VISUAL MODELING OF CYBER PHYSICAL SYSTEMS

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

Ing. Gabriela Măgureanu

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: 28.01.2013

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

To Daria – Children are the future.

Acknowledgement

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

Firstly, 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.

Secondly, I would also like to thank Assoc. Prof. Dr. Alex Doboli for his support, especially in the first years of my research.

Thirdly, special thanks go to my PhD colleague, Madalin Gavrilescu, who has worked by my side in all these years, even before he started his own PhD studies. I wish him the best of luck in finalizing his thesis.

Lastly, I would like to thank my family, who was there for me regardless of how difficult I proved to be at times. Special thanks go to my husband Milorad, my mother, my sister Alina and my brother-in-law Adi. This thesis was possible also because of you.

Măgureanu, Gabriela

VISUAL MODELING OF CYBER PHYSICAL SYSTEMS

Teze de doctorat ale UPT, Seria 14, Nr. 13, Editura Politehnica, 2013, 118 pagini, 43 figuri, 9 tabele.

ISSN: 2069-8216
ISSN-L: 2069-8216
ISBN: 978-606-554-612-7

Cuvinte cheie: Cyber Physical Systems, UM profiles, Model Driven Architecture approach, Goal-oriented approach

Rezumat: The PhD thesis presents a visual modeling methodology for modeling Cyber Physical Systems. Such an approach is useful for designers of Cyber Physical System applications. The presented methodology uses a goal-oriented approach for the specifications of applications of Cyber Physical Systems and Model Driven Architecture approach, at design level.

The thesis proves the utility of such an approach at design level by case studies which discuss applications for Cyber Physical Systems from different domains of activity, with different difficulty degree at specifications level and with different sizes.

Table of Contents

Motto:

"Lives of great men all remind us
We can make our lives sublime,
And, departing, leave behind us
Footprints on the sands of time"

A Psalm of Life - Henry Wadsworth Longfellow

Abstract

Cyber Physical Systems (CPSs) are massively distributed heterogeneous systems, which can be linked using wired or wireless connections. They integrate computation and physical processes and have a great economic and social potential. Regarding physical devices, such systems are mainly composed of sensors, actuators, communication units and decision nodes. At logical level, CPSs are tailored into several subsystems. Each subsystem aims to fulfill a specific objective. One of the main issues in developing CPS applications resides in asynchronous intercommunication between the subsystems and the influence of the exchanged information over the controlled devices. Using models in CPS design is intensively investigated nowadays.

The methodology introduced in this thesis is based on OMG Model Driven Architecture (MDA) approach. It supports the design of various computational models and allows customization based on the application requirements. Here, the design approach addresses both hardware and software aspects of a CPS application. As originated in MDA, it is based on Unified Modeling Language (UML). UML is characterized by an intuitive graphical approach for embedded systems design, making it easy for end-users to specify the requirements and constraints for the applications. UML profiles allow specific type definitions for families of applications. This thesis presents two defined UML profiles, one for hardware and one for software specification for CPS applications. The profiles are used for tailoring UML to application specific requirements. The profiles compose the Computational Independent Model (CIM) part of the presented MDA approach, by defining specific hardware requirements and behavioral constraints.

A CPS application consists in fulfilling a set of independent objectives. Each objective is assigned to a heterogeneous subsystem. In such a subsystem, the devices are logically coupled and they collaborate to meet the specific goal. Therefore, in the presented approach, the entire network is tailored into goal-oriented interconnected devices. The resulting models constitute the Platform Independent Model (PIM) of the system and contain the application's specifications. Using code specific transformations, a PIM can be later translated into a network deployable Platform Specific Model (PSM). The resulted code represents the final scope of using MDA approach.

List of Abbreviations

CCSL	Clock Constraint Specification Language
CIM	Computation Independent Model
CPS	Cyber Physical System
DM	Decision Module
DMA	Decision Module Area
DMP	Decision Module Perimeter
DMZ	Decision Module Zone
DRN	Declarative Resource Naming
DTM	Distributed Token Machine
EDA	Electronic Design Automation
EJB	Enterprise Java Beans
FSM	Finite State Machine
GMF	Graphical Modeling Framework
LP	Linear Programming
MARTE	Modeling and Analysis of Real-Time and Embedded Systems
MDA	Model Driven Architecture
MDP	Markov Decision Processes
MiXiM	Mixed Simulator
MOF	Meta-Object Facility
OMG	Object Management Group
OMNeT++	Objective Modular Network Testbed in C++
PC	Personal Computer
PIM	Platform Independent Model
PSM	Platform Specific Model
PSoC	Programmable System-on-Chip
SoC	System on Chip
SP	Spatial Programming
SysML	System Modeling Language
TML	Token Machine Language
UML	Unified Modeling Language
WNES	Wireless Network of Embedded Systems
WSN	Wireless Sensor Network

List of Figures

List of Tables

Chapter 1. Introduction

1.1 The Research Theme

Cyber Physical Systems (CPSs) are, as presented by Lee in [1], integration of computation and physical processes. They are massively distributed embedded systems, which can be linked through wired or wireless connections. At physical level, CPSs are composed of sensors, actuators, communication units and decision modules. At logical level, CPSs are tailored into several subsystems, where each subsystem aims to fulfill a specific objective.

In [1], Lee identifies the main challenges related to CPS applications. As expected, there are no general accepted solutions for all types of applications, although several problems regarding CPSs are already resolved. One of the main issues in developing CPS applications resides in asynchronous intercommunication between the subsystems and the influence of the exchanged information over the controlled devices.

Applications of CPSs can be found in various activity domains like: high confidence medical devices and systems, habitat monitoring, assisted living, aerospace, critical infrastructure monitoring, intelligent traffic management, energy consumption optimization for vehicles or buildings, distributed robotics, defense systems [1]. Such a large variety of application fields and the large number of actual applications developed, which are highly used and useful in practice, indicate that attention was given in efficient and, at the same time, optimal design and programming of CPSs.

Although it is a rather new area of research, CPSs domain is of interest for many researchers worldwide. Using models in CPS design is intensively investigated nowadays [2]. There are several proposed approaches in handling CPS applications, which are also discussed in this thesis.

The methodology presented in this thesis 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. UML profiles are used for tailoring UML to application specific requirements [5]. The main steps of the approach in this thesis are presented in chapter 1.3 and are detailed starting with chapter 3.

1.2 Thesis Objectives

This thesis intends to accomplish the following:

- From a theoretical point of view, to define a methodology for designing and programming CPS applications
  - To define two UML profiles, one for hardware specification and one for software specification, further used to customize CPS applications and to map them as MDA Computational Independent Model (CIM)
  - To define the goal-oriented tailoring the CPS network into several logical levels and the approach in specifying CPS applications requirements
  - To compose MDA Platform Independent Model (PIM) in a visual manner, starting from CIM, at both hardware and software levels
  - To identify PIM validation methodologies and possible transformations to PSMs

- From a practical point of view, to apply the MDA proposed approach to different types of CPS applications, in order to demonstrate that
  - The MDA presented approach is suitable for CPSs 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 that can be modeled in this manner belong to different activity domains and can be of various degrees of difficulty.

1.3 The Proposed Approach

Considering the multiple types of applications which use CPSs, an intuitive and easy to use design and programming model is required. Using already defined components, the user of such a programming methodology is able to design the network at structure level, by specifying the selected components and the relationships between them, without considering the hardware requirements and limitations. These are handled by the developer of the different types of predefined components. Also, the user has the possibility to simply specify the goals for the corresponding application, the created system having in care the actual implementation of the desired goals.

Such a programming methodology involves some well-defined steps. A UML profile which covers the basic hardware aspects for describing the structure of the different types of nodes, the connections between them, their customization and limitations is defined. Another UML profile groups the stereotypes which define the network behavior, in correlation with the hardware structure. These UML profiles are used to define the library of components that help users when constructing their CPS network, based on application requirements. The UML profiles form the CIM in MDA approach.

The UML profiles are used as starting point to define the MDA PIM. The PIM is defined using two types of diagrams. UML deployment diagrams are used to express the application topology, the types of nodes and the connections between them, as the CPS application requirements at hardware level. UML component diagrams are used to express the network behavior, at different logical levels. At the lowest logical level it is established the behavior at the actual device level. At the highest logical level, the CPS application behavior is established, considering the network as a whole. The UML software stereotypes define the reflection of the goals from higher logical levels to lower logical levels and the implications to higher logical levels generated by goals at lower logical levels.

The next step implies testing and verifying the PIM, before the transformation to a Platform Specific Model (PSM), depending on application requirements and user specifications. This can be achieved through simulation. Simulation of embedded system applications at node and network level is desirable before deployment on hardware devices. Simulation has major advantages, such as: errors detection in development phases as a result of testing, validation before deployment, the obtaining of a deterministic behavior for each node in the network, and also for the entire network.

Using code specific transformations, a PIM is later translated into a network deployable PSM. The resulted code represents the final scope of using MDA approach.

1.4 Thesis Organization

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

Chapter 2 reviews main research directions regarding programming models for sensor networks and CPSs, visual modeling of distributed systems and MDA approaches.

Chapter 3 presents the goal-oriented approach in CPS design and defines the tailoring into several logical levels for CPS networks.

Chapter 4 presents the proposed MDA approach in CPS design with emphasis on the CIM and PIM models and the models transformations.

Chapter 5 describes the UML artifacts defined for customizing CPS applications. The two subchapters describe the UML hardware and software defined profiles, respectively.

Chapter 6 presents the main ideas regarding model validation. Simulation is a useful validation method; specific case studies using the MDA presented approach are discussed in greater details. The applications are taken from different domains of activity and have different degrees of difficulty regarding specifications.

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

Chapter 2. State of the Art

2.1 Massively Distributed Embedded Systems, Sensor Networks, Cyber Physical Systems

The term of embedded systems was initially used to describe microprocessor-based systems that combine hardware and software processes, to control a particular function or a group of functions, as presented by Nessett in [6]. Embedded systems have predefined functionalities which are programmable by the user. However, the user cannot change the functionality of an embedded system by adding or replacing software, as in a regular personal computer (PC). Some of the advantages of using such systems are the improvement of mechanical performance or the replacement for analog circuits and discrete logic-based circuits. Applications of embedded systems were successfully designed over the years in many fields of activity. However, the major drawback of these systems is that most of them are "black boxes" to the outside, unable to link to other systems and share their computing abilities.

Massively distributed embedded systems group a large number of tiny, single-chip devices, equipped with sensors, actuators and communications units [7]. These systems are seen as a suitable solution for many modern applications. This has become feasible due to the low price and dimensions for sensing and electronic devices, which allowed connecting of a large number of nodes for a single application network. The challenges reside in solving the issues that appear for different sub-systems during the network life cycle, while maintaining the system robustness. Usually, these networks are deployed in a dynamic environment, where they permanently have to adapt. Also, due to the small size requirements for the nodes, the systems must handle the constraint resources.

In this thesis, the author is focused on CPS design, the challenges and methodologies of modeling a realistic CPS application, starting from application requirements and predefined, customized components. CPSs have evolved from massively distributed heterogeneous embedded systems over the last years and have an increased impact in various domains of activity. Applications of sensor networks and in particular of CPSs can be found nowadays in multiple fields like: intelligent traffic management, healthcare, aerospace, infrastructure management, energy consumption optimization in vehicles and buildings, critical infrastructure monitoring, habitat monitoring.

CPSs are integrations of computation and physical processes [1], composed mainly of sensors, actuators, communication and control devices. CPS nodes can be linked in networks through wired or wireless connections. Controlling such CPS networks is in a continuous evolution and major investments are made in the entire world to develop an efficient technology for CPSs. As sensors are an important part in CPSs, the author has considered that studying programming models for sensor networks can provide valuable ideas and examples for determining a suitable solution in modeling CPS applications. Some relevant programming models for sensor networks are discussed in the next subchapter.

Several summits [8] and research groups [9], [10] are interested in determining the major challenges imposed by CPSs and possible solutions. The research directions for CPSs can be summarized, as stated by Lee in [10]:

- Time must be considered in programming languages: physical time must be taken into consideration when designing reliable CPSs. Even if tasks are correctly completed in a CPS, the timing requirements for the entire network may not necessarily be met [1].
- Reconsideration of the operating system/programming language split: the way the operating systems are abstracted and isolate the programming language by the hardware must be rethought in case of CPSs. TinyOS/nesC [11] are a promising start as the programming language abstraction in nesC supports the design of thin wrappers around hardware.
- Reconsideration of the hardware/software split: the abstractions used for hardware differ from the ones used for software and a common "language" must be found, as CPSs integrate hardware with physical processes and software.
- Memory hierarchy with predictability: memory hierarchy techniques are important when considering scalability for CPSs networks. The software performance is increased at the expense of predictability. Solutions must be found in order to reduce the drawback of time predictability.
- Memory management with predictability: automatic memory management improves the productivity for the programmer and the software's general reliability, but again, the time predictability cannot be specified.
- Predictable, controllable deep pipelines: existing techniques for deep pipelines deliver high performance, but the time predictability problem still occurs.
- Predictable, controllable, understandable concurrency: CPSs are intrinsically concurrent and the existing solutions for concurrent programming (threads) make the program nondeterministic and the code unreliable.
- Concurrent components: physical components are qualitatively different from the object oriented software components; therefore standard abstractions are not suitable for CPSs.
- Networks with timing: the existing TCP/IP networking techniques cannot determine the important time predictability for a CPSs network; therefore other networking techniques must be considered.
- Computational dynamic system theory: the system theory must combine the existing independent purely physical with the purely computational theory.

The research results are encouraging for each of the research areas previously discussed. However, it cannot be stated that CPS design is no longer a challenge.

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 not predictable, therefore CPSs must be robust under unexpected conditions and be able to adapt to system failures. The principle to be followed, as stated by Lee in [1], is that components at each of the layers of abstraction defined for CPSs must be made predictable and reliable if this is to be technologically feasible case. If not, the higher level of abstraction must compensate with robustness.

This tailoring into several logical layers constitutes the basic idea of the goal-oriented programming model for CPSs presented in this thesis. The objective is to offer to the designer of a CPS application the possibility to specify the requirements at network level and the translation of commands to physical nodes to be controlled by this high level of abstraction.

The increased interest in CPSs over the last years is due to the large number of applications where they can be of interest, in multiple fields such as: intelligent traffic management, healthcare, aerospace, infrastructure management, energy consumption optimization in vehicles and buildings, critical infrastructure monitoring [12]. Also, the fact that the sensing and actuation devices are cheap and small and can therefore be deployed in a large number, made CPSs suitable for applications that require a high precision degree, ensured here by large amounts of information gathered from nodes. CPSs must have a high degree of reliability, as the malfunction of local nodes or even subsystems should not affect the entire system. Such a large system is hard to control as related problems are tackled together and not separately [13]. The control procedures in a CPS must understand and react to quality failures given by the specifications in an application. The necessary adjustments are not trivial, due to the large number of parameters existing over the network.

One approach in controlling the applications using CPSs is given by logically layering the devices that compose such systems. Decisions at lower computational levels are separated from, but influenced by, decisions at higher computational levels and vice versa. The lower logical levels are tackled with the physical level and mainly address local constraints. The decisions at higher logical levels refer to larger areas, where the parameters for a certain model change much slower. At lower logical levels, the decisions can be taken using reactive models, while at higher levels Task Graphs or Markov Decision Processes can be considered.

2.2 Programming Models for Sensor Networks and Cyber Physical Systems

2.2.1 Programming Models for Sensor Networks

Sensor networks come with an increased potential in various fields of activity. The difficulty in programming such networks is given by operating with unreliable communication, nodes and constrained resources [14]. Different programming models were proposed in literature to overcome such programming difficulties, each of them addressing only specific problems.

In [14], Sugihara and Gupta propose a taxonomy for the existing programming models, taking into consideration specific requirements for sensor network applications like energy-efficiency, scalability, failure-resilience and collaboration. Energy-efficiency is common for the wireless sensor network applications, as nodes must function with the same batteries for a large period of time. Bandwidth-efficient programs are desired, as applications may contain hundreds of nodes. Applications are intended to remain functional even when facing unreliable communications, nodes or other failures. Collaboration is a challenging requirement as the way information is gathered and distributed in the network is not standard.

The complexity of sensor nodes is both a hardware and software problem that requires a solution. Regarding software, programming levels can be placed at thre conceptual levels: node, group and network level. A few programming models, relevant to the author's research, are detailed in the next pages.

At node-level, programming models can be operating systems or virtual machines/middleware. This approach in sensor programming is suitable when speaking of small scale networks, because the software load necessary for the operating systems increases the costs in case of larger scale networks. Also, using specific operating systems for sensor networks can be a disadvantage, as programs written in nesC [15], for example, are not widely known. End-users can have difficulties when using such programming languages for specifying the applications. Node-level programming models are not suitable for the presented goal-oriented model, as it is intended for massively distributed embedded systems and CPSs. Such networks would have high costs and reduced energy efficiency, if implemented using node-level approaches.

Neighborhood-based group programming languages are efficient as they support aggregation at group level through data sharing. Definitions and operations are also made on groups. As the size of the network increases, the software overload also increases leading to slow and inefficient programming for applications. The size of the network is a very important parameter that needs to be taken into consideration for developing a suitable programming model.

The research approach regarding visual and goal-oriented programming of sensor networks is specified as a network-level programming model. Several programming models for sensor networks, operating at network level, were considered when initiating the presented approach. These are further grouped into: database and macroprogramming language. Macroprogramming language is useful in describing global behavior and resource naming for a system.

In [16], Heinzelman et al. present MiLAN, as a middleware designed to take into consideration the properties of the network itself. It consists of an abstraction layer that allows network-specific plugins to convert MiLAN commands to protocol-specific commands, passed through the usual network protocol stack.

This middleware contains two important parts: one that makes the connection to the possible sensor networks and another one that contains specific application programming interface. MiLAN takes into consideration the application needs and can use only sensors that fit a particular application.

Regarding macroprogramming languages, Semantic Streams is a representative example. In [17], Whitehouse et al. propose a framework that allows users to pose declarative queries for the interpretation of data gathered from sensors. The users can also place constraints. Multiple, independent users can query the network simultaneously, whereas the system allows sharing resources and solving conflicts between applications.

The framework is based on a semantic services programming model. A service is a process that infers semantic information about the surrounding world and incorporates it into an event stream. Each service has certain inputs and outputs. The framework was built in such a manner that it allows services to be composed and therefore to create new applications.

Newton and Welsh describe in [18] the design for Regiment, another macroprogramming language for sensor networks. The Regiment compiler transforms a macroprogram into an efficient program based on a token machine. The language is a purely functional one. The authors consider that sensor networks should be programmed at global level, while allowing the compiler to automatically generate behaviors for the nodes from a high-level specification of the global behavior of the network.

A program in Regiment can be best visualized as a dataflow graph; the compiler generates code for such graph by directly translating each edge into some number of token handlers. A standardized interface ensures that every distributed value produces at least a formation token handler and a membership token handler.

In [18], the authors state that the functional programming language is essential for capturing data parallelism and enabling the compiler to make informed decisions about the scheduling and placement of computation in the sensor network.

Newton et al. propose in [19] an intermediate language for sensor networks – Token Machine Language (TML). It is based on a simple abstract machine model, called Distributed Token Machines (DTMs). DTMs provide a model based on tokens. Communication is provided via typed messages containing a small payload (token messages). Tokens are associated with token handlers that execute when receiving a token message. TML is also used with Regiment. Each token also has a private memory (a fixed-size piece of state), accessible by the token handler.

A sensor node consists of a heap storing local token objects, a scheduler that processes the incoming messages and a collection of handler actions computable by the node. Token handlers can use some operations to access the token store and the scheduling mechanism. The language used in TML for handlers is a subset of the C syntax plus the DTM interface.

The DTM model permits returning subroutines. This can happen with the help of returning handler-calls built on top of core TML by using a continuation passing style transformation. An important example for applications of TML involves gradients (general purpose mechanisms for breadth-first exploration from a source node).

Gummadi et al. present Kairos in [20], which is a programming model that provides abstractions for expressing the global behavior in distributed computations, part of a single program. Kairos offers a set of extensions to a programming language, in the article, Python. It provides three abstractions:

- The node abstraction: programmers manipulate nodes and lists of nodes, while Kairos provides the data types and operators needed
- The list of one-hop neighbors of a node: a Kairos program is usually specified in terms of operations on the neighbor list for a node; therefore this is a normal abstraction for sensor network programming
- Remote data access: the ability to read the values of variables at named nodes. For the programmer this means *variable@node* notation.

The implementation of the Kairos extensions was made on Python, mostly because the authors were familiar with the internals. They claim that Kairos is language independent. The Kairos preprocessor uses Python extensibility interfaces and the runtime is implemented in C.

Kairos is easy to run on programmer side, as he only has to provide a program that contains distributed calculations. Taking into consideration the comparisons in [20], Kairos works well for distributed networks, compared to classical algorithms. Regiment and Kairos are suitable examples of describing global behavior in sensor networks.

Spatial Programming (SP), presented by Borcea et al. in [21], is a space-aware programming model, suitable for outdoor distributed embedded systems. The main idea is to offer network-transparent, fine-grained access to data and services from embedded systems. The programmers will be able to use the data provided by sensor nodes as if they would be simply using program variables.

The abstraction defined by SP contains Space Regions which are virtual representations of physical space and Spatial References – {*space: tag*} pairs, where *space* is a Space Region and *tag* is a name of a property or a service provided by the system. Programmers can use indexes to refer distinct systems. The spatial references are consistent during program execution.

A timeout mechanism is implemented, in case the node is no longer located in the space it is searched for. The timeout value can be specified. If the programmer has knowledge about certain node mobility, it can still access it through space casting. New space regions can be defined using the intersection and reunion operators. Also, the creation and removing of network resources is possible (similar to creating files).

Spatial Programming can be used when a high degree of transparency is desired for the programmer. He will query the data present in the nodes of the network as if he would use program variables. This can happen due to the introduced abstraction (the analogy with virtual memory and physical memory for traditional programming models).

The authors present in [22] Declarative Resource Naming (DRN), an abstraction that allows the description of desired resources by their run-time properties. The resources are bound to variables and the networking is transparent to the programmers.

A Wireless Network of Embedded Systems (WNES) is considered a single abstract machine and resource access can appear as simple variables access. For binding resources and variables, the programmer can specify the desired property for the target resource. Usually, a resource variable matches more resources. Each resource from the set can be accessed individually by using iterators (sequential access) or there can be parallel access, which can reduce energy consumption in the system (for example, while not taking into consideration resources that are lower that a desired max value at a moment).

Dynamic resource binding allows the programmers not to concern themselves with matching the resources variables to the resources of interest. This is enforced by the semantic regarding resource access; to prevent overhead and excessive energy consumption, programmers can relax the semantic of constraints.

Static binding is necessary if the program needs to access a resource that does not match the description anymore. A single resource or a set of resources can be retained for future use. By default, DRN uses dynamic binding.

SP and DRN are representative examples for the resource naming that should view the network as a single entity.

Ni et al. introduce in [23] SpatialViews, as a high-level programming language for ad-hoc networks. A spatial view is a collection of virtual nodes (programming abstractions for physical nodes). A virtual node is characterized by its location, time and services provided. In a virtual node, a program has access to the services the corresponding physical node provides (it hosts an object that implements a certain interface, corresponding to a certain service). A space can be an object of class Space or of a derived class of class Space. A space granularity for a class view denotes the spatial density of the virtual network.

In a defined spatial view, an iterator discovers the virtual nodes, gets access to the services they provide and performs the migration of the program execution between nodes. During iteration, unique nodes are visited. The program will migrate between virtual nodes using different techniques for choosing the next node. Infinite iteration can be performed, but it is not desirable. After all nodes have been visited, the results return to the node where the iteration was started.

Every variable for a SpatialViews program is a program variable or a service variable. The service variables support the access to services, do not migrate, are stored in the node space and can be created using the register operator. Program variables migrate from one node to the other, are stored in the program space and are created using the new operator. If a program variable does not match to one of the categories (local, container or reduction), a compile-time error can occur.

The current implementation of the SpatialViews includes a compiler, virtual machine, runtime library and debugging/visualization environment. The compiler extends javac from Sun. The low-level migration is provided by the SmartMessages system. At a higher-level, the compiler generates code, so that the migration appears transparent.

Aguiar et al. present in [24] a programming model for optimizing the energy consumption in heterogeneous WSN (wireless sensor network). The ability of sensor nodes to collect different phenomena is what makes the network heterogeneous. For developing the model, some aspects must be taken into consideration: the rate of variation for each measured phenomenon, for choosing the best sampling rate; the number of active nodes at a certain moment: too many nodes would result in higher energy consumption, too few could jeopardize the collection and transmission of data; bottlenecks for sensors that are part of too many transmitting processes and that can break.

The model was tested on homogenous and heterogeneous WSNs and the results show energy consumption reduction for the second network type. The demand points were placed in a grid, as well as the sensor nodes. The results became more obvious when the number of time intervals got larger. Tests were made considering one sink and also four sink nodes.

The main idea of the model is that by turning off a set of sensors at a certain time interval, the total energy consumption can be reduced and at the same time the routing of the network is still viable (the network is not partitioned).

Each of the proposed programming models is a relevant solution for the types of applications it intends to model. However, there is no general solution is solving each type of sensors network based application. As in this thesis the author aims to propose a high-level methodology for the design of CPS applications based on sensors and actuators, the network level programming models for sensor networks were analyzed in greater detail. From those ones, also summarized in the following table, the basic idea from Semantic Streams solution is relevant for the proposed approach. Declarative programming is more intuitive for users than low level programming for user, so application requirements can be declared at a high logical level. The approach presented by Heinzelman et al. in [16] is the starting point for the designed middleware, so that the current approach offers support for translating the goals stated at the highest computational level into goals to be established at lower computational levels and finally into commands to the physical nodes. The focus on the network being transparent for the programmer, defined in DRN [22], is identifiable in the approach presented in this thesis as the user of the CPS is required to specify at design general application objectives.

Programming model	Strong points	Weak points
Milan [16]	Takes into consideration the application needs.	Can use only sensors that fit a particular application.
Semantic Streams [17]	Declarative programming is more intuitive for regular users than low-level, distributed programming; The system can be queried by multiple users, with multiple tasks; The user can specify desired QoS.	The query processor cannot reason about runtime; The language does not support quantifiers and scoping.
Regiment [18]	Facilitate in-network processing by fold/map interface.	Applications with highly dynamic behavior are more difficult to encode; Is not designed to support rapid, repeated, short-lived queries; Explicit control of low level hardware features cannot be encoded.
Kairos [20]	Easy to use; Good results for distributed networks.	Authors wrongly claim Kairos is language independent.
Spatial Programming [21]	Offers a high degree of transparency for the programmer; Code migration uses Smart Messages; A timeout mechanism is implemented.	No information about SMWrapper.
DRN [22]	The networking is transparent to the programmer; In-network processing.	No information about testing or implementation.
Spatial Views [23]	User-specified quality of results; Dynamic binding.	The unit measurements are not specified; The lack of security or privacy in applications; Loop based compiler optimizations are investigated.

Table 1 Evaluation of programming models for sensor networks

2.2.2 Programming Models for Cyber Physical Systems

CPS design and programming is a relatively new topic in worldwide research, in comparison to subjects like sensor networks and embedded systems. However, there are several attempts that are worth considering when proposing a new approach in CPS design.

CPSs have been introduced as heterogeneous systems, which can be used in several types of applications, in [1]. Also, the author presents the open points in challenges in CPS design [10] and presents the CPS aspect as rather an intersection than a reunion of cyber and physical issues [25]. E. Lee’s research and articles have been the starting point for many articles that discuss programming models for CPSs.

In [26], Tabuada argues for a decoupling, within certain limits, between the physical characteristics of CPSs and the part that is available to the end user. He presents some notions on the topological abstractions of the physical devices, like the notion of locality, with different meanings in the physical environment and for the CPS network based on sensors and actuators. He also argues for in-network computation, where information gathered from sensors and the commands to actuators are managed within the network. The concepts in Tabuada’s proposal are very similar to the author of this thesis vision about CPS modeling. The separation into computational layers and the handling at the level of decision nodes, inside the CPS network, are aspects discussed in the present paper.

Gupta focuses in [27] on defining a programming support for location and time information, in CPS applications. He argues for the importance of a semantic support to use the physical location information and the validation of application’s models against spatial and timing requirements.

Derler et al. discuss in [28] CPS modeling from the heterogeneity, concurrency and sensitivity to timing perspectives. They take as CPS example a part of an aircraft vehicle, the fuel management system and propose several solutions. The authors use the Ptolemy project [29] for the fuel system modeling and also for simulating fuel flow between tanks.

Saeedloei and Gupta consider in [30] the hybrid automata as a possible solution for specifying, designing and analyzing several types of systems, including CPSs. They model hybrid automata using logic programming, with several extensions. They use the proposed framework on CPS examples, as the generalized railroad crossing problem and the reactor temperature control systems. This logical programming approach has good evolving perspectives, as the authors have progressed on their research and presented the results in several related papers.

In [31], Liu proposes the adaptation of unified object model and classical programming techniques to CPS programming. Indeed, already defined software engineering technologies and tools, including UML, can be put together to CPS design. Also, the strong points of Object Oriented Programming (OOP), encapsulation, code reuse or customization can have a great impact in CPS programming.

A summary of the programming models for CPS applications studied is presented in the following table. In this thesis, the author uses as one of the case studies a similar portion of an aircraft, the fuel management system [28], in order to illustrate the goal-oriented specifications. The fuel management system is a representative CPS, as it combines several subsystems, each with other tasks to fulfill. At the same time, this system is very difficult to design and the goal of using it is not to show that one can create an aircraft system from scratch. The objective of using it as a case study for this thesis is to show that the defined separation into logical layers can be used in case of difficult to manage systems, and not only in case of massively distributed systems.

Also, ideas like tailoring into computational levels or decisions taken inside the network, after gathering sensor information (as Tabuada proposes in [26]) are main concept in the goal-oriented approach presented in this thesis and are detailed in chapter 3. Similar to Liu's proposal in [31], the defined approach uses UML, as a classic and accepted solution in modeling distributed systems.

CPS Approach	Starting Year	Summary
CPS position paper	2006	Decoupling into levels of abstraction [26]; Decision management - inside network.
CPS support for location and time	2006	The author argues for the importance of location and time information in CPS semantics [27].

CPS position papers	2008	Defining CPS [1];
		Defining challenges in CPS design;
		Starting point for many CPS researches.

Heterogeneity, concurrency and sensitivity to timing perspectives	2010	Uses the fuel management system from an aircraft vehicle as CPS example [28]; Uses Ptolemy project as a solution for design, modeling and simulation of the CPS application.
Hybrid automata approach	2011	Model hybrid automata is modeled using logic programming [30]; Apply the approach on several CPS examples.
OOP concepts in CPSs	2011	Using OOP defined concept in design and simulation of CPS applications [31]

Table 2 Evaluation of programming models for CPSs

2.2.3 Goal-Oriented Programming Models

Goal-oriented programming allows raising the level of abstraction when specifying the requirements for an application. As this specification is a very important step in designing the application flow, interest has been shown in providing methods for posing the goals and constraints in a formalized way.

