows testing CPSs composed of PSoCs running at different clock frequencies. The penalty paid using the second model is the growth of simulation time proportionally with the number of distinct clock frequencies.

The third model can be used for simulating CPS networks consisting of distributed devices, which have different phase shifts at the end of the network's start-up. The test results show that the model is able to update the internal clock value of each node correctly by considering the clock offset of each node. Moreover, the scheduling and the rescheduling mechanisms of OMNeT++ simulator are extended internally by this model to be able to consider correctly the time when the message needs to be scheduled or to be received. The strong dependency of the simulation time with the dimension of the network may lead to situations in which the simulation time increases considerably. The advantage here is that the time accuracy is maintained higher.

The speeding up model was possible due to separation of times into PSoC node internal clock time, required overall network time and OMNeT++ simulation time. This model overcomes the penalty introduced by the previous models. From the comparison in Table 4 it can be observed that this model is suitable for networks even larger than 49 nodes, as the simulation time is reduced.

Chapter 4. Validation of Static Properties in UML Model Specifications for CPS Applications

UML consists of a set of general-purpose modeling elements [4]. It is widely used for specification, visualization and documentation in various engineering fields. The author of this thesis has investigated the opportunity of using complete specification and verification as an efficient approach in case of static properties of CPS applications, designed using UML models. This research is detailed in [84].

UML models can be customized by defining and using stereotypes, grouped in UML profiles [85]. Stereotypes contain tagged values and constraints, which add specific attributes and behavior to the customized UML elements. As already stated, scientific studies [5] have demonstrated the expressiveness of UML profiles in defining UML models. Strong points in favor of using UML are also the intuitive graphical notations, the structuring mechanisms and the standardized methodology the UML models use.

A major drawback of the UML notation is the lack of mathematical support. As the UML does not provide a complete syntax, the semantics of UML models cannot be fully defined and, therefore, verified and validated.

OCL [66] was introduced to describe rules and constraints applied to UML models. OCL is a declarative language which aims to bridge the gap between the ambiguity of natural language in which UML models are expressed and the difficulty of mathematical constructs. However, since the semantics of OCL are not mathematically defined, using OCL language for expressing UML models is still not enough for rigorous reasoning [64].

Therefore, in case of considering exclusively the UML, the models cannot be verified for correctness and completeness. Specification for UML models can be expressed with the help of rigorous constructs, specified in different languages. Several tools can use this rigorous specification for verifying the UML models. The purpose is to prove before deployment that the system will function according to the expected requirements [86].

4.1 Z Specifications of CPS UML Models

The growing complexity of software dedicated to massively distributed embedded systems, as CPSs, requires new modeling and programming paradigms. Visual modeling, relevant use cases and declarative programming are investigated in various CPS research projects.

Accurate modeling helps system designers to ensure the correctness and completeness of business functionality and of end-users requests [4]. Models are useful in evaluation of system robustness, security, stability and extensibility before actual code writing. This evaluation provides information used to reduce the number

of errors from the beginning of the implementation. Proper modeling and detailed system documentation can be the key of an easy extension of the overall project in case of future requirements.

UML can be used in various steps of the system development lifecycle. At the beginning it allows high-level specification of the system. Then the level of the abstraction can be decreased when focusing on the main aspects of the system or increased back by hiding details in the models [4]. Most UML tools allow switching perspectives, from a simple element of the scheme to the entire environment where the application is executed. They make possible visualization of connections between elements or even connections between related applications. When combined with MDA, UML is able to automatically generate code.

A major drawback for the verification of UML models is the lack of strong mathematical support. Therefore, models must be translated in a specification using a rigorous language. Validation tools take this specification and verify correctness and completeness of the described system. The Z language has been chosen for rigorous specification of the UML models for static description, defined in the PIM of the CPS applications, especially due to its expressiveness and rigor.

The basic principles regarding rigorous specification of UML specific elements using Z language are summarized next [68]. The focus is on UML deployment diagrams, as these diagrams are commonly used for the hardware representation of CPS applications, for units, nodes and network communication.

A schema is a data specification structure and the primary construct of the Z language. A schema contains two parts. The former is a declaration part, which contains the local variables and their types. The latter is a predicate part, which contains predicate logic expressions used to define the relations and constraints between these variables.

A type is an expression of a restricted kind. Z types can be basic, as given set names or compound from simpler types. There are three categories of composite types as Set, Cartesian product and Schema. This means that simpler schemas can be used in constructing more complex schemas, just like regular variables of basic types. Types are very important in Z as the type of expressions for a specification can be automatically calculated to check if the types make sense.

Schemas can be used to model static and dynamic aspects of systems. They are called state schemas and operation schemas, respectively.

In a state schema, the variables are defined in the declaration part, while the constraints on the state are defined in the predicate part. These schemas can be used to represent the hardware specifications for CPSs.

In an operation schema, the input and output variables represent the *before* and *after* states. The relationships between these states are represented in the predicate part. Operation schemas can be used to specify the behavior for CPSs.

Using Z, the UML deployment diagrams can be represented similar to class diagrams. Therefore, the author of this thesis presents in parallel the similarities between rigorous representation of class and deployment diagrams.

A first approach is presented in [72]. It is based on automatically translation of annotated UML class diagrams in Z specifications. The authors implement a tool called RoZ, which generates elementary operations and some proof obligations in order to validate the model constraints. Considering the similarities between class and deployment diagrams, the author of this thesis has considered that RoZ tool can be extended in order to allow automatic translation of deployment diagrams into Z specifications.

According to [71], a class has two distinct meanings. A class intension defines the properties common for all objects of a class, like attributes and methods. A class extension defines the class through existing objects of that type. The UML object diagrams used for the design of CPS applications can be treated as class extensions. This implies that schemas for class extensions and object diagrams will be created using the same rules.

Each class can be translated into a pair of schemas. First schema gives the type for all elements of the class and corresponds to the class intension. Therefore, it defines the general type of a class. The second schema describes the set of existing instances of the class in the system. This is achieved through a variable that records the finite set of all instances for the class. The schemas for class intension and class extension can be specified manually or generated using RoZ tool.

A deployment node in a deployment diagram can be considered similar to a class from a class diagram. Both a node and a class are characterized by attributes and operations. A class can have instances of other classes as attributes. Also, a node can contain instances of other nodes as attributes.

Similarly, the schema corresponding to the node intension describes the types for all elements of the node and the general type of the node. The schema for the node extension describes the set of existing instances for the node. It includes a variable that records the finite set of instances for the node in the system context.

The associations are defined in UML as structural relationships between classes. In a deployment diagram, links can be used to describe communication between nodes or inside a node. When specifying the design at hardware level for particular CPSs, the links can describe the physical communication channels. One link must be specified for communication between each two nodes or each two components, therefore links are similar to one-to-one associations.

In Z language, each role of the association must be specified by a function. Each function associates an instance or a set of instances from a class to an instance or a set of instances from the corresponding class. The constraints specify that an association binds existing instances of the classes as definitions of the domain and range for each role. As these constraints involve the extensions for both classes, the corresponding schemas are included as variables of the schema describing the association. The final constraint states that both functions are linked and they contain the same information.

The schema for representing a link in Z language differs from the one representing an association. Each function associates an instance of a node to an instance to a corresponding node. When generating Z specifications using RoZ, composition is considered a special case for an association. The corresponding schema follows the translation rules for the association, as described above.

In [78], UML definitions are used to describe UML aggregation in Z language. UML aggregation is a special type of association, which indicates a lifetime dependency between the parts involved. There are two types of aggregation in UML, physical and catalog aggregation. In a physical aggregation a part instance only belongs to one aggregate instance, while in a catalog aggregation, there is no such restriction.

In a Z schema for physical aggregation, some specific rules must be considered. The variable part contains the set of instances that has been created in the system and not yet destroyed. To express this, the predicate part must define some conditions. First, two elements from a set of instances are instances of the same class if they have at least a part common. Next, all parts of aggregate

instances must come from the set of existing instances. All created instances of the part classes must be part of the created aggregate instances. Moreover, the schema must contain a predicate for expressing the uniqueness of object identifiers.

In case of a Z schema for catalog aggregation, the restriction is that created instances of the part class must be part of created aggregate instances. Therefore, the instances of part class can be part of one or more created aggregate instances. The identifiers of the instances must also be unique.

In Z specifications, the superclass of a generalization structure is specified in the same way as a regular class [78]. The subclasses are considered subspaces of the superclass instance space. The Z state schemas for these subclasses contain also a variable of the superclass type. Additional constraints must be stated for a given hierarchy of classes. A Z state schema containing instances of both superclasses and subclasses must clearly state that all already created instances of the subclasses are also instances for the superclasses. As in case of aggregation, the schema will contain a predicate for expressing the uniqueness for object identifiers.

4.2 PVS Specifications of CPS UML Models

As discussed in the previous subchapter, the Z language is a common solution for rigorous specifying UML models for different types of applications. Z specifications can describe most object-oriented concepts. However, they must explicitly capture the object-oriented semantics and the modeled concepts [71]. Using Z language implies a strong mathematical background when constructing schemas. It is also limited in handling dynamic aspects for the systems. And, most important, there are no generally accepted tools for proving Z statements.

An alternative to Z language in describing UML structures is PVS [79]. The PVS tools have been developed for both description and verification of rigorous specifications. This system consists of a specification language, a type checker, a theorem prover and some other additional tools. The specification language can directly support reasoning about infinite traces, which is helpful in the verification process. The PVS theorem prover allows verification of the theorems defined along with the theories for a certain system.

A PVS system specification consists of theories. Theories can contain defined types, variables, constant declarations, definitions, axioms and theorems. A theory can be parameterized and specific constraints can be defined on the parameters.

PVS tools are characterized by a well-defined type system. Along with basic types, it includes type constructors like functions, sets and records. Also, complex predicate subtypes and dependent types can be defined. The type checker automatically verifies type correctness and generates, if necessary, proof obligations. These proof obligations are called type correctness conditions (TCCs). After the proof obligations are satisfied, the user has the certainty of correctness and completeness of the defined types.

A main advantage in using PVS tools for specification and validation of CPS UML models resides in expressivity and power of the specification language. Indeed, this system is capable to easily express most of object-oriented statements. The

type checker and theorem prover ensure a step-by-step verification for the components of the theories describing the applications.

As already stated, the UML deployment diagrams are the used constructs for CPS representation at hardware level, in the described visual programming model. In the context of rigorous representation, elements of deployment diagrams can be represented similarly to the ones of class diagrams.

An UML class diagram is represented in PVS by a theory that contains theories for all class elements, as discussed below. An interface can be represented using a theory that contains a declaration of a record type [63]. The fields in the record type are in this case the signatures for the externally visible operations in the interface.

The signatures for the operations of a class are represented similarly to the operations for an interface, using a PVS record type. The record type contains also fields that are declarations of the attributes of the class. If a class is a subclass of another class and/or it implements one or more interfaces, the record type defined must include fields for the attributes and operations of all superclasses and/or for the operations of all implemented interfaces. The record type for the theory represents in this case a union between the record type for local attributes and operations and the record type for imported attributes and operations. The types defined in the superclasses and/or interfaces are accessed using the importing mechanism. The same rules for representing a class can be applied to nodes in deployment diagrams, as nodes are also characterized by attributes and operations.

The objects of the class, represented in UML by object diagrams, become in PVS specification language instances of the record type of the theory.

In fact, all notions from UML class diagrams can be easily translated to PVS specification language. UML template classes correspond directly to PVS parameterized theories. Adding an axiom to the corresponding theory specifies an UML abstract class if it states that the set of all instances for the abstract class is empty.

In [63], the authors define a generic parameterized theory as template for class diagram associations, generalizations and aggregations. This is an efficient solution to represent class diagram elements, as it only requires a rigorous definition for a single theory for all the elements.

In the context of a generic parameterized theory, the list of generic parameters contains the classes whose instantiations are involved in the association. It contains also the corresponding roles of the objects and the multiplicities as subset of the natural numbers. To define the multiplicity, the theory should include an axiom regarding the number of instances of one class that can be associated to a corresponding class instance.

Resulting instantiated theory is a relation on a set of objects of the corresponding classes. The direction of the association is given by the order of objects in the instantiation. This generic parameterized theory for unidirectional associations can be then modified to support bidirectional associations by replacing the ordered pairs with corresponding records. The record fields are in this case the classifier, the multiplicity and the role.

Links from deployment diagrams are represented through instantiation of defined generic associations. The rule applied in this situation states that links are one-to-one associations between nodes or components of a node, with multiplicity equal to 1.

A naming conflict can occur during instantiation of generic template theories when variables or types with the same identifiers are declared. An attempt to solve

this problem is presented in [63]. Here, the authors use PVS theory abbreviation mechanism. This mechanism relies on theory name prefixes added to relations that specify associations.

Aggregation can be represented in PVS using the same generic template theory described above in this subchapter. In this case instantiating the theory with a whole class and a part class is required.

For generalization, the superclass is represented in PVS specification language like any other class. However, there are differences in case of subclasses as they are represented as theories that import the theories of the superclasses. These subclasses theories define a record type that includes imported attributes and operations, along with local defined ones.

In case of composite structure diagrams, the PVS representation follows the same rules as in case of class diagrams. Similarities between classes and composite structures go from variable declaration section to association between UML elements.

OCL constraints can be easily represented and verified in PVS tools. They are a common method to specify requirements and restrictions on stereotypes defined in UML profiles. Since the UML profiles are important for the design of CPS applications, this is another strong reason of using PVS for specification and verification of CPS UML models.

OCL is characterized by three-valued logic. For representing this in PVS, it has to be encoded into two-valued logic. Another aspect that should be considered is the fact that OCL uses partial functions, while PVS tools allow only total functions.

The OCL formulas are translated directly into the PVS specification language. However, some special cases have to be considered. They include partial functions, OCL semantics and undefined values [83]. Partial functions can be translated to total functions by restricting the partial functions to their domains. Therefore, the domain for each of the partial functions must be specified. Recursive user-defined OCL functions can be directly translated to PVS specification language if a ranking function in the PVS output is specified. The problem in this case is that OCL does not define such termination functions.

The OCL expansion using conjunctions or disjunctions of universal and existential quantification can lead to possible infinite values. It can be considered as an example the set of all existing instances for a class or a node. In this case, the quantified expression must be translated with the return type true. This approach eliminates the undefined return type from OCL.

4.3 Case Studies

The general guidelines described in the previous subchapters are applicable to most CPS UML models. In this subchapter are considered some case studies starting from several examples found in literature. For the first example, the author of this thesis has used the Z language to specify the systems. For the last example, the author has used PVS specification language. RoZ tool [72] has been used as support for Z specification. As stated earlier, it allows automatically generation of Z schemas for the UML class, object and deployment diagrams along with the elementary operations.

4.3.1 Z Specification: Case Study of *Adaptive Unsharper Image Filter*

The first case study is focused on identifying all required steps for rigorous specification. The considered example is the *Adaptive Unsharper Image Filter* system presented in [47]. This is a filter that enhances the contrast of an input color image. To model this embedded system, defined UML profiles at both hardware and software level are used. The first step in verifying the resulted model is to investigate the possibility of translation of the stereotyped UML representation into a specification in Z.

However, the intention for this case study is not to provide the complete Z schema for the *Adaptive Unsharper Image Filter* application. Instead, the goal was to determine and to detect the impact of the number of stereotypes used for defining an application over the time necessary to validate that application using Z language. Subchapter 4.4.1 analyzes these aspects and presents the conclusions for this case study, focusing on verifying the physical context constraint in which the image filtering operation has to be executed.