An early attempt to define goal-oriented programming is described in [32]. The author proposes an alternative to threads in dealing with events for distributed applications. He has shown that, by designing and implementing programs top-down and defining goals and dependencies between them, it is possible to obtain better performance than by using threads. The goals can be specified in Prolog. The completion of a goal is considered an event. This model for goal-oriented programming is different from the presented approach, as in [32] goals must be specified along with the actions necessary to accomplish the goal (the steps that need to be taken). In the discussed approach, the goals are set at a specific logical level and determine the goals at lower and upper logical levels. However, the performances obtained by van Renesse in [32] are important to keep in mind.

KAOS [33] is a goal-driven framework for systems that facilitates, after a preliminary identification of goals, identification of future goals, requirements and actions for the systems. In [33], the authors detail the implementation of a UML profile for specification of KAOS. Although this profile is oriented for KAOS framework and its needs, it is a strong point for the current research.

Navarro et al. propose in [34] a Goals Model included in ATRIUM, which is a methodology for concurrent definition for Software Architecture and Requirements. What is especially relevant for the goal-oriented model discussed in this paper is the UML profile developed to tailor this Goals Model with respect to the semantics. This UML profile has been a relevant example in defining the proposed CPS UML profile, with respect to functional and non-functional requirements applications are defined by.

At first glance, the approach stated by a group of researchers from MIT in [35] seems promising, but it appears to have been left unfinished. In the proposed system semantics, goals are formalized as language constructs and the values stated in them lead to an automatic construction of a component-based system. Regarding semantics, goals are similar to regular procedure calls, containing a name and parameters. However, there is no code associated with these goals. The system searches for the techniques that can solve the posed goals. Each of these techniques specifies a pattern that needs to be matched to considered goals, and eventually subgoals that need to be accomplished, and code to be run in case the tree of subgoals is satisfied, in order for the high level goals to be satisfied. This type of semantic is similar to Prolog.

The tree for the goals persists as long as a goal is active in order to record the choices that were made. If new requirements are posed, the considered path in the tree may not be the correct one anymore. In this case, the goal tree is reevaluated and alternative choices can be made in real time [35].

The similarity with the goal-oriented approach proposed by the author lies in the way goals are considered as requirements for the application and are purely declarative. This approach was initiated in [13]. The goals are purely declarative statements, customized for each application. They do not describe the steps to accomplish for achieving the goal. Another layer, the middle one, between application layer and hardware or simulation layer, implements the basic algorithms that will solve the expected goals. The goals and the constraints are customized for each application, but the algorithms remain basically the same, and independent from a specific application.

The applications do not present explicit interactions between entities, like the network structure. The basic routines introduced before are parameterized and the parameters are computed in an automated way, depending on the goals of the applications.

The main entities of this goal-oriented approach are the Decision Modules (DMs) [13]. They contain four main parts:

- Inputs: the physical data acquired for applications, in this case through sensors;
- Outputs: characterized by a set of attributes, they are the outputs of the applications;
- Goals: have to be maximized or minimized, depending on the requirements for the applications; they are mathematically expressed, such as in integer linear programming, using inputs, outputs and functions from middleware;
- Constraints: represent the physical capabilities and limitations of the platform on which the application is executed.

Decision modules are organized in hierarchical structures to handle the complexity of distributed applications, similar to the one used to group nodes inside a network. In fact, to each instance on a computational level corresponds a certain DM that handles the logic of the application. Therefore the DMs are grouped at low logical levels into Finite State Machines and at higher logical levels into less flexible models, like Data Flow Graph or Markov Decision Processes.

The ideas presented by Saif et al. in [35] were used to implement the "Just Play" framework [36]. This framework was proposed as a method to reduce the amount of configuration activities users must do in order to configure the possible applications for the consumer-level electronic devices. The framework consists of two layers, one that captures and satisfies user tasks and another one that tries to satisfy tasks when having as inputs several devices. The two layers are low coupled, as they are intended to evolve separately. As stated in [35], goals must be satisfied by techniques, without being bound to a particular technique until runtime. The goals and the techniques form a tree, and the planning means a heuristic search from the root of the tree to the leaves for the techniques that best satisfy each goal [36].

An interesting approach in representing the requirements of an application is detailed in [37]. De Sousa and de Castro propose a framework based on the separation of concern principle in order to improve the reusability, maintainability and comprehensibility of the specifications for the requirements in a system. It is important to analyze beyond the functional requirements of an application, especially in a case like the presented goal-oriented programming model. The target for functional requirements is to capture the intended functionality and for the non-functional requirements to impose restrictions on the system. The latter are more difficult to specify, yet this can be achieved through an optimal application of the separation of concern principle. The authors prove their theory through a relevant example for an Internet Banking System.

The approach presented by Kim et al. in [38] deals with the notion of facts and goals in networked CPSs. The authors provide a flexible reasoning framework to support goal requests when interacting with the environment and actions for achieving the goals.

The following table summarizes some of the most important approaches in the last years for goal-oriented programming. Considering all the studied goal-oriented methodologies, the one detailed in this thesis uses as starting point the concepts initiated in [13]. The application nodes are tailored into several logical levels, in each subsystem that acts in the CPS network. Application goals are stated at the higher logical level and the user does not deal with low level programming aspects.

Goal-Oriented Programming Approach	Starting Year	Summary
KAOS Framework	1991	The KAOS approach: Goal-driven requirements engineering [33].
ATRIUM Methodology	2003	A methodology for concurrent definition for Software Architecture and Requirements [34].
“Just Play” Framework	2006	A method to reduce the amount of configuration activities users must do in order to configure the possible applications for the consumer-level electronic devices [36].
Declarative goals	2009	Applications goals are purely declarative. The logic for the entire application is maintained in decision nodes [13].
An approach for networked CPSs.	2011	A distributed logic approach for networked CPS, with support for distributed goals and facts.

Table 3 Goal-oriented programming approaches

2.2.4 Summary

The goal-oriented model the author presents is similar to Region Streams and Semantic Streams. It allows aggregation of the data streams, same as Region Streams does, and is based on tasks which are periodically executed when input data exists. In this case, the algorithms selected from specialized libraries are dynamically receiving parameters in order to be periodically executed on the system. Semantic Streams approach is more suitable in applications where the network can be seen as a large database. The target applications for the programming model are deployed on networks with several sensor and actuator nodes and, at data pools, the decision nodes must manage the large quantity of data gathered from these nodes. Therefore, the concepts already defined for Semantic Streams can be used here.

The goals for an application must be specified as detailed as possible and by taking into consideration several aspects, such as the classification in functional and non-functional requirements for them. It is quite difficult to naturally specify all goals involved in an application, especially the non-functional ones; therefore the use of a separation of concern principle is desired. UML profiles allow a detailed specification of several aspects in the development life cycle of an application, including the requirements. Taking also into consideration the already defined UML profiles in the literature for embedded systems, the author of the thesis has proposed two UML profiles for customizing CPS applications, further detailed in chapter 5.

2.3 Visual Modeling of Distributed Systems

2.3.1 Modeling based on UML Profiles

Visual modeling for applications with applicability in various domains, a well-written documentation and use cases to facilitate the use of applications, came as a natural request with the growing complexity of software products. Through the usage of a model, designers of applications can ensure the correctness and completeness for business functionality and the end-user's requests [4]. The model is checked to ensure robustness, security, stability and extensibility. All these can be accomplished before actual code is written, leading to reduced number of errors at the beginning of the implementation. A proper modeling and a complex and detailed documentation can be the key of an easy extension of the overall project, in case of future requirements.

UML is used in computer engineering as general-purpose modeling language. UML is standardized by OMG, which is also the creator of this language. UML can be used in various steps for the software development life cycle. Another strong point in using UML is the fact that, when combined with MDA, it allows code generation. The amount of generated code in the total code written for an application and its precision depends on the accuracy and complexity of the transformation rules.

UML allows a high level of abstraction for specifications of applications. The level of abstraction can be increased by hiding details in the models or decreased, when focusing on the main aspects of the system [4]. UML allows the switching of perspectives starting from a simple element for the scheme, going to the entire environment where the application is executed, making possible visualization of connections between elements or even connections between related applications.

In UML, structure diagrams are used to describe the architecture of the system, while the behavior diagrams express the required functionality for the system which is modeled, the flow of data and control. A particularly interesting type of diagrams is the profile diagram. Such diagrams allow definitions for custom stereotypes, their "attributes" named tagged values and constraints, applicable to each stereotype. These customizations are grouped in an UML profile, which is a structure that can be applied to several elements in other UML diagrams. This application transfers the customizations to elements in the diagrams.

According to OMG, the organization that standardizes UML artifacts, including profiles, A UML profile is a specification that has one or more of the following characteristics [4]:

- Identifies a subset of the UML metamodel;
- Specifies "well-formedness rules" (constraints written in OCL), along with the ones specified by the identified subset of the UML metamodel;
- Specifies "standard elements" (standard instances of UML stereotypes, tagged values or constraints), along with the ones specified by the identified subset of the UML metamodel;

- Specifies semantics, in natural language, with the ones specified by the identified subset of the UML metamodel;

- Specifies common model elements, expressed in terms of the profile.

There are some already defined UML profiles that can be applied to embedded systems applications and were useful in defining the UML profiles detailed in chapter 5: one hardware profile for network topology and hardware components and one software profile for goal-oriented behavior.

SysML (System Modeling Language) is a general-purpose modeling language, designed to support specification, analysis and validation for complex systems, which include hardware, software, information, personnel, procedures, facilities and equipments [39]. SysML provides graphical representations, with a semantic foundation, for modeling system requirements, behavior, structure and parametric, used to be integrated with other engineering analysis models. SysML is an extension for UML and adds elements to the models not entirely supported by regular UML. When a user model applies the SysML profile, only the UML metaclasses referenced by SysML are available to the user of that model. This is ensured by the semantics of the UML profiles. If the profile is not strictly applied, the UML metaclasses that were not explicitly referenced can also be available.

MARTE (Modeling and Analysis of Real-Time and Embedded Systems) is a profile that supports specification, design, and stages for verification and validation and also MDA development for real-time embedded systems [40]. Concepts introduced by this profile help modeling hardware and software relevant aspects of real-time embedded systems. Several extensions of MARTE profile allow performance analysis and modeling platform specific services. MARTE offers support for real time and embedded systems, starting from specifications, going to the detailed design and model-based analysis. MARTE profile provides several benefits such as a common way for modeling hardware and software aspects of real time embedded systems, with the benefit of improving communication between developers. MARTE enables interoperability between development tools used for specification, design, development, analysis, verification and code generation. MARTE guides the models that can be used in predictions for properties of real time and embedded systems, considering both hardware and software features. There are two important aspects: one to model the features of real-time and embedded systems and another to annotate application models, as to support analysis of system properties. MARTE profile is intended as a foundation for applying transformations from UML models to different analysis models.

SoC (System on Chip) [41] profile has as target SoC applications based on SystemC [42]. SoC are the chips that integrate computational and communication components. The components can be digital, analogue or mixed-signal. SoC profile provides some representation capabilities: hierarchical representation of the fundamental elements of SoC (modules and channels); roles of modules; information transferred between modules using only one type of diagrams. Most diagrams used for SoC are internal structure diagrams, extended by the profile [4]. In the absence of a SoC module, the diagrams have to be described by using the description of the class constructor function with sequence diagrams. This method of representation fails in explicitly describing the role for a module, which is fundamental information when analyzing SoC. This profile does not require the use of a specific language; SystemC is used only for exemplification. The goal lies in reusing UML as much as possible, and to define the profile as a minimal extension for modeling SoC designs.

There are several environments that allow application visual modeling using UML diagrams. Some environments, like Papyrus [43], offer support for different UML elements like: Package, Class, Component, Property, Primitive Type, Template Signature, Stereotype, Import Metaclass, Comment and Constraint. These UML elements can be used to form different UML models. Also, Papyrus UML offers support for UML profiles definitions and UML profiles application in UML models described using Papyrus. A strong point for Papyrus is that it has already integrated an OCL verifier, which can be used for syntactic verification of the constraints for the models, written in OCL language. However, a semantic verifier for the OCL constraints had to be developed, in order to validate the logical statements the OCL constraints brought in the model. Along with this semantic verifier, a syntactic verifier was developed. This syntactic verifier can currently check only some OCL reserved words, the ones used in the OCL constraints written for the UML profiles, but it can be easily extended to support other particular OCL terms. As the syntactic verifier contains only the terms used already in OCL constraints, there is no need to overload the system with the entire OCL syntax. A weak point for Papyrus is the absence of support regarding UML deployment diagrams, which are very important in designing and modeling the hardware specifications for applications for distributed embedded systems.

Another environment which was taken into consideration when choosing the most suitable UML modeling environment was UML2 Tools [44]. UML2 Tools is a set of Graphical Modeling Framework (GMF) - based editors for viewing and editing UML models. It is focused on automatic generation of editors for all UML diagram types. UML2 Tools offers support for creating several type of UML diagrams like: activity diagram, class diagram, component diagram, composite structure diagram, sequence diagram, state machine diagram and use case diagram. Also, UML2 Tools allows the creation of profile definition diagrams, used in UML models. The weak point of UML2 Tools, important in modeling hardware specifications for applications of embedded systems, is the fact that deployment diagrams are not complete. Instance specification and component elements are missing, which leads to incompletely defined deployment diagrams, with lack of accuracy.

Smart Development Environment for Eclipse (SDE for Eclipse) [45] was the choice for UML modeling of applications for distributed embedded systems. SDE for Eclipse provides a unified development platform to the Java developer, enabling him to capture requirements, design software, design database, generate code, implement software and generate reports. SDE offers support for defining UML profiles and also detailed deployment diagrams.

The table below summarizes some of the studied UML profiles, used for customizing several types of applications, some of them standardized by OMG. In general, these profiles have been defined for real time embedded systems or for system-on-chip devices. However, as CPSs are composed of sensors, actuators and communication devices, the customizations related to applications follow the same principles. The two defined UML profiles try to customize the hardware part and the software behavior, respectively. As in the SoC profile [41], stereotypes are required for customizing the hardware sensor and actuator units and devices. The intention is to avoid the user having low level programming skills, which is for example required when modeling applications using MARTE [40].

Approach	Starting Year	Summary
Model using SDL	2002	A system design flow for real-time and embedded systems, using the formal strengths of the system specification and design language (SDL) [46].
UML SystemC Profile	2002	A design flow for SystemC language, starting from UML model [47].
SysML	2003	SysML is an extension of the UML2.0 for modeling of system engineering applications [39].
HASoC	2003	A design methodology for the lifecycle modeling of embedded systems, which are, not exclusively, SoC implementations [48].
Play-Engine Tool	2003	Play-Engine Tool is an approach that supports UML-based development for embedded systems using formal techniques, timing annotations and formal semantics [49].
MARTE Profile	2006	UML Profile for Modeling and Analysis of Real-Time and Embedded Systems [40].
SoC Profile	2006	An UML profile for integration of components of a computing or communication system in a single chip [41].
Holistic System Modeling	2008	A holistic system modeling and refinement of intern-connected micro-electronic systems in a platform-based approach [50].
Methodology using xUML and MDA	2009	A system-level design methodology using Executable-UML and MDA concepts [51].
MoPCoM methodology	2010	An UML/MDA for designing high quality real-time embedded systems. VHDL code is automatically generated using MDA techniques [52].

Table 4 UML in embedded systems design

2.3.2 Existing Approaches in Cyber Physical System Design

A complex model was proposed in [47] and detailed in [53] by Riccobene et al. These papers represent years of research in improving the system-on-chip and embedded system design, by unifying the capabilities of UML and SystemC/C to operate at system level. Similarly to the goal-oriented approach described in the next chapter, the authors use MDA for reducing abstract and coarse-grained PIMs to concrete and fine-grained PSMs. UML profiles are the starting points for both researches, as they help defining PIMs in the form of UML models for the types of nodes in a network and for the entire network topology.

Similarly to the goal-oriented approach proposed in this thesis, the authors define in [53] a design methodology and development flow for the hardware part, by defining a SystemC related UML profile, for different levels of abstraction. The hardware UML profile describes stereotypes for SystemC components. Separately, a multithread C UML profile for modeling software applications is defined. The software UML profile relates to the hardware UML profile and acts as business logic descriptor for the elements introduced in the hardware section. This approach is similar to the one described in this paper in the general steps taken by the methods: specifications using UML profiles and MDA approach. However, here the author has concluded that SystemC is not the most appropriate programming language for the types of applications to which the new programming model is intended.

System modeling at higher levels of abstraction is supported by both approaches. Also, the authors introduce, for the software section described in [53], support for mutex locks, semaphores and threads mechanisms. A bidirectional transformation between the hardware and the software perspectives, in order to ease the mapping process has been developed. Consequently, a prototype tool was initiated for hardware-software co-design to eliminate the necessity of creating two different validation contexts.

The actual UML artifacts used to express the connections at hardware level and the logic at software level are: class diagrams, composite structure diagrams, object diagrams and state machine diagrams. However, although authors give detailed explanations for the design flow, they leave out the OCL constraints which complete stereotypes definitions. OCL constrains are of high importance in an UML model, as they allow the verification of the defined model in accordance to the used stereotypes. This verification can be made using an OCL general validation tool already defined or by defining a specific validation tool that parses only a part of the OCL language, directly involved in defining OCL constraints in a UML profile.

Transformations from UML models to SystemC have been considered in other papers as well. For example, Andersson and Host propose in [54] an automatic code generation method related to hardware interconnections, based on mapping rules between UML hardware profile and SystemC hardware descriptors. The UML hardware profile defined in [54] is similar to the one defined in [53]. The contribution for the authors is represented by the automation of the hardware interconnection specifications, in order to minimize the design effort. However, the initiative for hardware-software co-design in [53] already covers such aspects.

In [55], Nguyen et al. describe the communication infrastructure and timing features of SystemC using high-level UML modeling. As in the previous two described papers, communication primitives from SystemC, like interfaces and channels, were stereotyped. Clock sensitivity and timing constraints were also considered. Class diagrams and state diagrams were used to capture the internal behavior of embedded modules. The programming methodology the authors report allows describing executable platforms at UML level and also translation of applications described in UML to SystemC level. However, a major drawback of the proposal in [55] is the fact that the user must manually define a top level class for instantiating objects of already defined types. This has the effect of constraining the user behavior when specifying an application. Another drawback is that the model allows only one level of nesting in state diagrams.

The models presented above have in common the specification of hardware configurations for the types of nodes in a network and the communication between nodes in UML profiles. The software approach relates to another UML profile. The validation for the UML artifacts is made using automatic code generation for SystemC simulator. Although SystemC is a system level modeling language and allows simulation at different levels of abstraction, it was not the author's choice regarding validation of the „proof of concept" stated in the UML profile defined for hardware and software specification of applications using CPSs. The arguments that make SystemC not to be the perfect candidate for simulating applications for large distributed embedded systems are detailed in [56].

In [57], the authors propose an UML 2.0 profile for embedded system design. This UML profile defines stereotypes and design rules for application, platform and mapping. The profile classifies application and platform components, at the same time enabling their parameterization. The application is seen as a set of active classes with an internal behavior, UML modeled. The platform is seen as a component library with a parameterized presentation in UML 2.0 for each library component. This approach of parameterized component library is similar to the presented design methodology, where a middleware which recognizes the types of components used is required to be developed. The middleware ensures the correct functionality of the goals described by the users. Library functions are also made available, in order to help defining custom applications. The library functions ensure general application type specific requirements and constraints. The custom specific application goals and constraints are inserted by the user, and the middleware correlated with the specific application library is able to handle them.

When designing CPS applications, one of the main problems resides in complex interconnections management, although each component has clear specifications. The dynamic aspects of these components interconnections make the handling even more difficult. A recent solution in handling dynamic aspects of CPS subsystem and component is based on using Programmable System-on-Chip (PSoC) devices [58].

CPSs can be composed of PSoC devices, which integrate reconfigurable analog and digital circuits and memory. These devices are managed by an embedded microcontroller [59]. Although CPS design seems simpler using dedicated devices like PSoCs, the problem resides here in synchronizing PSoC devices for a homogenous functionality. At communication level, synchronization mechanisms are required in order to achieve cooperation between components in a CPS. More details about simulation models for CPSs, optimization methods and how the clock synchronization problem can be solved are discussed in subchapter 6.4, which summarizes the efforts made by the author, together with the research team, regarding CPS validation using simulation.

Developing suitable CPS modeling techniques represents a continuous challenge for researchers worldwide. Here are summarized some of the recent projects and approaches. The National Science Foundation (NSF) periodically organizes the CPS program through which it offers funding for research groups that consider CPS approaches. The considerable number of proposed projects states the growing interest in CPS challenges. Lee's studies ([1], [10]) define the main issues in CPS design that need to be addressed in a realist programming methodology. Projects like Ptolemy [29] propose a solution from design until simulation for CPS applications, using specific simulation tools. In this thesis, the objective is to keep the modeling independent of the specific deployment platform which will be used.

The main idea in the approach presented in this paper is to define a way to raise the level of abstraction in CPS applications. The user will deal with application requirements at the highest logical level and these goals will be translated into commands at lower logical levels, until the physical layer.

University/Research Group	Project Name	Starting Year	Summary
MathWorks	Simulink language	1984	Simulink is an environment for multidomain simulation and Model-Based Design for dynamic and embedded systems [60].
Electronic Design Automation Consortium	EDA	1989	Electronic Design Automation is a category of software tools for designing electronic systems such as printed circuit boards and integrated circuits [61].
TTTech Computertechnik AG (Vienna University of Technology)	TTP	1994	Time-Triggered Protocol (TTP) is an open and modular control system platform technology that supports the design of upgradeable, reusable and easy-to-integrate systems [62].
UC Berkeley	Giotto	2001	The Giotto system is a programming methodology for embedded control systems that can run on possibly distributed devices [63]. The Giotto system aims at hard real-time applications with periodic behavior.
UC Berkeley EECS Department	Ptolemy Project	2003	The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems [29]. The focus is on assembly of concurrent components.
IBM T.J. Watson Research Center	The Metronome	2003	A family of techniques that delivers bounded pause time garbage collections [64].
Vanderbilt University	C2WT	2006	Modeling and simulation environment for Command and Control systems (Command and Control Wind Tunnel - C2WT) [65].
Vanderbilt University	MURI Project	2007	Frameworks and Tools for High-Confidence Design of Adaptive, Distributed

			Embedded Control Systems MURI Project – on High-Confidence Design for Distributed Embedded Systems [66].
Carnegie Mellon University	PhyNet		The project intends to develop methods to monitor and ensure the robustness of Physical Networks, in the presence of contingencies [67]. This project is funded under NSF's Cyber-Physical Systems program.

Table 5 CPSs – List of ongoing projects

2.3.3 Summary

UML has the capability to unify the specifications of the applications with the design of the hardware and software parts of systems. This supports lightweight modeling, especially meant for early stages in design. Specification using UML allows a better understanding of the model and raises the level of abstraction the programmer must use. The PIM established for applications consists of deployment and component diagrams.

Studying the approaches in CPS design presented over the last years allows defining the open points and challenges in using such CPSs in different applications. Also, the various research directions are valuable, as they help identifying the possible solutions to the defined problems. In this paper, the author is focused on defining a design methodology, when starting from the application’s specifications. The goal is to be able to define the general requirements at a high level of abstraction and these to be translated to lower logical levels by the methodology.

2.4 Model Driven Architecture Approach

The increasing complexity of software over the years has determined the need for a higher level of abstraction. As before, models are used to reduce the gap between vision and implementation for a product. The goals are to detect the occurred errors and the missing parts in the early stages of the design and to conduct the implementation [68]. MDA is a software design approach in the form of a set of standards intending to conduct to a “model-based, standards-driven, and tool-supported” [69] handling of applications development. MDA was proposed in 2001 by OMG, a consortium focused on model-based standards.

There are several reasons for which MDA was chosen by the author of this thesis as a modeling approach for CPS applications. First, the MDA models allow a structural approach in application design. The possible customizations for the applications are grouped at CIM level.

These are used at PIM level, when constructing the application models. The transformations from PIM to PSMs allow automatic code generation and this is the final scope of using MDA.

2.4.1 Models and Transformations

The main concepts for the MDA are the models on which this architecture is based. These models are: the Computation Independent Model (CIM), the Platform Independent Model (PIM), the Platform Specific Model (PSM) and the Platform Model.

The CIM shows a certain system in the environment where it will evolve, without giving any details about the specific implementation. This model is realized by a business analyst that understands well the requirements for the class of applications and the business process [68]. UML is used as a modeling technique and the final product of CIM contains only basic class modeling and some behavioral modeling, in terms of use cases. The requirements are modeled in a more strict way than the previous business model. The functional requirements have to be expressed in a clear and unambiguous way. Therefore, a better knowledge of the UML artifacts and the design by contract is expected.

The PIM is a more detailed view of the system and contains more details but from a platform independent point of view [70]. Although this model contains information about behavior and business functionality, it does not inform about specific implementation details or platform. The previous model only states the goals the system must achieve. PIM is focused on how to achieve these targets. PIM is independent of information formatting technologies, possible programming languages to use, distributed middleware and messaging middleware [68]. A PIM usually consists of a class diagram and behavioral information. There can be a thin separation line between the PIM and the following model, the PSM, because the PIM can already contain some language specific concepts. As example, one can speak of multiple inheritances or the type of collections to use [68]. In the future, as more of the exemplified concepts will be placed in the underlying architectures, the PIM will become more abstract than today.

The PSM is a view of the system from specific platform point of view. The PSM contains the specifications described by the PIM but containing the details related to the concrete platform in use. The PSM should contain enough details as to allow code generation [70]. A modality to construct a PSM is to annotate the corresponding PIM with the technical details specific for the target platform. Another modality for obtaining a PSM is to use a generator, aimed to generate PSMs from PIM. If the PIM is based on a virtual machine, the regular transformations from PIM to PSM are no longer necessary.

The Platform Model contains the necessary technical information regarding a specific platform and the services it can provide. These concepts provided by this model are used in the PSM.

The final step in using MDA is the code generation from the PSM. The code generation should be as automated as possible, so modifications in previous step will not need human intervention for code. Transformation can be applied to the source of the code, the PSM, in order to obtain an optimal code.

The transformation between models can be done by mapping, which provides specifications for the transformations of a PIM to a PSM. There are different types of mapping. An example is the Model type mapping which defines how models built using the types defined in the PIM language can become types defined in the PSM language. A concrete example is metamodel mapping, where the types of model elements both in PIM and PSM are specified as Meta-Object Facility (MOF) metamodels. The mapping contains the rules and algorithms to produce the previous explained transformations. Another type of mapping is defined as Model Instance Mapping. The elements in the PIM that need to be transformed for the PSM are marked at first and then the rules of transformation to a PSM are applied to the marked elements. If a combination of the types of mapping is used, a more robust and detailed PSM can be created, as more information will be available.

The Record of transformation contains information about the creation of every PSM element from a source PIM element. This gives the ability to "round-trip around models" [70]. Sometimes the PSM can act as a PIM and will be refined into a more detailed PSM that will allow relevant code generation. At other times, the PIM can be directly translated into code if the source PIM contains enough information.

2.4.2 Using Model Driven Architecture in Embedded Systems

MDA approach, its definition, use, benefits and drawbacks have been over the past years subjects for workshops and conferences. An example of such a workshop, the discussed topics and drawn conclusions is presented in [69]. Some of the conclusions gathered during discussions were as follows. MDA is more an approach for software, than only a set of standards. The MDA tools existing today allow the generation of code from the posed models. Nonetheless, nonfunctional semantics such as security, performance, and reliability are not currently modeled or reasoned about. The strength of a software lies not only in the generated code, but also in the characteristics for such code, which goes "beyond generating infrastructure and code "skeleton"" [69].

One example of reliability support following the MDA principles is given in [71]. Rodrigues et al. propose a bottom-up model, followed by a relevant example for EJB applications to express reliability for MDA constructs. The steps to ensure reliability at PSM level can be summarized as follows: design of a reliability profile in case of specific application of interest; mapping between the specific domain already obtained and a platform-independent domain, resulting in a preliminary PIM; choosing a component model related to the application domain, in order to achieve a PSM to PIM model, with respect to the PIM reliability profile; automation of such mapping; providing future models for real-life case studies. The described model can be used mainly if the users are aware of the reliability mechanism supported by different platforms used in the application domains. This allows describing a suitable profile, in terms of reliability.

In [72], Fong presents a project for a bank that supports a variety of services and transactions. The goal of the project was to streamline and integrate the transactional messaging system with other systems (database storage, messaging system, legacy applications). MDA was the chosen approach for design and development. The author presents detailed steps for the PIM and the construction of the various PSMs and also some samples of generated code.

MDA can be used to "zoom in/out" different levels of abstraction for a system. A PIM can be "zoomed in" to find a corresponding PSM, while a specific model can be "zoomed out" to check the higher level of abstraction. Mos and Murphy use in [73] such an approach for MDA, for performance modeling and prediction of a framework that extracts performance information and automatically creates performance models written in UML.

In [74], Deelstra et al. relate MDA to a configurable software product family and observe some benefits to such an approach: for application engineering, binding time and selection mechanism is postponed to a further level of abstraction and the domain concepts, product family assets and the transformation technique run independently.

Reference [3] is a very useful example that shows in details mapping rules from PIM to a specified platform, in this case CORBA, which is also characterized by a profile. The PIM is characterized by a profile for Component-Object based Software Architecture, which contains components and connectors. The authors guide us step by step into transformation from PIM to PSM, using graphical and coding explanations.

2.4.3 Benefits and Drawbacks

The important benefits that result in using MDA show that, although being a relatively young approach, it has a clear future in software design. At first, the platform independency, resulted from PIM, and domain specificity, resulted from PSM are to be mentioned. The same PIM can be used to generate different PSMs, using different transformation rules. Therefore, the time, costs and complexity are reduced. MDA allows moving from one vendor to another without considering the application from scratch. The productivity as benefit must be mentioned, as in MDA the developers focus on UML diagrams that allow keeping the code up to date. Following the presented standards, one of the implemented goals for MDA was a proper documentation. This is a very useful part in keeping track of the state the application can be found, the various changes that have been made and allows the evaluation of software, in search for weak points. Consistency is ensured by the fact that complying with the OMG’s standards, the transformations are bidirectional.

MDA works well for business and data centric-applications, but for domains that require a large vendor support or a lot of tailoring, it did not prove its usefulness [69]. Also, MDA requires some more attributes like: the possibility to generate code for a great variety of platforms, the possibility of debugging at model level, therefore avoiding error propagation at the next abstraction level, an easier integration with the tools for analysis and validation. All of these would increase the role MDA has in software life cycle [69].

2.4.4 Summary

The CPS application design approach proposed in this thesis is based on a goal-oriented approach in conjunction with MDA methodology. It is based on a set of models which constitute the PIM and cover both hardware and software aspects in CPS applications. The PIM contains deployment and component diagrams, respectively. These models are further transformed into PSMs, which allows automatic code generation. Code generation is one of the main reasons for using MDA, as it reduces the effort to produce valid code, creates correctly, completely and consistently header files and offers a great support for model and document maintenance.

The UML defined profiles for hardware and software specification of CPS applications are the equivalent of the CIM in MDA approach. The UML profiles contain stereotypes that define the goal-oriented approach, more specifically the tailoring into logical layers and the handling of the application objectives at each of these levels.

The PIM is established using the defined UML profiles, the network topology defined by the programmer of the network, the expressed goals and constraints. The PIM correlates all the desired functionalities and deals with them at an abstract level, without taking into consideration the hardware limitations induced by the physical components of the network.

The PSM evaluates the defined PIM against the physical requirements and limitations of the specified hardware to be used. At this level of MDA, PSM is able to show the impact of physical hardware over PIM. Also, error cases are defined and signaled accordingly.

The following table summarizes some of the recent approaches in using MDA for embedded systems. The case studies presented in these approaches indicate that MDA is a suitable approach for designing and validating embedded and distributed systems.

Approach	Starting Year	Summary
MDA to design SoCs embedded systems	2003	A construction of metamodels based on MDA to support a co-design methodology.
An RTES metamodel	2005	A simulation-based method for the development of real time embedded systems, using MDA and separation of concerns (introducing communication, application and connection layer).
ModTransf	2005	ModTransf is an MDA Transformation engine that allows the application of MDA in the context of Intensive Signal Processing (ISP).
UML/MOF meta-modeling infrastructure	2006	A meta-data repository implementation, automatically providing an API suitable to the manipulation of the UML meta-models and models and to the implementation of design tools for embedded systems.
"On-the-fly" modeling of Musical Real Time Applications	2009	The method is based on the use of the Executable and Translatable UML approach (xtUML), with focus on modifying user PIM on-the-fly.
MDA for Embedded Devices	2010	Software design and code generation for a low-cost mobile phone.

Table 6 MDA in embedded systems

Chapter 3. Goal-Oriented Approach for Cyber Physical Systems

This chapter presents a goal-oriented approach regarding specification of requirements in CPS applications. The objective is to raise the level of abstraction in applications specifications and to ease the work for a designer of CPS applications. He is not required to have low level programming skills, as the set-up of the application is made at a high level of abstraction. For this to be possible, the CPS network is tailored into several levels of abstraction, which will be explained in the theoretical part of this chapter.

3.1 Theoretical Approach

CPSs integrate computational and physical processes and at logical level are tailored into several subsystems. Each of these subsystems aims to fulfill a specific task or objective and behaves unaware of the existence of the other subsystems. This characteristic determines one of the main issues in developing realistic CPS applications. The subsystems pose different objectives, even contradictory ones, over the controlled devices and these must be accomplished.

The goal-oriented approach proposed by Wang et al in [13] and continued by Magureanu et al. in [77] tries to handle the complexity of distributed applications, and in particular of CPS applications, by expressing the goals and constraints in a purely declarative way. There are no algorithmic or procedural constructs to define the steps to be taken for achieving the stated application goal.

The algorithms required for goals processing are named strategies and are selected from specialized, predefined libraries. These basic algorithms remain the same, independent of the applications, while the goals and constraints can change dynamically. Goals descriptions do not explicitly include the interaction mechanisms between distributed entities. The best interaction scheme is inferred automatically, based on the goals and constraints. This is the reason why also the logical tailoring is dynamically configurable.

Decision Modules (DMs), as introduced in [13], are the main entities in this goal-oriented approach, as they hold the entire logic for each layer, in each subsystem of the application. The DMs are composed of inputs, outputs, actual goals and constraints. The inputs are constituted from the physical data acquired for the applications, mainly from sensors. Inputs are a set of attributes, in the form of equations and constraints (invariants), composed using linear programming (LP). Specific examples of using LP systems are presented in chapter 6. The DM outputs are the physical outputs generated by the application, by solving the inputs system, to which constraints are applied. The DM constrains are the physical capability of the used platform (embedded nodes and communication infrastructure are included) and requirements of the application (like timing constraints or precision). The goals are mathematical expressions, involving inputs and outputs and need to be maximized or minimized.

The communication between levels of abstractions is maintained through DMs. Therefore, the DMs can produce a hierarchical structure, as the outputs of one DM become inputs to another DM. At each logical level, different execution semantics can be employed. At the lowest computational level, similar to the physical level of the controlled devices, reactive models like Finite State Machines (FSMs) are more suitable. At higher logical levels, less flexible models, but with a more predictable performance are used. These models offer a transition from the behavior at the embedded node level to the application level. Top-down and bottom-up constraint transformations are used to model the interactions between DMs at different logical levels. The top-down mechanism imposes constraints for the lower logical levels, so that the goals of the higher levels are met. The bottom-up mechanism identifies accordingly when the goals set by upper levels cannot be satisfied at lower levels.

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

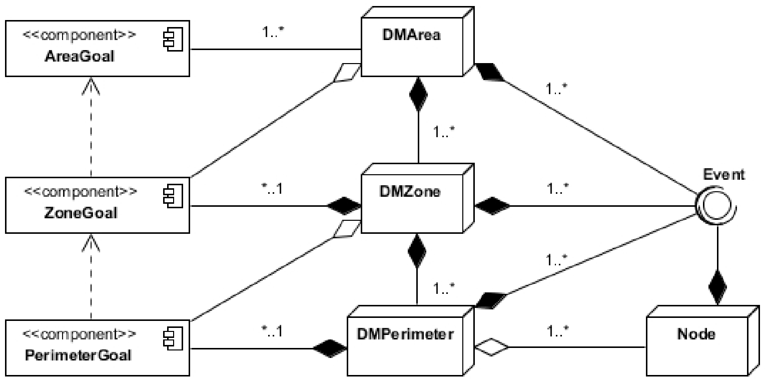


Figure 1 System DMs model

Considering the heterogeneous aspect of a CPS network, a node can be part of different subsystems at any given moment of time. Therefore, at each node level, several independent objectives have to be fulfilled.

In [76], the author of this thesis has proposed a system tailoring into several logical levels based on user goals. Furthermore, this tailoring applies to each subsystem of the CPS application. The goal management logic is handled within DM nodes. They are responsible for controlling the nodes within the logical grouping. Also, they ensure communication with DMs from upper and lower computational levels, as already stated.

At the highest level, each subsystem boundaries are specified. In goal-oriented terms, this represents the Area level and the objective is described in a general accepted form. At this level, the goal of the Area deals with the lower levels of abstraction, presented next. The entire business logic of the Area model is kept in a particular Decision Module Area (DMA) node.

The next logical level, named Zone, represents a subset boundary of the physical devices located inside the Area. The goal is managed as a specialization of the inherited Area goal over the internal managed devices. It adds more specific requirements regarding the implementation of the objective.

The implementation of the goal is hold in a Decision Module Zone (DMZ) node. The lowest computational level brings together local logically coupled hardware devices, like sensors and actuators, acting over the same environment "point". This represents the Perimeter level and consists of clearly stated management aspects for the physical devices. The objective implementation consists in well-defined execution blocks, describing the co-working manner in particularly identified cases. This logic is comprised in a Decision Module Perimeter (DMP) node and represents the most detailed implementation of the objective which needs to be fulfilled.

In this modeling methodology for CPS applications, a primary constraint related to logical tailoring is that the Perimeters are shared between the subsystems. The goals of the subsystems are expressed as a goal union at Perimeter level at every given moment of time. The result is represented by the set of commands that need to be achieved by the grouping level. The feedback obtained after executing the instructions at Perimeter level is communicated to upper logical layers, so that each subsystem will adjust its goals.

In case of Zones, the application requirements may determine a Zone to be shared among subsystems. In this case, the Zone's goals are unified as described in case of Perimeters. However, in this situation the managed Perimeters implied in fulfilling the goal are required to send the feedback only to this Zone. As the Zone is part of all the cyber-subsystems, it constructs feedbacks to all the Areas that contain the Zone in question, to update their main objectives.

Changing the current goal at subsystem level can be achieved using two different approaches. The first case implies a DMA of a particular subsystem that monitors a set of sensors. In the second case a DM node is responsible with monitoring a particular part of the environment.

For the first approach, the set of sensors is managed by other subsystems of the CPS containing the Area. Therefore, changing the goal requires identification of all involved Zones. Then, the Area level goal has to be translated into new specific goals for every DMZ. Each Zone operates on the Perimeters selected for satisfying the DMZ goal related to specific Perimeter goals. The DM node notifies the upper level in case it fails to fulfill the assigned goal. In that case, the upper level assigns a new goal to the lower level or cancels all the goals already assigned to the other DMs.

Each change in the sensed environment causes the node to trigger an event. The upper grouping level can react to it by processing the event without any other actions or by initiating a new notification to the higher level. If the upper level decides to consume the received event without changing its internal goal, no feedback will be sent to the triggering DM node.

When an upper level modifies the current goal, the event handling part has impact only on the environment part managed by it. The change of the goal at upper level induces change of lower level goals.

The actual usage of the CPS tailoring at logical level will be detailed in a case study for a fuel management system in an aircraft vehicle, in chapter 6.3. This example is not part of the current chapter, as further notions of MDA approach and defined UML Profiles will be introduced in chapters 4 and 5, respectively.

The practical example chosen for this goal-oriented related chapter presents goal optimizations for distributed control in traffic management applications.

3.2 Practical Example

The results presented in this subchapter are included in [77] (Magureanu et al.). The author of this thesis discusses here the main services for the supporting architecture in case of optimizing the goals for distributed control applications.

3.2.1 Goal-Oriented Control Model

The objectives for CPS applications can be stated mathematically as local or global constraints that need to be solved, so that the global cost function to be minimized or maximized.

This global cost function defines the main objectives for an application. The case study considered here is the one of intelligent distributed traffic management (Figure 2). The studies regarding traffic management were started in [78]. The first ideas referring to traffic management were already stated in [13]. Here, the described artifacts refer more to control procedures at different logical levels and their UML and MARTE representation.

The goals for the traffic management applications can be expressed as constraints. Expression (1) states that the total number of vehicles passing through the exit points of intersections is to be maximized. The set of exit points S_{exit} includes the locations of the modeled area at which vehicles can leave the area. Functions $N_i(t)$ express the number of vehicles over time passing through point *i*.

$$\max \sum_{i \in S_{exit}} \int^{\infty} N_i(t)dt \qquad (1)$$

Expression (2) defines a global timing constraint, available for all vehicles. The overall travel time of a vehicle through the area (until exiting at point *i* from S_{exit}) must be less than the maximum travel time T_{Max}. The Act_i indicates the set of activities that a vehicle must perform to reach exit point *i*. In this case study, the activities indicate the road portions (i.e. number of blocks) that the car must travel to the exit point. Time $T_{k,v}$ is the time that vehicle v needs to cover the road portion k (Figure 2(b)).

$$\max_{\forall v \in Veh, \forall i \in S_{exit}} \sum_{k \in Act_i} T_{k,v} \leq T_{Max} \qquad (2)$$

Expression (3) defines the overall precision of the control procedure. It indicates that the difference between the outcomes (e.g., the number N_i of passing vehicles) computed based on estimated and real traffic parameters must be less than bound ε .

$$\sum_{i \in S_{exit}} \int_{\Delta T} \left| N_i^{real}(t) - N_i^{optim}(t) \right| \le \varepsilon \qquad (3)$$

Expression (4) illustrates a local constraint, which requires that the waiting time for vehicle *v* at a certain traffic light *ts* must be less than a defined maximum value T_{Local}. Other local constraints, such as the maximum lengths of the queues at the traffic lights, can also be considered.

$$\max_{\forall v \in Veh, \forall ts} T_{v,ts}^{waiting} \le T_{Local} \qquad (4)$$

The constraints formulated in the previous equations are continuously solved, which implies continuously optimized decision making by the distributed control environment. This can become a difficult task for large-scale, distributed applications. Therefore, the optimization problem can be approximated by introducing a hierarchy of logical levels and the previous equations are expressed at these different logical levels. The detailed physical levels hold local interaction and the more abstract levels hold broad physical areas. This determines a good compromise between the frequency of updating the equations and the physical environment on which the data is collected.

For the constraints (1)–(4) to be solved by the control model, some approximations are introduced for time range and functions descriptions. Equation (1) is a cost function that considers infinite time range because the traffic management system is always on. However, this expression is difficult to implement as predictions about the future behavior have a limited accuracy. The mathematical form of the functions in (1) for the number of transited vehicles must be identical at runtime, as traffic conditions are dynamic. The identified function approximations differ depending on the number of parameters of the expressions, the mathematical expressions of the functions and the control variables, deciding when a new approximation is needed.

The global constraints, as expression (2) must also be approximated. The approximation for global constrains refer to constrain decomposition and to interactions. The amount of time required to achieve a global task, as for example to traverse a zone, depends on the structure of the road and the time needed to traverse each portion. The decomposition can be arbitrary, as some portions of the road are not defined based on physical properties (such as roads crossings or blocks), but are selected based on the required control precision (expression (3)).

In expressions (2) and (4), the time to travel for a car depends not only on the attributes of the individual car, but also on the interactions with other cars on the same road and with cars along intersecting lanes. As an example, a car can be slowed down by the slower cars and must wait at traffic light signals, as long as traffic lights on orthogonal paths have green light.

The constraints, expressed as equations, must be solved at different logical levels. The higher computational level is an area, corresponding to an entire region to be managed. The area contains zones (Figure 2 (a)) and the traffic flow inside each zone is defined by different scenarios with well-defined attributes (such as speed, inter-car distance). As in Figure 2 (b), each zone includes several traffic lights (*ITU*$_1$ and *ITU*$_2$). The control algorithm of a traffic light, as shown in Figure 2 (c), represents the lowest, physical level in the defined abstraction hierarchy.

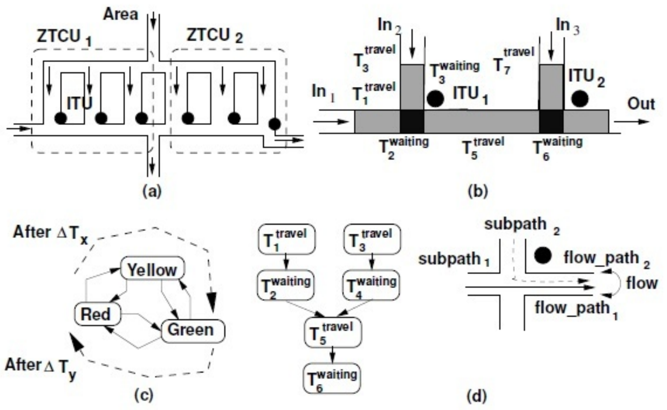


Figure 2 Overview of distributed traffic management [77]

Figure 3 presents the approximations for equations (1)-(4), on the hierarchy of abstractions.

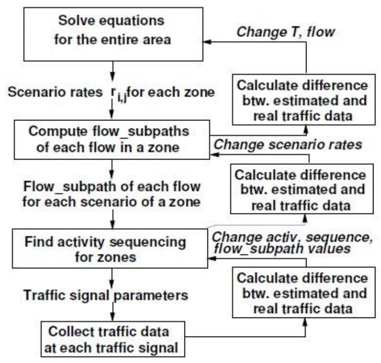

Solve equations for the entire area
Change T, flow
Scenario rates r i,j for each zone
Calculate difference btw. estimated and real traffic data
Compute flow_subpaths of each flow in a zone
Change scenario rates
Flow_subpath of each flow for each scenario of a zone
Calculate difference btw. estimated and real traffic data
Find activity sequencing for zones
Change activ. sequence, flow_subpath values
Traffic signal parameters
Calculate difference btw. estimated and real traffic data
Collect traffic data at each traffic signal

Figure 3 Distributed control optimization [77]

At the highest logical level, the area level, the infinite time range in equation (1) is approximated as a finite horizon *T* for which the optimal decision parameters are computed. Using the upper feedback loop in Figure 3, the value *T* of the time horizon is dynamically adjusted based on the dynamics of the traffic model parameters.

$$\sum_{i \in S_{exit}} \int^{\infty} N_i(t)dt \approx \sum_{i \in S_{exit}} \int_{T} N_i(t)dt \qquad (5)$$

Solving the model equations for the area level determines having different traffic scenarios at each of the composing zones.

For example, rate $r_{i,j}$ defines the fraction of time for zone *i* to have scenario *j.* As each scenario is characterized by different attributes values (for example, different number of vehicles exiting an exit point *i*), this step approximates the integral over time *T* of functions $N_i(t)$ (in expression (1)) as a sum of integrals defined over the constant vehicle flows corresponding to the scenarios. $flow_{i,j}^{veh}$ is the number of vehicles that pass per unit of time for scenario *j*:

$$\int_T N_i(t)dt = T\sum_j r_{i,j} \times flow_{i,j}^{veh} \tag{6}$$

The distributed control environment computes the differences between the estimated and the real traffic flow parameters (for example, parameters $flow_{i,j}^{veh}$). If the computed difference exceeds the acceptable limit of the precision constraint (3), a feedback is given to the upper level to suggest that the model equations must be re-solved for different flow parameters and time horizon *T*.

Equation (6) assumes the simpler situation, in which the scenarios of a zone are independent of each other. A more complex case is characterized by a causal dependency between the scenarios, such as reaching scenario *j* is possible from a scenario *k* with a probability of $p_{j,k}$. For example, reaching the fast moving traffic scenario is possible only after going through the traffic at average speed scenario. In this case, the behavior of the scenarios for zone *i* can be modeled as a Markovian Decision Process described by the following equations:

$$p_{i,j}r_{i,j} - \sum_k p_{k,j}r_{i,k} = 0 \tag{7}$$

$$\sum_j r_{i,j} = 1 \tag{8}$$

$$\sum_k p_{j,k} = 1 \tag{9}$$

$$p_{j,k} \geq 0, r_{i,j} \geq 0 \tag{10}$$

In equation (6), the $flow_{i,j}^{veh}$ parameters are estimated as follows. A scenario of a zone *i* results from aggregating multiple traffic sub-flows along different paths of the same zone. A path is a pair of points, the first is a point through which vehicles enter the zone and the second is a point through which vehicles leave the zone. In this case,

$$flow_{i,j}^{veh} = \sum_k flow\_subpath_{i,k,j} \times I_k \tag{11}$$

where *flow_subpath* expresses the amount contributed to scenario *j* of zone *i* by the zone's entry point k.

Figure 2 (d) presents two sub-flows, subpath1 and subpath2, which form together a scenario described by the flow attribute. In the case the sub-flow does not interact with other sub-flows then parameter I_k equals one. The other possible scenario is when the sub-flows interact with each other, for example as by sharing the same road infrastructure (intersections) or correlations with other vehicles (like the speed constraints due to the slower cars). In this situation the parameter I_k reflects the flow due to the interactions. The distributed control environment approximates parameters I_k as shown in Figure 3.

Parameters $flow\_subpath_{i,k,j}$ and I_k are estimated as follows. Each zone *i* can be decomposed into road portions *k* based on the position of the traffic signals (Figure 2 (b)). Each road portion is characterized by a set of parameters, such as the travel time $t_{i,k}^{veh}$ of a vehicle along the portion. The travel time $t_{i,k}^{veh}$ depends on the interaction with other vehicles and other flows. These interactions are approximated as follows in equation (12).

$$t_{i,k}^{veh} = \sum_s t_{i,k,s}^{travel} + \sum_t t_{i,k,t}^{waiting} \tag{12}$$

- $t_{i,k,s}^{travel}$ is the time required to a vehicle to travel the portion *s* of the zone *i*.
- $t_{i,k,t}^{waiting}$ is the time a vehicle has to wait at intersection *t* because of vehicles moving in the alternative direction.

As already stated in [13], the two time values $t_{i,k,s}^{travel}$ and $t_{i,k,t}^{waiting}$ can be estimated by performing a separate scheduling procedure for each zone, where the procedure assumes different possible situations for the queues of vehicles waiting at a traffic light. Then, parameters $flow\_subpath_{i,k,j}$ can be estimated using t^{veh}. Parameters I_k characterize the extra delay between t^{veh} and the same travel time, in case no interactions exist.

Figure 2 (d) presents the data flow graph defining the sequencing of the activities shown for the first intersection presented in Figure 2 (b). The two parallel sequences correspond to the two perpendicular roads crossing each other at the first intersection. The two waiting times $T_2^{waiting}$ and $T_4^{waiting}$ of vehicles moving along the two directions depend on the parameters of the traffic light FSM controller (for example time ΔT_x for state *Red* and time ΔT_y for state *Green* in Figure 2 (c). Therefore, optimizing the sequencing of the data flow graph activities to maximize the traffic volume defines the FSM parameters of the traffic light.

3.2.2 UML Support for Distributed Optimizations

The services for distributed control are summarized in Table 7. They are detailed for each logical level, starting with the physical level, in relation to abstractions from the same or upper levels. The four rows represent the four levels of the abstraction hierarchy: (a) at the physical level, FSM-based control for traffic signals, (b) at the zone level, activity sequencing using data flow graph scheduling, (c) at the zone scenario level, using Markov Decision Processes, and (d) at the area level using relations similar to (1)-(4). The three columns present the services required for a level to interact with entities at the same and lower levels, at the upper levels and to perform model approximation for distributed optimization.

Level	From Same or Lower Levels	To Upper Levels	Used for Approximation
Intersection traffic signal (ITU) (FSM, reactive)	(1) Identify traffic signals in a zone (2) Set traffic signal period, split (3) Set traffic signal # substates (4) Set timing for each direction (5) Set local constraints (e.g. (4))	(1) Send # vehicles in each direction (2) Send distribution + outliers of travel time (3) Send distribution + outliers of interval distances (4) Send change in parameter distribution (5) Send information to identify activities	(1) Communicate with traffic signals of the same ITU (2) Difference between expected and real # of vehicles passing by the traffic signal (3) Modify activities in a zone
Flow optimization inside zone (Data flow graph)	(1) Set activities and their sequencing (2) Set situations for each activity (3) Set parameters of different situations (4) Set expected queue lengths for situations (5) Set time ratios for competing activities (6) Set number of expected cars entering the zone (7) Set local constraints (e.g. (4))	(1) Send # of vehicles entering zone (2) Send # of vehicles exiting zone (3) Send queue lengths at traffic signals (4) Send waiting time at traffic signals (5) Send waiting time due to slow cars	(1) Modify description of activities (2) Modify situations for activities (3) Modify parameters for situations (4) Difference between expected and real # of vehicles passing through the zone
Zone scenario optimization (Markov Decision Processes)	(1) Set the scenarios for zone (2) Set scenario parameters ($flow_{i,j}^{veh}$) (3) Set transition problems between scenarios ($p_{i,k}$)	(1) Send scenario rates $r_{i,j}$ of zone	(1) Difference between expected and measured flow parameters $flow_{i,j}^{veh}$ (2) Difference between expected and measured flow rates $r_{i,j}$ (3) Communicate between *ZTCUs* of

			the same area
Area control optimization (linear)	(1) Set time horizon value T (equation 1) (2) Set exit points in set S_{exit} (3) Set descriptions of functions for # of vehicles $N_i(t)$ (4) Identify the zones of an area (5) Communicate the related constraints, e.g. from expression (2) to zones	-	(1) Adjust time horizon using received parameters (2) Adjust functions N_i (3) Send the allocated approximation error to zones

Table 7 Summary of distributed control services [77]

The services at the physical level include identifying the traffic signals that are defined in a zone, setting the parameters of a traffic signal controller (FSM) and setting local constraints. Examples of parameters for a FSM are period, split time, time range of states. A local constraint is related to the maximum time a vehicle should wait at the traffic signal. The services for interaction with the upper level (data flow graphs at the zone level) include transmitting the parameters required for setting-up and solving the scheduling algorithm or activity sequencing, for example statistical descriptions of the traffic in each direction, travel time, number of vehicles. Another type of services includes those for model approximation, such as monitoring the differences between the expected and real traffic parameters in the vicinity of the traffic signal, like the number of vehicles, travel time distributions.

The services for the FSMs in case of traffic signals can be described using UML diagrams, stereotyped with artifacts corresponding to MARTE profile. To specify the functionality of a traffic light, the author of this thesis considers only a simplified part from the intersection in Figure 2(b), composed of 4 traffic lights (*ITU*). The traffic lights are considered to function all in the same way, in couples, the one in the Nord (N) region with the one in the South (S) region, the one in the West (W) region with the one in the East (E) region of the intersection.

For each logical level, there is a part for interpreting the goals arrived from an upper computational level and transforming them into local goals and another part that handles the execution of the commands at that level of abstraction. The execution at ITU's level is reduced to proper coordination of traffic light states.

Each traffic light works following the same scheme from the FSM, the order of the states is known and the passing to the next state happens after a determined period of time, which is variable in the state. It can be considered that the duration of Yellow state is fixed and well-known.

MARTE::TimeLibrary contains the description of an *IdealClock*, a class stereotyped with *ClockType* and an instance *idealClk* of such "ideal" clock. Starting from these artifacts, chronometric clocks can be defined. To simulate the passing of time, a reference clock is required. This is *ChronometricClock* and is defined using MARTE stereotype *ClockType.* The attributes in the stereotype have specific values for the application, as can be seen in Figure 4.

The resolution is the minimum interval between two values. The offset is the temporal shift at the first instance. *isLogical* defines the existence of a logical or chronometric clock. *unitType* is of type *TimeUnitKind*. The time units can be continuous or discrete.

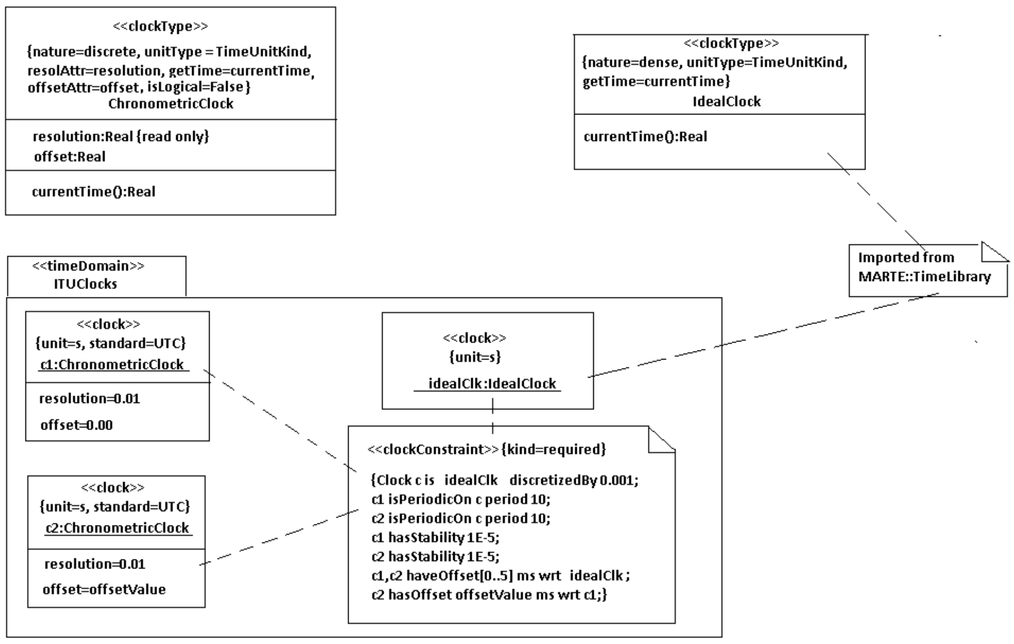

<<clockType>>
{nature=discrete, unitType = TimeUnitKind, resolAttr=resolution, getTime=currentTime, offsetAttr=offset, isLogical=False}
ChronometricClock
resolution:Real {read only}
offset:Real
currentTime():Real
<<clockType>>
{nature=dense, unitType=TimeUnitKind, getTime=currentTime}
IdealClock
currentTime():Real
Imported from MARTE::TimeLibrary
<<timeDomain>>
ITUClocks
<<clock>>
{unit=s, standard=UTC}
c1:ChronometricClock
resolution=0.01
offset=0.00
<<clock>>
{unit=s}
idealClk:IdealClock
<<clock>>
{unit=s, standard=UTC}
c2:ChronometricClock
resolution=0.01
offset=offsetValue
<<clockConstraint>> {kind=required}
{Clock c is idealClk discretizedBy 0.001;
c1 isPeriodicOn c period 10;
c2 isPeriodicOn c period 10;
c1 hasStability 1E-5;
c2 hasStability 1E-5;
c1,c2 haveOffset[0..5] ms wrt idealClk;
c2 hasOffset offsetValue ms wrt c1;}

Figure 4 Reference clock description in UML [77]

The constraints between clocks are defined using Clock Constraint Specification Language (CCSL). There are two constraints for the chronometric clocks against the ideal instance. Other constrains can be expressed relative against another clock using offset parameter *offValue*.

A model in MARTE that contains clocks makes possible the association of time intervals to behaviors in an UML model. Therefore, the UML behaviors including sequence diagrams can be stereotyped using *timedProcessing*. The main result in this association is that is offers the possibility to the designer of the applications to express temporal characteristics and constraints in behaviors. In MARTE, these constraints are *timedValueSpecification*, which represents a duration in which the behavior must execute.

The behavior of a traffic signal is better described as an UML sequence diagram, as shown in Figure 5.

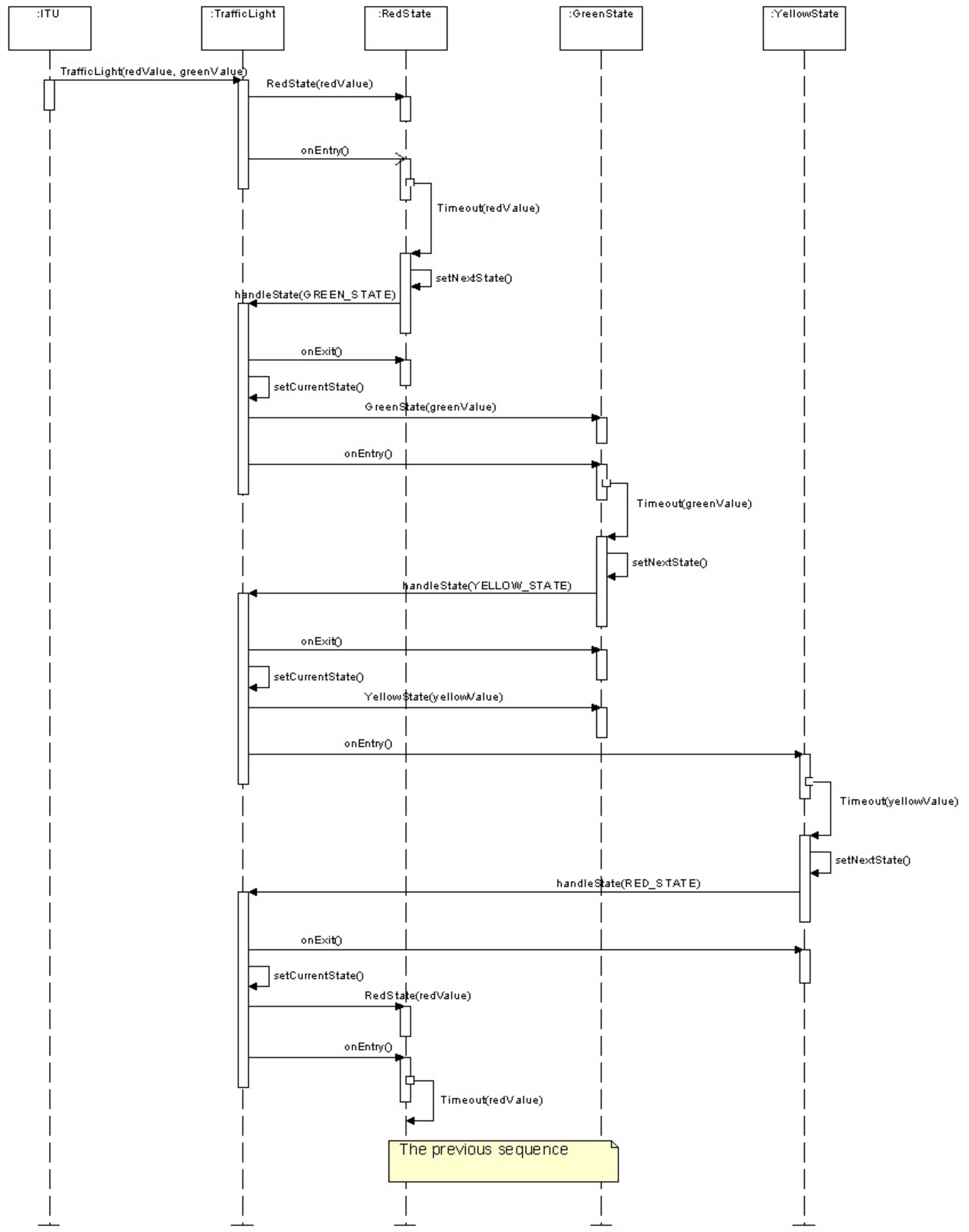


Figure 5 UML sequence diagram for traffic signal [77]

This diagram is stereotyped using *timeProcessing* and the attributes specific to this stereotype are expressed as follows.