The rigorous specification flow starts with defining the Z types required by the system configuration. Thus, the hierarchy of UML stereotypes must be translated into Z language. The specification of a stereotype is similar to Z schema for class intension. Stereotypes that are specialization of other stereotypes must include the Z schema of the latter stereotypes in the variable part, as schema invocation. As a result, *pt_struct* stereotype defined in [47] is a specialization of the *pt_module* stereotype. Therefore, the Z schema for *pt_module* stereotype must be defined first and then included in the variable part of the Z schema for *pt_struct* stereotype.

Considering the example, types like *SEM_T* for logical semaphores, *IMAGE_TP* for the image which needs to be processed, *SPLIT_IM* and *UNION_IM* for splitting into red\green\blue colors and reconstructing the image, respectively, are defined using Z schemas. These particular defined types are not described in detail as the scope is here to identify the required steps for a specification of the *UNSHARP_IM* module type.

In Figure 16 the Z specification of *UNSHARP_IM* module type of the *Adaptive Unsharper Image Filter* example is illustrated. All defined modules are customized using *pt_struct* stereotype. This implies that the Z schemas for all modules intensions must contain the Z schema for the stereotype, as variable. In this case it corresponds to *pt_struct* stereotype. To facilitate visibility, only variables that are relevant for the presented flow were presented in Figure 16. Figure 16 (a) presents the Z schema for the intension of the *UNSHARP_IM* module type. Figure 16 (b) depicts a Z schema for the module type extension.

Each functionality provided by one of the system's modules needs to be described in its own corresponding Z schema. As an example, Figure 17 presents the schema for *doFilter* operation provided by the *UNSHARP_IM* module type. The first line in the variable part of *doFilter* schema indicates that the effects of this operation are limited to modification of *UNSHARP_IM* module instances.

The below presented methodology must be applied to each new stereotype, module and functionality involved in application specification as described already.

Figure 16 (a) Partial Z specification of the *UNSHARP_IM* module type; (b) Partial Z specification of the set of instances for *UNSHARP_IM* module type

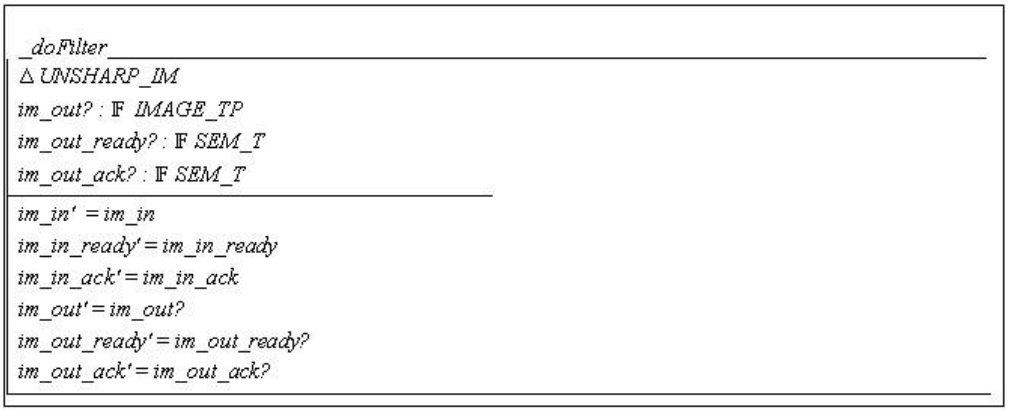

Figure 17 Partial Z specification of *doFilter* operation

This represents in the end specifying the Z schema for the entire application. The resulting application schema will contains references to all schemas defined for stereotyped modules and nodes. It also contains references to all relations defined between nodes.

4.3.2 Z Specification: Case Study of a *Sensing Node* Model

In the previous subchapter, the author of this thesis has presented rigorous specification at stereotype level and introduced Z specification at the node level. Here, he details Z specification at the node level in a CPS application. As case study we consider the system presented in [16]. It describes a CPS for traffic management in a regular urban intersection.

This application contains several types of nodes. A decision node computes optimal green color duration for each traffic light node. The calculations are based on the information provided by the sensor nodes. The sensor nodes from the presented CPS gather information about the number of cars waiting at the red color of the semaphores.

Each node specification is composed of hardware and software models. The hardware model is customized using stereotypes of the UML profile detailed in [16]. The communication between the hardware units of a node is wired. The nodes also communicate through wires.

Figure 18 presents the UML model of hardware configuration for the sensing node used in this CPS.

The specification of the sensing node internal architecture is expressed using a UML deployment diagram. The UML components are instances of deployment nodes. These artifacts are customized with stereotypes, which hold constraints related to different unit types used. The internal hardware connections and the external connection with the corresponding traffic light node are also visible in this figure.

The internal hardware structure of the sensing node is described following the guiding rules for the representation of class diagrams, as described in subchapter 4.1. In this case, the rules are applied to deployment diagrams.

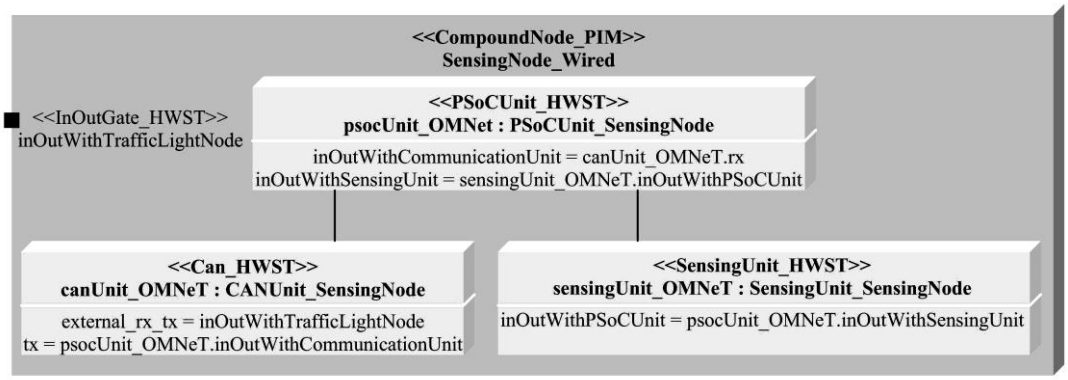

Figure 18 UML model for hardware configuration of a sensing node

A set of Z schemas is defined for each deployment node stereotype. Each schema is named according to the stereotype name. The variable part of it contains all tagged values defined by the stereotype. The predicate part contains the OCL statements expressed using Z language.

Figure 19 presents the tagged values and OCL constraints of the *Can_HWST* stereotype used in this case study.

```
Can_HWST
self.base_Node.ownedPort -> exists(a | a.name='tx' and a.oclIsTypeOf(OutputGate_HWST))
self.base_Node.ownedPort -> exists(a | a.name='rx' and a.oclIsTypeOf(InputGate_HWST))

bool_define_tx_enPort: BOOLEAN

if self.bool_define_tx_enPort = 'true'
then
  self.base_Node.ownedPort -> exists(a | a.name= 'tx_en' and a.oclIsTypeOf(InOutGate_HWST))
  else true
endif
```

Figure 19 OCL Constraints for *Can_HWST* stereotype

The *Can_HWST* stereotype defines the communication ports for the component it customizes. As the focus in this example is on the Z specification at sensing node level, this stereotype is considered being represented in its corresponding Z schema. As presented in the previous subchapter, the Z schema of the sensing node's CAN unit contains the *Can_HWST* schema as a variable. This variable indicates the corresponding stereotype. The Z specification prover tool will later determine whether the stereotyped component is correctly customized with its corresponding stereotype in terms of variables and constraints.

In Figure 20 (a), the variable part of the *SENSINGNODE_WIRED* schema defines the sensing node type along with its internal hardware unit types. For each of these components, a Z schema has to be also provided. Figure 20 (b) describes the set of existing sensing nodes types inside of the network. Figure 20 (c) illustrates a concrete instantiation for the node already presented in Figure 18.

Figure 20 (a) Z specification of the *SENSINGNODE_WIRED* node; (b) Z specification of the set of instances; (c) Z specification for a particular sensing node type

As presented in Figure 20 (c), *SensingNode_Wired* is an instance of the *SensingNode* type and expresses a concrete node used in the CPS network. It also contains the actual instantiations for the internal node unit components. The connections between these units are described using Z functions.

The links between nodes in the deployment diagram or between components of a node are represented using functions defined for both roles of the association. The constraints are meant to define the domain and range of these functions. The constraints also define the fact that both functions are linked and they refer to the same information. More details about possible types of connections are provided in the paragraphs of subchapter 4.1 describing the UML relations between two nodes in a deployment diagram.

In Figure 21, the author of this thesis presents the Z specification for the bidirectional link between a sensing node and its corresponding traffic light node.

Actual values for the domain and range of these functions are set in the network specification. The network representation at physical level contains the schemas of node intensions and node extensions for all nodes in the traffic network, along with the links between nodes. These schemas are represented also as functions.

In this case study, the focus is on specification and validation of the application at the node level, therefore the network schema is not presented in details. Based on presented specification steps for a CPS node, specification of the network schema becomes intuitive.

Subchapter 4.4.2 covers the validation process of the elementary operations generated using the RoZ tool [72]. The generated elementary operations along with the evaluation are discussed there in more details.

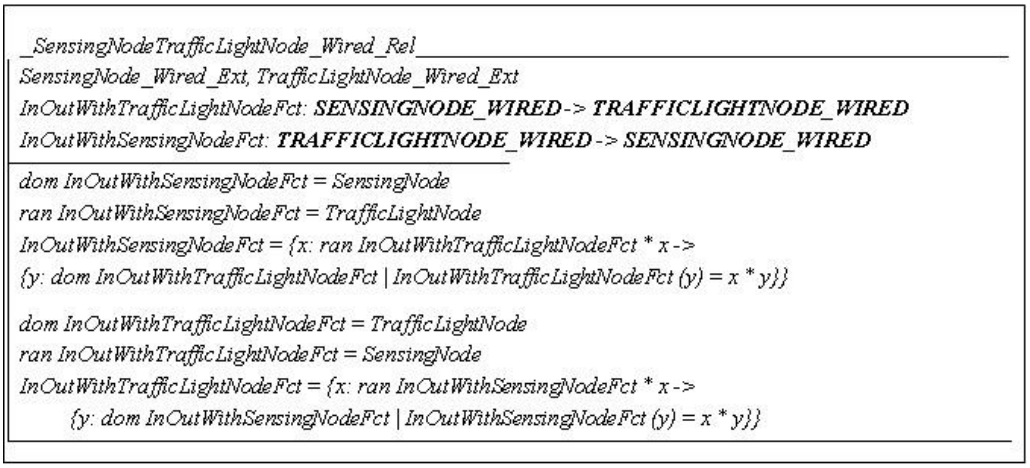

Figure 21 Z specification for relationship between sensing nodes and traffic light nodes

4.3.3 PVS Specification: Case Study of a *WSN Monitoring* Application

To investigate problems in specification of wireless communication in a CPS at network level, the author of this thesis has considered an application for a gas distribution monitoring system. Various references present such kind of applications. However, they consider different approaches in handling the monitoring issues. For example, in [87] the authors present a type of pipeline leak detection and also a localization method based on hierarchical model pattern. In this subchapter, the author of this thesis discusses the case of a monitoring system application for an urban gas distribution CPS network, introduced in [88].

A gas distribution system requires a large number of interconnected pipes. Therefore, to manage such a large network a great amount of sensors is required, along with local management nodes and valve actuators. Compared to a real gas management application, the example presented here tries to minimize the usage of

nodes. The aim is to keep the network as simple as possible, for a better understanding of how is it topologically structured. This allows revealing the benefit of using PVS tools for evaluating such CPSs. The author of this thesis has chosen PVS tools because it is easier to specify and verify complex networks using PVS specification constructs, along with type checker and theorem prover.

The gas pipes network presented in [88] is modeled first using UML. One particularity of this case study is the usage of wireless communication between nodes. Wireless communication operating frequency is considered in ISM band. Basically, a stereotype inserts its tagged values as mandatory attributes for the customized node. As an example, the author presents the tagged values inserted by *CompoundNode_PIM* stereotype. These tagged values are shown in Figure 22.

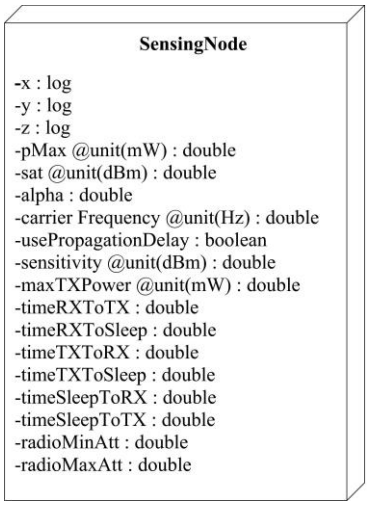

Figure 22 UML deployment for sensing node [84]

When deploying such nodes, it is necessary to specify the position (*x, y*, *z*), transmission and reception parameters for the node. The application goal is to control the gas flooding in case of pipe leaks. Due to the need of an efficient monitoring, the network has been logically tailored into perimeters, zones and areas. For expressing this tailoring at UML level, stereotypes definitions are required for each modeling level. The OCL constraints for these stereotypes express their meaning. Next, for better understanding, the author of this thesis presents these OCL constraints in natural language.

A grouping stereotyped with *Perimeter* expresses a set of nodes containing a single instance of a node customized with *DM_PerimeterCompoundNode_HWST* stereotype. All the other nodes are instances of nodes customized with *CompoundNode_PIM* stereotype.

The *Zone* stereotype expresses a set of nodes containing a single instance of a stereotype named *DM_ZoneCompoundNode_HWST* whereas all other nodes are stereotyped with *Perimeter* or are instances of *CompoundNode_PIM* stereotype.

The *Area* stereotype expresses a set of nodes containing a single instance of a node customized with *DM_AreaCompoundNode_HWST* stereotype. All other contained nodes are stereotyped with *Zone* or *Perimeter* or are instances of nodes customized with *CompoundNode_PIM* stereotype.

The communication between areas and zones, zones and perimeters and perimeters and nodes is enssured using beacon approach [89]. In order to focus on relevant aspects in our discussion, it is considered the simplified example of unidirectional communication: areas communicate to zones, zones communicate to perimeters, whereas perimeters communicate to sensors and valves. Figure 23 shows the UML model of this CPS. Similar to the approach presented in [63], the author of this thesis has represented the stereotypes in PVS specification language.