- *clock* attribute defines the reference clock for measuring the time intervals and events, like *c1* for traffic lights in the N and S and *c2* for traffic light in W and E
- *start* attribute defines the event that triggers execution
- *stop* attribute defines the event when ending execution.

Every state instance (for Red, Green and Yellow states) is stereotyped using *timedDurationConstraint*. For each state instance, the attributes for the specified stereotype will be defined:

For instance *RedState*:
<<timedDurationConstraint>> {period = 0.01, *deadline* = *redValue on c1*}
For instance *GreenState*:
<<timedDurationConstraint>> {period = 0.01, deadline = greenValue on c1}
For instance *YellowState*:
<<timedDurationConstraint>>{period = 0.01, *deadline* = *yellowValue on c1*}

The attributes for *timedDurationConstraint* can be expressed as above for traffic lights in N and S, for example. In case of traffic lights in W and E, the deadline depends on *c2.* The time spent in a certain state is indicated in the sequence diagram by *Timeout(values)* function.

Figure 6 presents the data flow graph for flow optimization inside a zone. This corresponds to the second row in Table 7. Inside a zone, information acquired at the traffic signal level (ITUs in Figure 2 (b)) is used to set-up the parameters of the nodes in the data flow graphs, for example the execution and waiting times of the nodes. Information about the number of vehicles entering the zone, through In_1, In_2, and In_3 is used in the analysis to estimate the expected queue lengths at the traffic signals. The results for this flow graph are the timing parameters for the traffic light inside the specified zone, like the time length of Red, Yellow and Green states.

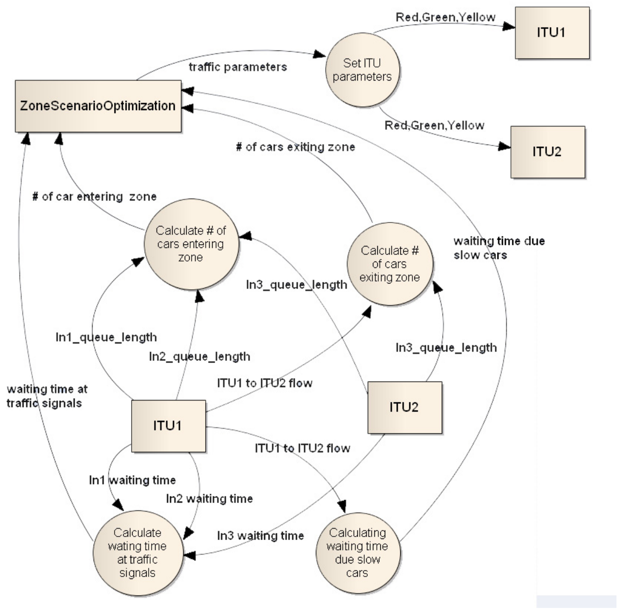


Figure 6 Data flow-graph level services inside zone [77]

Figure 7 illustrates the UML statechart for the services which correspond to the third row in Table 7 regarding zone scenario optimization using Markov Decision Processes (MDP). The description is based on the method presented by Jansen et al. in [79]. For simplicity, the shown MDP is for a zone with traffic moving along two, single-lane directions, for example N-S and W-E directions. There are several possible scenarios: fast-moving traffic in both directions, average-speed traffic and slow moving traffic, including when traffic is stopped. The transition probabilities between MDP states represent predictions on "switching" between the corresponding scenarios. The probabilities, for example *PN* fast, *PN* slow, are computed based on the information collected at the FSM and flow optimization levels (in first two rows in Table 7), for example, waiting time at traffic signal, number of cars entering zone, and number of cars exiting zone.

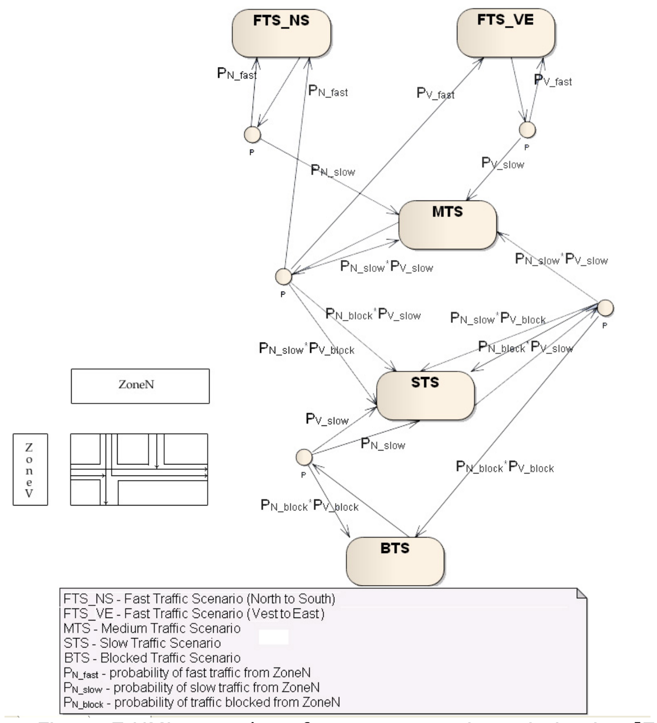

FTS_NS
FTS_VE
MTS
STS
BTS
ZoneN
ZoneV
PN_fast
PV_fast
PN_slow
PV_slow
PN_slow*PV_slow
PN_block*PV_slow
PN_slow*PV_block
PN_block*PV_block
P
FTS_NS - Fast Traffic Scenario (North to South)
FTS_VE - Fast Traffic Scenario (Vest to East)
MTS - Medium Traffic Scenario
STS - Slow Traffic Scenario
BTS - Blocked Traffic Scenario
PN_fast - probability of fast traffic from ZoneN
PN_slow - probability of slow traffic from ZoneN
PN_block - probability of traffic blocked from ZoneN

Figure 7 UML statechart for zone scenario optimization [77]

3.3 Summary

The increased interest in CPS applications determines the need for a design methodology, in which the user's effort to be minimal. The goal-oriented approach of handling the application's requirements and goals by using the tailoring into several logical levels, along with MDA approach, allows an efficient, intuitive and easy to use design methodology in CPS applications.

A CPS application consists in fulfilling a set of independent objectives. Each objective is assigned to a heterogeneous subsystem. These subsystems are characterized by the fact that the devices are logically coupled and they collaborate to meet the specific goal. Therefore, in the presented thesis, the entire network is tailored into goal-oriented interconnected devices. The resulting models constitute the PIM of the system and hold the application's specifications. From this step, more details about the MDA approach considered here are required. They will be given in chapter 4.

Based on the proposed tailoring into computational levels, the practical example in this chapter discusses the main services of the architecture for real-time optimization of the goals of distributed CPS applications. The case study considered is of a large-scale traffic management. UML/MARTE descriptions are presented for some of the main services, at each hierarchical level. The services address an optimization problem that includes both global (at higher levels of abstraction) and local constraints (at physical level) with some parameters being collected in real-time over distributed, physical areas. For scalability, the distributed control scheme implements a hierarchical approximation scheme. Decisions at the physical level are local and decisions at higher levels refer to larger geographical areas, where the model parameters change slower. The proposed services are based on a well-defined, mathematical model for control optimization. This is useful in dependable control, as the performance of decision making can be reasoned out based on the model.

Chapter 4. MDA Approach for Cyber Physical System Design

The methodology presented in this thesis is a visual one in order to help the users in easier and more intuitively defining their applications, with a minimal effort.

When discussing the users' perspective, they are able to use different types of already defined UML components for CPS networks. These components can be used by simply customizing them depending on application needs, or can be composed into more complex components, taking into consideration communication and links between components, limitations and requirements. The components and connections between them form the hardware specifications for an application. For the software specifications, the users specify the behavioral requirements for the applications, in form of goals for the highest logical level in the system. Goals at higher logical levels are translated into goals at lower logical levels. These lower computational levels have also an important role in specifying the software behavior of the application, as their fulfillment or lack of accomplish influences goals at higher computational levels.

The users can test and validate their defined UML models for hardware and software management for a CPS network. The UML models related to hardware are defined for the types of nodes in the network and for entire network topology and customized for the requirements of the applications. The UML models related to software represent the high-level software specifications, in form of goals for the applications. These models are tested by generating specifications for a simulation environment regarding hardware and software configuration of the network. The correctness and completeness of the UML models defined by the user is inspected by checking the OCL constraints previously defined and attached to UML models, through used stereotypes. In case of success, the users test their applications in a simulation environment that allows them to verify the defined network and goals for the applications. In case testing and validation is proven unsuccessful, the users must return to the high-level UML specifications and correct them. After several steps from UML modeling to testing in simulation environment, if necessary, the specifications for the applications are validated. Then, the users can deploy the network in a physical environment, having the certainty that hardware modeling errors and at least some behavioral errors have been already resolved.

This means that the visual design and programming model are able to generate simulation code for validation of the desired applications. Hardware validation implies network topology, the nodes contained in the network and the communication between them. Software validation implies the goals for the application, the requirements and limitations translated from UML high-level modeling. After the validation of the application in simulation environment, the approach allows generating efficient business-logical code, which is loaded on the physical nodes. The physical network is able to collaborate with the generated code and is acting as close as possible to the desired behavior. The generated code considers the hardware limitations of the physical components which define the network.

The specific application library helps the visual design and programming model in generating the necessary drivers for the hardware components of the network.

When discussing from the developers of components perspective, for high-level UML modeling, some UML profiles are first defined. An UML profile for defining the hardware for CPS network components is developed. This profile is detailed in chapter 5.1 and has been introduced by Magureanu et al. in [80]. The UML hardware profile allows defining particular standard elements based on stereotypes. A set of stereotypes defines node specific requirements like localization or type of communication. These defined stereotypes contain specific tagged values and OCL constraints, for expressing the customizations required for the further stereotyped UML elements. Well defined stereotypes for specifying the hardware components of a network help achieving a clear separation of devices and grouping into families of hardware devices. The work in [80] was continued by Magureanu et al. in [81], with defining customizations for wireless communication between nodes in a CPS network.

Another UML profile is defined for software specification of CPS applications. This has been introduced by Gavrilescu et al. in [82] and detailed in [83]. It contains a set of stereotypes which define the goal-oriented approach, by specifying logical levels to which the node belongs to, the reflection of the goals from higher levels to lower logical levels, and the extension of the goals from the lower logical levels in a collection of goals derived from the higher logical levels.

The UML goals definitions consider the customized expression for each goal and should cover any goal that is possibly to occur. UML goals definitions are handled in a common manner by the visual design and programming model. The design methodology handles requirements for several types of CPS applications and has been tested on three types of applications: management of traffic systems in intersections, management of gas distribution systems and fuel management in a portion of an aircraft vehicle, the fuel system.

Studies have indicated that using UML defined stereotypes for specifying different requirements for applications help improving the overall understanding of the models in question. Such a study is detailed by Kurniarz et al. in [5].

Using the defined UML hardware profile, it is possible to start a distributed embedded application design with clear specifications regarding network topology, the types of nodes participating in its construction and the hardware units involved in specifications for the families of nodes used. It is the task of the developers of the components to define customization for the possible types of nodes, for the network topology and other hardware units based on UML hardware profile. These customized components are placed in a library that can be accessed by the users in order to define their applications.

In order to achieve this design methodology, a middleware which recognizes the types of components used is required. The middleware ensures the correct functionality of the hardware components of the network and the software components, in the form of goals to be completed by the users. The library functions are available to the users, in order to help defining custom CPS applications. The library functions ensure general application requirements and constraints. The custom specific hardware requirements and application goals and constraints are inserted by the users, and the middleware correlated with the specific application library functions is able to handle them.

Also, the developers of the predefined components for the CPS applications should ensure the verification of the users defined applications in a simulated environment.

They provide the tools for checking the OCL constraints for the UML user defined models. After validation of the UML models, a code generator ensures the generation of specifications for hardware and software requirements in the simulation environment. After validating the simulated application, the visual design and programming model is able to generate executable code, which can be deployed on the hardware network.

The goal-oriented method proposed by the author of this thesis uses MDA approach for the design of CPS applications. The steps required to be accomplished by the developers of the components and the users which define the applications can be translated into the models described by the MDA approach. MDA has been chosen as it is a rigorous approach, that allows creating specific models for each development phase for embedded distributed applications in general, and CPS applications in particular. The CIM holds the UML defined profiles, which are used to customize the PIM defined for hardware and software specifications. Also, the PIM can be tested and validated, by verifying it in a formalized manner and by simulating the models and correcting the encountered errors.

Each of the models that characterize the MDA approach has well-defined objectives. The CIM is specified in the visual programming model. It indicates a certain system in the environment where it will evolve, without giving any details about the specific implementation. In the present thesis, the CIM is used to describe a particular family of CPS applications. It defines types of nodes, network variables, internal node strategies, the network boundaries and requirement abstractions. The author of this thesis has defined two UML profiles (in [80], [81], [82] and [83]), one for hardware and interconnection constraints and the other for software specification and internal business logic. These profiles are used to design CPS application CIM.

The PIM is defined using the defined UML profiles, indicating the network topology defined by the user, the expression of the functional and non-functional requirements, the types of nodes required by the user and their behavior. The objective of PIMs is to correlate all the desired hardware and software functionalities and to deal with them at an abstract level, without taking into consideration the hardware limitations inducted by the actual physical components. The PIM makes use of deployment diagrams, component diagrams and CIM to correlate all required hardware and software functionalities. It is based on a hierarchy of diagrams, starting from the network interconnections between all nodes and continuing with tailoring based on application goals. Each grouping level has attached its own deployment diagram containing lower grouping levels, as deployment nodes. These diagrams depict also internal constituents of all system nodes as customized components and relationships between them. Component diagrams define behavior of system nodes. They are attached to deployment diagrams. A node internal behavior and strategies is specified using component stereotypes. They enable mapping of external customized goals to internally recognized format. Then predefined external events are assigned to nodes.

At this time, the stub application defined by the user is able to be compiled by the visual design and programming model at a static level, in order to detect the lack of correlations between goals and network topology, or between goals themselves, expressed by OCL constraints.

The PIM can be tested and validated before deployment into a physical network, on a PSM. A possible and recommended validation methodology is testing in a simulated environment.

Using directly a physical network as PSM implies some risks because undetected errors before deployment may lead to the increase of costs for the application and also may slow down the application. It is more time consuming to repeatedly load the application on physical network after correcting some errors than is solving the errors and then loading the application on physical devices. Error cases should be defined and signaled accordingly.

Designing CPSs for physical networks involves significant challenges. At the same time, simulation must be as realistic as possible in order to obtain best results from simulating a network before deployment in a physical environment. CPSs can consist of distributed devices, each of them having its own internal clock and being able to operate at a different clock frequency than another device. Although there are clear specifications for each component of CPSs, the main issue in designing this type of systems resides in the management of complex interconnections. Dynamic aspects for the interconnections increase the difficulty of the stated problem. Proposed techniques for concurrent programming of massively distributed embedded systems raise the same problem [84]. A recent solution for handling dynamic aspects of CPS is based on Programmable System-on-Chip (PSoC) devices [58].

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, and this leads to using a small number of components for each specific task [59]. PSoC devices support a wide range of communication protocols.

Using PSoC technology for designing CPSs applications does not solve all the problems. There are still open issues regarding a correct temporal semantic for all concurrent processes involved [85]. 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 [86], [87], [88] and [89] and are detailed in chapter 6.

Chapter 5. UML Profiles for Cyber Physical System Applications

UML approach allows high level-modeling of PSoC based CPS applications. It allows static and behavioral descriptions of distributed applications. The UML models allow the user to define several hardware and software parts for embedded system applications. Each hardware component has assigned a specific behavior based on the application specifications. The hardware specifications for the network topology and components, along with the behavior of the components in an application are later used as input for simulation and, after testing and validation, as input for application deployment. Simulation on a simulation environment and deployment of the network on physical devices are considered PSMs, which can be obtained from the same PIM of network topology, network components and network behavior, from high logical levels to low logical levels.

Using UML modeling, the user can define the structure of a network, first as a black box of hardware components, connected through each other, depending on the established connection paths, and then at a lower computational level, each type of node in the network, with its internal components, connected for being able to exchange messages. The UML models for hardware description consist of UML predefined hardware components. For these customized components, the author of this thesis has defined an UML hardware profile. This is detailed in subchapter 5.1. The components behavior at lower logical levels and the network behavior at the highest logical level compose the UML software profile. The stereotypes for software description correspond to the ones for hardware description. The UML software profile is detailed in subchapter 5.2.

Studies have shown that using defined stereotypes in UML specifications and customizing the models with stereotypes help improving the overall understanding of the UML models in question [5]. This is a desired goal, as it 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.

5.1 UML Profile for Hardware Specification

The stereotypes defined in this UML hardware profile, along with the tagged values and the OCL constraints, are used in defining the hardware components, which instances are actually used in modeling the application. Well defined stereotypes for specifying hardware components help achieving a clear separation and grouping between families of devices.

Although UML profile appliance is similar with the inheritance relationship, there are some important specific features when using UML profiles. The instance-of relationship is not transitive [92]. This means that when a stereotype is applied to a certain element, the tagged values are applicable only to that element and not to instances of that element. Here, for the hardware specification of a network, deployment diagrams are required and therefore stereotypes which extend metaclass *Node (Deployment)* are used. The tagged values corresponding for all nodes stereotyped with a specific stereotype are defined in the stereotype of metaclass *Node (Deployment)*, and have the same value, independent of the instances created. At the same time, if it is necessary for a tagged value to have different values on instances, this tagged value must be defined in a stereotype which extends metaclass *Instance Specification.* The OCL constraints required here imply that when a node is stereotyped with a certain stereotype of metaclass *Node (Deployment)*, each instance of that node must be stereotyped with the corresponding stereotype of metaclass *Instance Specification*. The correspondence is made using naming convention. For example for a stereotype *ADC_HWST* of metaclass *Node (Deployment)*, the corresponding stereotype for metaclass *Instance Specification* is *ADC_HWST_Instance*. In this way, the node will have some specific characteristics, general for all instances, given by the stereotype of *Node (Deployment)* and each instance will be able to customize other characteristics, given by the stereotype of metaclass *Instance Specification*.

5.1.1 First Level Stereotypes

For high-level hardware configuration of nodes in a network and for the entire network, it is necessary to use deployment diagrams to specify the hardware requirements for the components of an application. Instance specifications of nodes 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 connection 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 on the stereotype for the node, which specifies 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 first level hierarchy of stereotypes which extend the *Node (Deployment)* metaclass is described visually in Figure 8.

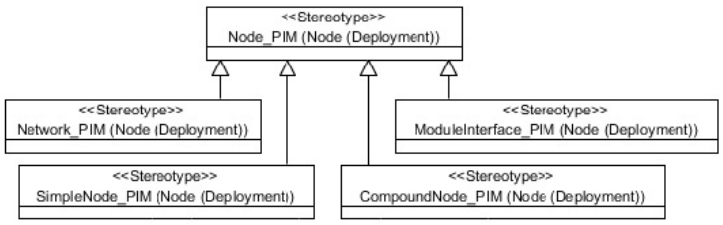

<<Stereotype>>
Node_PIM (Node (Deployment))
<<Stereotype>>
Network_PIM (Node (Deployment))
<<Stereotype>>
ModuleInterface_PIM (Node (Deployment))
<<Stereotype>>
SimpleNode_PIM (Node (Deployment))
<<Stereotype>>
CompoundNode_PIM (Node (Deployment))

Figure 8 First level Node (Deployment) stereotypes

- *Node_PIM* is a stereotype which extends Node *(Deployment)* metaclass. It is an abstract stereotype, base for all stereotypes of *Node (Deployment)* in the UML hardware profile.
- *Network_PIM* stereotype is the customization for each network, as part of an application. The networks can contain wired or wireless connections, depending on the type of communications between nodes, as components of the network.

- *SimpleNode_PIM* stereotype is used for customizing a unit in a network which contains only attributes and ports. This stereotype is mainly used for grouping purposes, more specialized stereotype being 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, 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.

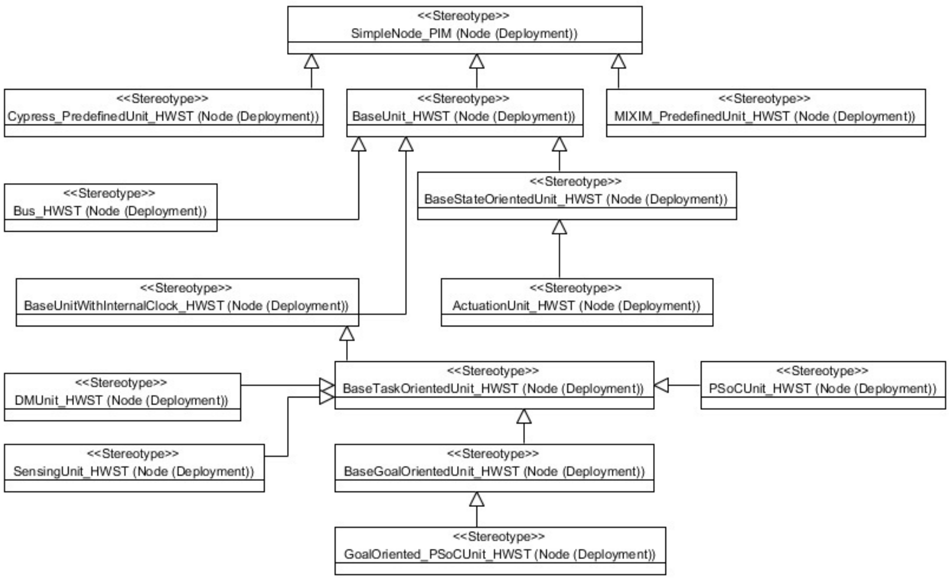
5.1.2 Simple Modules Stereotypes

Cypress_PredefinedUnit_HWST, *BaseUnit_HWST, MIXIM_PredefinedUnit_HWST* are first level abstract stereotypes for simple units.

- *Bus_HWST* stereotype customizes a hardware component which facilitates communication between components in a distributed wired network, acting as a regular data bus.
- *PSOCUnit_HWST* stereotype is a relevant stereotype derived from *SimpleNode_PIM* stereotype. It defines the PSoC device contained by the hardware node.

Most distributed embedded systems are composed of nodes which contain communication, sensing, actuation and decision units. This is the reason stereotypes like *SensingUnit_HWST*, *ActuationUnit_HWST* or *DMUnit_HWST* were created, as shown in Figure 9. As for communication in embedded systems networks, this can be wired or wireless. Wired communication is described by stereotypes grouped in *Communication_HWST* stereotype, derived from *Cypress_PredefinedUnit_HWST* [93]. Wireless communication uses stereotypes created using as model the MiXiM project for wireless sensor networks [94].

The base units for nodes in a network can require an internal clock or not. This is the reason for layering the stereotypes, depending on the internal clock, for example using *BaseUnitWithInternalClock_HWST* stereotype. Actuation units do not require an internal clock, therefore *ActuationUnit_HWST* stereotype is not derived from stereotype *BaseUnitWithInternalClock_HWST.* Decision units, sensing units and PSoC units require an internal clock, therefore the corresponding stereotypes generalize *BaseUnitWithInternalClock_HWST* stereotype. At the same time, some units can be found in different states, while other units execute tasks. Therefore the layering into components customized with *BaseStateOrientedUnit_HWST* and *BaseTaskOrientedUnit_HWST* stereotypes is required. Goal oriented units are derived from task oriented units. Corresponding stereotypes are also in a generalization relationship. In case of goal-oriented applications, the PSoC unit requires a customization, realized with *GoalOriented_PSoCUnit_HWST* stereotype.



<<Stereotype>> SimpleNode_PIM (Node (Deployment))
<<Stereotype>> Cypress_PredefinedUnit_HWST (Node (Deployment))
<<Stereotype>> BaseUnit_HWST (Node (Deployment))
<<Stereotype>> MIXIM_PredefinedUnit_HWST (Node (Deployment))
<<Stereotype>> Bus_HWST (Node (Deployment))
<<Stereotype>> BaseStateOrientedUnit_HWST (Node (Deployment))
<<Stereotype>> BaseUnitWithInternalClock_HWST (Node (Deployment))
<<Stereotype>> ActuationUnit_HWST (Node (Deployment))
<<Stereotype>> DMUnit_HWST (Node (Deployment))
<<Stereotype>> BaseTaskOrientedUnit_HWST (Node (Deployment))
<<Stereotype>> PSoCUnit_HWST (Node (Deployment))
<<Stereotype>> SensingUnit_HWST (Node (Deployment))
<<Stereotype>> BaseGoalOrientedUnit_HWST (Node (Deployment))
<<Stereotype>> GoalOriented_PSoCUnit_HWST (Node (Deployment))

Figure 9 Simple nodes stereotypes

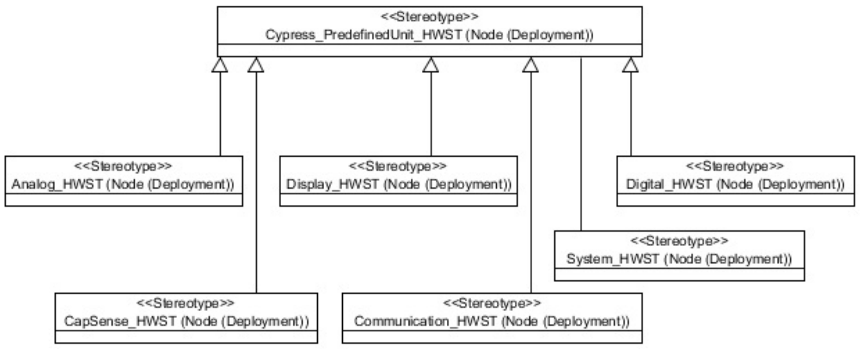
5.1.3 Stereotypes for PSoC based Cyber Physical Systems

PSoC devices integrate reconfigurable analog and digital circuits, all of them managed by an embedded microcontroller. PSoC devices contain also memory and programming circuits and support a wide range of communication protocols. Such 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 time spent on designing the system. PSoC devices are characterized by low power consumption. The overall costs for the system are reduced. The reconfiguration capabilities allow to the designer of the system to connect internal resources on the fly. This leads to usage of a reduced number of components for each specific task. The reduced number of components, the low power consumption and the reconfiguration capabilities in PSoC devices make them suitable for being used as CPS devices. The main challenge in designing PSoC based CPS applications resides in fulfilling a complex interconnection management.

PSoC Creator is an IDE provided by Cypress Semiconductor Corporation [93], suitable for developing PSoC based applications. PSoC Creator offers support for loading the user application into device and therefore it helps creating a PSM starting from the PIM based on UML stereotypes. It is also possible to describe the applications using only PSoC Creator and then loading them on the device, but by using MDA, it is possible to specify only a set of PIMs in UML high-level modeling from which several PSMs can be created: one for OMNeT++ simulation environment and another one for deployment on physical network, with the facilities offered by PSoC Creator. Therefore, for performing a realistic simulation and for a precise mapping between PIMs constructed using UML modeling and PSoC Creator IDE, all hardware configuration possibilities that can be realized on PSoC Creator side must be possible also in UML models. This is the reason for defining stereotypes for all analog and digital predefined devices on PSoC Creator side. These stereotypes have a common base stereotype named *Cypress_PredefinedUnit_HWST*, which is derived from *SimpleNode_PIM* stereotype. *Cypress_PredefinedUnit_HWST* stereotype was defined for a better hierarchical view regarding the types of stereotypes contained by UML hardware profile.

Each PSoC Creator predefined component unit is expressed in UML models as a simple node and requires gates for interconnecting with other elements in the models. Each hardware component has associated a detailed technical data sheet, which expresses the component's functionality and its terminals, therefore the gates do not have to be described in detail by the UML defined stereotypes. A naming convention for these can is used. The naming convention is constructed from the terminal name with a *Gate_HWST* suffix attached, as a result having stereotypes: *InputGate_HWST*, *OutputGate_HWST* and *InOutGate_HWST*.

There are some most relevant groups of stereotypes for the most frequently used hardware components in distributed network design, available in PSoC Creator and in UML hardware profile. The hierarchy for these groups of stereotypes is presented more explicitly in Figure 10. Most stereotypes are abstract and cannot be directly applied to UML models for hardware components because their role is to represent the groups of stereotypes with similar functionalities. The information regarding features for hardware components and their descriptions are taken from PSoC Creator Component Data Sheets documents.



<<Stereotype>>
Cypress_PredefinedUnit_HWST (Node (Deployment))
<<Stereotype>>
Analog_HWST (Node (Deployment))
<<Stereotype>>
Display_HWST (Node (Deployment))
<<Stereotype>>
Digital_HWST (Node (Deployment))
<<Stereotype>>
System_HWST (Node (Deployment))
<<Stereotype>>
CapSense_HWST (Node (Deployment))
<<Stereotype>>
Communication_HWST (Node (Deployment))

Figure 10 Stereotypes group for PSoC based CPSs

Analog_HWST is an abstract stereotype indicating that the stereotypes that use this stereotype as base stereotype customize analog hardware components. The derived stereotypes from Analog_HWST stereotype are: *Amplifier_HWST*, *Mutex_HWST*, *ADC_HWST*, *DAC_HWST*, *Comparator_HWST*, *Mixer_HWST*.

CapSense_HWST stereotype defines a group of hardware devices able to measure the capacitance for applications, such as touch-sensing buttons, sliders and proximity detectors.

There are two hardware elements for capacitive sensing documented in PSoC Creator: CapSense and CapSense CSD. CapSense component supports different combinations of independent and slider capacitive sensors; supports parallel and serial scanning configurations; offers guided slot and terminal assignments using the CapSense Configure dialog. Capacitive sensing systems can be used in applications in place of conventional buttons, switches and other controls, even exposed to hard weather conditions. As for the types of applications in question, here can be included ATMs, outdoor equipment, automotive systems, public access systems, portable devices (cell phones and PDAs), and kitchen and bathroom home appliances. CapSense CSD supports combinations of buttons, sliders, touch pads and proximity capacitive sensors, defined by the user. Capacitive sensing provides an efficient capacitance measurement for different types of applications.

- *Communication_HWST* abstract stereotype represents the units responsible with communication on different buses and protocols. The derived stereotypes from *Communication_HWST* stereotype are: *CAN_HWST*, *I2C_HWST*, *I2S_HWST*, *SPI_HWST*, *UART_HWST* and *USBFS_HWST*.
- *Digital_HWST* abstract stereotype defines a group representative for hardware components acting as logic devices, registers of functions. The derived stereotypes from *Digital_HWST* stereotype are: *Logic_HWST*, *Register_HWST* and *Function_HWST*.
- *Display_HWST* abstract stereotype represents the group of display modules. The derived stereotypes from *Display_HWST* stereotype are: *CharacterLCD_HWST*, *GraphicLCD_HWST* and *SegmentLCD_HWST*.
- *System_HWST* abstract stereotype groups other system digital components stereotyped with: *BusConvertor_HWST*, *Clock_HWST*, *Temperature_HWST*, *DMA_HWST*, *EEPROM_HWST*, *RealTimeClock_HWST*, *SleepTimer_HWST*, *Synchroni-zationBlock_HWST* and *OverClock_HWST.*

As a general observation regarding the method stereotypes for Cypress hardware components were defined: the constraints associated with these stereotypes express mainly the hardware requirements for the stereotyped components. In case for all components from a certain sub-group of some main abstract group, a port is mandatory, this will appear as an OCL constraint, associated with the corresponding stereotype. In case a port is optional for components from certain sub-groups, a tagged value with prefix *bool_define_* will be defined for the corresponding stereotype. This tagged value is transformed into a parameter on a PIM, for example to validate it on OMNeT++ simulator. In case the parameter changes the default *false* Boolean value given to the corresponding tagged value in the UML PIM, then the stereotyped hardware component must define a port with a specific name and functionality. This is also expressed through an OCL constraint. The constraints, both their natural expressions and the OCL expressions, are grouped in a table later in this chapter.

For a sub-group represented by a stereotype, there can be more possible components, defined in PSoC Creator, for example *ADC_HWST* stereotype can be used for customizing two different analog components: Delta Sigma Analog to Digital Convertor or ADC Successive Aproximation Register. A tagged value *ADC_HWST_Type* is used to indicate which of these two possible components is referred. An OCL constraint is defined for this purpose and it helps defining a restriction for the exact type of components that can be customized using *ADC_HWST* stereotype.

The stereotypes derived from *Analog_HWST* stereotype are presented visually in Figure 11.

- *Amplifier_HWST* stereotype is mapped to the amplifier hardware units. There are four different types of amplifiers described in PSoC Creator documents: Inverting Programmable Gain Amplifier (PGA_Inv), Operational Amplifier (Opamp), Programmable Gain Amplifier (PGA) and Trans-Impedance Amplifier (TIA). Which one of these components is used in the hardware description for PSoC based applications is specified by *string_Amplifier_HWST_Type* tagged value from

Amplifier_HWST_Instance stereotype. The specification for the actual component used from the amplifiers' group is helpful for the software description on the PSM. The *Amplifier_HWST* stereotype defines only a mandatory output port, *Vout*, and no optional ports.

- *Mutex_HWST* stereotype describes different multiplexer units, which can be constructed in PSoC Creator. The multiplexers described in PSoC Creator documents are Analog Hardware Multiplexer (AMuxHw), Analog Multiplexer (AMux), Analog Multiplexer Sequencer (AMuxSeq) and Virtual Mux. There are no optional ports for *Mutex_HWST* stereotype. The mandatory input port named *inputs* is specified using an OCL constraint.
- *ADC_HWST* stereotype customizes the components providing analog to digital conversion. There are two hardware elements for analog to digital conversion documented in PSoC Creator: Delta Sigma Analog to Digital Convertor (ADC_DelSig) and ADC Successive Aproximation Register (ADC_SAR). The optional ports for this group of components are: *minusInput*, *soc* and *aclk*. In case it is required for these ports to be activated, this must be specified in the UML PIM of the node customized with *ADC_HWST* stereotype by setting the values of the Boolean variables *bool_define_minusInuputPort*, *bool_define_socPort* and *bool_define_aclkPort*, respectively, to *true*.
- *DAC_HWST* stereotype customizes different types of digital to analog convertors, described in PSoC Creator: 8-Bit Current Digital to Analog Converter (IDAC8) and 8-Bit Voltage Digital to Analog Converter (VDAC8). The optional input ports are *strobe* and *data,* specified in the UML PIM for the node customized with DAC_HWST stereotype by setting to *true* the Boolean values for the variables *bool_define_strobePort* or *bool_define_dataPort*, respectively. There are no mandatory ports defined by *DAC_HWST* stereotype.
- *Comparator_HWST* stereotype is used for specifying comparator hardware units. This stereotype uses OCL constraints to define the mandatory ports: *PositiveInput* and *NegativeInput* as input gates and *ComparatorOut* as output gate. The optional input port is *clock*, with the corresponding *bool_define_clockPort* value.
- *Mixer_HWST* stereotype customizes mixer hardware units which are components that can be used for frequency conversion of an input signal with a sampling clock obtained from a fixed Local Oscillator. The mandatory ports defined by *Mixer_HWST* stereotype using OCL constraints are *Fin*, *LO* si *Vref*, as input ports and *Fout* as output port.

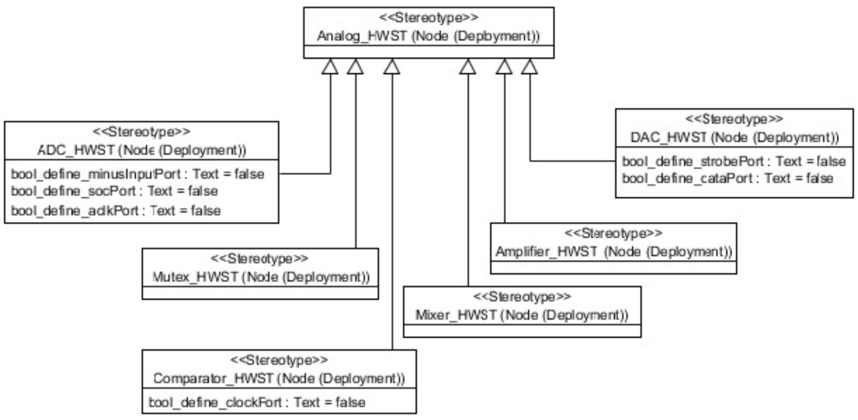

<<Stereotype>>
Analog_HWST (Node (Depbyment))
<<Stereotype>>
ADC_HWST (Node (Deployment))
bool_define_minusInpuPort : Text = false
bool_define_socPort : Text = false
bool_define_adkPort : Text = false
<<Stereotype>>
DAC_HWST (Node (Deployment))
bool_define_strobePort : Text = false
bool_define_cataPort : Text = false
<<Stereotype>>
Amplifier_HWST (Node (Deployment))
<<Stereotype>>
Mutex_HWST (Node (Deployment))
<<Stereotype>>
Mixer_HWST (Node (Deployment))
<<Stereotype>>
Comparator_HWST (Node (Deployment))
bool_define_clockFort : Text = false

Figure 11 Analog stereotypes

The stereotypes derived from *Communication_HWST* stereotype are presented visually in Figure 12. The naming convention for the defined stereotypes allows an easy recognition for the customized buses and protocols, as described in PSoC Creator documents.

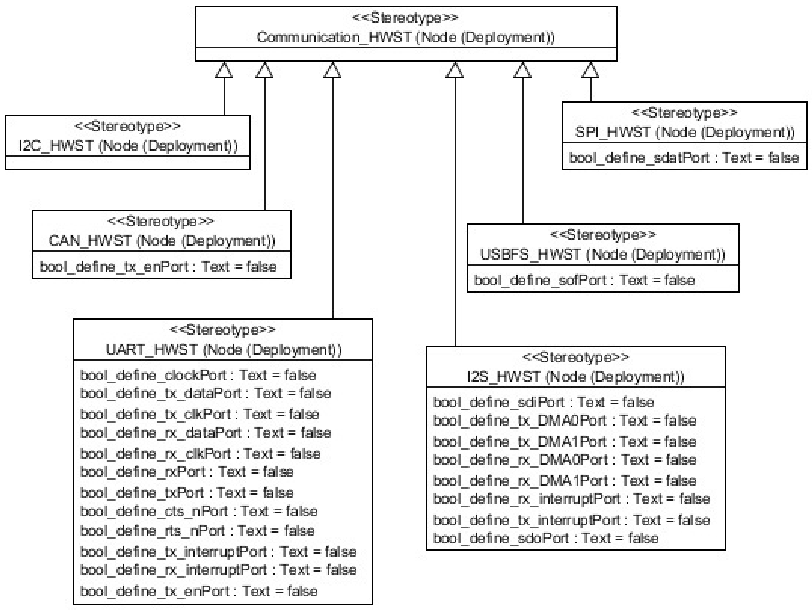

<<Stereotype>>
Communication_HWST (Node (Deployment))
<<Stereotype>>
I2C_HWST (Node (Deployment))
<<Stereotype>>
SPI_HWST (Node (Deployment))
bool_define_sdatPort : Text = false
<<Stereotype>>
CAN_HWST (Node (Deployment))
bool_define_tx_enPort : Text = false
<<Stereotype>>
USBFS_HWST (Node (Deployment))
bool_define_sofPort : Text = false
<<Stereotype>>
UART_HWST (Node (Deployment))
bool_define_clockPort : Text = false
bool_define_tx_dataPort : Text = false
bool_define_tx_clkPort : Text = false
bool_define_rx_dataPort : Text = false
bool_define_rx_clkPort : Text = false
bool_define_rxPort : Text = false
bool_define_txPort : Text = false
bool_define_cts_nPort : Text = false
bool_define_rts_nPort : Text = false
bool_define_tx_interruptPort : Text = false
bool_define_rx_interruptPort : Text = false
bool_define_tx_enPort : Text = false
<<Stereotype>>
I2S_HWST (Node (Deployment))
bool_define_sdiPort : Text = false
bool_define_tx_DMA0Port : Text = false
bool_define_tx_DMA1Port : Text = false
bool_define_rx_DMA0Port : Text = false
bool_define_rx_DMA1Port : Text = false
bool_define_rx_interruptPort : Text = false
bool_define_tx_interruptPort : Text = false
bool_define_sdoPort : Text = false

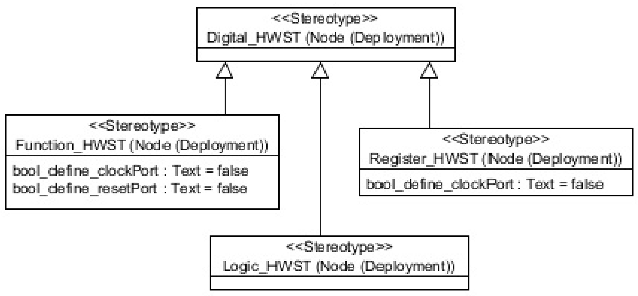
Figure 12 Communication stereotypes

- *CAN_HWST* stereotype is used for integrating Controller Area Network (CAN) devices which implement a standard communication protocol for motion oriented machines controlled networks (CANOpen) and factory automation applications (DeviceNet). *CAN_HWST* stereotype defines a mandatory input port *rx* and a mandatory output port *tx* using OCL constraints. The optional output port *tx_en* is specified using the *bool_define_tx_enPort* Boolean variable.
- *I2C_HWST* stereotype is used for configuring I2C hardware components which support I2C slave, master and multi-master configurations. The differentiation is made using *I2C_HWST_Type* tagged value, defined in *I2C_HWST_Instance* stereotype. The I2C hardware components implement the industry standard I2C Bus, compatible with other third-party devices. *sda* and *scl* are the mandatory ports defined in *I2C_HWST* stereotype, which can have role of both input and output gates.
- *I2S_HWST* stereotype helps constructing the Integrated Inter-IC Sound Bus (I2S) serial bus standard interface, used for interconnecting digital devices. The *I2S_HWST* stereotype defines two mandatory output ports, *sck* si *ws*, and a large number of optional ports. For the optional ports, the corresponding *bool_define_* prefixed tagged values are visible in Figure 12.
- *SPI_HWST* stereotype is used for customizing the Serial Peripheral Interface (SPI) components, similar to the ones described in the PSoC Creator document. The SPI components can be master or slave. The *reset* input port is defined as

mandatory in the *SPI_HWST* stereotype. The input-output optional port is *sdat*, with the corresponding *bool_define_* prefixed variable.

- *UART_HWST* stereotype describes the Universal Asynchronous Receiver Transmitter (UART) hardware unit which provides asynchronous communications. The UART component can be configured for Full Duplex, Half Duplex, Rx only or Tx only version. This explains the large number of optional ports in the *UART_HWST* stereotype, for which, the attached *bool_define_* prefixed tagged values are visible in Figure 12.
- *USBFS_HWST* stereotype is used in the definition of a Full Speed USB (USBFS) component, which provides a USB full speed compliant device framework. The *USBFS_HWST* stereotype provides an optional output port, named *sof.*

The stereotypes derived from *Digital_HWST* stereotype are presented visually in Figure 13.



<<Stereotype>>
Digital_HWST (Node (Deployment))
<<Stereotype>>
Function_HWST (Node (Deployment))
bool_define_clockPort : Text = false
bool_define_resetPort : Text = false
<<Stereotype>>
Register_HWST (Node (Deployment))
bool_define_clockPort : Text = false
<<Stereotype>>
Logic_HWST (Node (Deployment))

Figure 13 Digital stereotypes

- *Logic_HWST* stereotype is used for devices that cover binary logic operations like: logic high '1', logic low '0', and, or, nand, nor, not, xor, xnor, virtual multiplexer, de-multiplexer, lookup table, multiplexer, three state buffer and digital value keeper. The exact type for the device is expressed by *Logic_HWST_Type* tagged value, defined in *Logic_HWST_Instance* stereotype.
- *Register_HWST* stereotype is used for customizing hardware components like control or status registers. The exact type for the device is expressed by *Register_HWST_Type* tagged value, defined in *Register_HWST_Instance* stereotype. The stereotype defines an optional input port, named *clock*.
- *Function_HWST* stereotype is used for tagging a hardware device which provides one of the following functions: counting for events, cyclic redundancy check from a given serial bit stream, pseudo random bit stream generation based on internal linear feedback shift register, pseudo random sequence generation based on linear feedback shift register, pulse width modulator for generating single or continuous control signals, data shift in and out of a parallel register or interval timing between hardware events [80]. All devices which implement these functions are already defined in PSoC Creator tool. The exact type for the device is expressed by *Function_HWST_Type* tagged value, defined in *Function_HWST_Instance* stereotype. The stereotype defines optional input ports, named *clock* and *reset*.

The stereotypes derived from *Display_HWST* stereotype are presented visually in Figure 14 and define different display configurations predefined in PSoC Creator tool.

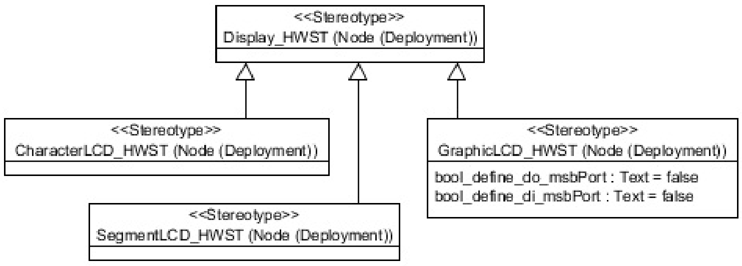

<<Stereotype>>
Display_HWST (Node (Deployment))
<<Stereotype>>
CharacterLCD_HWST (Node (Deployment))
<<Stereotype>>
GraphicLCD_HWST (Node (Deployment))
bool_define_do_msbPort : Text = false
bool_define_di_msbPort : Text = false
<<Stereotype>>
SegmentLCD_HWST (Node (Deployment))

Figure 14 Display stereotypes

- *CharacterLCD_HWST* stereotype is used for customizing a component similar to the one created in PSoC Creator which can create user defined custom characters and horizontal and vertical bar graphs. The mandatory port is named LCD_Port; the description is available in PSoC Creator documents.
- *GraphicLCD_HWST* stereotype is used for customizing different types of graphic LCD display devices, detailed in PSoC Creator: Graphic LCD Interface (GraphicLCDIntf), Graphic LCD Controller (GraphicLCDCtrl). The Graphic LCD Interface (GraphicLCDIntf) component provides the interface to a graphic LCD controller and driver device. The Graphic LCD Controller (GraphicLCDCtrl) component provides the interface to an LCD panel that has an LCD driver, but not an LCD controller. c*lock*, *di_lsb* and *do_lsb* are mandatory ports, more detailed described in technical data sheets, while *di_msb* and *do_msb* are optional ports, their presence on the stereotyped component being established by the value of the Boolean tagged value *GraphicLCD_HWST_Type*.
- *SegmentLCD_HWST* stereotype is used for customizing different types of segment LCD display devices, detailed in PSoC Creator: Segment LCD (LCD_Seg) and Static Segment LCD (LCD_SegStat). The type of the component is given by the value of the *SegmentLCD_HWST_Type* variable.

The stereotypes derived from *System_HWST* stereotype are presented in Figure 15. They are representative for bus convertor, clock, temperature, DMA, EEPROM, real time clock, sleep timer, signals synchronization and over clock component units, respectively.

- *BoostConvertor_HWST* stereotype is used for describing a hardware component which provides the ability to configure and control the PSoC boost converter hardware block.
- *Clock_HWST* stereotype is used for describing a hardware component which provides two main features: it provide the means to create local clocks and it provides the means to connect designs to system and design-wide clocks. This stereotype defines a mandatory port named *clock* and an optional port named *digitalDomain*.
- *Temperature_HWST* stereotype is used for describing a hardware component which provides an API to acquire the temperature of the die.
- *DMA_HWST* stereotype helps customizing a hardware component which allows data transfer to and from memory, components and registers. The *DMA_HWST* stereotype defines one mandatory output port and two optional input ports.

- *EEPROM_HWST* stereotype is used for describing a hardware component which provides and API to write a row of data to the EEPROM.
- *RealTimeClock_HWST* stereotype is used for describing a hardware component which provides accurate time and data information for the system.
- *SleepTimer_HWST* stereotype is used for describing a hardware component which can be used to wake the device from Alternate Active and Sleep modes at a configurable interval. It can be configured to issue an interrupt at a configurable interval; therefore the SleepTimer_HWST stereotype defines a mandatory output port, named *interrupt*.
- *SynchronizationBlock_HWST* stereotype is used for describing a hardware component which resynchronizes a set of input signals to the rising edge of the clock signal. Two mandatory input gates and a mandatory output gate are identified.
- *UDBClock_HWST* stereotype is used for describing a hardware component which supports precise control over clocking behavior. The stereotype defines two mandatory input gates and a mandatory output gate.

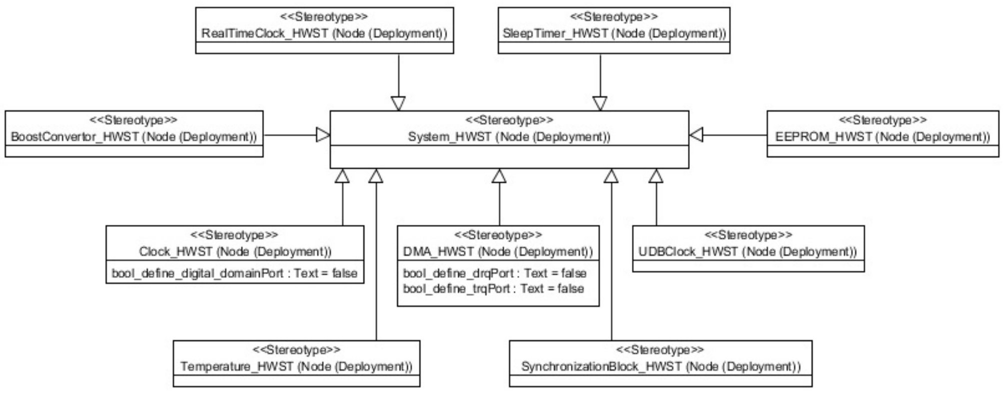

<<Stereotype>>
RealTimeClock_HWST (Node (Deployment))
<<Stereotype>>
SleepTimer_HWST (Node (Deployment))
<<Stereotype>>
BoostConvertor_HWST (Node (Deployment))
<<Stereotype>>
System_HWST (Node (Deployment))
<<Stereotype>>
EEPROM_HWST (Node (Deployment))
<<Stereotype>>
Clock_HWST (Node (Deployment))
bool_define_digital_domainPort : Text = false
<<Stereotype>>
DMA_HWST (Node (Deployment))
bool_define_drqPort : Text = false
bool_define_trqPort : Text = false
<<Stereotype>>
UDBClock_HWST (Node (Deployment))
<<Stereotype>>
Temperature_HWST (Node (Deployment))
<<Stereotype>>
SynchronizationBlock_HWST (Node (Deployment))

Figure 15 System stereotypes

5.1.4 Stereotypes for MiXiM

For UML high-level describing of applications that use wireless communication between components, MiXiM project [94] was used as example. MiXiM is an OMNeT++ modeling framework created for mobile and fixed wireless networks, such as wireless sensor networks. It offers different models of wireless communication. In this thesis, the author defines stereotypes for supporting wireless communication in CPS networks, based on the MiXiM models. The defined stereotypes are grouped in Figure 16.

- *MIXIM_PredefinedUnit_HWST* abstract stereotype is used for grouping the stereotypes used for customizing the components similar to the one in MiXiM project.
- *BaseLayer_HWST* stereotype is a generalization for *BaseMacLayer_HWST* and *BaseNetwLayer_HWST* stereotypes, used for customizing component units for basic nodes.

- *ConnectionManager_HWST* stereotype is used for describing the module that coordinates the connections between nodes and handles dynamic gate creation.
- *BaseWorldUtility_HWST* stereotype customizes a module which provides utility methods and information used by the entire network, as well as simulation wide black board functionality.
- *BaseBattery_HWST* stereotype is used for customizing the battery module for a node.
- *BaseMobility_HWST* stereotype is used for customizing the mobility module for a node. This module handles the physical localization for the node.
- *BlackBoard_HWST* stereotype is used for customizing the module that provides black board like information exchange between the other modules of a host.

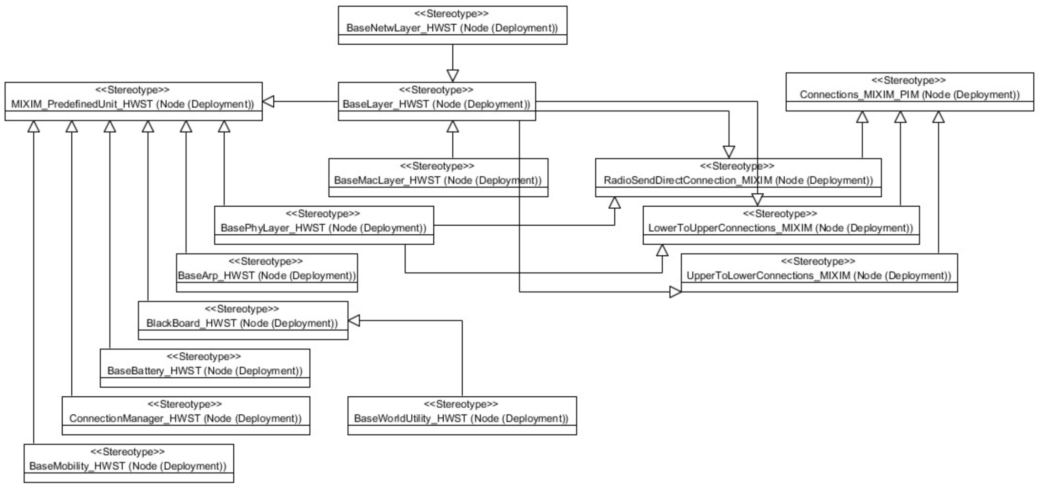

<<Stereotype>>
BaseNetwLayer_HWST (Node (Deployment))
<<Stereotype>>
MIXIM_PredefinedUnit_HWST (Node (Deployment))
<<Stereotype>>
BaseLayer_HWST (Node (Deployment))
<<Stereotype>>
Connections_MIXIM_PIM (Node (Deployment))
<<Stereotype>>
BaseMacLayer_HWST (Node (Deployment))
<<Stereotype>>
RadioSendDirectConnection_MIXIM (Node (Deployment))
<<Stereotype>>
BasePhyLayer_HWST (Node (Deployment))
<<Stereotype>>
LowerToUpperConnections_MIXIM (Node (Deployment))
<<Stereotype>>
BaseArp_HWST (Node (Deployment))
<<Stereotype>>
UpperToLowerConnections_MIXIM (Node (Deployment))
<<Stereotype>>
BlackBoard_HWST (Node (Deployment))
<<Stereotype>>
BaseBattery_HWST (Node (Deployment))
<<Stereotype>>
ConnectionManager_HWST (Node (Deployment))
<<Stereotype>>
BaseWorldUtility_HWST (Node (Deployment))
<<Stereotype>>
BaseMobility_HWST (Node (Deployment))

Figure 16 MiXiM units stereotypes

- *Connections_MIXIM_PIM* stereotype provides the customizations for the connections for a module of a node to the upper layers, the lower layers of with the exterior of the node.

5.1.5 Module Interfaces Stereotypes

The stereotypes defined for this group are shown in Figure 17.

- *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. At the same time, simple modules can implement an interface, instead of extending other modules.
- *INic_HWST* stereotype is used for customizing an interface for network interface card.
- *IBaseMobility_HWST* stereotype is used for customizing an interface which is implemented by modules responsible with mobility in a node.
- *IBaseCommunicationUnit_HWST* stereotype is used for customizing the interface that must be implemented by communication units in a network.

- *IBaseNetwLayer_HWST* stereotype is used for customizing an interface for network layer modules.
- *IBaseApplLayer_HWST* stereotype is used for customizing an interface for application layer modules.
- *IBasePSoCLayer_HWST* stereotype is used for customizing an interface used for the PSoC layer for an application.
- *IBaseGoalOriented_PSoCLayer_HWST* stereotype is used for customizing an interface used for the PSoC layer for goal-oriented applications.

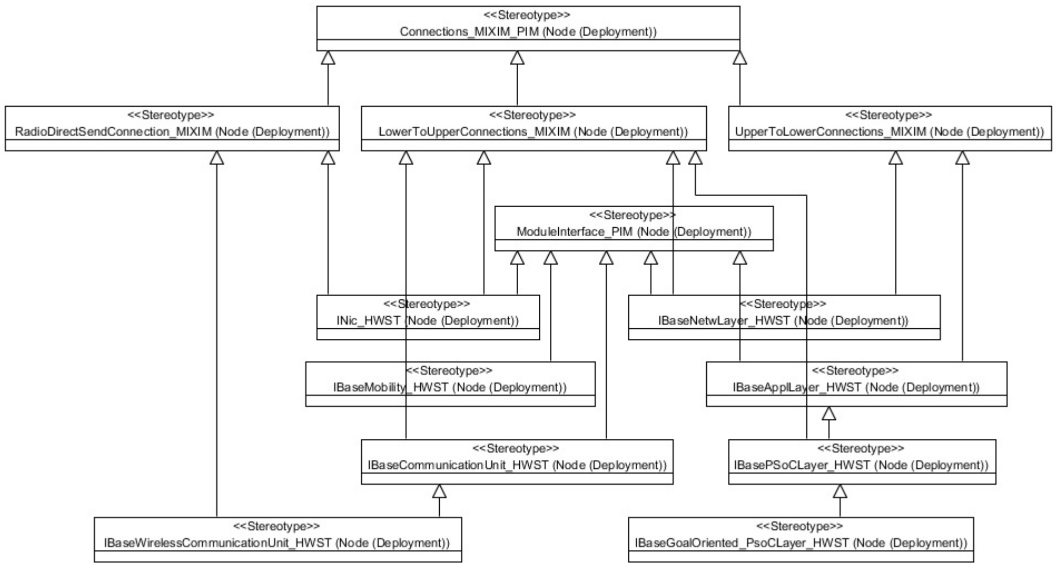

<<Stereotype>>
Connections_MIXIM_PIM (Node (Deployment))
<<Stereotype>>
RadioDirectSendConnection_MIXIM (Node (Deployment))
<<Stereotype>>
LowerToUpperConnections_MIXIM (Node (Deployment))
<<Stereotype>>
UpperToLowerConnections_MIXIM (Node (Deployment))
<<Stereotype>>
ModuleInterface_PIM (Node (Deployment))
<<Stereotype>>
INic_HWST (Node (Deployment))
<<Stereotype>>
IBaseNetwLayer_HWST (Node (Deployment))
<<Stereotype>>
IBaseMobility_HWST (Node (Deployment))
<<Stereotype>>
IBaseApplLayer_HWST (Node (Deployment))
<<Stereotype>>
IBaseCommunicationUnit_HWST (Node (Deployment))
<<Stereotype>>
IBasePSoCLayer_HWST (Node (Deployment))
<<Stereotype>>
IBaseWirelessCommunicationUnit_HWST (Node (Deployment))
<<Stereotype>>
IBaseGoalOriented_PsoCLayer_HWST (Node (Deployment))

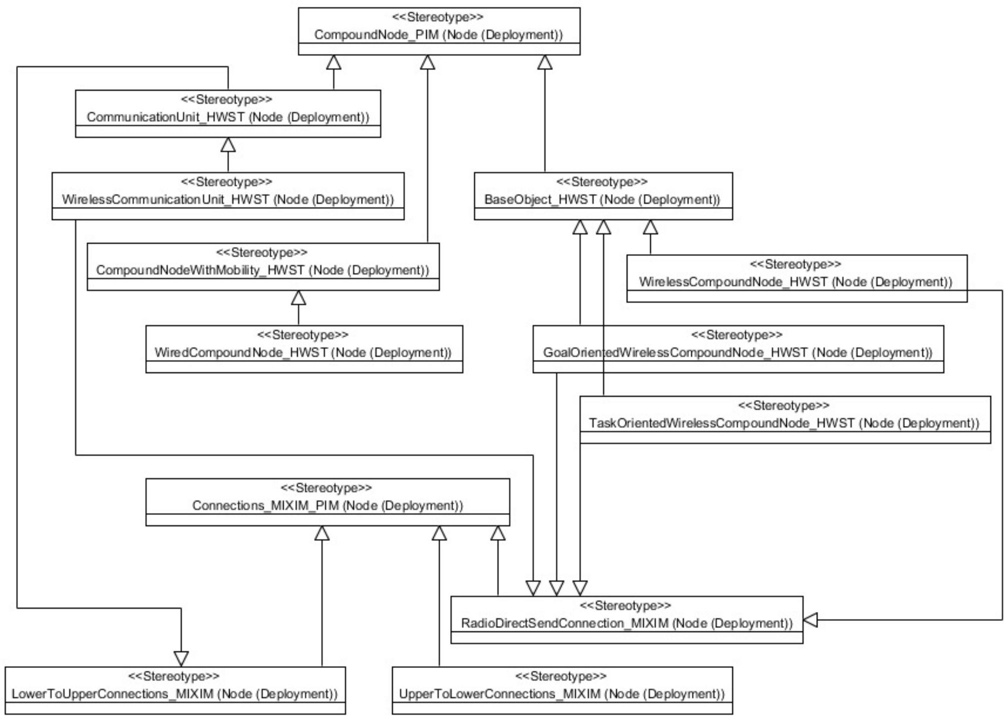
Figure 17 Modules interfaces stereotypes

As expressed in Figure 17, the stereotypes for interfaces must contain also the possible connections for modules which implement these interfaces, expressed by the following stereotypes: *RadioDirectSendConnection_MIXIM*, *LowerToUpperConnections_MIXIM*, *UpperToLowerConnections_MIXIM*.