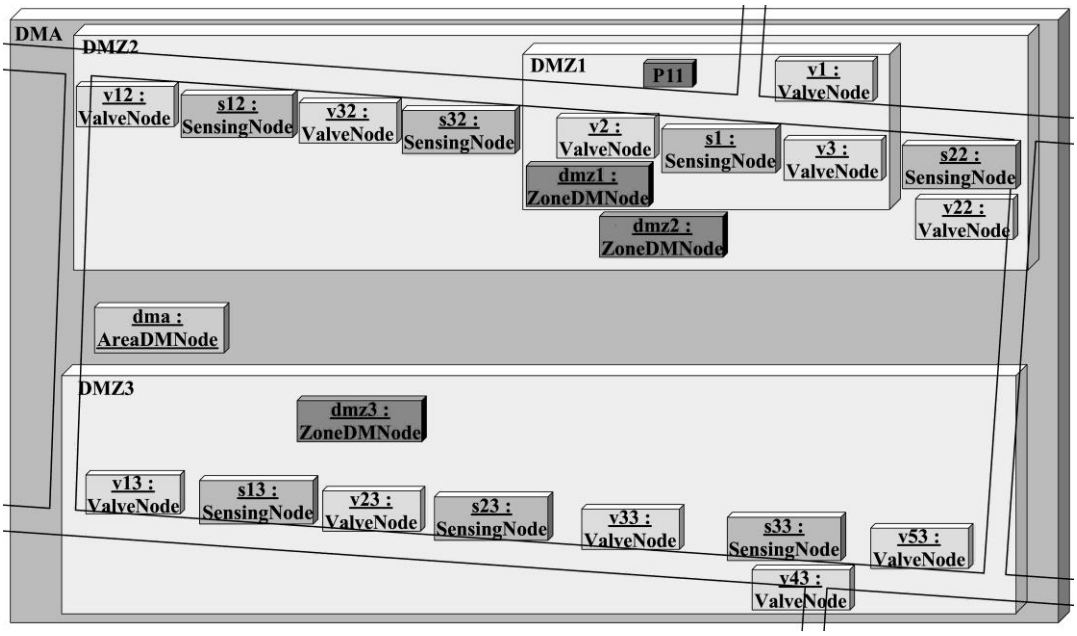

Figure 23 UML model for distributed gas monitoring topology

Figure 24 depicts stereotypes used for defining the network physical nodes as PVS theories. For each of these stereotypes the author of this thesis has defined a record type, where the fields are declarations of tagged values. When necessary, signatures of operations are also included in the record type. As in this case study it is considered a specific scenario in which the definitions at area level are evaluated, Figure 24 displays only the relevant tagged values for each stereotype.

The *X*, *Y*, *Z* types are subsets of integer values expressing the bounding box of the network's deployment area. Generally, in PVS the types defined in a theory are available to other theories by the importing mechanism. The inherited tagged values and operations are used along with local defined tagged values and operations.

The author of this thesis proposes the same approach of importing types and usage of attributes for defining stereotyped nodes. Thus, the theory describing the stereotype is imported in the theory describing the node itself. This ensures availability of the stereotype record type in the node theory.

Figure 25 presents stereotypes used for describing the tailoring of the network as PVS theories. All of them are based on the previous specified theories.

```
DM_AreaCompoundNode_HWST : THEORY
  BEGIN
  IMPORTING AREA_CARRIER_FREQUENCY, PMAX, SAT, ALPHA, X, Y, Z
    DM_AreaCompoundNode_HWST :TYPE = [# carierFrequency : CARRIER_FREQUENCY, pMax : PMAX, sat :
SAT, alpha : ALPHA, x : X, y : Y, z : Z #]
  END DM_AreaCompoundNode_HWST
DM_ZoneCompoundNode_HWST : THEORY
  BEGIN
  IMPORTING SENSITIVITY, X, Y, Z
    DM_ZoneCompoundNode_HWST:TYPE = [# sensitivity : SENSITIVITY, x : X, y : Y, z : Z #]
  END DM_ZoneCompoundNode_HWST
DM_PerimeterCompoundNode_HWST : THEORY
  BEGIN
  IMPORTING SENSITIVITY, X, Y, Z
    DM_PerimeterCompoundNode_HWST :TYPE = [# sensitivity : SENSITIVITY, x : X, y : Y, z : Z #]
  END DM_PerimeterCompoundNode_HWST
CompoundNode_PIM : THEORY
   BEGIN
   IMPORTING SENSITIVITY, X, Y, Z
     CompoundNode_PIM : TYPE = [# sensitivity : SENSITIVITY, x : X, y : Y, z : Z #]
   END CompoundNode_PIM
```

Figure 24 PVS Theories for *CompoundNode*, *PerimeterDM*, *ZoneDM* and *AreaDM* stereotypes

```
AREA : THEORY
  BEGIN
  IMPORTING ZONE, DM_AreaCompoundNode_HWST
    AREA : TYPE=[# dm : DM_AreaCompoundNode_HWST, zones : setof[ZONE],
    perimeters : setof[PERIMETER], nodes : setof[CompoundNode_PIM] #]
  END AREA
ZONE : THEORY
  BEGIN
  IMPORTING PERIMETER, DM_ZoneCompoundNode_HWST
    ZONE : TYPE=[# dm : DM_ZoneCompoundNode_HWST, perimeters : setof[PERIMETER],
    nodes : setof[CompoundNode_PIM] #]
  END ZONE
PERIMETER : THEORY
   BEGIN
   IMPORTING CompoundNode_PIM, DM_PerimeterCompoundNode_HWST
     PERIMETER : TYPE=[# dm : DM_PerimeterCompoundNode_HWST,
    nodes : setof[CompoundNode_PIM] #]
   END PERIMETER
```

Figure 25 PVS Theories for *Perimeter*, *Zone* and *Area* stereotypes [84]

In this subchapter, the author of this thesis has shown how PVS language can be used to specify subsystem parts of CPS applications. The entire system can be specified in the same manner. In subchapter 4.4.3 it is discussed how verification

can help detecting lacks in initial specification. This task is accomplished with the help of PVS type checker and theorem prover.The aim is to adjust UML models before starting the development process.

4.4 Verification of CPS UML Specifications

The previous subchapter has detailed how CPS UML models can be represented using Z and PVS specification languages. The aim is to use rigorous specification for model verification. Through verification, one can check the ambiguities, the correctness and the completeness of the UML models.

4.4.1 Verification of Z Specification for *Adaptive Unsharper Image Filter*

Z is a well-defined language, displays a precise semantics and can be used to represent object-oriented constructs. The validation of Z specifications implies usage of a specialized proof tool.

The UML model and Z specification of the *UNSHARP_IM* module type from the *Adaptive Unsharper Image Filter* [47] were presented in the subchapter 4.3.1.

The verification of Z descriptions should consider the constraints defined for stereotypes used in customizing modules. To facilitate this procedure, constraints are expressed similar to Z schemas for operations. Z schemas for customized module types will be verified against these Z schemas for constraints. Proof obligations are then generated from specification analysis. The process can be automated using existing verification tools for Z specifications.

The duration of the verification process for Z specification is influenced by the size of the specified constructs for the system. If each module and component is customized using different stereotypes, the specification code size will increase significantly. In the *Adaptive Unsharper Image Filter* case study, all modules are customized using *pt_struct* stereotype, which allowed maintaining the size of the Z specification at reasonable levels. This led to a better verification time, compared to other applications of the same complexity and which require defining larger number of schemas for their contained stereotypes.

4.4.2 Verification of Z Specification for the *Sensing Node* Model

Each type of UML diagram construct can be manually transformed into rigorous specifications using the set of rules described in subchapter 4.2. This process can be optimized using dedicated tools. RoZ tool [72] generates a complete Z specification from an annotated UML class diagram. This approach was presented in detail in subchapter 4.2. RoZ tool can also generate specifications of elementary

operations over the UML elements and proof obligations. A common example of such elementary operations deals with modification of class attributes. This is significantly helpful in validation of the UML model constraints.

As deployment diagrams can be specified similarly to class diagrams, modification operations can be generated for attributes of deployment nodes. Following the same reasoning, modification operations can be also automatically generated for attributes of composite structure diagrams.

As a method for determining the impact of defined operations over the UML model constraints, the author of this thesis has used computation of corresponding operations preconditions. This implies determining the conditions that must be satisfied before the operations takes place. The aim is to preserve the constraints at the end of the operations [90].

When considering the traffic management example from the previous section, there are several operations that can be analyzed in the model.

Figure 26 (a) Z specification of *ModifySensingUnit* operation; (b) Z language statements for sensing node model; (c) Z language statements for sensing node model; (d) Z language evaluation of sensing node theorem

For flexibility, the model ensures the possibility of modifying sensing unit types at the sensing node level. Therefore, a sensing unit from another producer could replace the unit included in the original design, which results in a new type of the sensing node.

For simplicity, only *ModifySensingUnit* operation is discussed, which is described by the Z operation schema from Figure 26 (a). The first line in the schema delimitates the domain of the effect of this operation to objects of type *SENSINGNODE_WIRED*. The predicates state that all attributes keep their initial values except for the one representing the sensing unit. This attribute receives the value of the *newSensingUnit_SensingNode* input parameter. The precondition for this operation verifies the existence of values for the new types of sensing units. Figure 26 (b) presents this precondition represented as Z specification.

Furthermore, the information can be used to describe a theorem used to validate the precondition. This theorem is depicted in Figure 26 (c). This theorem is generated using the RoZ tool.

To validate Z theorems, several tools were proposed in literature. The author of this thesis uses in his work the Z-EVES theorem prover [77], similar to authors of [72]. The commands required for proving the precondition for the discussed operation demonstrate the validity of the defined precondition, as presented in Figure 26 (d).

4.4.3 Verification using PVS Tools of a *Wireless Network Area* Model

In case of UML models, rigorous specification is followed by verification, with the support of PVS tools. The author of this thesis considers the example of the gas distribution network presented in Figure 23. The system designer has already defined for it the position of each node and the physical parameters described in Figure 22. For a complete specification at the hardware level, the author of this thesis considers also the OCL constraints from the definitions of perimeters, zones and areas, along with the specific OCL constraints for nodes.

However, the constraints do not evaluate the ability of the *dma*, *dmz1*, *dmz2* and *dmz3* nodes to communicate. There is no guarantee at this point that nodes are in the communication range of each other and a valid route can be established. To ensure this functionality, one needs to evaluate the values of the parameters implicated in the construction of the transmitting-receiving signals in the specific application context. The application context is the sum of all constraints of nodes implied in communication. Depending on these values, a possible situation is when *dma* node is able to communicate to *dmz3*, but the transmission cannot reach *dmz2* or when *dmz3* node cannot receive of the signal correctly due the weak antenna sensibility of *dmz3*. Robust verifications are required to avoid these cases at design time, before the effective deployment of the CPS.

Signal transmission in wireless networks is influenced mainly by the signal frequency, path loss, receiver sensitivity and noise. Path loss, known as attenuation, is influenced by deployment configuration, distance between the transmitter and the receiver, height and location of the antennas, obstacles and weather conditions

[91]. The power of the received signal is gained by multiplying the power of the transmitted signal with every attenuation mapping.

In literature, several propagation models are used to estimate the radio signal propagation distance. These models rely on information specific to a considered scenario. One of main challenges is to collect sufficiently large data to cover different situations in the scenario. For complete verification of the system, one needs to consider a proper radio model in PVS theory. The appropriate radio model is chosen based on the application's specific operating environment, the existing obstacles and the technology used to implement wireless communication [91].

The author of this thesis starts his investigation with a simplified radio model as described in [92] and presented in Figure 27 (a). This tiny radio model does not take into consideration obstacles and other environment limitations. Figure 27 (b) details the theory used to evaluate the capability of the decision module of the area to communicate with the zones included in its boundaries. Changes in radio model to take into consideration obstacles have no direct influence on the already specified PVS theory used for evaluating the area transmission.

(a)

```
DMA : THEORY
  BEGIN
  IMPORTING TINY_RADIO_MODEL
  DMA : TYPE FROM AREA
  receive(x, y, z, x_dm, y_dm, z_dm, node) : bool =
    (IF  (sqr(x - x_dm) + sqr(y - y_dm) + sqr(z - z_dm) <= sqr(range_receiver(node))) THEN true
        ELSE false
    ENDIF)
  inRange(x, y, z, x_dm, y_dm, z_dm, range) : bool =
    (IF  (sqr(x - x_dm) + sqr(y - y_dm) + sqr(z - z_dm) <= sqr(range)) THEN true
        ELSE false
    ENDIF)
  checkCommunicationWithAllNodes: CONJECTURE FORALL (node: zones): bool =
    (IF inRange(x(node), y(node), z(node), x(dm), y(dm), z(dm),
          calculateRange(carierFrequency(dm), pMax(dm) , sat(dm) , alpha(dm)))
       THEN receive(x(node), y(node), z(node), x(dm), y(dm), z(dm), node) == true
        ELSE false
    ENDIF)
  END DMA
```

(b)

```
TINY_RADIO_MODEL : THEORY
  BEGIN
  IMPORTING PI_CONSTANT, LIGHT_CONSTANT, AREA
    TINY_RADIO_MODEL  : TYPE=[#carierFrequency : CARRIER_FREQUENCY, pMax : PMAX,
    sat : SAT, alpha : ALPHA #]
    calculateRange(carierFrequency, pMax, sat, alpha) : nat =
         pow((sqr(speedOfLight_value/carierFrequency) * pMax)
        / (16 * sqr(pi_value) * pow(10, sat / 10)) , 1.0 /alpha)
  END TINY_RADIO_MODEL
```

Figure 27 (a) PVS Theory for *dma* model; (b) PVS Theory for a tiny radio model

Conditions for calculating the communication capacity are expressed in form of PVS theorems. These become objectives to be verified using PVS theorem prover. Type checking is a precondition for the evaluation of the theorems. It represents an intermediary step between a completely specified theory and theorem proving. The

PVS type checker searches for semantic errors, like ambiguous types or undeclared variables in the theory.

In *DMA_Theory*, the *receive* function determines if an external node is in the receiving range of the considered node. The decision is based on the receiver sensitivity. In this specification, verification determines if the node with which *dma* wants to communicate is in *dma*'s range, based on evaluation of *receive* function. In this case, the evaluation of *checkCommunicationWithAllNodes* theorem fails.

An extended model is presented in Figure 28. It includes also the sensitivity of the receiver in the theorem condition.

```
checkCommunicationWithAllNodes: CONJECTURE FORALL (node: zones): bool =
  (IF inRange(x(node), y(node), z(node), x(dm), y(dm), z(dm) ,
       calculateRange(carierFrequency(dm), pMax(dm) , sat(dm) , alpha(dm)))
       AND inRange(x(dm), y(dm), z(dm), x(node), y(node), z(node), range_receiver(node))
       THEN receive(x(node), y(node), z(node), x(dm), y(dm), z(dm), node) == true
      ELSE false
  ENDIF)
```

Figure 28 Corrected PVS method for validating the *dma* model [84]

The author of this thesis concludes that performing rigorous verification of all theorems defined for CPS model ensures a proper validation of the assumptions made in design. Initial decisions risk to remains incomplete without the help of PVS theorem prover.

4.5 Summary

CPS applications are present nowadays in various fields of activity. High-level UML modeling is a promising approach. A deficiency of UML informal language used to express CPS models resides in lack of support for automatically verification and validation. Although OCL offers expressiveness in UML design, it is not enough for a proper validation. Rigorous specification of UML models is therefore required.

The author of this thesis has presented in this chapter a research on using Z language and PVS tools to specify and verify CPS UML models. He uses Z as it provides a rigorous mathematical specification. As there are no general accepted tools to verify Z specifications, he uses Z to specify examples of reduced complexity.

To overcome the lack in complex verification tools, the author of this thesis considers PVS as an alternative. PVS consists of a specification language, a type checker, a theorem prover and additional tools. The author has demonstrated the advantages of PVS in a more complex reasoning, involving specifications for large-scale CPS models based on wireless communication. From this perspective, Z involves a strong mathematical background, while PVS is more close to programming languages and object-oriented design.

The author of this thesis concludes that CPS UML design can benefit from using OCL constraints to improve model expressiveness but the correctness and completeness of models require rigorous specification for automat verification and validation.

Chapter 5. Handling Event-Driven Scenarios in CPS Applications Simulation

Event-driven simulators for distributed applications are preferred instead of other types of simulators for several reasons [93]. These reasons are the simulation time control facility in different points of the simulation process and the possibility for elaborating specific simulation scenarios. The simulation time control facility allows the developer to analyze the overall status of the distributed application, at specific simulation time.

When using such simulators, simulation scenarios can be defined in a consistent manner, having the same network behavior and results over several runs. Re-running already defined simulation scenarios when the business logic implementation changes, allows behavior comparisons and performance analysis on the distributed applications.

However, there is no general format for specifying simulation scenarios for distributed applications. The simulation scenarios are very application specific and, based on the goals of the distributed application, can be specified using different formats.

5.1 Event-Driven Model Specification

The observations of the author of this thesis regarding common situations in which events are sent by a device determined the event-driven model specification. The common situations behave similar, in patterns, for the time for sending or changing the state on the corresponding device. The event-driven simulation of a particular device is commonly developed as state machine pattern. Based on a particular list of recognized events, the device can change its state, can start a particular task or can send other events to other devices.

This first aspect is expressed in Figure 29, as an XML-based event-driven model. The devices which communicate with each other are identified by their name. In case of wireless communication, they are also identified by their network address. Each device must specify a list of its internal possible states, in which that device can be found.

For each state, a transition list to other states must be defined, along with the list of recognized received events at which the device being in that state should react to.

Figure 30 details the specification of the transition list. This list consists of the possible conditions for entering a new particular state. These conditions for a transition can be defined as a list of particular recognized received events or as a maximum time the device can remain in the current state. Both these conditions sets can be fulfilled. However, the first condition to occur will trigger the transition

to the next defined state. As from one state a device may get into several possible other new states, for each possibility a transition must be defined.

```
<device name = str_Device_1 srcAddr = int_addr>
 <state name = str_CurrentState_1>
  <transitionsList> ... </transitionsList>
  <eventsList> ... </eventsList>
 <\state>
 ...
 <state name = str_CurrentState_s> ... <\state>
</device>
...
<device name = str_Device_n srcAddr = int_addr> ...
</device>
```

Figure 29 XML specification model for known devices and their possible internal states [93]

```
<transitionsList>
 <transition nextState = str_NextState_1>
  <receivedEvent name = str_receivedEvent_1 />
  ...
  <receivedEvent name = str_receivedEvent_m />
  <timeout value = int_timeoutValue />
 </transition>
 ...
 <transition nextState = str_NextState_t> ...
 </transition>
</transitionsList>
```

Figure 30 XML specification model for the transition list of a particular internal state [93]

Figure 31 presents and details the list of recognized events for which the system can react by sending new constructed events to the other devices. The recognized events can be of two types: external sent events from other devices or internally scheduled events.

When the device is in a particular state, each event for this device contains the following properties: the event type, which is specified by the name tag, the input receiving gate through which the device gets the event, the index of the gate used for a vector receiving gate, and, optionally, in case of wireless interconnections, the device network address which has sent the event. The device can react differently on two events with identical names and can make the difference between them, based on the other receiving parameters.

In such a case, the reaction on the device side consists in a list of events which need to be triggered. The user can define a delay by specifying the timeout value, from the time when receiving an external event and the time for sending events to other devices. By default, the timeout value is left empty and the device will react immediately.

```
<eventsList>
<receivedEvent name = str_receivedEvent_1
inputGate = str_inputGate index = int_gateIndex
srcAddr = int_senderAddr>
  <timeout value = int_timeoutValue />
  <triggeredEventsList> ...
  </triggeredEventsList>
</receivedEvent>
...
<receivedEvent name = str_receivedEvent_r
inputGate = str_inputGate index = int_gateIndex
srcAddr = int_senderAddr> ... </receivedEvent>
<timestamp value = int_timestampValue>
  <triggeredEventsList> ...
  </triggeredEventsList>
</timestamp>
...
<timestamp> ... </timestamp>
</eventsList>
```

Figure 31 XML specification model for the possible events and their particular timestamps of an internal defined state [93]

Discussing the *eventsList* tag, particular moments from the period when the device is in that particular state can be defined using *timestamp* tag definition. If the timestamp is set to zero, the list of external events required to be triggered must be taken into consideration, at the moment when the device enters in that particular state. If timestamp is set to minus one then, the device will trigger the events found in the encompassing *triggeredEventsList* tag, by exiting that particular state.

```
<triggeredEventsList>
  <triggeredEvent name = str_eventFormat_1>
    <outputGate name = str_outputGate_1>
      <index value = int_gateIndex_1>
        <destAddr value = int_addr_1 />
        ...
        <destAddr value = int_addr_k />
      </index>
      ...
      <index value = int_gateIndex_i> ...
      </index>
    </outputGate>
    ...
    <outputGate name = str_ outputGate _o> ...
    </outputGate>
  </triggeredEvent>
  ...
  <triggeredEvent name = str_eventFormat_p> ...
    </triggeredEvent>
</triggeredEventsList>
```

Figure 32 XML specification model for the possible events to be triggered in case of a recognized event or timeout occurrence of an internal defined state [93]

The developer is allowed to define special moments for sending external events. This can be done by setting the timestamp value equal to the period necessary to pass from the moment when the device enters in that particular state. The attributes required to define a list of external events to be triggered for the previous described situations are detailed in Figure 32.

Each event must have a format. Here, the format is the message string required to be sent, concatenated with formatting specifications for inserting particular data into the message. The formatting specifications are similar to C/C++ input-output formatters ("%s" or "%d"). The output gate parameters must be specified after constructing the event string required to be triggered. In case of wireless communication, a list of destination addresses describing the devices which will receive the events can be composed.

The discussed solution for describing the event-based specification represents an effective manner for displaying the flow of the events and their interactions. Based on the discussed descriptions, event-driven scenarios can be deployed and updated very easily.

5.2 Event-Oriented Programming Model

An event-driven programming model must be defined, in order to complete the event-driven specification. This programming model must be able to handle a particular scenario designed by the developer. The event-driven programming model must follow the state machine pattern. The reason for this requirement is that most of the event-driven simulation applications are written as event reactive applications, based on this pattern.

Figure 33 defines in pseudocode a states manager. Some actions need to be taken, when changing the current state to a new one.

First, the manager calls the onExit() method on the old state, and allows it to make modifications before exiting. Next, it changes the current state to the new requested one. In order to perform particular initializations, the manager calls the *onEntry()* method on the new already set state. Then, it calls the *performAction()* method. This method is responsible with the state's main job.

```
CLASS GenericTaskManager
BEGIN
 VAR GenericTask currentTask
 void changeTaskTo(GenericTask newTask)
 BEGIN
  currentTask.onExit()
  currentTask = newTask
  currentTask.onEntry()
  currentTask.performAction()
 END
 bool handleEv(str recvEv)
 BEGIN
  currentTask.handleEv(recvEv)
 END
END
```

Figure 33 *GenericTaskManager* implementation in pseudocode [93]

The pseudocode implementation, called *GenericTask,* of the proposed event-driven programming model is presented in Figure 34. This model is the basic implementation of a state. The code is structured based on the reusability and extendibility principles. The user is responsible for extending this implementation to particular states for a device. This can be done by adding customized implementation for the abstract methods defined in Figure 34. While using these principles, the effort for elaborating particular application dependent devices behaviors is reduced. The development models presented in [26] and [27] are based on the same principles.

These models, used together with the presented event-driven programming model, dramatically reduce the application specific code that needs to be written.

```
CLASS GenericTask
BEGIN
  const str PATH_TO_XML_MODEL
  const str ON_ENTRY
  const str ON_EXIT

  virtual void onEntry()
  BEGIN
    transitionChain = readTransitionChain()
    constrTransitionScheme(transitionChain)
    IF (hasDefinedTimeout())
        schedTimeout()
    END IF
    triggerExtEvIfAny(ON_ENTRY)
  END

  virtual void onExit()
  BEGIN
    triggerExtEvIfAny(ON_EXIT)
  END

  bool handleEv(str recvEv)
  BEGIN
    bool result = false
    IF (isTimeoutEv(recvEv))
      result = true
    ELSE
      IF (isIntSchedEv(recvEv))
        result = handleIntInEv(recvEv)
      ELSE // is external received ev
        result = handleExtInEv(recvEv)
      END IF
    END IF
    triggerExtEvIfAny(recvEv)
    checkIfChangeToState(recvEv)
    return result
  END

  void triggerExtEvIfAny(str recvEv)
  BEGIN
    ExtIEvFList list = getExtIEvs(recvEv)
    WHILE( list.hasElements()){
      str extEvFormat = list.getExtEvFormat()
      str extEv = constrExtOutEv(extEvFormat)
      send(extEv, list.getGate(),
          list.getGateIndex(),
          list.getDestAddrList())
      list = list.getNext()
    END WHILE
  END

  abstract void performAction()
  abstract str constrExtOutEv(str evFormat)
  abstract bool handleExtInEv(str ev)
  abstract bool handleIntInEv(str ev)
END
```

Figure 34 Pseudocode implementation for the event-driven programming model [93]

The author of this thesis details the possible implementations in a particular state, for each abstract method defined in the basic state. *performTask()* method has been already discussed. The *handleExtInEv()* and *handleIntInEv()* methods are called each time an external incoming event or an internal scheduled event are received, respectively.

By implementation, the user can create particular behavior responsible for internal updates or modifications of contained variables. Also, the user is responsible for retrieving a Boolean answer in case of correct detection of the event type. In the *constrExtOutEv()* method, the developer must define specific implementations for the event formats recognized by the state. The method must return the formatted event text, which contains the message string concatenated with the replacements for the format specifications.

Regarding the business logic implementation of *GenericTask*, when the *onEntry()* method is called, the transition chain for this state is read and an internal representation of it is stored, for future use, in a transition scheme.

As a next step, the programming model verifies if there is any timeout defined and schedules it. Timeouts are the periods during which the device must maintain the current state, along with timestamps defined by the developer for triggering particular lists of external events. If defining a state timeout, the user must not define a timestamp greater than the timeout already set, otherwise the timeout will not be taken into consideration by the *schedTimeout()* method.

The code calls the *triggerExtEvIfAny()* method to verify if there is any timestamp set to zero for a list of external events to be triggered. Such a situation means that by entering in this new state, the developer wants to send the external events list. By calling the same *triggerExtEvIfAny()* method, the *onExit()* method checks if there is any timestamp set to minus one for an external event list.

The method *triggerExtEvIfAny()* is responsible for detecting the external events list mapped to the event received as parameter from the constructed transition scheme. The event parameter can fetch a timestamp or can be an internal scheduled or external event received by that device. If finding any external events list to be triggered, it sends those events through the specified gates and gate indexes. The method uses the specified device destination addresses for each event from list, if set.

The main method of the event-driven model is the *handleEv()* method. First, it checks the type of the received event. This event can be an already scheduled event, a timeout event, a timestamp event or an external received event. For an already scheduled event, the method calls the customized implementation of the method *handleIntInEv()*. If the received event is an external one, the method calls the implementation of the method *handleExtInEv()* from the particular state extension. If tge event if of type timeout or timestamp, no specific handling is required at this step. Next, independent of the event type, the *triggerExtEvIfAny()* method is called. After triggering any possible external events list, it calls *checkIfChangeToState()* method to check if a change to another state is required to be performed.

The next possible state verification is performed in the *checkIfChangeToState()* method. First, the method verifies if the event is of external or internal scheduled type. If the result is positive, the method checks if the corresponding transition list of the current state contains the event. If a timeout event type occurs, the state's timeout tag is checked to see if a change to a new state is required, based on the period finalization. In both cases, the method will cancel any timeout events before starting the *GenericTaskManager*'s state change method. If exiting the current state, the manager will handle the call of the *onExit()* method of the current state.

The discussed event-driven programming model can be implemented in any event-driven simulator. Once designed, this programming model can be reused in other distributed applications.

5.3 Experimental Results

The tests were focused on the reduction of the number of lines of code required for simulating distributed applications. For simulation, the author of this thesis has used the OMNeT++ framework. The experiments were focused on reasonable-sized networks. Clear illustrations of the advantages in using this model can be obtained with simpler experiments. The majority of the event-driven specification attributes necessary for constructing the simulation scenario were used. The other attributes can be deducted.

The first test consisted of simulating the management of a simple traffic intersection constructed of four traffic light nodes. All nodes had the same type. This implied that it was required to elaborate an event-driven simulation specification consisting of a single device type. The implemented device type contained only three states, corresponding to the colors of a real traffic light. The transitions were similar to the ones for a real traffic light device. The simulated traffic lights changed their colors based on timeout, as there was no necessity to communicate to each other. Therefore, no external events list had to be constructed. The task in the simulated application was to add specific color management behavior in the inherited *performAction()* method. The other abstract methods were not extended, as no particular behavior was required. Evaluated this simple case study indicated an overall improvement of 45% by using the implementation of the presented model, compared to the regular case in which writing of the entire simulation application from scratch is required. Using also the first development model presented in [26], the simulation model for PSoC devices functioning at the same clock frequency, the improvement increased to 85% compared to the previous already described approach.

Next, the author of this thesis has investigated a more complex case study, simulating an interconnected set of traffic intersections. Each traffic light intersection had the task to optimize its internal traffic lights based on the received events from the adjacent intersections.

A traffic light intersection consisted of three types of compound devices, each of the compound devices consisting of several internal hardware units. The types of compound devices used in the construction of the simulation model for the set of traffic intersections were as follows: the sensing node type, the traffic light node type and the decision node type. Each intersection consisted of four traffic light nodes.

Each traffic light node had associated a sensing node for detecting the cars waiting at red color of the semaphore and for transmitting the sensed data to a local intersection management node. Internally, the sensing node type consisted of a communication unit, a video-based sensing unit and a central processing unit. Internally, the traffic light node type consisted of a communication unit, an actuation unit and a central processing unit. The actuation unit was used for color switch.

The compound device, named decision node, characterized each intersection and was capable to analyze traffic data received from the sensing nodes located on each of the four directions of the intersection. Based on the received data, the decision node was able to refresh the timeouts of the traffic lights located in the

area. It was also able to communicate its decision to the other decision nodes located in the adjacent intersections.

Internally, the decision node type consisted of a communication unit, a decision unit and a central processing unit. The internal decision unit was used by the decision node in computing the optimized traffic lights timeouts based on linear programming equations. The used linear programming equations described the state of the intersection, at a given moment of time in simulation.

In this example, the communication between the compound nodes was considered wireless. Therefore, the type of the communication unit used in this simulation consisted of a physical transmission layer, a MAC layer and a network layer. The central processing unit was used as application layer.

Each unit was managed by its node's application layer. Inside the compound node, the communication between the units was based on wires. For a precise synchronization, the application layers had to have the same internal state as their controlled particular units.

The MIXIM platform ([94]) was used for simulating wireless communication. Each sensing node expected a particular command to be triggered by the corresponding traffic light located on its sensing direction. At the time when displaying the red color, the traffic light node tried to start the sensing operation on the proper sensing node.

Simulation Configuration	**Actuation node**	**Sensing node**	**Decision node**
Without XML communication schema	Around 1000 lines of code	Around 1400 lines of code	Around 2200 lines of code
With XML communication schema	Around 600 lines of code	Around 1200 lines of code	Around 2000 lines of code

Table 5 Evaluation of lines of code based on the node model required for designing a traffic management application, with and without communication schema in place

Table 5 shows the results obtained in terms of lines of code necessary to develop the nodes needed for traffic management system. As it can be seen, for the improvement for developing the node using the already discussed communication XML schema is around 40%. Sensing node development shows also an improvement of 14 %, whereas the decision node development implies a 10% improvement in terms of written code lines.