5.1.6 Compound Modules Stereotypes

The hierarchy for compound modules stereotypes is visually expressed in Figure 18. The compound nodes are differentiated based on the communication modality, using stereotypes for customization like: *WiredCompoundNode_HWST* or *WirelessCompoundNode_HWST*. At the same time, compound nodes with wireless communication can be classified depending on the requirements of the applications into goal-oriented and task-oriented. The corresponding stereotypes are: *GoalOrientedWirelessCompound_HWST and Node_HWST TaskOrientedWirelessCompoundNode_HWST*, respectively.

The stereotypes for the required connections for communication with other modules in a node and with the exterior of the node are also exemplified in the figure.


<<Stereotype>>
CompoundNode_PIM (Node (Deployment))
<<Stereotype>>
CommunicationUnit_HWST (Node (Deployment))
<<Stereotype>>
WirelessCommunicationUnit_HWST (Node (Deployment))
<<Stereotype>>
BaseObject_HWST (Node (Deployment))
<<Stereotype>>
CompoundNodeWithMobility_HWST (Node (Deployment))
<<Stereotype>>
WirelessCompoundNode_HWST (Node (Deployment))
<<Stereotype>>
WiredCompoundNode_HWST (Node (Deployment))
<<Stereotype>>
GoalOrientedWirelessCompoundNode_HWST (Node (Deployment))
<<Stereotype>>
TaskOrientedWirelessCompoundNode_HWST (Node (Deployment))
<<Stereotype>>
Connections_MIXIM_PIM (Node (Deployment))
<<Stereotype>>
RadioDirectSendConnection_MIXIM (Node (Deployment))
<<Stereotype>>
LowerToUpperConnections_MIXIM (Node (Deployment))
<<Stereotype>>
UpperToLowerConnections_MIXIM (Node (Deployment))
Figure 18 Compound modules stereotypes

5.1.7 OCL Constraints

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. There are a large number of OCL constraints specific for the UML profile but only a few types of constraints. This thesis presents the main types of OCL constraints, with examples.

The next table presents constraints, written both in natural language and OCL, for a better understanding. Also there is specified the stereotype to which the OCL constraint is applied.

Constraints similar to first constraint from the table are applicable to *Node_PIM*, *Gate_PIM*, *Instance_PIM*, *Connections_MIXIM_PIM* and state that the stereotyped element should not be named as the stereotype.

Constraints similar to the second constraint in the table are applicable to stereotypes that impose definitions of mandatory ports on the stereotyped element.

The third constraint is defined for stereotypes where the stereotyped element contains an optional port. Depending on the value of the Boolean tagged value, prefixed with *bool_define*, the port must be defined on the customized element or not.

If some tagged values and constraints must be established for each instance of an element, separately, the stereotype that customizes the element must contain an OCL constraint similar to the fourth one in the table.

The instance name is obtained using naming convention rules and contains *_Instance* as suffix. The fifth constraint specifies which value some tagged value should have. Any other value is considered false in the stereotype context. The sixth constraint is applicable to elements and refers to the instances of those elements.

The seventh constraint imposes that a certain component to be stereotyped with the Cypress like stereotype, which defines the OCL constraint.

Stereotype name	Constraint in natural language	Constraint in OCL

Node_PIM	Stereotyped node name should not be identical to stereotype base name	let stereotypeBaseName: String = self.name.substring(1, length(self.name) - 5) in
		if length(self.base_Node.name) = length(sterotypeBaseName)
		then let nodeBaseName: String = self.base_Node.name.substring(1, length(stereotypeBaseName)) in nodeBaseName <> stereotypeBaseName
		else self.base_Node.name <> stereotypeBaseName endif

DMUnit_HWST	Each stereotyped node must define a port named *dataIn*, stereotyped with *InputGate_HWST*	self.base_Node.ownedPort -> exists(a | a.name='dataIn' and a.oclIsTypeOf(InputGate_HWST))
Comparator_HWST	Each stereotyped node can define a port named *clock*, stereotyped with *InputGate_HWST*	if self.bool_define_clockPort = 'true' then self.base_Node.ownedPort -> exists(a | a.name= 'clock' and a.oclIsTypeOf(InputGate_HWST)) else true endif
DAC_HWST	Each instance must be stereotyped with *DAC_HWST_Instance*	self.base_Node.allInstances() -> forAll(o | o.oclIsTypeOf(DAC_HWST_Instance))
CharacterLCD_HWST_Instance	The tagged value int*_LCD_PortMultiplicity* must have value 7	self.int_LCD_PortMultiplicity = 7
BaseObject_HWST	Each stereotyped instance must contain a reference to a *IBaseMobility_HWST* stereotype	self.base_Node.allInstances() -> forAll(o | o.allInstances() -> exists (i | i.oclIsTypeOf(IBaseMobility_HWST))
SPI_HWST	Each instance must have an element named *SPI_HWST_Type* , with value *SPI_Master* or *SPI_Slave*	self.base_Node.allInstances() -> forAll(i -> exists (type = i.ownedElement -> type.name='SPI_HWST_Type' and (type.value='SPI_Master' or type.value='SPI_Slave')))

Table 8 Constraints in UML hardware profile

5.2 UML Profile for Software Specification

The hardware UML profile introduced in the previous subchapter allows designing a distributed embedded application, with clear specifications regarding the network topology, the types of nodes participating in its construction and the hardware units involved in the specifications for the families of nodes used. Deployment diagrams are used for the hardware representation of distributed embedded systems.

The UML software profile presented below allows customization of the application behavior at network, node and unit level.

5.2.1 First Level Stereotypes

The component diagrams are used to express the software requirements for distributed applications. A resulting software component model must correspond to each designed hardware deployment model.

The defined software stereotypes are grouped into several sets, all being inherited from *Component_PIM* stereotype, as shown in Figure 19. The presented groups of stereotypes aim to offer a solution for specifying and modeling the software strategies available in an external library for the different types of units in CPS nodes.

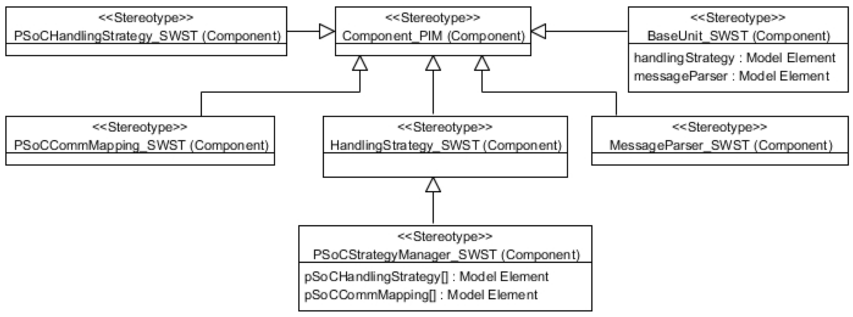

<<Stereotype>>
PSoCHandlingStrategy_SWST (Component)
<<Stereotype>>
Component_PIM (Component)
<<Stereotype>>
BaseUnit_SWST (Component)
handlingStrategy : Model Element
messageParser : Model Element
<<Stereotype>>
PSoCCommMapping_SWST (Component)
<<Stereotype>>
HandlingStrategy_SWST (Component)
<<Stereotype>>
MessageParser_SWST (Component)
<<Stereotype>>
PSoCStrategyManager_SWST (Component)
pSoCHandlingStrategy[] : Model Element
pSoCCommMapping[] : Model Element

Figure 19 First level inheritance of the stereotypes composing the defined software profile

Therefore, 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 translator has to identify, bind and integrate all the customized software strategies and to provide a functional CPS application, customized based on the user's need.

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

The hierarchy derived from the *Message_Parser_*SWST stereotype deals with the application communication part, whereas the Handling*_Strategy_SWST* inherited stereotypes deal with the application business logic which has to be attached to the hardware unit.

The *PSoCStrategyManager_SWST* type stereotypes are meant for keeping the application software logic related to hardware PSoC units involved in the CPS nodes construction. They act as software containers and have the constraint to host software artifacts inherited from the *PSoCHandlingStrategy_SWST* and *PSoCCommMapping_SWST* stereotypes, respectively.

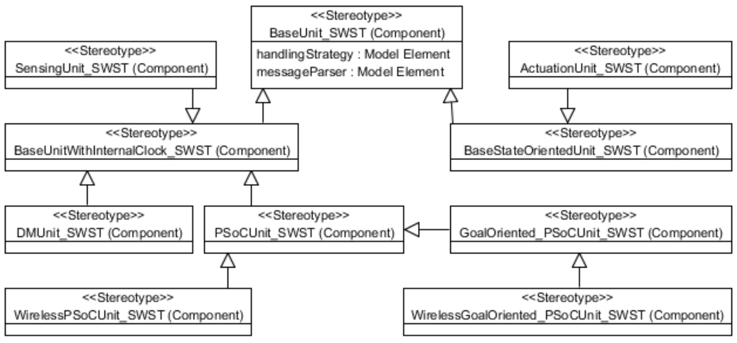
The inheritance of the *PSoCHandlingStrategy_SWST* keeps the actual internal PSoC related business logic and offers the user the opportunity of customizing it, whereas, the *PSoCCommMapping_SWST* stereotype inheritance deals with the message mapping and exchange between different hardware units managed by PSoC. The latter stereotype hierarchy offers in other words 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.

5.2.2 Software Part Definition Stereotypes

This group of stereotypes is presented in Figure 20 and it offers a mapping between the hardware and the software which needs to be attached to it.

The *PSoCUnit_SWST* stereotype defines the software part of a PSoC hardware unit of a node. *SensingUnit_SWST* defines the software part of an internal hardware unit, responsible for sensing. Similar to these, there are also other several units like *ActuationUnit_SWST*, *CANUnit_SWST*, and so on. The *DMUnit_SWST* is related to DM hardware units from a node which is able to collect data from other nodes, to analyze it based on its current goal and to send reconfiguration commands to external nodes in a goal-oriented approach. The main constraint for this set of stereotypes is that each stereotype must have defined the ports similar to their complementary hardware units. Therefore, the deployment model corresponding to the hardware part has to be specified as tagged value.

Each of these stereotypes must have declared internal components stereotyped with specific inheritances from *MessageParser_SWST* and with *HandlingStrategy_SWST*, respectively. The stereotyped internal artifacts must have a link relationship.



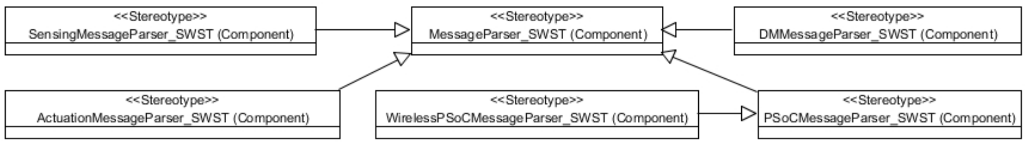
<<Stereotype>> BaseUnit_SWST (Component)
handlingStrategy : Model Element
messageParser : Model Element
<<Stereotype>> SensingUnit_SWST (Component)
<<Stereotype>> ActuationUnit_SWST (Component)
<<Stereotype>> BaseUnitWithInternalClock_SWST (Component)
<<Stereotype>> BaseStateOrientedUnit_SWST (Component)
<<Stereotype>> DMUnit_SWST (Component)
<<Stereotype>> PSoCUnit_SWST (Component)
<<Stereotype>> GoalOriented_PSoCUnit_SWST (Component)
<<Stereotype>> WirelessPSoCUnit_SWST (Component)
<<Stereotype>> WirelessGoalOriented_PSoCUnit_SWST (Component)

Figure 20 Stereotypes used for mapping the software part to a specific hardware unit

BUPT

5.2.3 Message Handling Stereotypes

The stereotypes inherited from *MessageParser_SWST* define the communication software management part. These stereotypes are connected to the communication hardware unit of the node and are responsible for handling the received commands and data and also for sending information through the unit ports designated for other nodes. Figure 21 shows the defined stereotypes.



<<Stereotype>>
SensingMessageParser_SWST (Component)
<<Stereotype>>
MessageParser_SWST (Component)
<<Stereotype>>
DMMessageParser_SWST (Component)
<<Stereotype>>
ActuationMessageParser_SWST (Component)
<<Stereotype>>
WirelessPSoCMessageParser_SWST (Component)
<<Stereotype>>
PSoCMessageParser_SWST (Component)

Figure 21 Inheritance of stereotypes used for handling the communication aspects

For wireless communication, the message handler part must be stereotyped with *WirelessPSoCMessageParser_SWST*. Using this stereotype, the communication hardware component is able to detect the sender and destination node, respectively.

As an internal communication constraint, they must be in a one-to-one association with a component stereotyped with *HandlingStrategy_SWST*. It uses internal collaboration interfaces for communication with the linked handling strategy artifact. Therefore, when receiving a message, the message handling part is able to send the port name, the message and the node identifier to the handling strategy component. The linked handling strategy artifact uses the same contract for sending a message to one of its managed units, or to send it externally.

5.2.4 Handling Strategy related Stereotypes

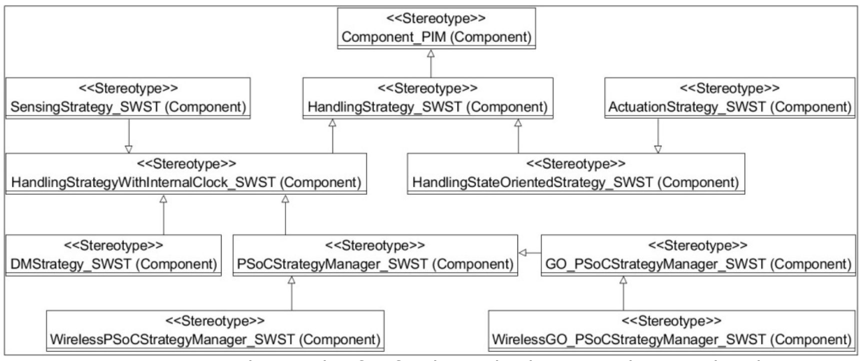
Figure 22 lists the inheritance of *HandlingStrategy_SWST.* The listed stereotypes specify the concrete commands handling and the internal chosen strategy which has to be attached to a hardware unit.

The stereotype *HandlingStateOrientedStrategy_SWST* defines hardware units able to change their internal state based on commands received from external units. *HandlingStrategyWithInternalClock_SWST* stereotypes units provide the ability to change their current state based on internal or external clock management.

The customized components stereotyped with *SensingStrategy_SWST*, *DMStrategy_SWST* and *ActuationStrategy_SWST* specify the basic internal behavior of their corresponding hardware units. They are managed by the PSoC software constructs, connected to them. The resulted PIM diagrams of these units are used in customizing the simulation PSM part and the selected strategies mapped from a third party library. It is considered that the physical units contain the required internal drivers and therefore they are able to run the compiled application resulting by integrating the selected strategies.

The PSoC devices represent programmable circuits, able to interconnect and manage physical units. Therefore, the software components stereotyped with *PSoCStrategyManager_SWST* and *GO_PSoCStrategyManager_SWST*, respectively, must have internally declared software constructs stereotyped with

PSoCHandlingStrategy_SWST and with *PSoCCommMapping_SWST*, being in a different collaboration relationships.

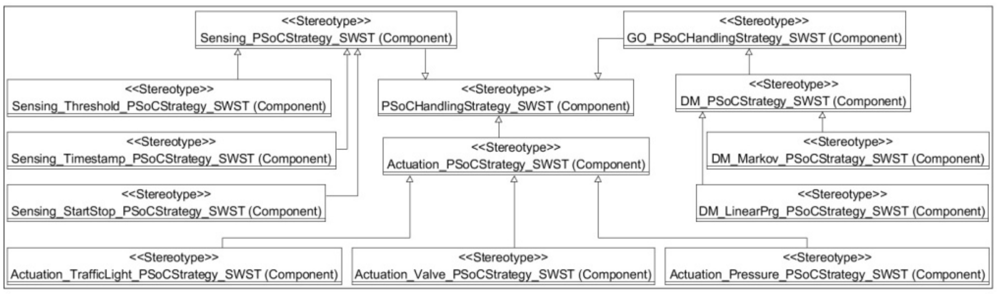

<<Stereotype>> Component_PIM (Component)
<<Stereotype>> SensingStrategy_SWST (Component)
<<Stereotype>> HandlingStrategy_SWST (Component)
<<Stereotype>> ActuationStrategy_SWST (Component)
<<Stereotype>> HandlingStrategyWithInternalClock_SWST (Component)
<<Stereotype>> HandlingStateOrientedStrategy_SWST (Component)
<<Stereotype>> DMStrategy_SWST (Component)
<<Stereotype>> PSoCStrategyManager_SWST (Component)
<<Stereotype>> GO_PSoCStrategyManager_SWST (Component)
<<Stereotype>> WirelessPSoCStrategyManager_SWST (Component)
<<Stereotype>> WirelessGO_PSoCStrategyManager_SWST (Component)
Figure 22 Stereotypes hierarchy for fetching the business logic at hardware unit level

5.2.5 PSoC Handling Strategy related Stereotypes

In case of simple units connected to a PSoC device, the handling strategy component processes the message. In case of PSoC units, the message is forwarded by the software component manager to one of the internal handling components.

The manager components act as software artifact containers. Therefore, the actual handlings are ensured by their internal software defined components, customized using stereotypes presented in Figure 23.

For each hardware unit connected to the PSoC device, a corresponding internal handling strategy is established. This handling strategy can be chosen from various domain-specific predefined strategies. Each stereotype defined in Figure 23 specifies an internal behavioral model of PSoC unit for managing the attached device. The tagged values defined by these stereotypes allow the customization of the selected predefined software components. The constraints are related to port interconnectivity issues between PSoC and the managed unit. The developer specifies the connected ports to that unit using the tagged values provided by the stereotypes.


<<Stereotype>> Sensing_PSoCStrategy_SWST (Component)
<<Stereotype>> GO_PSoCHandlingStrategy_SWST (Component)
<<Stereotype>> Sensing_Threshold_PSoCStrategy_SWST (Component)
<<Stereotype>> PSoCHandlingStrategy_SWST (Component)
<<Stereotype>> DM_PSoCStrategy_SWST (Component)
<<Stereotype>> Sensing_Timestamp_PSoCStrategy_SWST (Component)
<<Stereotype>> Actuation_PSoCStrategy_SWST (Component)
<<Stereotype>> DM_Markov_PSoCStratagy_SWST (Component)
<<Stereotype>> Sensing_StartStop_PSoCStrategy_SWST (Component)
<<Stereotype>> DM_LinearPrg_PSoCStrategy_SWST (Component)
<<Stereotype>> Actuation_TrafficLight_PSoCStrategy_SWST (Component)
<<Stereotype>> Actuation_Valve_PSoCStrategy_SWST (Component)
<<Stereotype>> Actuation_Pressure_PSoCStrategy_SWST (Component)
Figure 23 Stereotypes hierarchy for fetching the management of connected hardware devices at PSoC unit level

5.2.6 Communication related Stereotypes

For communication between units residing inside of a physical node, and also between nodes, a new set of stereotypes inherited from *PSoCCommMapping_SWST* is added in the UML software profile, as listed in Figure 24.

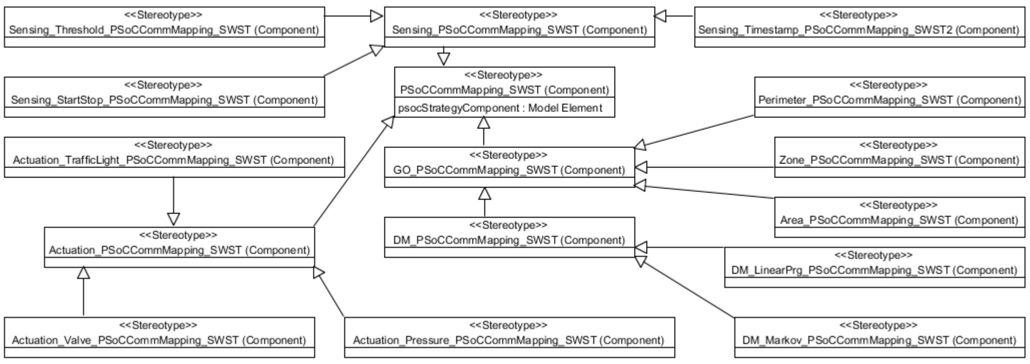
These stereotypes have as tagged values the union of possible internal software states and the commands the units recognize, acting as possible triggering events. The inheritance for this kind of stereotypes follows the inheritance of possible internal strategies for the customized device.

The developer defines different application dependent message formats which he intends the device to use during the communication with other units and nodes. Each already defined message format has to be bound to a precondition matching criteria. The list of the tagged values of the stereotype act as possible matching preconditions for triggering the message are listed in. Thus, every time the internal software state changes, internal watchdog timeout occurs or event is received, the software implementation of the stereotype checks its list of preconditions to find out if there is any message to be sent.

Each communication mapping component must have also specified the corresponding output gate. In case of wireless communication, it is also required to set as tagged value the destination node identifier. In this case, the destination gate is not required, since the external communication ports of the unit are mapped internally in the model component used by the manager component for sending messages.

In case of communication with multiple internal units, a component stereotyped with an inheritance of *PSoCCommMapping_SWST* must be specified for each unit. Therefore, the internal strategy is found in a one-to-many relationship with its related communication mapping components.

To ensure proper communication, the message sent from a unit has to be recognized based on its format by the message receiving unit either as being a precondition for one of its internal *PSoCCommMapping_SWST* stereotyped artifacts, or as being an action request for one of its *HandlingStrategy_SWST* stereotyped artifact. Thus, intra-node communications schemas like *request-feedback*, *request-act*, *send-forward* and *send broadcast* are possible and easy to design.



<<Stereotype>> Sensing_Threshold_PSoCCommMapping_SWST (Component)
<<Stereotype>> Sensing_PSoCCommMapping_SWST (Component)
<<Stereotype>> Sensing_Timestamp_PSoCCommMapping_SWST2 (Component)
<<Stereotype>> Sensing_StartStop_PSoCCommMapping_SWST (Component)
<<Stereotype>> PSoCCommMapping_SWST (Component)
psocStrategyComponent : Model Element
<<Stereotype>> Perimeter_PSoCCommMapping_SWST (Component)
<<Stereotype>> Actuation_TrafficLight_PSoCCommMapping_SWST (Component)
<<Stereotype>> GO_PSoCCommMapping_SWST (Component)
<<Stereotype>> Zone_PSoCCommMapping_SWST (Component)
<<Stereotype>> Area_PSoCCommMapping_SWST (Component)
<<Stereotype>> Actuation_PSoCCommMapping_SWST (Component)
<<Stereotype>> DM_PSoCCommMapping_SWST (Component)
<<Stereotype>> DM_LinearPrg_PSoCCommMapping_SWST (Component)
<<Stereotype>> Actuation_Valve_PSoCCommMapping_SWST (Component)
<<Stereotype>> Actuation_Pressure_PSoCCommMapping_SWST (Component)
<<Stereotype>> DM_Markov_PSoCCommMapping_SWST (Component)

Figure 24 Unit and node intra-communication stereotypes

5.2.7 OCL Constraints

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

Stereotype	Constraint in natural language
SensingStrategy_SWST	Each component must have instantiated the tagged value called *controlIn* with the name of a port contained by the parent component and of type *InputGate_HWST*
SensingStrategy_SWST	Each component must have a link to a component stereotyped with *SensingMessageParser_SWST*
DM_CommunicationInterface_SWST	Each class must declare a method having the following signature void setEquation(string eq)
PSoC_CommunicationInterface_SWST	Each class must declare a method having the following signature boolean isResponsibleForHandlingPort(string port, index portIndex = 0)
DM_PSoCInternalStrategy_SWST	Each component must have instantiated the tagged value called *controlIn* with the name of a port contained by the grandparent component and of type *InputGate_HWST*
DM_PSoCInternalStrategy_SWST	Each component must have instantiated the tagged value called *dataOut* with the name of a port contained by the grandparent component and of type *OutputGate_HWST*
Predefined_PSoC_ModelStrategy_SWST	Each component must publish a class stereotyped with *SendOutInterface_SWST* stereotype
Predefined_WirelessPSoC_ModelStrategy_SWST	Each component must have instantiated the tagged value called *controlIn* with the name of a port contained by the grandparent component and of type *InputGate_HWST*
PSoCCommunicationMapping_SWST	Each component must have a link to a component stereotyped with *PSoCInternalHandlingStrategy_SWST*
PSoCCommunicationMapping_SWST	The parent must be stereotyped with *PSoCStrategy_SWST*

Table 9 Constraints in UML software profile

The previous table presents constraints written in natural language, for a better understanding, along with the stereotypes to which the OCL constraints are applied. There is a large number of OCL constraints specific for the UML profile but only a few types of constraints. Each next two constraints are related to: handling strategies, interfaces, PSoC internal strategies, model strategies and communication mapping, respectively.

5.3 Summary

UML modeling allows a high level of abstraction in designing applications. At the same time, it provides a facile switching of perspective between simple elements and the entire network.

This is very useful, as the user can describe the specifications and the requirements for the distributed applications at different logical levels. The model can describe first the network as a black-box of nodes, then going into details for each node.

UML profiles help customizing different UML models. The current chapter presents two defined UML profiles, one for hardware specification and one for software specification of the components of the nodes and of the entire network.

The stereotypes, tagged values and OCL constraints defined in the UML profiles are used in customizing the hardware and software components, which instances are actually used in modeling the application. The stereotypes are hierarchically built and help achieving a clear separation and grouping into families of devices.

The UML hardware profile helps defining the network topology, the hardware components and connections between nodes while the UML software profile defines possible behaviors corresponding to each hardware component.

Chapter 6. Approach Validation using Case Studies

This chapter discusses the validation of the presented methodology, based on goal-oriented specifications and MDA approach in CPS applications.

The first subchapters discuss the case studies considered by the author of this thesis in order to demonstrate the utility of the UML hardware and software profiles defined for CPS design. Also, the presented case studies contribute to validation of the MDA approach considered in CPS design, for applications where specifications or requirements are represented in a goal-oriented manner. In each of these examples, one can distinguish the use of UML artifacts, both for hardware and software customization in CPS applications, and the correlation between specification at hardware and software level, respectively. The usage of the defined stereotypes allows creating the PIM for each specific application, which contains UML models of different parts of the CPS application, depending on the requirements.

The case studies presented in this chapter have been discussed in the context of the MDA approach for goal-oriented CPS application design and programming in several articles. These articles have been published into several international ISI and IEEE conferences, between September 2010 and Ianuary 2012.

The management for a gas distribution network has been introduced in [75], with a discussion on goals interdependency and hardware customization and has been continued in [82] with a presentation of a software model for a gas node.

The traffic light management system has been discussed in [80] from a perspective of wired connections between nodes in the CPS network and [81] and [95], considering wireless communication between the nodes of the network.

The example with the aircraft fuel management system is part of paper [76].

The next subchapter presents briefly the efforts made by the research team the author of this thesis is member of in constructing simulating models for PIM in CPS applications.

6.1 Management for a Gas Distribution Network

The first case study considered in this thesis is a monitoring system for an urban gas distribution network. The main goal of such an application is to secure the perimeters by interrupting the gas flooding through some pipes, in case of a possible gas leak situation. Along with the security goal, another main requirement is to affect the end users as little as possible when the gas flow is interrupted in a certain point. Therefore, a smart control must be implemented over the network. To support the requirements, the gas network is composed of a collection of pipes, meters, street and house valves, thus a good candidate for a CPS network. The implementation must be extendable to larger networks, cheap and efficient. The network for the gas distribution being wide spread, it will be divided into areas and the solution will be implemented for each of those areas.

The solution provided here follows some steps and uses goal-oriented approach in application goal handling and MDA approach. First of all, the settings at area level are provided and the application is tailored in a goal-oriented manner. The objective is to handle the application requirements in a homogenous way. Then, the application is configured at each logical level, constructing the PIM. In the end, an example of a goal handling at perimeter level is given.

6.1.1 Network Settings at Area Level

In the design of the application, it is considered that a gas leak can occur in any point of the gas distribution network. The dangerous zone must be isolated and closed, in order to prevent a possible explosion. At the same time, the zone must be as reduced as possible, so that end users will not to be unnecessarily affected. The area is divided into zones characterized by homogenous pipe architecture. Special separations inside zones, revealing a greater possible danger, are called perimeters. When a gas leak occurs in one of the established zones, detected by the sensors, the valves corresponding to the zone are closed, to secure the zone. The valves corresponding to other zones, not related to the point of the gas leak are not affected and therefore, not closed. Figure 25 shows a representation of gas pipes for the network, with details only for the north and the south zone, for image clarity.

A complex building can have many points of gas consumption. When a gas leak appears, it is necessary to shut down the gas flow for the entire building, without affecting gas distribution for buildings in the neighborhood. This separation is achieved by implementing distinct regions at the building level. The new abstract level is called a perimeter. The network is also divided into zones with the purpose of improving the communication speed and to avoid the possible bottlenecks at stressed zones. This is also a way to reduce the energy consumption with communications and to increase the network lifetime, if some nodes are battery operated.

Once the network is divided into areas, zones and perimeters, goals are established at each separation level, to reflect the requirements of the application. For the considered case of the gas distribution network, the area level goal is to minimize the time for perimeter isolation, by closing the corresponding valves, and the time of reporting the event at the central distribution point. Another goal for the area refers to determining the optimal point for storing information about the event, in order to later determine the source and remedy the situation by human intervention [75].

The goal at each zone level is to maintain the maximum gas consumption below a certain value. A value greater than the one considered as threshold can lead to a gas leak event. Setting up specific goals for perimeters and zones is necessary due to different risk characteristics for these regions. Considering the topology for the zones, the goals are established accordingly.

In the example considered (Figure 25), the maximum admitted gas level for *Zone-3* is below the one for *Zone-2*. This is due to the fact that *Zone-3* covers a part of the street that includes several houses.

All these houses have a surrounding perimeter between them and the sidewalk. The potential false alarms are reduced if the goal takes into consideration this specific situation. *Zone-2* mainly contains houses close to the sidewalk. To avoid false event localization, *Zone-2* requires a closer monitoring than *Zone-3*. *Zone-2* also contains a gas refueling station. Considering the greater danger involving such a facility, it is required to set a more complex goal for this zone.