As a consequence, the overall network simulation development results shows a reduction of 20% of the development effort, compared to the usage of the implementation of the presented model with the regular case in which the writing of the entire simulation application from scratch is required. When using the model presented in [27], a more accurate modeling of physical time, and the reduction was calculated to 40% compared to the previous already described approach.

5.4 Summary

The author of this thesis presents in this chapter a static event-driven model specification for CPSs distributed applications. The event-driven model is described in XML format. The goal of this research is to allow the developer to describe specific simulation scenarios for distributed applications.

The event-driven specification represents an effective manner for detailing the flow of the events and their interactions. Clear state-to-events and state-to-transition mappings improve understanding of the overall application architecture. While the refactoring process is simplified, it allows the user to update the constructed scenarios to future simulation changes.

Additionally to the event-driven specification, the author of this thesis has presented an event oriented programming model for handling specific scenarios for CPSs applications. The programming model is a generic one and can be implemented in any event-driven simulation environment. Once the programming model is designed, it can be reused in other distributed applications.

The presented approach contributes to the reduction in lines of code required in implementing different types of distributed applications, with an average of 20% and by up to 45%, by using the implementation of the presented model, compared to the regular case where writing the entire simulation application from scratch is required. This depends on the specific goals of the application which needs to be simulated. The presented event-driven simulation model can be integrated very well with the development models for simulating at clock level, presented in [26] and [27].

These approaches reduce dramatically the application specific code which is required to be written and they improve the overall effort of the development process in simulating distributed applications, with an average of 40% and by up to 85% compared to the already described approach.

Chapter 6. Error Handling in CPS Applications Implemented using Goal-Oriented Approach

The command, communication and control aspects in CPSs are encapsulated into dedicated hardware devices nodes, DM nodes, shared among the subsystems which are acting over the same environment region. These physical nodes are able to accomplish several parallel asynchronous requests which are posed in different subsystems contexts.

The DM nodes must be robust, reliable and to provide error and failure detection mechanisms. Therefore, this type of node must be dynamic reconfigurable and capable to take the correct decisions, during a reasonable time frame.

In order to achieve these design constraints, a first research on the usage of DM device management based on LP systems was made by the author of this thesis, inside the research team, and presented in [88]. The case study was represented by a CPS application for monitoring a gas distribution network. The LP equations can be used to express the DM goals and can be evaluated for finding the optimal solution [95].

Along with the computational aspect, an LP based approach can be used for solving error detection and dynamical self-reconfigurable aspects within CPS applications design and implementation. This is the topic of this chapter and was also discussed in details in [96].

6.1 The Proposed Methodology

The DM nodes are characterized by computation and decisions capabilities at each of these logical layers. Their behavior is similar to the one of beacon nodes, they control the actions taken inside their managed network region.

A common situation met in a CPS application is when several tasks are submitted asynchronously from different CPS subsystems and are targeting a particular network region. The corresponding DM node for that region must have the ability to compute the region's task as being a sum of the incoming tasks from the upper logical layers. Also, the region's task must be determined by taking into account the application specific constraints, with respect to the posed tasks. The resulting region task is the optimal one, with respect to all the requests coming from the upper logical layers. A suitable approach for finding the optimal solution resides in using an LP system. A LP approach was already discussed in [20], with respect to CPS dynamical control aspect, however without taking into consideration the error handling aspect.

The *Perimeters* are the first logical tailoring level, which corresponds to the physical nodes. Each *Perimeter* is managed by a corresponding DM.

The *Zones* are the next logical tailoring level of a CPS network, upper from the *Perimeters*. They group inner zones, perimeters and physical nodes, all

managed by the corresponding DM. The LP system associated to the DM node of each *Zone* must calculate the requests. These requests will be later used as input values for solving the LP systems corresponding to every inner *Zone* and *Perimeter*.

If the LP system for an inner *Zone* or *Perimeter* fails to provide a valid solution, the DM of the upper *Zone* must relax the constraints and adapt the LP system accordingly.

The *Area* tailoring level represents the CPS subsystem as a whole and consists of all the *Zones*, *Perimeters* and physical nodes involved in the CPS sub-network composition. The goal at CPS subsystem level is translated into *Area* tasks. Each new task is used as input in the LP system of the *Area* DM. The solutions are then used as requirements sent to the controlled regions and physical nodes.

The LP system used at the *Area* level consists of constraints related only to the CPS subsystem context whose requirements are described by the *Area* DM, unlike the other LP systems already presented at the lower logical levels.

CPS network lifecycle encounters situations which bring parts of the CPS network near to their limitations based on regular application functionality. These situations represent application specific special cases which need to be handled accordingly, and not network or application errors.

The scenarios corresponding to such situations illustrate the general case in the inter-dependency context when an action taken in a CPS subsystem influences the decisions already taken in the other CPS subsystems, by determining them to reevaluate their current tasks. In the CPS application perspective, such a situation is considered an exception which appears often in CPS networks. Subchapter 6.2.1 presents in detail such a special situation and the step by step handling methodology.

Subchapter 6.2.2 discusses in details the node access failure for the main artifacts of the CPS network and the way of handling such malfunctions.

After several unsuccessful call attempts to a node, the managing DM node can detect a sensor or valve node access failure. The literature includes various approaches for detecting a node access failure. However, the author of this thesis is focused on handling at DM level such situations.

The approach involves first the dynamical removing of all the terms corresponding to that sensor or valve and then updating the LP relations accordingly, while starting from the LP system correlated to a specific DM.

The next step is the evaluation of the resulting LP relations and verification of their correctness, meaning the definition completeness for a mathematical relation. More specifically, it is analyzed if every relation still contains a valid left and right hand side.

If there is found a relation containing an empty left or right hand side, but containing terms on the other one, the approach carries on with identification of the nodes feeding those terms. On the next step, the procedure identifies all the terms used in the LP system and fed by those nodes and proceeds with removing all the identified ones from the LP system.

Furthermore, the procedure investigates again the remaining relations and continues with the already described removal and update parts in a loop manner. The algorithm ends when there is no relation remaining in the LP system or the remaining ones are all mathematically correct. If any of the removed terms belong to an external DM located in the proximity of the current one, the same procedure must be considered against the LP system used by that DM.

The reason behind this loop procedure is the fact that the remaining relations from an LP system must be mathematically correct. Otherwise, these

relations cannot be kept in the LP system. Due to the risk that the remaining relations do not describe the entire set of constraints related to a particular node having one of its attributes ignored, the procedure cannot remove only the incorrect detected relations. In such a case, the risk of introducing errors would persist.

6.2 Case Study: An Aircraft Fuel Management System

For a better understand of the topic, the author of this thesis uses the design of an aircraft fuel management system, with focus on the handling aspects with respect to the application specific special cases and node failures. This application of managing the fuel transfer inside of an aircraft vehicle was introduced in [97] and continued in [98]. Inside the research team, for this application, the first approach from a CPS network perspective was made in [20].

This CPS application consists of two subsystems, the *Center of Gravity (CG) Management* and *Fuel Management* for feeding the engines. Each of them has its own objective, as the naming also suggests. The two subsystems share and act over the same physical network nodes (fuel tanks, sensors, actuators and pipes).

For organizing the network, the hierarchical logical layering into *Areas*, *Zones* and *Perimeters*, which can be shared between subsystems, can be applied, as shown in Figure 35. The *CG* subsystem is characterized by dynamically reconfigurable zones, organized based on the data received from a gravity sensor.

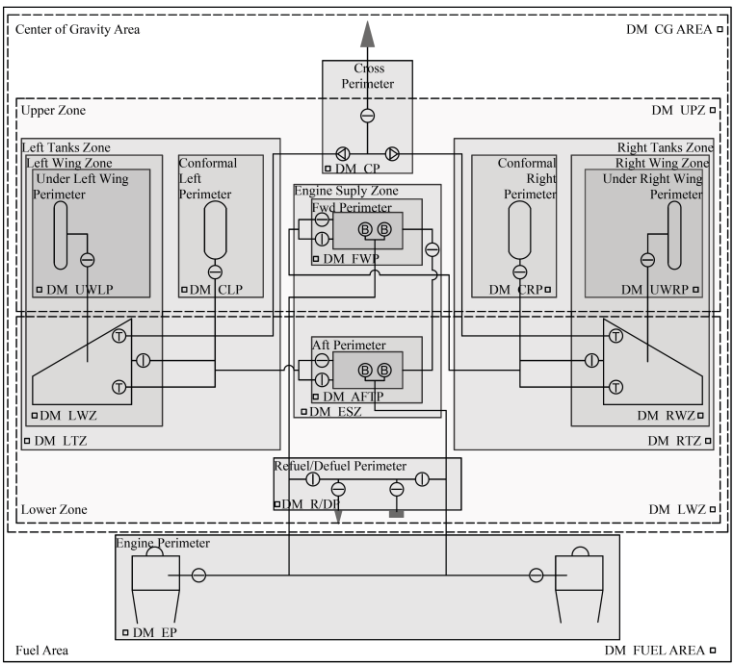

Figure 35 Logical tailoring in *Fuel Management* and *Center of Gravity Management* CPS subsystems of a typical military aircraft top-level fuel system [20]

In the given case of the aircraft fuel management study, the task of feeding the engines has a higher priority compared to the one of computing the center of gravity. Therefore, this prioritization schema is defined for every *Perimeter* presented in Figure 35. Each *Perimeter* manages the internal tank and its corresponding valve connected through a pipe. As a specific example, equation (11) illustrates the LP system for the *Conformal Left Perimeter*.

In (11), *currentDeb* represents the debit transferred with respect to the targeted subsystem or physical value. *dir* variable is the sense of the fuel flowing through the valve and pipe. *dir* can take one of the following values:

- *-1*: the fuel is transferred outside the region
- *0*: no transfer is performed
- *1*: the fuel is transferred inside the region.

$$\begin{cases} dir_{CLP} = dir_{CLP\_Fuel} + \left\| dir_{CLP\_Fuel} \right| - 1 \left| \cdot dir_{CLP\_CG} \right. \\ currentDeb_{CLP\_Fuel} + currentDeb_{CLP\_CG} = currentDeb_{CLP} \\ currentDeb_{CLP} \le \max Deb_{CLP} \\ dir_{CLP} * currentDeb_{CLP} + currentCap_{CLP} \le \max Cap_{CLP} \\ dir_{CLP} * currentDeb_{CLP} + currentCap_{CLP} \ge \min Cap_{CLP} \end{cases} \quad (11)$$

Considering the resulting LP solutions, the debit requested at *Perimeter* level in the *CG* subsystem accepted and added to the actual fuel debit flow through the managed pipe or valve or it can be declined.

The *C*ross *Perimeter* is a special case among the *Perimeters* presented in Figure 35. This *Perimeter* is characterized by two internal valves, named *Cross Left Valve (CLV)* and *Cross Right Valve (CRV)*. Based on the *Cross Perimeter* level sense and debit (dir_{Cross} and $currentDeb_{Cross}$, respectively), these valves must synchronize their senses and debits. The *Cross Perimeter* has no fuel storage functionality. The related LP system is presented in (12) and reflects these particular requirements.

$$\begin{cases} currentDeb_{Cross} = currentDeb_{Cross\_Fuel} + currentDeb_{Cross\_CG} \\ currentDeb_{Cross} \le \max Deb_{Cross} \\ dir_{Cross} = dir_{Cross\_Fuel} + \left\| dir_{Cross\_Fuel} \right| - 1 \left| \cdot dir_{Cross\_CG} \right. \\ currentDeb_{Cross} = currentDeb_{CLV} = currentDeb_{CRV} \end{cases} \quad (12)$$

The parameters $dir_{Cross}, dir_{Fuel}, and dir_{CG} \in \{-1,0,1\}$, as they represent the sense of the fuel flowing through the valve and pipe, already presented.

The possible values for dir_{Cross} determine the sense values for the internal managed valves, as shown in (13).

$$\begin{cases} dir_{Cross} = 0 \rightarrow dir_{CLV} = dir_{CRV} = -1 \\ dir_{Cross} = -1 \rightarrow dir_{CLV} = -1 \wedge dir_{CRV} = 1 \\ dir_{Cross} = 1 \rightarrow dir_{CLV} = 1 \wedge dir_{CRV} = -1 \end{cases} \quad (13)$$

Figure 36 presents the *Left Tanks Zone* that manages the *Left Wing Zone*, the *Conformal Left Perimeter* and their interconnected pipes. The *CLV* is the pipe connection between the *Left Tanks Zone* and the *Cross Perimeter*. The *LB* acronym represents the pipe connection with the *Engine Supply Zone*.

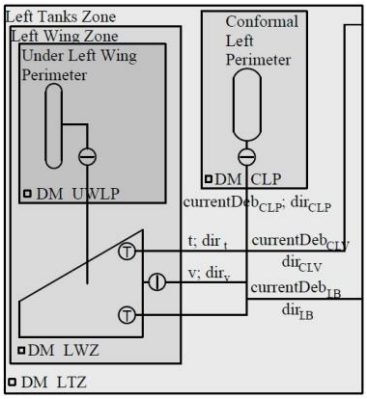

Figure 36 Tailoring the *Left Tanks Zone* illustrating the physical fuel debits and directions flowing through internal pipes and valves [20]

The LP system used by DM LTZ to determine the request values like (t, dir_t), (v, dir_v), ($currentDeb_{CLP}$, dir_{CLP}) for the inner regions is presented in (14).

$$
\begin{cases}
dir_t \cdot t + dir_v \cdot v = dir_{LB} \cdot currentDeb_{LB} + dir_{CLV} \cdot currentDeb_{CLV} + dir_{CLP} \cdot currentDeb_{CLP} \\
dir_{LB} = dir_{LB\_Fuel} + \left\| dir_{LB\_Fuel} \right| - 1 \left| \cdot dir_{LB\_CG} \right. \\
dir_{CLV} = dir_{CLV\_Fuel} + \left\| dir_{CLV\_Fuel} \right| - 1 \left| \cdot dir_{CLV\_CG} \right. \\
dir_{CLP} = dir_{CLP\_Fuel} + \left\| dir_{CLP\_Fuel} \right| - 1 \left| \cdot dir_{CLP\_CG} \right. \\
dir_t = dir_{t\_Fuel} + \left\| dir_{t\_Fuel} \right| - 1 \left| \cdot dir_{t\_CG} \right. \\
dir_v = dir_{v\_Fuel} + \left\| dir_{v\_Fuel} \right| - 1 \left| \cdot dir_{v\_CG} \right. \\
currentDeb_{LB\_Fuel} + currentDeb_{LB\_CG} = currentDeb_{LB} \\
currentDeb_{CLV\_Fuel} + currentDeb_{CLV\_CG} = currentDeb_{CLV} \\
currentDeb_{CLP\_Fuel} + currentDeb_{CLP\_CG} = currentDeb_{CLP} \\
currentDeb_{t\_Fuel} + currentDeb_{t\_CG} = currentDeb_t \\
currentDeb_{v\_Fuel} + currentDeb_{v\_CG} = currentDeb_v \\
currentDeb_{LB} \le \max Deb_{LB} \\
currentDeb_{CLV} \le \max Deb_{CLV} \\
currentDeb_{CLP} \le \max Deb_{CLP} \\
currentDeb_t \le \max Deb_t \\
currentDeb_v \le \max Deb_v \\
currentCap_{LTZ} = currentCap_{LWZ} + currentCap_{CLP} \\
\max Cap_{LTZ} = \max Cap_{LWZ} + \max Cap_{CLP} \\
currentCap_{LTZ} + dir_t \cdot t + dir_v \cdot v \le \max Cap_{LTZ} + dir_{CLP} \cdot currentDeb_{CLP} \\
currentCap_{LTZ} + dir_t \cdot t + dir_v \cdot v \ge \min Cap_{LTZ} + dir_{CLP} \cdot currentDeb_{CLP} \\
0 \le currentCap_{LTZ} \le \max Cap_{LTZ} \\
0 \le currentCap_{LWZ} \le \max Cap_{LWZ} \\
0 \le currentCap_{CLP} \le \max Cap_{CLP} \\
0 \le t \le \max Deb_t \\
0 \le v \le \max Deb_V
\end{cases}
\tag{14}
$$

where the fuel senses can take values as follows: $dir_v \in \{0;1\}$,

$$dir_{LB\_Fuel}, dir_{LB\_CG}, dir_{LB}, dir_t \in \{-1;0\}$$

$$dir_{CLV\_Fuel}, dir_{CLV\_CG}, dir_{CLV}, dir_{CLP} \in \{-1;0;1\}$$

The LP system corresponding to the *Right Tanks Zone* can be constructed using the LP presented in (14), by replacing the *L* (Left) variables with the corresponding *R* (Right) variables.

In this case study, the LP related to the *Fuel Area*, with respect to the *Fuel Management* subsystem, consists of context related relations and constraints, as presented in (15). The relations refer to debits and sense for the fuel passing from one region to another. The flowing is made using a shared pipe controlled by the *DM Fuel Area*. At *Area* level, these relations ensure a proper adaptation to all the other affected regions, when the fuel debit and sense are modified by one region.

$$\begin{cases} dir_{CLV\_Fuel} = -dir_{Cross\_Fuel} \\ dir_{CRV\_Fuel} = dir_{Cross\_Fuel} \\ currentDeb_{CLV\_Fuel} = currentDeb_{Cross\_Fuel} = currentDeb_{CRV\_Fuel} \\ dir_{LB\_Fuel} = -dir_{AftSuply\_Fuel} \\ currentDeb_{LB\_Fuel} = currentDeb_{AftSuply\_Fuel} \\ dir_{RB\_Fuel} = -dir_{FwdSuply\_Fuel} \\ currentDeb_{RB\_Fuel} = currentDeb_{FwdSuply\_Fuel} \\ dir_{FwdB\_Fuel} = -dir_{LEngine} \\ currentDeb_{FwdB\_Fuel} = currentDeb_{LEngine} \\ dir_{AftB\_Fuel} = -dir_{REngine} \\ currentDeb_{AftB\_Fuel} = currentDeb_{REngine} \end{cases} \quad (15)$$

Similar to *Fuel Management* subsystem, the *CG* subsystem expresses relations between shared pipes. The *CG* subsystem also holds relations between regions capacities for determining the fuel quantity available for being moved, from one region to another based on its gravity sensor output. The gravity sensor value determines the configuration of the regions which have to be taken into account. For every possible configuration, a LP system is defined and activated at *CG Area* level. Due to the paper size limitation, we present below the LP system corresponding to the Right → Left fuel transfer for adapting the airplane center of gravity, when the airplane is turning left (16):

$$\begin{cases} dir_{CLV\_CG} = -dir_{Cross\_CG} \\ dir_{CRV\_CG} = dir_{Cross\_CG} \\ currentDeb_{CLV\_CG} = currentDeb_{Cross\_CG} = currentDeb_{CRV\_CG} \\ dir_{LB\_CG} = -dir_{AftSuply\_CG} \\ currentDeb_{LB\_CG} = currentDeb_{AftSuply\_CG} \\ dir_{RB\_CG} = -dir_{FwdSuply\_CG} \\ currentDeb_{RB\_CG} = currentDeb_{FwdSuply\_CG} \\ boostTransf_{LTZ} = period * dir_{AftSuply\_CG} * currentDeb_{AftSuply\_CG} \\ boostTransf_{RTZ} = period * dir_{FwdSuply\_CG} * currentDeb_{FwdSuply\_CG} \\ crossTransf = period * dir_{Cross\_CG} * currentDeb_{Cross\_CG} \\ currentCap_{LTZ} + boostTransf_{LTZ} = currentCap_{RTZ} + boostTransf_{RTZ} + crossTransf \end{cases} \quad (16)$$

In (16), *period* represents the time frame and it is calculated along with the debits based on the airplane left turning angle value. The fuel transfer senses are determined based on the turning sense received also from the gravity sensor.

6.2.1 Handling Application Specific Special Cases

In this subchapter, the author of this thesis considers a scenario which presents the situation when the minimum level of fuel is reached inside of a specific fuel tank. The goal is to illustrate the manner of using LP systems presented in the previous subchapter.

In the *Fuel Management* subsystem context, a fuel transfer is ongoing from *Left Tanks Zone* to *Engine Supply Zone*. Considering (14), $currentCap_{CLP}$ and dir_{CLP} are set to 0 and the transfer is accomplished from the *Left Wing Zone*. At a certain moment of time, the transfer to *Engine Supply Zone* is not allowed anymore due to the minimum fuel level reached inside both fuel tanks managed by the *Left Wing Zone*.

At this moment, the resulting solutions from (14) redirect the requested fuel to be transferred from the *Conformal Left Perimeter*. This implies an internal request to be sent to *DM CLP* in the context of the *Fuel Management* subsystem. The request consists in $desiredDeb_{CLP\_Fuel}$ to be equal to the requested quantity and dir_{CLP_Fuel}=-1. *DM CLP* uses relations (11) to find a valid solution to the received request. However, considering $currentCap_{CLP}$ close as value to $minCap_{CLP}$, the last relation from (11) fails. As a result, the request is denied by *DM CLP*.

DM CLP is unable to satisfy the addressed request, therefore *DM LTZ* reconsiders its LP system in (14) and fails also to provide a valid solution. *DM LTZ* sends to *DM Fuel Area* a request for it to provide the fuel quantity, using the *CLV* pipe ($currentDeb_{CLV\_Fuel}$ is equal to the requested quantity and dir_{CLV_Fuel} = 1).

DM Fuel Area uses relations (15) to determine the requests for the other involved regions. In this case, based on the results obtained from solving (15), it requests the *Cross Perimeter* to set the $currentDeb_{Cross\_Fuel}$ to the requested value and to set its internal valves according to dir_{Cross_Fuel} =1. The internal cross valves are set accordingly, based on the solutions from (12) and (13).

The *DM CP* can encounter the case when the calculated $currentDeb_{Cross}$ from its LP system presented in (12) is greater than its $maxDeb_{Cross}$. Therefore, *DM CP* cancels the asynchronous task set in the *CG* context ($currentDeb_{Cross\_CG}$ = 0 and dir_{Cross_CG} = 0). This allows *DM CG Area* to recalculate the solutions for equations (6), leading to no valid solutions and fuel transfer cancelation for all the other regions involved in the *CG* context.

At the same time, *DM Fuel Area* commands the *Right Tanks Zone* to provide fuel (dir_{CRV_Fuel} = -1 and $currentDeb_{CRV\_Fuel}$ set to the requested fuel level for the *Left Tanks Zone*).

A similar subroutine flow schema applies to maximum level reached inside a tank. In this situation, the fuel directions on all the pipes and valves considered are inverted comparing to the already presented scenario.

6.2.2 Handling CPS Equipment Failures

Using as example support the presented case study, the author of this thesis discusses the node failure handling procedure in the situation when the sensor located on the tank managed by *DM CLP* fails. The corresponding LP system for the DM presented in (11) must to be updated by removing the $currentCap_{CLP}$, $minCap_{CLP}$ and $maxCap_{CLP}$ terms. The next step implies verification of the resulting relations. As a result of the verification, the last two relations do not have any right hand side terms anymore. Next, the procedure continues with removal of $currentDeb_{CLP}$, dir_{CLP}, $currentDeb_{CLP\_Fuel}$, dir_{CLP_Fuel}, $currentDeb_{CLP\_CG}$ and dir_{CLP_CG} terms. Removing the terms presented above leads to no relation being available for this LP system.

The next step is represented by *DM LTZ* announcing to update its internal LP system (14), as a consequence of removal of the $currentDeb_{CLP}$ and dir_{CLP} terms from the LP system of *DM CLP*. Therefore, the updated LP equations for *DM LTZ* are presented in (17).

A DM node access failure can be detected if a DM located in the immediate upper logical level performs calls and does not receive an answer after a certain period of time.

Such detection has as result the disposal of the inner *Perimeter* or *Zone* and the update of the upper logical level related LP system, by adding all the inner relations to it. This implies transfer of the command, communication and control over all the physical existing nodes located inside of the dismissed *Perimeter* or *Zone* to the upper logical level DM.

$$
\begin{cases}
dir_{LB} = dir_{LB\_Fuel} + \left\| dir_{LB\_Fuel} \right| - 1 \left| \cdot dir_{LB\_CG} \right. \\
currentDeb_{LB\_Fuel} + currentDeb_{LB\_CG} = currentDeb_{LB} \\
currentDeb_{LB} \le \max Deb_{LB} \\
dir_{CLV} = dir_{CLV\_Fuel} + \left\| dir_{CLV\_Fuel} \right| - 1 \left| \cdot dir_{CLV\_CG} \right. \\
currentDeb_{CLV\_Fuel} + currentDeb_{CLV\_CG} = currentDeb_{CLV} \\
currentDeb_{CLV} \le \max Deb_{CLV} \\
dir_t \cdot t + dir_v \cdot v = dir_{LB} \cdot currentDeb_{LB} + dir_{CLV} \cdot currentDeb_{CLV} \\
currentCap_{LTZ} = currentCap_{LWZ} \\
\max Cap_{LTZ} = \max Cap_{LWZ} \\
currentCap_{LTZ} + dir_t \cdot t + dir_v \cdot v \le \max Cap_{LTZ} \\
currentCap_{LTZ} + dir_t \cdot t + dir_v \cdot v \ge \min Cap_{LTZ} \\
0 \le currentCap_{LTZ} \le \max Cap_{LTZ} \\
0 \le currentCap_{LWZ} \le \max Cap_{LWZ} \\
0 \le t \le \max Deb_T \\
0 \le v \le \max Deb_V
\end{cases}
\tag{17}
$$

As a physical constraint, the physical nodes transferred have to be in the communication range of the new DM which will be responsible to manage them. Otherwise, a reorganization of the entire network logical levels is required, with respect to the subsystem context not being able to fulfill this network communication requirement.

In the presented case study, *DM CLP* failure leads to updating the LP related to *DM LTZ* (14), by importing all the *Conformal Left Perimeter* related relations (11),

with respect to all the subsystems contexts in which *DM CLP* resided earlier. Equation (18) presents the updated LP system in case of the *Left Tanks Zone* region.

$$\begin{cases} dir_{LB} = dir_{LB\_Fuel} + \left\| dir_{LB\_Fuel} \right| - 1 \left| \cdot dir_{LB\_CG} \right. \\ currentDeb_{LB\_Fuel} + currentDeb_{LB\_CG} = currentDeb_{LB} \\ currentDeb_{LB} \le \max Deb_{LB} \\ dir_{CLV} = dir_{CLV\_Fuel} + \left\| dir_{CLV\_Fuel} \right| - 1 \left| \cdot dir_{CLV\_CG} \right. \\ currentDeb_{CLV\_Fuel} + currentDeb_{CLV\_CG} = currentDeb_{CLV} \\ currentDeb_{CLV} \le \max Deb_{CLV} \\ dir_t \cdot t + dir_v \cdot v = dir_{LB} \cdot currentDeb_{LB} + dir_{CLV} \cdot currentDeb_{CLV} \\ currentCap_{LTZ} = currentCap_{LWZ} + currentCap_{CLP} \\ \max Cap_{LTZ} = \max Cap_{LWZ} + \max Cap_{CLP} \\ currentCap_{LTZ} + dir_t \cdot t + dir_v \cdot v \le \max Cap_{LTZ} + dir_{CLP} \cdot currentDeb_{CLP} \\ currentCap_{LTZ} + dir_t \cdot t + dir_v \cdot v \ge \min Cap_{LTZ} + dir_{CLP} \cdot currentDeb_{CLP} \\ 0 \le currentCap_{LTZ} \le \max Cap_{LTZ} \\ 0 \le currentCap_{LWZ} \le \max Cap_{LWZ} \\ 0 \le currentCap_{CLP} \le \max Cap_{CLP} \\ 0 \le t \le \max Deb_T \\ 0 \le v \le \max Deb_V \\ dir_{CLP} = dir_{CLP\_Fuel} + \left\| dir_{CLP\_Fuel} \right| - 1 \left| \cdot dir_{CLP\_CG} \right. \\ currentDeb_{CLP\_Fuel} + currentDeb_{CLP\_CG} = currentDeb_{CLP} \\ currentDeb_{CLP} \le \max Deb_{CLP} \\ dir_{CLP} * currentDeb_{CLP} + currentCap_{CLP} \le \max Cap_{CLP} \\ dir_{CLP} * currentDeb_{CLP} + currentCap_{CLP} \ge \min Cap_{CLP} \end{cases} \quad (18)$$

6.3 Summary

The author of this thesis discusses within this chapter a linear programming based approach for CPS application dynamics, asynchronous tasks fulfillment, application specific exceptions and nodes failure handlings. Beside regular behavior, according to specifications, in a CPS application lifecycle various exceptional situations can appear. The approach presented here allows the developer to deal with these special cases, along with specifying, design, verification and validation of CPS applications. The benefits for this approach have been showed with a case study representative for CPS applications.

There are particular cases when linear programming is not able to retrieve valid solutions. In such situations the LP system is considered as being too restrictive. To overcome these situations, the author of this thesis considers approximation of the valid solution based on changing the domain range of certain variables, or by updating the control equation from the LP system that needs to be optimized. Such an approach promises to be able to obtain the optimal solutions.

A possible drawback of this approach is the fact that all physical nodes require to be static. As future work, the author of this thesis intends to develop a dynamical LP system based mechanism, able to deal with mobile nodes. This approach promises to provide a suitable solution in case of complex hybrid CPS applications.

Chapter 7. Conclusions, Contributions, Publications and Perspectives

7.1 Conclusions

The methodology for specification, design and validation of CPS applications developed by the research team the author of this thesis is part of is intended to ease the work for designers of CPS networks. This visual methodology is based on MDA approach and the specification is started from a high level of abstraction, the CIM, goes to a more specific level, the PIM, and finally is implemented on the physical network. The objectives for the CPS application are expressed in a goal-oriented manner, while the entire network is tailored into logical layers. The modeling of the CPS applications is done with the help of UML stereotypes, defined for both hardware and software levels. The specification of the CPS application at PIM level implies creation of the UML hardware and software models for network topology and network behavior, respectively.

The main objective of this thesis was to be able to validate the models before deployment on physical network. The novelty introduced by this thesis is the rigorous and complex testing and validation process, the extended verifications posed on both static and dynamic aspects in a CPS network, at component level, node level and network level, the focus being on errors identification as well as errors handling mechanisms.