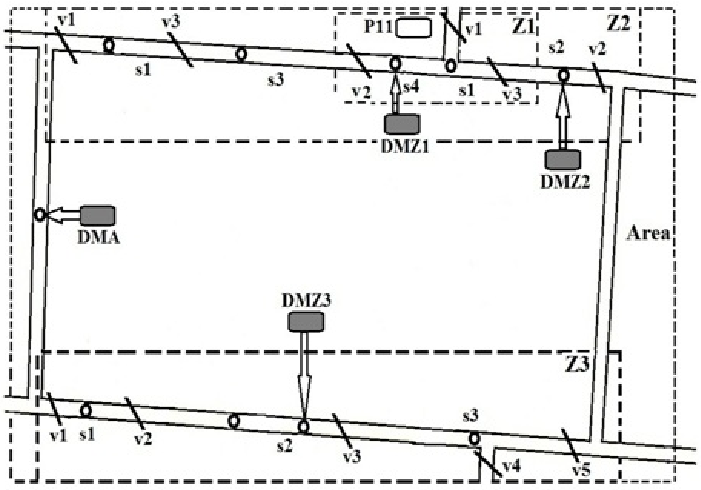


Figure 25 Gas pipes representation [75]

This is the first reason to consider a nested zone (*Zone-1*) inside *Zone-2*, which contains inside the gas refueling station. This gas refueling station is placed inside a perimeter and inside a zone, to make a clear distinction between possible leaks inside buildings and outside them and the different measures to be taken.

An additional goal for each zone is the minimization of the closing time for the corresponding valves in the case of overpassing the maximum allowed gas level. Concurrently, the goal is to ensure the minimization of the affected distribution points. In case of an event, the area decision module is notified. The gas refueling station is enclosed in a house perimeter and in a nested zone, in order to minimize the number of affected nearby neighbors. This allows closing only its corresponding valves when an event occurs.

The perimeter goal is to maintain the gas level below an established maximum value and to prevent any gas leak. As soon as this value is exceeded, the perimeter handler must minimize the time for closing the appropriate gas valves and announce the upper level. Another reason for designing the gas refueling station inside a perimeter is to clearly distinguish between gas leaks inside the building and gas leaks in the proximity, with potentially higher explosion danger. This distinction is covered by event type and helps on the decision of closing a certain grouping of valves. Thus, for the event of a gas leak outside the building, the goal is to minimize the time for signaling to the DM of the *Zone-1*, which will take decision for shutdown all the *Zone-1* valves. This is done without effectively closing the gas refueling station valves. Indeed, those valves are responsible for handling a gas leak in the *P11* perimeter [75].

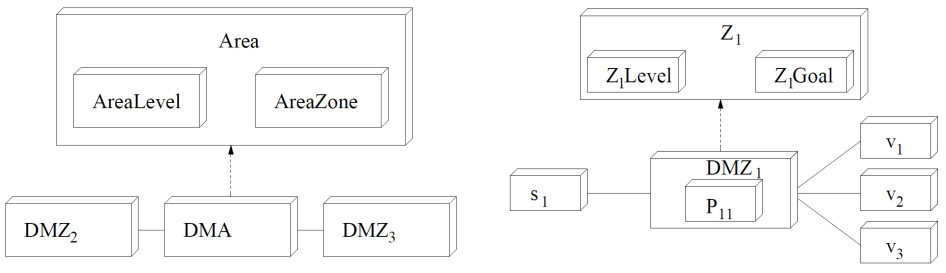
6.1.2 Configuration at Each Logical Level

The next step in the network configuration is to determine the components of logical levels and the models that construct the application PIM. The deployment of a CPS wireless linked network is considered, in order to monitor the area. The network is composed of a series of nodes with different functionalities, connected also with gas meters already installed on the gas pipes.

In the deployment diagrams in Figure 26 can be seen that each logical level contains a DM. The DMs implement the goals at the corresponding level. DMs represent the interface with the neighbor logical levels and they handle the events fired inside the abstraction zone. Figure 26 shows this architecture and uses denominations for each DM according to the addressed logical level. In the first part of the figure, the *DMZ*$_x$ corresponds to the *x* zone level. A unique *DMA* is responsible for at the area level. The *DMP*$_{xy}$ represents the DM for the each *xy* perimeter level, where *x* is the zone index and *y* the index of the perimeter. An example can be seen in Figure 26 (b). DMs represent the interface between their corresponding logical level and the one immediately above or below. Therefore, communication to the outside of the event detected at a node level is ensured via the DM associated to the logical level that contains the respective node.

Actuation units, described by the corresponding stereotypes, are added to each zone, in order to control the house and junction valves (v_1, v_2, v_3 in Figure 26 (b)). The action of opening or closing the valves is commanded by the DMs. Complementary wireless sensing units (s_1 in Figure 26 (b)) are placed on the streets and inside the buildings to measure the gas level at certain periods of time.

The results of their measurements are interpreted by the DMs according to the goals of the corresponding logical level. The first task was to model the topology of the network using deployment diagrams. Such diagrams are presented in Figure 26 (a) and (b). The next task is discussed in the next subchapter.



(a) (b)
Figure 26 (a) Area model; (b) Zone-1 model

Article [82] continues the work started by the author of this thesis in [75] and extends the case study of the gas system management with software customization artifacts, defined in the UML software profile. Figure 27 represents the PIM for a gas node, which inspects a physical perimeter and if the gas value in that perimeter reaches a maximum specified threshold value, it has to secure the gas pipe and to announce the managing DM node.

When receiving a sensed value greater than the maximum allowed one, the *SensingHandlingStrategy* component requests the *SensingToActuationUnitComm* component for retrieval of the mapped message. The message *ev_closeValve* is returned and it is referring to the actuation unit connected to the same PSoC device. This message requires no formatting and therefore it is sent to *PSoCHandlingModel* component. This component detects that this is a message referring to *ActuationHandlingStrategy* component, which is responsible for managing the connected actuation unit.

The *ActuationHandlingStrategy* component has internally mapped as control gate the port of *PSoCUnitDMNode_MessageParser* component, called *leftControlOut*. It requests in turn to *PSoCHandlingModel* component to send the same command using the *leftControlOut* port. The *PSoCHandlingModel* component forwards the requested message along with the sending parameters to *HandlingStrategyManagerPSoCUnitGasNode* component, which forwards the request to *PSoCUnitDMNode_MessageParser* component. This one sends the message using its requested port. The message is then forwarded to the *upperControlActOut* port to the input command port of the actuation unit.

6.1.3 Goal Design at Perimeter Level

The goals at each logical level can be considered in the form of constrained optimization equations. For solving the equation system, integer linear programming is used. As an example, the author of this thesis considers a *Goal* for a perimeter. This is the minimization of the number of pipe segments that need to be closed in case of a gas leak.

The goal can be expressed as a system of linear functions that need to be minimized. The system expresses the problem constraints using variables to designate the pipe segments covered by sensors and corresponding valves [75].

The goal is implemented by a minimization function (13).

$$G_1(v_1, v_2, ... v_{nv}) = v_1 \sum_{i=1}^{m1} s_{1,i} + v_2 \sum_{i=1}^{m2} s_{2,i} + ... + v_{nv} \sum_{i=1}^{m_{nv}} s_{nv,i} \qquad (13)$$

`

The associated goal constraint is denoted by (14), expressing the requirement for the set of cut-off pipe segments to include affected segments that belong to the set of pipe segments affected by a gas leak.

$$\bigcup_{k, v_k = 1} \{s_{k,i}, i = \overline{1, mk}\} \supseteq AF \qquad (14)$$

The variables are of type integer and take values in *{0,1}*. The meanings of the notations for the equations are expressed as follows:

- $S = \{s_1, s_2, \dots s_{np}\}$ – the set of pipe segments belonging to considered perimeter
- n_p – the number of pipes covered by the perimeter
- $V = \{ v_1(\{s_{1,1}, s_{1,2}, \dots s_{1,m1}\}), v_2(\{s_{2,1}, s_{2,2}, \dots s_{2,m2}\}), \dots v_{nv}(\{s_{nv,1}, s_{nv,2}, \dots s_{nv,mnv}\}) \}$ – the set of valves that control the perimeter

- $s_{k,j} \in S$
- $\{s_{k,1}, s_{k,2}, \ldots s_{k,mk}\}$ – the set of pipe segments controlled by the valve v_k
- np – the number of available valves
- $AF = \{s_{a1}, s_{a2}, \ldots s_{an}\}$ – the set of pipe segment affected by the gas leak

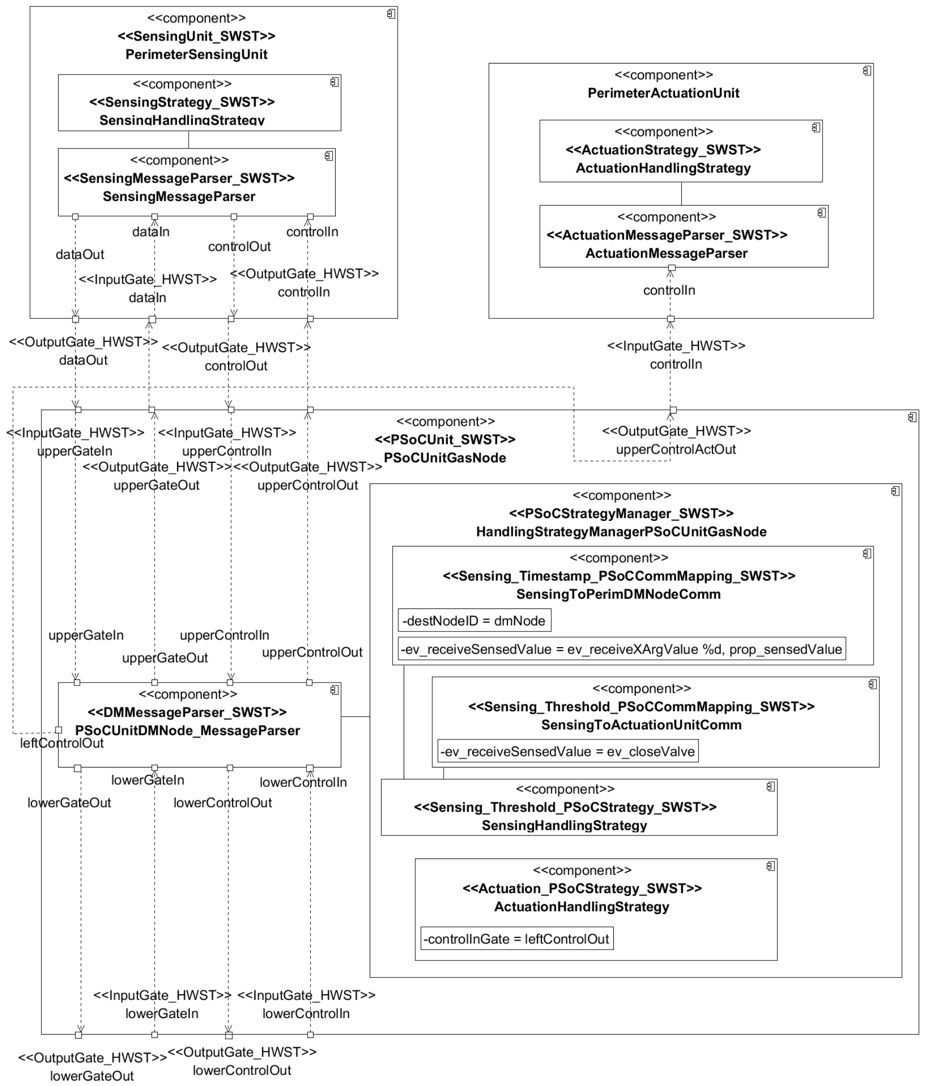


Figure 27 PIM of a gas node containing sensing and actuation units

A similar set of functions is designed for each perimeter belonging to application area. These functions are string attributes for nodes customized by the UML artifacts. The functions are injected into the generated code. The system's middleware layer has an interpreter designed to dynamically solve the considered functions through linear programming algorithms [75].

6.1.4 Summary

The approach of CPSs design presented in subchapter 6.1 leads to an efficient development of a gas distribution controlling application that ensures a minimal disruption in the network in case of gas leaks.

After the configuration is completed, creating similar UML models to the ones already presented and applying linear programming to goals, code can be automatically generated for a PSM. In the example it is considered a gas distribution application over a PSoC based CPS network. The generator produces C code adapted to the PSoC compiler. Analyzing the code, one can observe that more than 34% of code lines are automatically generated from the UML description of the model. The remaining lines have to be written manually according to application specifications. One advantage in using generated code resides in the fact that all header files are correctly, completely and consistently generated from UML diagrams, while the effort to produce valid code is reduced. The other important benefit is the ease of model and documentation maintenance.

6.2 Management for a Traffic Light Network

The examples presented in this subchapter are two case studies for system design of an intersection traffic management. Figure 28 presents a view for an intersection in Timisoara, where the proposed traffic management can be applied. The actual intersection is zoomed in the picture and the places where the traffic light nodes can be found are evidenced with red marks. The goal of the application is to optimize the traffic through the intersection, by giving a higher priority to one or another direction in traffic.

For simplicity, it is considered that the traffic lights are commanded in pairs, following N-S or E-W directions. A higher priority for a traffic direction is given by increased green color duration for the traffic lights that control the direction. At the same time, the green time duration estimation is carefully planned in order to avoid traffic bottlenecks on one direction, which is considered to have a lower priority. The decision for which direction to have a higher priority is taken a priori considering general traffic flow characteristics in that area or are taken dynamically, considering the information provided by video cameras installed on sensors related to each traffic light.

The video cameras record the number of cars that are waiting at the red color of the corresponding traffic light. In the examples presented in the next paragraphs, the decision for which direction to have a higher priority is taken considering sensors information. Communication between nodes for sensors, traffic lights and a decision module in a network can be made using wired or wireless connections. The examples presented next consider both wired and wireless connections. For a better understanding, the author of this thesis has created two separate examples, considering the types of connections.

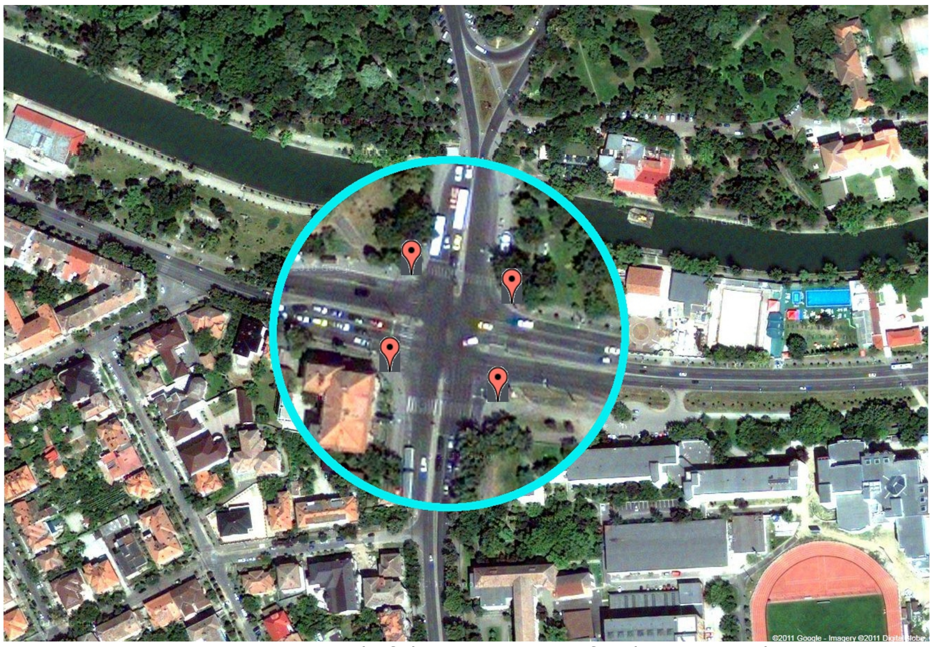


Figure 28 Detail of the environment for the case studies

6.2.1 Cyber Physical System Networks using Wired Connections

A simple management example is considered for an easier understanding and a better focus on the steps that need to be taken for the construction of the PIM for hardware that describes CPS network and components.

The system consists of four traffic light nodes, displayed as in Figure 29, controlled by a decision node, which must compute optimal green color duration for each traffic light node. For simplicity, the traffic light nodes are considered to be working in pairs, and therefore, at a moment only two green color durations must be computed, and not four. Each traffic light node is connected to a sensing node and it commands the time when the sensing node starts counting the number of cars waiting at the red color for that semaphore. The resulting values for the sensing node are computed using image handling algorithms, using as inputs the video cameras placed on the sensing nodes.

The sensed values are forwarded by the traffic light nodes to the decision node. The latter interprets the received data and computes the next optimal green color periods for N-S and E-W intersection directions, respectively. The decisions taken on the decision node can favor a certain direction, considered having a higher priority in traffic or can be influenced on the number of cars waiting at traffic lights red color.

For designing such application at hardware level using UML high-level modeling, it is necessary first a library of possible components, from which the user chooses the most suitable components for his application. Using predefined components, the user can also create customized nodes, which are valid as long as the hardware and software requirements for the nodes are completely satisfied.

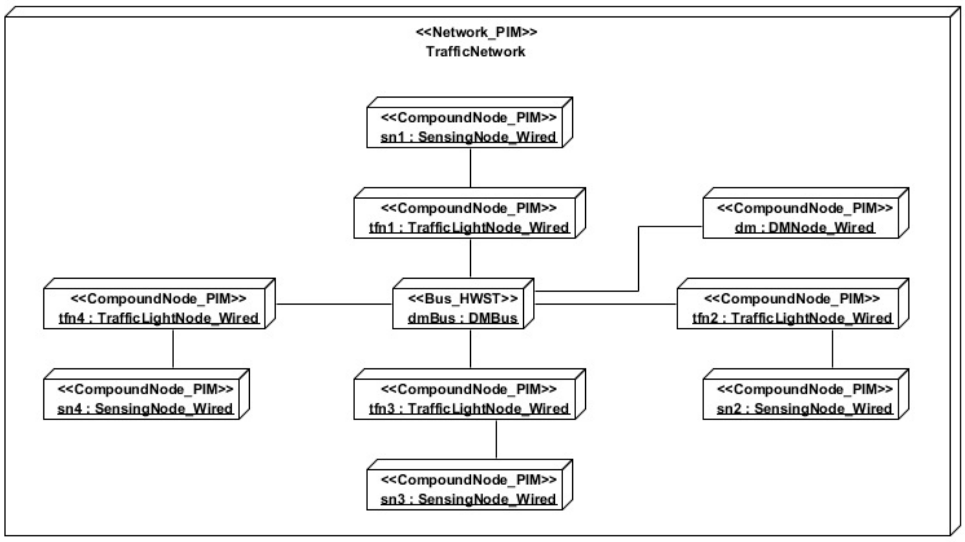


Figure 29 UML representation for intersection traffic management network

At first, the developer of the library of available components uses the UML defined profiles in order to create valid and complete components. The developer must also satisfy the constraints stated by the UML stereotypes using to customize different components. Examples of such component can be seen in Figure 30, as UML defined components for sensing nodes and in Figure 31, as UML defined components for traffic light nodes, respectively. These figures show the mapping of the simple modules using connections.

The *PSoCUnit*, used in constructing sensing nodes and traffic light nodes, respectively, is customized using *PSOCUnit_HWST* stereotype, which extends *Node (Deployment)* metaclass. This stereotype imposes, using an OCL constraint, the fact that all instances of a *PSoCUnit* must be stereotyped using *PSoCUnit_HWST_Instance* stereotype. The *PSoCUnit* represents a PSoC device which contains the user application code representing the business logic of the corresponding nodes. The *PSoCUnit* exchanges data with the other two modules using input output gates. These gates of the PSoC device represent the pins at which the communication and sensing units are connected.

PSoCUnit_HWST_Instance stereotype defines tagged values like *debug*, used in debugging purposes, a Boolean value specifying whether the node does or does not have a wireless behavior, the header length and an integer information about the host identity.

The communication unit can be *CANUnit*, customized using *CAN_HWST* stereotype. There are several types of CAN units, expressed by Cypress like stereotypes. The developer can specify customized components using each of those communication modalities.

Sensing units and actuation units are specific components for different types of nodes, in this case for sensing nodes and traffic light nodes, respectively. They are customized using stereotype that follow a certain naming convention.

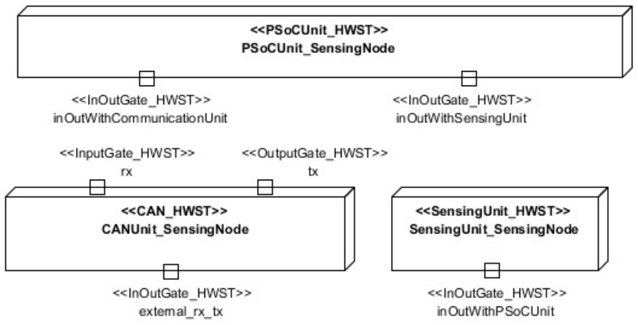


Figure 30 UML components for sensing nodes

Some of the user defined nodes used in traffic management applications using PSoC based CPSs are presented in Figure 32 and Figure 33. At the same time, the connections between ports for each of the components of the composed nodes are detailed in the figures. For example, the connection of the *SensingNode_Wired* compound module, named *inOutWithTrafficLightNode*, is mapped to the internal communication unit and participates in data exchange with a *TrafficLightNode_Wired*

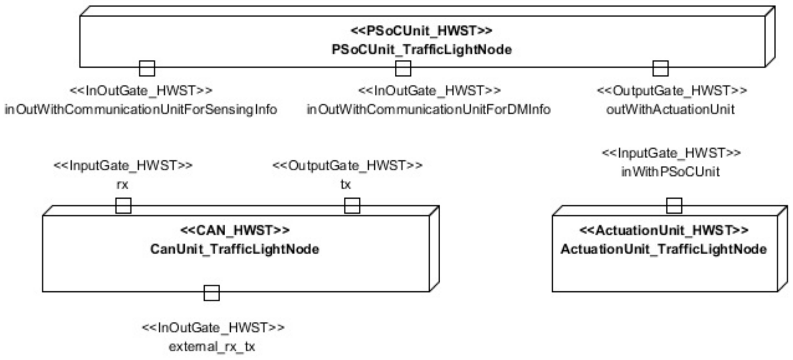
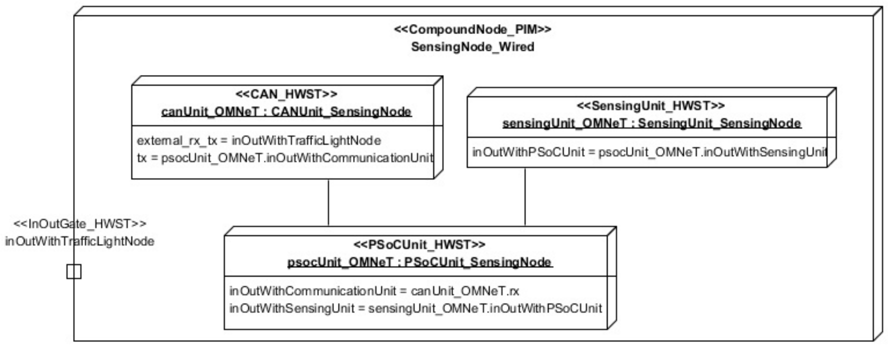


Figure 31 UML components for traffic light nodes



<<CompoundNode_PIM>>
SensingNode_Wired
<<CAN_HWST>>
canUnit_OMNeT : CANUnit_SensingNode
external_rx_tx = inOutWithTrafficLightNode
tx = psocUnit_OMNeT.inOutWithCommunicationUnit
<<SensingUnit_HWST>>
sensingUnit_OMNeT : SensingUnit_SensingNode
inOutWithPSoCUnit = psocUnit_OMNeT.inOutWithSensingUnit
<<InOutGate_HWST>>
inOutWithTrafficLightNode
<<PSoCUnit_HWST>>
psocUnit_OMNeT : PSoCUnit_SensingNode
inOutWithCommunicationUnit = canUnit_OMNeT.rx
inOutWithSensingUnit = sensingUnit_OMNeT.inOutWithPSoCUnit

Figure 32 UML compound sensing node

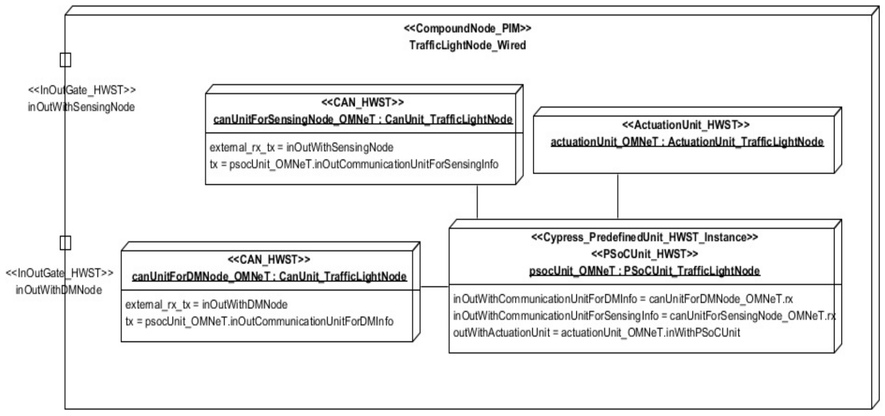

<<CompoundNode_PIM>>
TrafficLightNode_Wired
<<InOutGate_HWST>>
inOutWithSensingNode
<<CAN_HWST>>
canUnitForSensingNode_OMNeT : CanUnit_TrafficLightNode
external_rx_tx = inOutWithSensingNode
tx = psocUnit_OMNeT.inOutCommunicationUnitForSensingInfo
<<ActuationUnit_HWST>>
actuationUnit_OMNeT : ActuationUnit_TrafficLightNode
<<InOutGate_HWST>>
inOutWithDMNode
<<CAN_HWST>>
canUnitForDMNode_OMNeT : CanUnit_TrafficLightNode
external_rx_tx = inOutWithDMNode
tx = psocUnit_OMNeT.inOutCommunicationUnitForDMInfo
<<Cypress_PredefinedUnit_HWST_Instance>>
<<PSoCUnit_HWST>>
psocUnit_OMNeT : PSoCUnit_TrafficLightNode
inOutWithCommunicationUnitForDMInfo = canUnitForDMNode_OMNeT.rx
inOutWithCommunicationUnitForSensingInfo = canUnitForSensingNode_OMNeT.rx
outWithActuationUnit = actuationUnit_OMNeT.inWithPSoCUnit

Figure 33 UML compound traffic light node

After modeling the application at UML level and validating the OCL constraints imposed on the components with OCL constraints, the PSM constituted by the simulation model using OMNeT++ environment [96] is developed.

The hardware parser presented by Magureanu et al. in [80] enables transformation from UML specifications into NED configuration files. However, at this moment the user must manually specify the behavior for the simulation components.

The reasoning for giving priority to this aspect, before the software programming aspect, resides in increasing the overall efficiency of designing process. It allows starting a distributed embedded application design with clear specifications regarding the network topology, type of nodes involved in its construction and hardware units participating in specification of the families of nodes used.

As the requirements for the behavior of the network are not influenced by the type of connections between nodes, the software specification for the traffic light management system will be discussed in the case of wireless connections between network nodes.

6.2.2 Cyber Physical System Networks using Wireless Connections

Another example regarding intersection traffic management uses wireless connections for communication between nodes in a distributed network. The wireless communication model is inspired by MiXiM project [94], therefore the UML defined stereotypes for the UML hardware profile follow the main ideas presented in MiXiM.

The steps necessary to be taken in modeling an application for which nodes communicate wirelessly are similar to the ones described in the previous subchapter. The differences are made by some extra units required for nodes that communicate without using wires.

The next three figures present the types of compound nodes that are necessary to be modeled by the user in a distributed traffic application.

All three types of nodes present some similarities. For example, communication with the exterior of the node is made using a port named *radioIn,* stereotyped with *InputGate_HWST*, which means this is an input port. All types of nodes contain a mobility component which specifies the *x*, *y* and *z* coordinates of the node position in geographical space. Also, the nodes contain a *BaseArp* component, which is a module responsible with address resolution.

Each node contains a component responsible with the application part for that node. In particular it must implement the required behavior for the type of node. The communication for each node is ensured by a component which implements a base wireless communication unit predefined interface.

Each of the presented types of nodes is composed of several units. Each of these units is an instance of a predefined component, customized depending on application needs. If it is required by OCL constraints that the instances of the components also have to be stereotyped, on the UML representation for a certain instance will appear both used stereotypes: the one extending *Node (Deployment)* metaclass and the one extending Instance Specification metaclass. Such an example is the *sensingUnit*, which is an instance of the defined *VideoSensingUnit*. The stereotypes here are *SensingUnit_HWST* for *Node (Deployment)* and *SensingUnit_HWST_Instance* for *Instance Specification*.

The software implementation part is detailed in [83]. The components' collaboration at unit level is satisfied from the point of view of PIM development by respecting the constraints of each stereotype used for defining the units, like interconnectivity, type check and port connections. At node level, the software components interconnectivity is satisfied by respecting the mapping between hardware and software relationship.