The author of this thesis used rigorous specification for identifying the lack of requirements in static properties of CPS applications, for units, nodes and at network level. The author also developed simulation models, which allow a faster and more accurate verification by simulation. The author has defined clock cycle level tests, for units testing, node level tests and workflow tests, at network level. The author of this thesis has defined a methodology to handle particular cases which can appear during a CPS network lifecycle.

The testing and verification of the units used in the CPS application have formed the clock cycle level tests, integrating the units led to node level tests while the entire application's verification implied workflow tests, at network level. The author of this thesis has considered also the particular cases which can appear during a CPS network lifecycle, which are not seen as errors but require an appropriate handling.

Validation is a very important phase when developing any programming model for CPS applications. For the case of the MDA approach discussed here, a tested and verified CPS network can be deployed on the physical environment with the certainty that the specifications errors and a large part of the behavioral errors have been already removed. This leads to the increased credibility for the programming methodology.

7.2 Contributions

The contributions of the author of this thesis regarding CPS applications validation research have been discussed in Chapters 2-6. All these proposals regarding CPS applications have been gathered into 17 articles, published in ISI and BDI journals, presented at international ISI and IEEE conferences and published in the proceedings and presented at theme specific workshops. The contributions with each article are presented in subchapter 7.3.

The main contributions in CPS applications validation follow the initial research objectives, described in the proposal for the theme of research, presented in September 2011. These objectives have also been summarized in subchapter 1.3.

The main contributions of the author of this thesis regarding CPS applications validation are:

- At theoretical level
  - A comprehensive study and systematization of publications in simulation frameworks for sensor networks and modeling and validation of CPS applications
  - A methodology for testing and verifying CPS applications, at each level for the CPS design, for both static and dynamic aspects
  - A rigorous specification of the static properties defined for units, nodes and network communication capabilities. Based on the full specification at each of these levels, validation can be performed using already defined tools.
  - The definition of simulation models for CPS applications composed of PSoC devices. One or another of these simulation models can be applied, depending on the types of PSoC devices used (PSoC devices running at the same clock frequency, PSoC devices running at different clock frequencies or PSoC devices with different clock shift).
  - Validation approaches at each level of network components: testing and verification of the units using in composing the nodes in the CPS applications (clock cycle level tests), units integration and collaboration inside CPS networks (node level tests) and workflow tests (network level tests).
  - The definition of a handling methodology for event-driven scenarios in simulating CPS applications. This methodology contains both an event-driven simulation model and an event-driven programming model, used in testing the dynamic aspects of the CPS application.
  - The definition of a methodology for handling special cases which can appear during the lifecycle of a CPS application, which are not seen as errors. Also, a handling manner in case equipment failures for CPS networks has been defined.

- At practical level
  - The verification methodology has been successfully applied to CPS networks composed of sensors, actuators, communication units and decision nodes.
  - The methodology has been successfully applied to CPS networks of variable size.
  - The methodology has been successfully applied to CPS applications with different degrees of difficulty regarding requirements.

7.3 Publications

This subchapter presents the articles that were published during the PhD research of the author of this thesis. The articles have been published during 2009-2012 in several ISI conferences, IEEE international conferences, BDI journals and POSDRU workshops. The papers are presented below in significance order.

7.3.1 Article Published in ISI Journal

1. G. Magureanu, M. Gavrilescu, D. Pescaru, "Validation of Static Properties in UML Models for Cyber Physical Systems", in Journal of Zhejiang University Science C, Impact factor = 0.297 (2012), ISSN 1869-1951, doi: 10.1631/jzus.C1200263.

7.3.2 Articles Published in ISI Proceedings

1. D. Pescaru, C. Istin, F. Naghiu, M. Gavrilescu, D. Curiac, "Scalable metric for coverage evaluation in video-based wireless sensor networks", in Proceedings of the 5th International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, Romania, ISBN 978-1-4244-4478-6, May 2009, pp. 323 - 328.

2. M. Gavrilescu, G. Magureanu, D. Pescaru, I. Jian, "UML Software Models for Cyber Physical System Applications", in the 20th Telecommunications Forum (TELFOR), Belgrad, Serbia, November 2012.

3. (Pending indexing) - M. Gavrilescu, G. Magureanu, D. Pescaru, "CPS Design Using Model Driven Architecture Approach: Aircraft Fuel Management System Case Study", in 2012 Third International Conference on Theoretical and Mathematical Foundations of Computer Science (ICTMF), Indonesia, in Lecture Notes in Information Technology, ISSN 2070-1918, 2012.

7.3.3 Articles Published in IEEE Proceedings

1. M. Gavrilescu, G. Magureanu, D. Pescaru, A. Doboli, "A Simulation Framework for PSoC Based Cyber Physical Systems", in Proceedings of the IEEE International Joint Conferences on Computational Cybernetics and Technical Informatics ICCC-CONTI 2010, Timisoara, Romania, pp. 137 – 142, May 2010, doi: 10.1109/ICCCYB.2010.5491313.

2. G. Magureanu, M. Gavrilescu, D. Pescaru, A. Doboli, " Towards UML Modeling of Cyber-Physical Systems: A Case Study for Gas Distribution", in Proceedings of the in 8th IEEE International Symposium on Intelligent Systems and

Informatics SISY 2010, Subotica, Serbia, pp. 471 – 476, September 2010, doi: 10.1109/SISY.2010.5647314.

3. M. Gavrilescu, G. Magureanu, D. Pescaru, A. Doboli, "Accurate Modeling of Physical Time in Asynchronous Embedded Sensing Networks", in Proceedings of the in 8th IEEE International Symposium on Intelligent Systems and Informatics SISY 2010, Subotica, Serbia, pp. 477 – 482, September 2010, doi: 10.1109/SISY.2010.5647308.

4. G. Magureanu, M. Gavrilescu, D. Pescaru, A. Doboli, "UML Support for Optimizing the Goals of Distributed Control in Traffic Management Applications", in International Workshop on Robotic and Sensors Environments ROSE 2010, Pheonix, Arizona, USA, pp.1 - 6 October 2010, doi: 10.1109/ROSE.2010.5675288.

5. G. Magureanu, M. Gavrilescu, I. Tal, A. Toma, D. Pescaru, I. Jian, "Generating OMNeT++ Specifications from UML Models for PSoC Distributed Applications", in Proceedings of the 6th International Symposium on Applied Computational Intelligence and Informatics SACI 2011, Timisoara, Romania, pp. 85 – 90, May 2011, doi: 10.1109/SACI.2011.5872977.

6. G. Magureanu, M. Gavrilescu, D. Pescaru, I. Jian, "UML Profile for Cyber-Physical System Wireless Communication Specification", in Proceedings of the 7th International Symposium on Applied Computational Intelligence and Informatics SACI 2012, Timisoara, Romania, pp. 383 – 388, May 2012, doi: 10.1109/SACI.2012.6250034.

7. M. Gavrilescu, G. Magureanu, D. Pescaru, "Error Handling using Linear Programming in CPS Applications: Aircraft Fuel Management System Case Study", in 9th International Conference on Computational Cybernetics ICCC 2013, Tihany, Hungary, July 2013.

7.3.4 Articles Published in BDI Journals

1. M. Gavrilescu, G. Magureanu, D. Pescaru, A. Doboli, "Time Models for PSoC Based Cyber Physical Systems Simulation", in Scientific Bulletin of "Politehnica" University of Timisoara, Transactions on Automatic Control and Computer Science BS-UPT TACCS, Volume 56 (70) No. 1, pp. 27-34, March 2011.

2. M. Gavrilescu, G. Magureanu, D. Pescaru, I. Jian, "Handling Event-Driven Scenarios in CPS Application Simulations", in Carpathian Journal of Electronic and Computer Engineering, Volume 4, No. 1, ISSN 1844 – 9689, October 2011.

7.3.5 Articles Presented in POSDRU Workshops

1. M. Gavrilescu, G. Magureanu, D. Pescaru, I. Jian, "Optimization of Sensor Networks Physical Time Modeling in OMNeT++", in Workshop no. 1 "Cercetari doctorale in domeniul tehnic", Craiova, Romania, February 2011.

2. M. Gavrilescu, "A Survey of Simulation Environments for Applications of Massively Distributed Embedded Systems", in Workshop nr. 1 "Interdisciplinaritatea si Managementul Cercetarii", Timisoara, Romania, November 2011.

3. M. Gavrilescu, "UML Software Models for Cyber Physical System Applications", in Workshop no. 2, "Interdisciplinaritatea si Managementul Cercetarii in Studiile Doctorale", Oradea, Romania, June 2012.

4. M. Gavrilescu, "Towards Verification and Validation of Cyber Physical System Applications", in Workshop no. 3, "Interdisciplinaritatea si Managementul Cercetarii", Pitesti, Romania, May 2013.

7.4 Future Research Perspectives

Future work will be focuses on improving the testing and validation process for CPS applications before deployment on the physical environment, to further reduce the possible behavior errors which can appear. Future work will cover investigation of the approach required so that rigorous specification and validation to be used in verifying dynamic aspects in CPS application.

References

[1] E. Lee, "Cyber Physical Systems: Design Challenges", University of California, Berkeley Technical Report No. UCB/EECS-2008-8, 2008.

[2] G. Magureanu, "Visual Modeling of Cyber Physical Systems", Ed. Politehnica, 2013, Series 14, No. 13, ISSN 2069-8216.

[3] A. Alti, T. Khammaci and A. Smeda, "Integrating Software Architecture Concepts into the MDA Platform with UML Profile", *Journal of Computer Science 3 (10)*, 2007, pp. 793-802.

[4] Object Management Group - UML v. 2.3 Specifications. http://www.omg.org/spec/UML/2.3/, 2013.

[5] L. Kuzniarz, M. Staron, C. Wohlin, "An empirical study on using stereotypes to improve understanding of UML models", in *Proceedings of the International Workshop on Program Comprehension*, pp. 14-23, doi:10.1109/WPC.2004.1311043.

[6] V. Subramanian, M. Gilberti, A. Doboli, "Online adaptation policy design for grid sensor networks with reconfigurable embedded nodes", in *Proceedings of Design, Automation & Test in Europe Conference and Exhibition (DATE)*, Nice, 2009.

[7] M. Wang, V. Subramanian, A. Doboli, D. Curiac, D. Pescaru, C. Istin, "Towards a Model and Specification for Visual Programming of Massively Distributed Embedded Systems", in *IFSA Sensors and Transducers Journal*, ISSN 1726-5479, Vol. 5, March 2009, pp. 69-85.

[8] E. Lee, "Cyber-Physical Systems – Are Computing Foundation Adequate?", Position Paper for *NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, 2006.

[9] E. Lee, "CPS Foundations", in *Proceedings of the 47th Design Automation Conference (DAC),* ACM, June 2010, pp. 737-742.

[10] P. Tabuada, "Cyber-physical systems: Position paper", in *Proceedings of the 2006 National Science Foundation Workshop on Cyber-Physical Systems*, 2006.

[11] R. Gupta, "Programming Models and Methods for SpatioTemporal Actions and Reasoning in Cyber-Physical Systems", Position Paper, in *Proceedings of the 2006 National Science Foundation Workshop on Cyber-Physical Systems*, 2006.

[12] P. Derler, E. Lee, A. Sangiovanni-Vincentelli, "Modeling Cyber-Physical Systems", in *Proceedings of the IEEE (Special Issue on CPS)*, Vol. 100 (1), January 2012, pp. 13-28.

[13] E. Lee, S. Tripakis, "Modal Models in Ptolemy", in *Proceedings of 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)*, October, 2010.

[14] N. Saeedloei, G. Gupta, "A logic-based modeling and verification of CPS", in ACM SIGBED Review – *Work-in-Progress (WiP) Session of the 2nd International Conference on Cyber Physical Systems*, Vol. 8, Issue 2, June 2011, pp. 31-34.

[15] Y. Liu, "Toward a unified object model for cyber-physical systems", in *Proceedings of the 2nd Workshop on Software Engineering for Sensor Networks Applications (SESENA)*, 2011.

[16] G. Magureanu, M. Gavrilescu, I. Tal, A. Toma, D. Pescaru, I. Jian, "Generating OMNeT++ specifications from UML models for PSoC Distributed Applications", in *Proceedings of the 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI 2011)*, Timisoara, Romania, May 2011, pp. 85-90, doi:10.1109/SACI. 2011.5872977.

[17] G. Magureanu, M. Gavrilescu, D. Pescaru, "UML profile for Cyber-Physical System wireless communication specification", in *Proceedings of the 7th International Symposium on Applied Computational Intelligence and Informatics (SACI 2012)*, Timisoara, Romania, May 2012, pp. 383-388, doi:10.1109/SACI.2012.6250034.

[18] M. Gavrilescu, "UML Software Models for Cyber Physical System Applications", in *Workshop no. 2, "Interdisciplinaritatea si managementul cercetarii in studiile doctorale"*, Oradea, Romania, June 2012.

[19] M. Gavrilescu, G. Magureanu, D. Pescaru, "Towards UML software models for Cyber Physical System applications", in *the 20th Telecommunications Forum (TELFOR)*, Belgrad, Serbia, November 2012, pp. 1701-1704, doi:10.1109/TELFOR.2012. 6419554.

[20] M. Gavrilescu, G. Magureanu, D. Pescaru, "CPS Design Using Model Driven Architecture Approach: Aircraft Fuel Management System Case Study", in *2012 Third International Conference on Theoretical and Mathematical Foundations of Computer Science (ICTMF)*, Indonesia, in *Lecture Notes in Information Technology*, ISSN 2070-1918, 2012.

[21] J. Boydens, E. Steegmans, "Model Driven Architecture The next abstraction level in programming", in *European Conferences on the Use of Modern Information and Communication Technologies (ECUMICT)*, March 2004.

[22] N. Zeldovich, A. Yip, F. Dabek, R. Morris, D. Mazieres, F. Kaashoek, "Multiprocessor support for event driven programs", in *Proceedings of the USENIX Annual Technical Conference*, San Antonio, TX, USA, June 2003, pp. 239-252.

[23] V. Subramanian, A. Doboli, "PNet: A Grid type Sensor Network of Reconfigurable Nodes," in *Proceedings of the IEEE International Conference on Distributed Computing Systems Workshops*, Montreal, 2009, pp.7-13.

[24] L. Wang, E. A. Johannessen, P. A. Hammond, Li Cui, S. W. J. Reid, J. M. Cooper, and D. R. S. Cumming, "A Programmable Microsystem Using System-on-Chip for Real-time Biotelemetry," *IEEE Transactions on Biomedical Engineering*, Vol. 52, No. 7, July 2005.

[25] Y. Zhao, Jie Liu, and Edward A. Lee, "A Programming Model for Time-Synchronized Distributed Real-Time Systems", in *Proceedings of the 13th IEEE Real-Time and Embedded Technology and Application Symposium*, RTAS'07, WA, USA, April 2007, pp.259-268.

[26] M. Gavrilescu, G. Magureanu, D. Pescaru, A. Doboli, "A Simulation Framework for PSoC Based Cyber Physical Systems", in *IEEE ICCC-CONTI'10*, Timisoara, Romania, May 2010.

[27] M. Gavrilescu, G. Magureanu, D. Pescaru, A. Doboli, "Accurate Modeling of Physical Time in Asynchronous Embedded Sensing Networks", in *Proceedings of the 8th International Symposium on Intelligent Systems and Informatics SISY*, Subotica, Serbia, September 2010, pp. 477–482.

[28] M. Gavrilescu, G. Magureanu, D. Pescaru, I. Jian, "Optimization of Sensor Networks Physical Time Modeling in OMNeT++", in *CDDT Workshop*, Craiova, Romania, February 2011.

[29] M. Gavrilescu, G. Magureanu, D. Pescaru, A. Doboli, "Time Models for PSoC Based Cyber Physical Systems Simulation", in *Scientific Bulletin of "Politehnica" University of Timisoara, Transactions on Automatic Control and Computer Science BS-UPT TACCS*, Volume 56 (70) No. 1, pp. 27-34, March 2011.

[30] G. Magureanu, M. Gavrilescu, D. Pescaru, A. Doboli, "UML Support for Optimizing the Goals of Distributed Control in Traffic Management Applications", in *International Workshop on Robotic and Sensors Environments (ROSE)*, Pheonix, Arizona, USA, October 2010.

[31] C. Atkinson, T. Kuhne, B. Henderson-Sellers, "To Meta or not to Meta – That is the Question", in *Journal of Object-Oriented Programming*, Vol. 13, No. 8, 2000, pp. 32-35.

[32] MiXiM Project – http://mixim.sourceforge.net/, 2013.

[33] M. Gavrilescu, "A Survey of Simulation Environments for Applications of Massively Distributed Embedded Systems", in *Workshop nr. 1 "Interdisciplinaritatea si Managementul Cercetarii"*, Timisoara, Romania, November 2011.

[34] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, and M. Karir, "ATEMU: A fine-grained sensor network simulator", in *Proceedings of SECON'04, First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[35] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications", in *Proceedings of SenSys'03, First ACM Conference on Embedded Networked Sensor Systems*, 2003.

[36] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, D. Culler, "TinyOS: An operating system for sensor networks", in *Ambient Intelligence*, Springer-Verlag, 2004.

[37] Ben L. Titzer, Daniel K. Lee, and Jens Palsberg, "Avrora: scalable sensor network simulation with precise timing", in *Proceedings of the 4th international symposium on Information processing in sensor networks*, Los Angeles, California, April 2005.

[38] Yi-Ran Sun, Shashi Kumar, and Axel Jantsch, "Simulation and Evaluation for a Network on Chip Architecture Using Ns-2", in *Proceedings of the 20th IEEE Norchip Conference*, 2002.

[39] C++SIM – http://www.c-sim.zcu.cz/, 2013.

[40] H.-Y. Tyan and J. C. Hou. "JavaSim: A component-based compositional network simulation environment", in *Proceedings of Western Simulation Multiconference, CNDS'01*, 2001.

[41] Xiang Zeng , Rajive Bagrodia, and Mario Gerla, "GloMoSim: a library for parallel simulation of large-scale wireless networks", in *Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation*, Banff, Alberta, Canada, May 1998.

[42] H. Dai and R. Han, "TSync: A Lightweight Bidirectional Time Synchronization Service for Wireless Sensor Networks", in *ACM SIGMOBILE Mobile Computing and Communications Review*, Vol. 8, No. 1, Jan. 2004, pp. 125–139.

[43] Distributed Computing Group, "Sinalgo, Simulator for Network Algorithms", http://dcg.ethz.ch/projects/sinalgo, 2007.

[44] P. Sommer, R. Wattenhofer, "Gradient Clock Synchronization in Wireless Sensor Networks", in *the 8th ACM/IEEE IPSN'09*, San Francisco, USA, April 2009.

[45] Open SystemC Initiative – http://www.systemc.org, 2013.

[46] E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio, "A UML 2.0 Profile for SystemC: Toward High-level SoC Design", in *Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT)*, New York, USA, 2005.

[47] E. Riccobene, P. Scandurra, S. Bocchio, A. Rosti, L. Lavazza, L. Mantellini, "SystemC/C-based model-driven design for embedded systems", in *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 8, No. 4, 2009.

[48] P. Andersson, M. Host, "UML and SystemC - A Comparison and Mapping Rules for Automatic Code Generation", in *Forum on Specification and Design Languages (FDL)*, Barcelona, Spain, 2007.

[49] K.D. Nguyen, Z. Sun, P.S. Thiagarajan, "Model-Driven SoC Design Via Executable UML to SystemC", in *Proceedings of the IEEE International Real-time Systems Symposium (RTSS'04)*, Lisbon, Portugal, 2004.

[50] Kai Huang, I. Bacivarov, F. Hugelshofer, L. Thiele, "Scalably distributed SystemC simulation for embedded applications", in *Proceedings of the 3rd International Symposium on Industrial Embedded Systems (SIES)*, Montpellier, France, 2008, pp.271-274.

[51] S. Meftali, A. Dziri, L. Charest, P. Marquet, J.-L. Dekeyser, "SOAP based distributed simulation environment for System-on-Chip (SoC) design", in *Forum on Specification and Design Languages, FDL'05*, Lausanne, Switzerland, December 2005.

[52] M. Damm, J. Moreno, J. Haase, C. Grimm, "Using transaction level modeling techniques for wireless sensor network simulation", in *Proceedings of the*

Design, Automation and Test in Europe (DATE'10), Dresden, Germany, 2010.

[53] Cypress Semiconductor Corporation – http://www.cypress.com, 2013.

[54] I.-K Rhee, J Lee, J. Kim, E. Serpedin, Wu, Y.-C., "Clock Synchronization in Wireless Sensor Networks: an Overview", in *Sensors*, 2009, pp. 56-85.

[55] http://www.omnetpp.org/doc/omnetpp41/manual/usman.html – OMNeT++ User Manual ver. 4.1, 2013.

[56] A. Varga, R. Hornig, "An overview of the OMNeT++ simulation environment", in *Proceedings of the 1st Iinternational Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, Marseille, France, March 2008.

[57] C. Mallanda, A. Suri, V. Kunchakarra, S. S. Iyengar, R. Kannan, A. Durresi, and S. Sastry, "Simulating Wireless Sensor Networks with OMNeT++", *Submitted for Publication to IEEE*, 2005.

[58] Mobility Framework – http://mobility-fw.sourceforge.net/, 2013.

[59] D. Pediaditakis, Y. Tselishchev, A. Boulis, "Performance and Scalability Evaluation of the Castalia Wireless Sensor Network Simulator", in *3rd International ICST Conference on Simulation Tools and Techniques, SIMUTOOLS'2010*, Malaga, Spain, 2010.

[60] I. Baumgart, B. Heep, S. Krause, "OverSim: A Flexible Overlay Network Simulation Framework", in *IEEE Global Internet Symposium*, Anchorage, AK, May 2007, pp. 79-84.

[61] INET Framework – http://inet.omnetpp.org/, 2013.

[62] Mixnet Project – http://sourceforge.net/apps/trac/mixim/wiki/mixnet, 2013.

[63] D. B. Aredo, I. Traore, K. Stolen, "Towards formalization of UML class structure in PVS", Research Report 272, Department of Informatics, University of Oslo, Norway, August 1999.

[64] R. France, A. Evans, K. Lano, B. Rumpe, "The UML as a formal modeling notation", in *Computer Standard Interfaces,* Vol. 19, 1998, pp. 325–334.

[65] A. Bhutto, D.M. Akbar Hussain, "Formal verification of UML profile", in *Australian Journal of Basic and Applied Sciences*, 5(6), 2011, pp. 1594–1598.

[66] Object Management Group - Object Constraint Language Specifications. http://www.omg.org/spec/OCL/2.0/, 2013.

[67] A. Hamie, F. Civello, J. Howse, S. Kent, R. Mitchell, "Reflections on the object constraints language", in *Lecture Notes in Computer Science*, No. 1618, pp. 137-145, doi:10.1007/978-3-540-48480-6_13.

[68] J. M. Spivey, "The Z notation: A reference manual", 2nd edition, Prentice-Hall International, Ltd. Hertfordshire, UK, 1992.

[69] L. Martins, "An Empirical Study Using Z and UML for the Requirements Specification of an Information System", in *The Experimental Software Engineering Latin American Workshop*, Uberlândia, Brasil, October 2005, Paper 1.

[70] D. Roe, K. Broda, A. Russo, "Mapping UML models incorporating OCL constraints into Object-Z", Report No. 9/2003, Imperial College, London, UK.

[71] N. Amalio, S. Stepney, F. Polack, "Modular UML semantics: Interpretations in Z based on templates and generics", presented at *the Workshop on Formal Aspects of Component Software*, Pisa, Italy, September 2003.

[72] S. Dupuy, Y. Ledru, M. Chabre-Peccoud, "An overview of RoZ: A tool for integrating UML and Z specifications", in *Proceedings of 12th Conference on Advanced information System Engineering (CAiSE 2000)*, volume 1789 of Lecture Notes in Computer Science, Stockholm, Sweden, June 2000, pp. 417–430, doi:10.1007/3-540- 45140-4_28.

[73] S. C. Dupuy, J. Freire, M. Chabre-Peccoud, Y. Ledru, "Formal and informal specifications: a proposal for a coupling", in *Proceedings of 13th International Conference on Software and Systems Engineering and their Applications*, Paris, France, December 2000.

[74] S. C. Dupuy, L. du Bousquet, "Validation of UML Models Thanks to Z and Lustre", in *Proceedings of the International Symposium of Formal Methods Europe: Formal Methods for Increasing Software Productivity*, Berlin, Germany, March 2001, Springer-Verlag LNCS 2021, London, UK, 2001, pp. 242-258.

[75] Y. Ledru, "Using Jaza to Animate RoZ Specifications of UML Class Diagrams", in *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*, Washington DC, SUA, 2006, pp. 253-262.

[76] N. Halbwachs, F. Lagnier, C. Ratel, "Programming and Verifying Real-Time Systems by Means of Synchronous Data-Flow Programming Language LUSTRE", in *IEEE Trans. Software Engineering,* 1992, Vol. 8, pp. 785 – 793.

[77] M. Saaltink, "The Z/EVES system", in *Proceedings of the The Z Formal Specification Notation: 10th International Conference of Z Users*, London, UK, April 1997, in Lecture Notes in Computer Science, Springer-Verlag, 1997, pp. 72–85, doi:10.1007/BFb0027284.

[78] M. Shroff, R.B. France, "Towards a formalization of UML class structures in Z", in the *Proceedings of the 21st International Computer Software and Applications Conference (COMPSAC '97)*, Washington DC, USA, August 1997, pp. 646–651, doi:10. 1109/CMPSAC.1997.625087.

[79] J. Crow, S. Owre, J. Rushby, N. Shankar, M. Srivas, "A tutorial introduction to PVS", in *Workshop on Industrial Strength Formal Specification Techniques (WIFT '95)*, Florida, SUA, April 1995.

[80] D. B. Aredo, "Formal semantics of UML statecharts in PVS", in *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003)*, Florida, USA, July 2003.

[81] D. B. Aredo, "A framework for semantics of UML sequence diagrams in PVS", in *Journal of Universal Computer Science*, Volume 8, 2002, pp. 674–698, doi:10.3217/jucs-008-07.

[82] B. Jacobs, J. van den Berg, M. Huisman, M. van Berkum, "Reasoning about Java Classes: preliminary report", in *Proceedings of Object-Oriented*

Programming Systems, Languages and Applications, Vancouver, BC, Canada, October 1998, ACM, New York, New York, USA, 1998, pp. 329-340.

[83] M. Kyas, H. Fecher, F.S. de Boer, J. Jacob, J. Hooman, M. van der Zwaag, T. Arons, H. Kugler, "Formalizing UML models and OCL constraints in PVS", in *Proceedings of the Second Workshop on Semantic Foundations of Engineering Design Languages (SFEDL 2004), Volume 115 of Electronic Notes in Theoretical Computer Science*, Elsevier 2004, pp. 39–47, [doi:10.1016/j. entcs.2004.09.027].

[84] G. Magureanu, M. Gavrilescu, D. Pescaru, "Validation of Static Properties in UML Models for Cyber Physical Systems", In *Journal of Zhejiang University Science C*, Impact factor = 0.297 (2012), ISSN 1869-1951, doi: 10.1631/jzus.C1200263.

[85] Object Management Group - UML Profile Specifications. http://www.omg.org/technology/documents/profile_catalog.htm, 2013.

[86] A. Bondavalli, I. Majzik, I. Mura, "Automated Dependability Analysis of UML Designs", in *Proceedings of Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Saint-Malo, France, May 1999, pp.139-144.

[87] J. Wan, Y. Yu, Y. Wu, R. Feng, N. Yu, "Hierarchical Leak Detection and Localization Method in Natural Gas Pipeline Monitoring Sensor Networks" in *Sensors* journal, 2012, Vol. 12, pp. 189-214.

[88] G. Magureanu, M. Gavrilescu, D. Pescaru, A. Doboli,"Towards UML Modeling of Cyber-Physical Systems: A Case Study for Gas Distribution", in *Proceedings of the 8th IEEE International Symposium on Intelligent Systems and Informatics*, Subotica, Serbia, September 2010, pp. 471 – 476.

[89] C. Buratti, A. Conti, D. Dardari, R. Verdone, "An Overview on Wireless Sensor Networks Technology and Evolution", in *Sensors* journal, 2009, Vol. 9, pp. 6869-6896.

[90] Y. Ledru, "Identifying pre-conditions with the Z/EVES theorem prover", in *Proceedings of the 13th IEEE International Conference on Automated Software Engineering*, Honolulu, HI , USA, October 1998, pp. 32 – 41.

[91] T. Stoyanova, F. Kerasiotis, A. Prayati, G. Papadopoulos, "A Practical RF Propagation Model for Wireless Network Sensors", in *Proceedings of the Third International Conference on Sensor Technologies and Applications*, Glyfada, Athens, August 2009, pp. 194 - 199.

[92] J. Rousselot, J.-D. Decotignie, "A High-Precision Ultra Wideband Impulse Radio Physical Layer Model for Network Simulation", in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques - 2nd International Workshop on OMNeT++*, March 2009, article no. 79.

[93] M. Gavrilescu, G. Magureanu, D. Pescaru, I. Jian, "Handling Event-Driven Scenarios in CPS Application Simulations", in *Carpathian Journal of Electronic and Computer Engineering*, Vol. 4, No. 1, ISSN 1844 – 9689, October 2011.

[94] A. Kopke, M. Swigulski, K. Wessel, D. Willkomm, P.T. Klein Haneveld, T.E.V. Parker, O. W. Visser, H. S. Lichte, S. Valentin, "Simulating Wireless and

Mobile Networks in OMNeT++. The MIXIM Vision", in *Proceedings of the 1st International Workshop on OMNeT++*, March 2008.

[95] J. Beasley, editor, "Advances in Linear and Integer Programming", Oxford Science, 1996.

[96] M. Gavrilescu, G. Magureanu, D. Pescaru, "Error Handling using Linear Programming in CPS Applications: Aircraft Fuel Management System Case Study", in *9th International Conference on Computational Cybernetics ICCC 2013*, Tihany, Hungary, July 2013.

[97] I. Moir and A. Seabridge, "Aircraft Systems: Mechanical, Electrical, and Avionics Subsystems Integration", 3rd Ed., AIAA Education Series, Wiley, 2008

[98] P. Derler, E. A. Lee, A. S. Vincentelli, "Addressing modeling challenges in Cyber-Physical Systems", Technical Report UCB/EECS-2011-17, EECS Department, UC Berkeley, USA.

This work was partially supported by the strategic grant POSDRU/107/1.5/S/77265 Project ID77265 (2010), co-financed by the European Social Fund – Investing in People, within the Sectoral Operational Programme Human Resources Development 2007 – 2013.