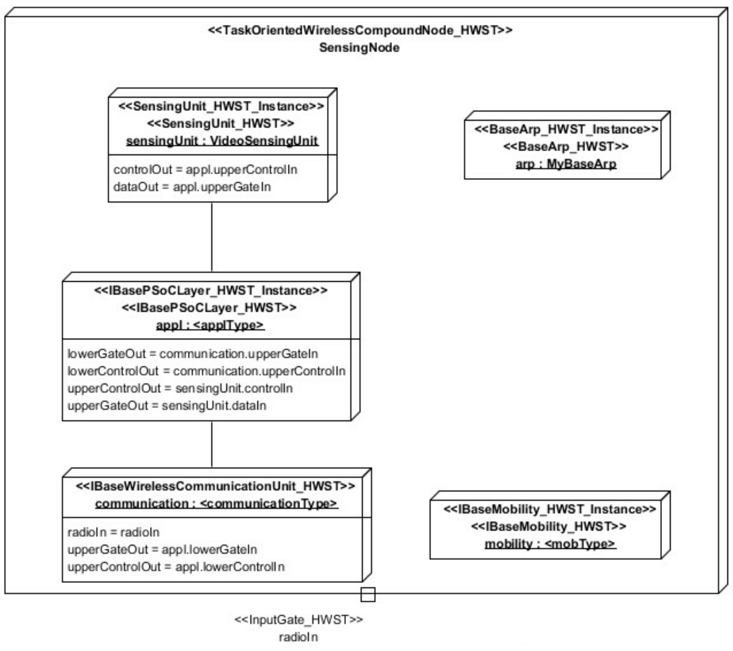

<<TaskOrientedWirelessCompoundNode_HWST>>
SensingNode
<<SensingUnit_HWST_Instance>>
<<SensingUnit_HWST>>
sensingUnit : VideoSensingUnit
controlOut = appl.upperControlIn
dataOut = appl.upperGateIn
<<BaseArp_HWST_Instance>>
<<BaseArp_HWST>>
arp : MyBaseArp
<<IBasePSoCLayer_HWST_Instance>>
<<IBasePSoCLayer_HWST>>
appl : <applType>
lowerGateOut = communication.upperGateIn
lowerControlOut = communication.upperControlIn
upperControlOut = sensingUnit.controlIn
upperGateOut = sensingUnit.dataIn
<<IBaseWirelessCommunicationUnit_HWST>>
communication : <communicationType>
radioIn = radioIn
upperGateOut = appl.lowerGateIn
upperControlOut = appl.lowerControlIn
<<IBaseMobility_HWST_Instance>>
<<IBaseMobility_HWST>>
mobility : <mobType>
<<InputGate_HWST>>
radioIn

Figure 34 UML compound MiXiM like sensing node

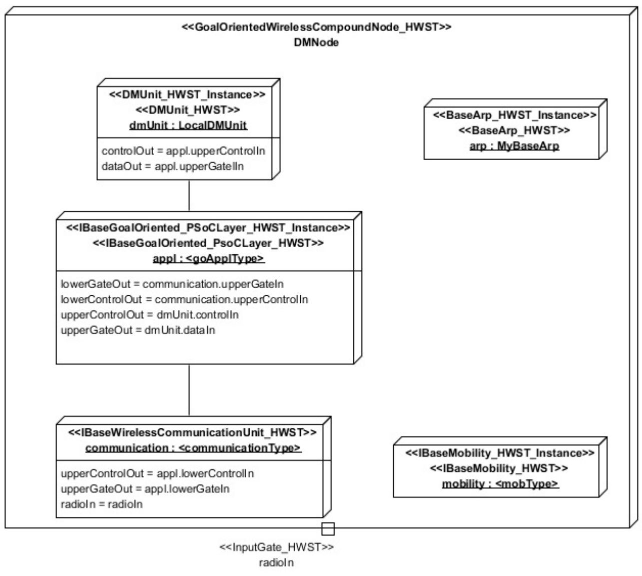

<<GoalOrientedWirelessCompoundNode_HWST>>
DMNode
<<DMUnit_HWST_Instance>>
<<DMUnit_HWST>>
dmUnit : LocalDMUnit
controlOut = appl.upperControlIn
dataOut = appl.upperGateIn
<<BaseArp_HWST_Instance>>
<<BaseArp_HWST>>
arp : MyBaseArp
<<IBaseGoalOriented_PSoCLayer_HWST_Instance>>
<<IBaseGoalOriented_PsoCLayer_HWST>>
appl : <goApplType>
lowerGateOut = communication.upperGateIn
lowerControlOut = communication.upperControlIn
upperControlOut = dmUnit.controlIn
upperGateOut = dmUnit.dataIn
<<IBaseWirelessCommunicationUnit_HWST>>
communication : <communicationType>
upperControlOut = appl.lowerControlIn
upperGateOut = appl.lowerGateIn
radioIn = radioIn
<<IBaseMobility_HWST_Instance>>
<<IBaseMobility_HWST>>
mobility : <mobType>
<<InputGate_HWST>>
radioIn

Figure 35 UML compound MiXiM like decision node

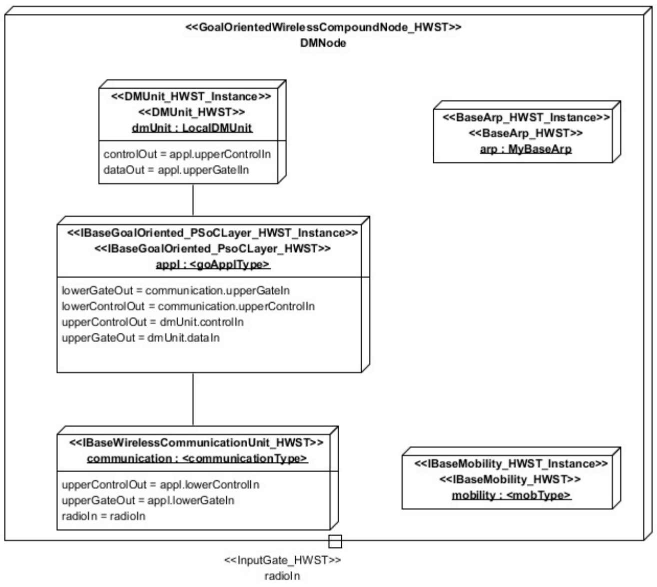

<<GoalOrientedWirelessCompoundNode_HWST>>
DMNode
<<DMUnit_HWST_Instance>>
<<DMUnit_HWST>>
dmUnit : LocalDMUnit
controlOut = appl.upperControlIn
dataOut = appl.upperGateIn
<<BaseArp_HWST_Instance>>
<<BaseArp_HWST>>
arp : MyBaseArp
<<IBaseGoalOriented_PSoCLayer_HWST_Instance>>
<<IBaseGoalOriented_PsoCLayer_HWST>>
appl : <goApplType>
lowerGateOut = communication.upperGateIn
lowerControlOut = communication.upperControlIn
upperControlOut = dmUnit.controlIn
upperGateOut = dmUnit.dataIn
<<IBaseWirelessCommunicationUnit_HWST>>
communication : <communicationType>
upperControlOut = appl.lowerControlIn
upperGateOut = appl.lowerGateIn
radioIn = radioIn
<<IBaseMobility_HWST_Instance>>
<<IBaseMobility_HWST>>
mobility : <mobType>
<<InputGate_HWST>>
radioIn

Figure 36 UML compound MiXiM like traffic light node

The software handling is discussed related to communication aspects at network level. At network level, the communication is ensured by adding internal software components stereotyped with *PSoCCommMapping_SWST* inheritances. In these components resides the contract for communicating, along with the communication context. The context is satisfied by specifying the node identifier and gate name, along with the list of messages that need to trigger external message formats. The external messages are constructed based on the specified custom message concatenated with the parameters that need to be sent.

The handling components from a PSoC device are able to retain the unit's internal state along with the data received from the managed devices. This information can be used to transmit messages to other external nodes which require data regarding the managed unit.

Each time the internal state of the managed unit changes or the handling component receives/sends a command from/to the managed component, the handling strategy requests the bounded communication mapping components for sending messages. The internal handling component receives the external message format and the sending parameters. Then it constructs the message by adding the requested information and it sends the message along with the parameters to the PSoC model component, to manage the sending process.

In Figure 37 the component named *SensingToDMNodeCommunication* is customized for sending a message to the external node called *dmNode*, when *ev_receiveSensedValue* message occurs. The message format is constructed by adding the integer internal value named *sensedValue* to the *ev_receiveXArgValue* string.

In Figure 38, it is presented the strategy manager of a DM node. The DM node is managing a DM unit and it is able to communicate with four traffic light nodes.

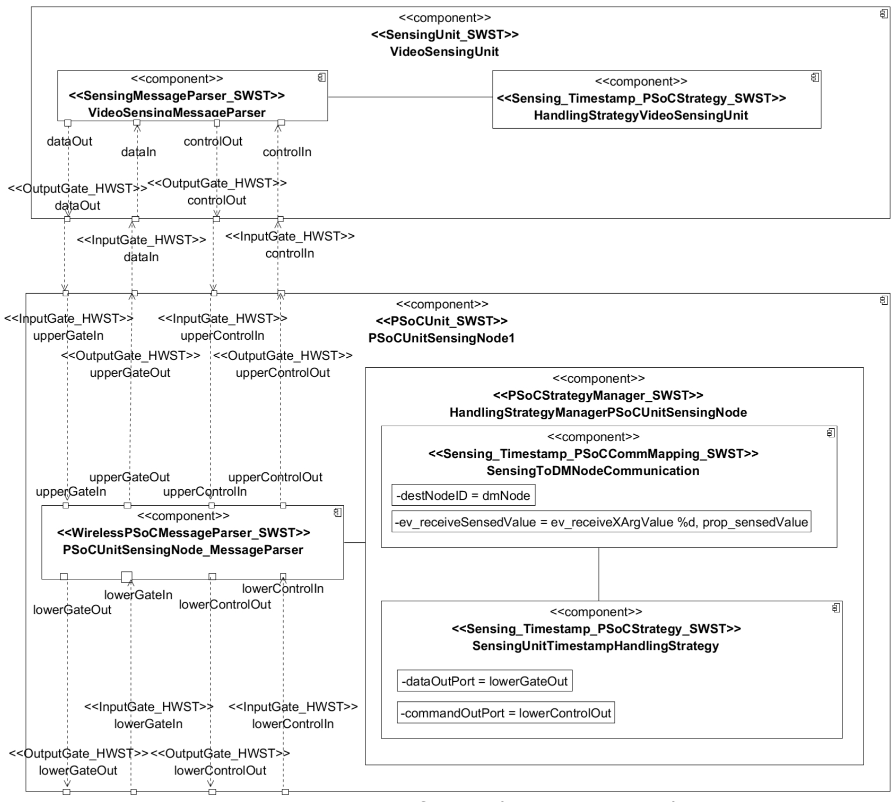
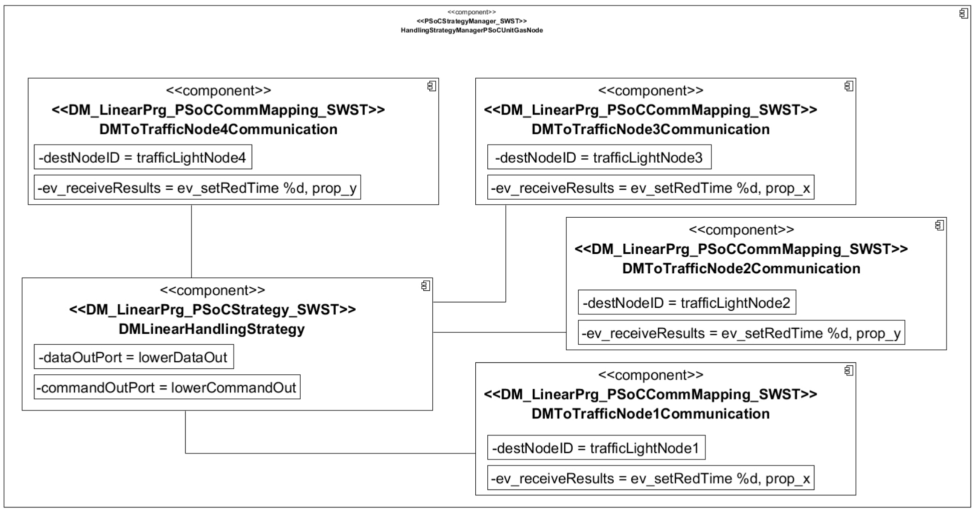


Figure 37 PIM for wireless sensing node


<<component>>
<<PSoCStrategyManager_SWST>>
HandlingStrategyManagerPSoCUnitGasNode
<<component>>
<<DM_LinearPrg_PSoCCommMapping_SWST>>
DMToTrafficNode4Communication
-destNodeID = trafficLightNode4
-ev_receiveResults = ev_setRedTime %d, prop_y
<<component>>
<<DM_LinearPrg_PSoCCommMapping_SWST>>
DMToTrafficNode3Communication
-destNodeID = trafficLightNode3
-ev_receiveResults = ev_setRedTime %d, prop_x
<<component>>
<<DM_LinearPrg_PSoCCommMapping_SWST>>
DMToTrafficNode2Communication
-destNodeID = trafficLightNode2
-ev_receiveResults = ev_setRedTime %d, prop_y
<<component>>
<<DM_LinearPrg_PSoCStrategy_SWST>>
DMLinearHandlingStrategy
-dataOutPort = lowerDataOut
-commandOutPort = lowerCommandOut
<<component>>
<<DM_LinearPrg_PSoCCommMapping_SWST>>
DMToTrafficNode1Communication
-destNodeID = trafficLightNode1
-ev_receiveResults = ev_setRedTime %d, prop_x
Figure 38 Business logic specification for the PSoC unit of decision module node

All the four mapped external messages are sent when the data is received from the DM unit. Therefore, for constructing the messages, the strategy binds the integer value of the *x* property to the *trafficLightNode1* and *3*, and the integer value of the *y* property to the *trafficLightNode2* and *4*, respectively.

6.2.3 Summary

The examples presented in the previous subchapters are two case studies for system design regarding traffic management in an intersection. The difference between the examples resides in the type of communication between the nodes in the network, which can be accomplished using wired or wireless connections. First, the author of this thesis has presented the types of components required for a traffic management application. These types of nodes differ for the two examples because the communication method is different: in case of wired connections, the communication components are customized using Cypress like stereotypes, while for wireless connections, the communication components are customized using MiXiM like stereotypes. These predefined components are used in creating the types of nodes for the application in question: traffic light nodes, sensor nodes and a decision node. Instances of these types of nodes are used in the actual network, represented by a deployment diagram.

For the PIM for hardware specification to be valid, the OCL constraints associated with the stereotypes used for customizing the components used in the application design are checked. When the PIM is proved valid, it is used to generate simulation specifications for OMNeT++ environment. These simulation specifications are included in NED configuration files. While the hardware configuration for components of the nodes and for the entire network is specified starting with UML design and then generating OMNeT++ configuration files, the used must manually specify the characteristics of the behavior of the simulation components.

6.3 An Aircraft Fuel Management System

A modern aircraft consists of several monitoring and management systems, responsible for various objectives regarding keeping the aircraft fully functional during the flight.

The fuel system determines one of the most critical aspects in aircraft design. An aircraft is typically equipped with several fuel tanks and a variety of valves, pumps, probes, sensors and switches. The main purpose of the fuel system is to reliably supply engines with fuel. Other important functions include fuel transfer, based on the center of gravity (CG) at a given moment of time during the flight, engine oil coolant, pressurization and vent or refueling systems. All these functions work together to maintain overall balance and optimal performance of the aircraft during the flight.

6.3.1 Cyber Physical System Network Tailoring in Logical Levels

Given the great complexity of all these systems, this subchapter discusses two critical functionalities: engine feed in the flow subsystem and fuel transfer in the CG subsystem, with focus on illustrating the way they influence each other.

For a better understanding, in this subchapter is considered a typical aircraft fuel system, as shown in Figure 39.

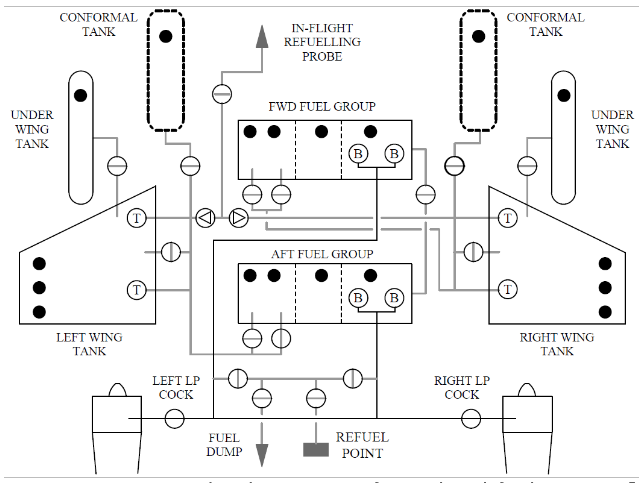


Figure 39 Typical military aircraft top-level fuel system [97]

The fuel is collected from the fuel tanks before being fed to the engines. The *FWD Fuel Group* and the *AFT Fuel Group* collector tanks have the role of ensuring the required fuel quantity for the engines. The fuel tanks are represented by the conformal tanks, under wing tanks, and left and right wing tanks, respectively.

The role of the booster pumps (*B* in Figure 39) inside collector tanks is to provide a normal delivery pressure to the engines. Two booster pumps are available for each collector tank, so that, in case one fails, the other one can take over the fuel transfer.

Two transfer pumps (*T* in Figure 39) are available in each wing tank and in each fuel group. From every pair, the second transfer pump acts as backup, in case of a pump failure. These pumps ensure the pressure for the corresponding transfer and are supposed to run period based.

There are several shut-off valves associated with the fuel transfer, both for the engine feed and CG functionalities. A shut-off valve is available to maintain the desired fuel level between *FWD* and *AFT* fuel groups at the moment when the airplane turns up or down. Other valves have the role to control the transfer from the wing tanks to the collector tanks.

The two cross valves are used to establish the desired flow direction, both during refueling and for fuel transfers from right to left or vice versa.

All tanks are equipped with specialized sensors for constantly measuring the fuel level during flight operations. Maximum capacitance reached or critical fuel level is properly signaled so that the system takes the most appropriate decisions over the managed pumps and valves.

Based on the fuel system model presented in Figure 39, the flow and the CG subsystems are tailored, at different logical levels in Figure 40. As both subsystems act over the same physical devices, they are represented overlapped.
The flow subsystem contains statically configured logical groups, depending on the requirements to be established. The perimeters usually contain a specific sensor tank and the associated valve or group of valves (such as *Under Wing Left Perimeter*, *Conformal Left Perimeter* and *Aft Perimeter*). The *Cross Perimeter* groups the valves used for refueling and left - right transfer. The *Refuel/Defuel Perimeter* controls the fuel feed to the engines and the fuel removal, in case of danger.

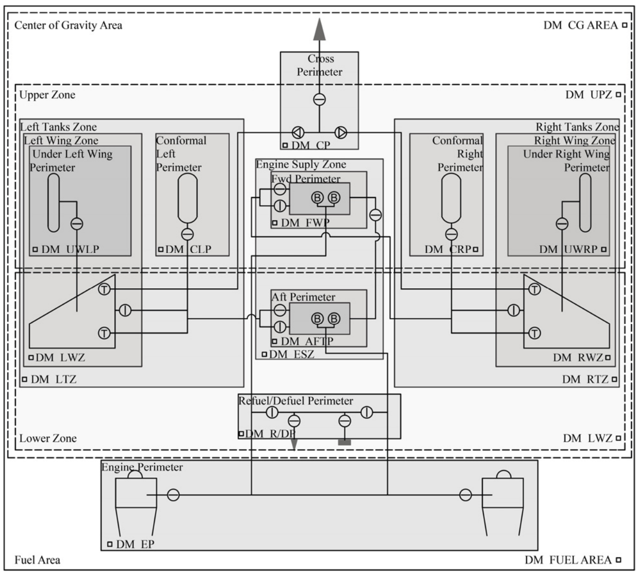


Figure 40 Logical tailoring in flow and CG subsystems for the fuel system [76]

Perimeters along with physical devices are grouped into zones. A meaningful example is represented by the tanks clustered into the *Left Tanks Zone*. The zone's main task is to provide fuel to the *Fuel Supply Zone*, to be later used for feeding the engines.

The *Fuel Area* correlates the functionalities of all devices, perimeters and zones mapped in the fuel system. The objectives are constructed at perimeter level and are transferred up until reaching the area level. Next, the area level establishes specific commands for the designated zones that will lead to accomplishing the goal.

In case of CG subsystem, here are presented only the differences with the flow subsystem, as the tailoring at perimeter level is rather similar. Unlike the flow subsystem, the CG subsystem contains logical zone groups which are dynamically configurable. The zones configuration depends on the actual position of the aircraft. In case of left or right turnings, the zones overlap with the ones in the flow subsystem. In case of up or down movements, the zones are Upper and Lower Zone, respectively, as shown with dotted line in Figure 40. Another difference compared to the flow subsystem is that the objectives are established directly at area level based on the aircraft position and the current capacity level from the tanks.

The objectives are internally set as goals and translated into commands for the lower grouping levels.

6.3.2 Constructing PIM Models at Zone Level

After tailoring the application, the next step represents PIM models construction for every node type used in the CPS network. The DM node for *LTZ* is detailed in Figure 41 and has associated the PIM model presented in Figure 42 and Figure 43. The DM *LTZ* node synchronizes the received data gathered within its local zone (*LWZ*) and perimeter (*CLP*) with the asynchronous goals requested by the upper layers. The PIM models for sensing and actuation nodes are presented in [80] and [81] (subchapter 6.2) and are adapted for these CPS application requirements. They are not subject of this case study, as they are only slightly different than the already presented models.

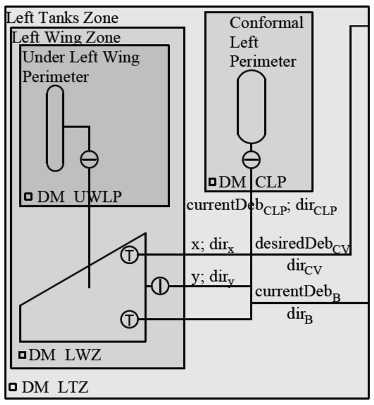


Figure 41 Tailoring of the Left Tanks Zone [76]

Figure 42 uses deployment diagram to list the hardware interconnections between the internal devices of the presented node. The hardware ports are marked as deployment gates. Most of the hardware characteristics are imported directly from the tagged values corresponding to the used stereotypes. The remaining hardware specifications represent application dependent characteristics and are specified as attributes of the deployment nodes. The hardware characteristics of the devices are not detailed in Figure 42, as they are presented in [80] and [81] and subchapter 6.2.

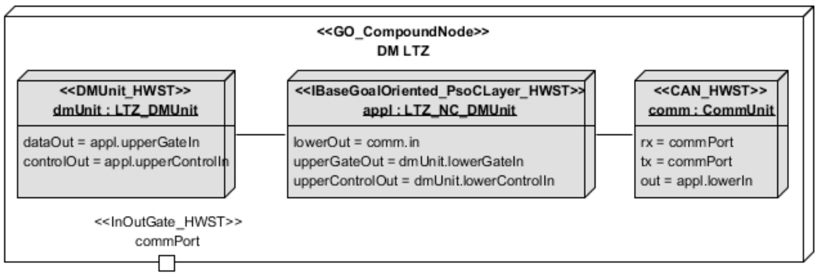


Figure 42 PIM hardware model of DM LTZ node [76]

The PIM software model completes the hardware model and consists of the desired behavior attached to the node. Figure 43 illustrates the software models of *LTZ_NC_DMUnit* and *LTZ_DMUnit* devices from Figure 42.

Both components have an associated strategy stereotyped with an inheritance from *HandlingStrategy_SWST*. All stereotypes are considered having a default PSM implementation, which can be configured by PIM. The *DMHandlingStrategyManager* artifact contains the handling strategy for the *LTZ_NC_DMUnit* device.

It defines the communication link between the internal events related to the managed device and the external ones.

The *LTZ_CommLWZ* and *LTZ_CommCLP* artifacts are responsible with collecting the *LWZ* and *CLP* related fuel capacities, respectively. Also, they transfer to their corresponding DM nodes the desired goals, as requested fuel capacities and directions, illustrated in Figure 41. The *LTZ_CommFuelArea* and *LTZ_CommCGArea* are responsible with maintaining the communication with the two subsystem upper levels in which this zone operates. The goals sent by the areas DMs are transmitted in asynchronous mode as desired fuel directions and capacities.

The goals received by DM *LTZ* from both subsystems are interpreted and validated on demand, before calculating the optimal results. As goals from both subsystems act over the same zone, the following constraints are considered. In case the goals require different fuel directions over the same pipe, the desired direction and capacity of the flow subsystem are considered as the current valid goal. Otherwise, the resulting desired capacity is determined as the sum of the desired capacities set by both subsystems and it is compared with the maximum allowed capacity for that pipe.

Considering *LTZ* presented in Figure 41 the validated goal related to *Engine Supply Zone* is stored in the $desiredDeb_B$ and dir_B variables. Also, the $desiredDeb_{CV}$ and dir_{CV} variables contain the validated goal related to the *Cross Perimeter*.

The desired direction for each pipe is computed based on the equation (15).

$$dir_{pipe} = dir_{Flow} + \left| \left| dir_{Flow} \right| - 1 \right| \cdot dir_{CG}; where\ dir_{pipe}, dir_{Flow}, and\ dir_{CG} \in \{-1,0,1\} \quad (15)$$

The fuel direction *dir* of a pipe is represented as:

- *-1* in case the fuel is transferred outside the grouping level
- *1* when the fuel is added to the tanks managed by the grouping level
- *0* when there is no transfer for that pipe.

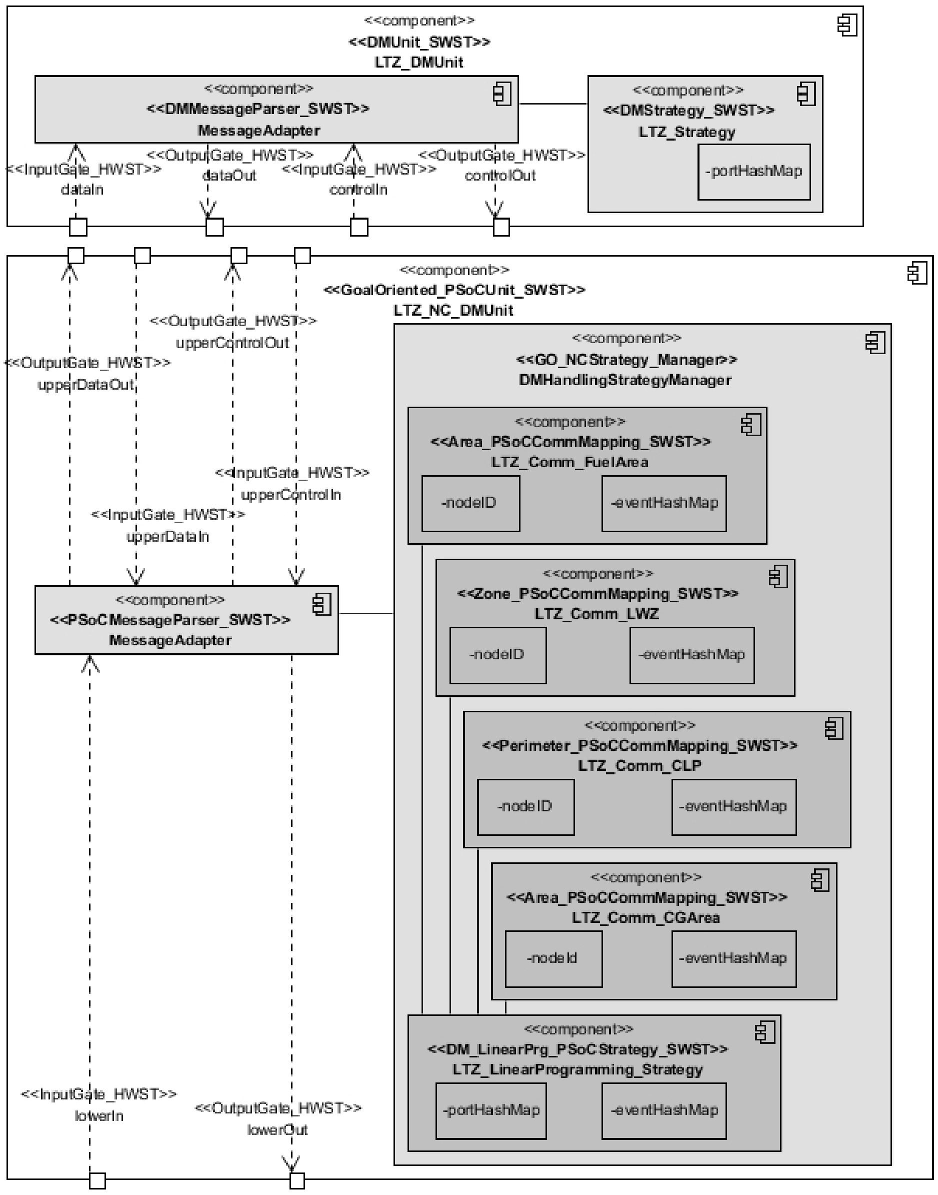


Figure 43 PIM software model of DM LTZ node [76]

LTZ_LinearPrograming_Strategy artifact represents the internal strategy used by NC to manage a DM device, based on linear programming (LP) [98]. LP equations express the DM goals and are used to find an optimal solution.

The relationships between subsystems are expressed in terms of LP constraints. All parameter characteristics, which need to be optimized (*t* and *v* in (2)), are translated into LP equation. The *LTZ_DMUnit* is responsible for solving the LP system.

As an example, in case of *Left Tanks Zone*, the resulting LP is shown below.

$$\begin{cases} dir_t \cdot t + dir_v \cdot v = dir_B \cdot currentDeb_B + dir_{CV} \cdot desiredDeb_{CV} \\ 0 \le t \le \max Deb_T \wedge 0 \le v \le \max Deb_V \\ dir_t = -1 \wedge dir_v = 1 \\ currentDeb_B = desiredDeb_B \\ currentCap_{LTZ} = currentCap_{LWZ} + currentCap_{CLP} \\ \max Cap_{LTZ} = \max Cap_{LWZ} + \max Cap_{CLP} \\ currentCap_{LTZ} + dir_t \cdot t + dir_v \cdot v \le \max Cap_{LTZ} + dir_{CLP} \cdot currentDeb_{CLP} \\ currentCap_{LTZ} + dir_t \cdot t + dir_v \cdot v \ge \min Cap_{LTZ} + dir_{CLP} \cdot currentDeb_{CLP} \end{cases} \tag{16}$$

The abbreviations are as follows:

- *dir* represents the fuel flow direction through a pipe
- *currentDeb* represents the debit of a valve or pipe
- *t* and *v* represent the optimal resulted debits of the pipe connected to the active transfer pump *T* and to the valve tank *V*
- *currentCap, minCap and maxCap* represent the capacity fuel values of a tank.

Using model transformation, PIM and goals can be translated to specific PSM. The simulation middleware introduced in [87] is under development. It is based on the recognition of the types of components and provides basic functionality for handling the goals and constraints described by the user.

6.3.3 Summary

The subchapter presents as case study the design of an aircraft fuel management. The design methodology is based on MDA approach and the starting point is the CIM composed of the defined UML profiles for hardware and software specification of CPS applications. To simplify development, system behavior is tailored using goal-oriented specifications. The user has the possibility to simply specify the goals for the CPS application, the created system being responsible to translate these goals to specific commands for the subsystems. The CPS application goals are expressed as linear programming systems at every abstraction level, similar to the example at zone level given in subchapter 6.3.2.

6.4 Model Validation Through Simulation

The PIM construction in an MDA approach can be followed directly by a deployment to a PSM, on a physical network, as PSM is the final model to build. However, it is recommended to take an intermediary step between PIM and PSM which is PIM validation.

A suitable approach is performing PIM simulation and validation before the deployment of the network in the physical environment, on hardware devices.

There are strong points for performing a simulation before a deployment for an application. Testing applications written for CPSs and tracking the execution steps after deployment is a difficult task due to the great number of messages exchanged between hardware devices.

Maintaining a detailed logging in case of distributed devices is not very feasible. At the same time, even if logging is no longer in question, the fact that PSoC devices usually have a reduced hardware display, offers little information during testing. Additional hardware, as debugging devices, external of the network composed of physical nodes, is required for detecting errors at runtime. These devices are also an expensive solution.

Testing and validating an application through simulation, before deploying it on the physical network, is a more suitable approach and a less expensive one. Along with the material costs reduced, the time necessary for validating an application is reduced in case of simulation than in case of deploying. Loading the application on physical devices after correcting some errors and then continuing with the debugging 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.

The author of this thesis, together with the research team, has defined some simulation models that are helpful in testing and validating CPS applications through simulation. The simulation models are developed using OMNeT++ environment. OMNeT++ simulator overcomes some of the issues that make SystemC unsuitable for simulating distributed embedded applications. OMNeT++ is an object-oriented modular discrete event network simulation framework [99], which allows network simulation on a large scale. Being modular and customizable, it also allows embedding simulations into larger applications and offers support for parallel simulation. OMNeT++ handles each event in sequence and maintains its own virtual clock, independent of the processor's clock [96], which a step forward related to SystemC. Therefore, 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. In case of complex systems simulations, this property allows the elimination of the problems related to real-time synchronization constraints.

OMNeT++ also facilitates the development for applications' models and analysis of the obtained results. The facilities of visualizing and debugging the simulation models are helpful in reducing debugging time. In OMNeT++, communication between modules of the application is made through messages. OMNeT++ provides support for scheduling, sending and receiving messages and provides a library for multithreading applications.

Studies [100] have shown that simulations performed using OMNeT++ are executed at least an order of magnitude faster than the ones performed using Ns-2 [101]. At the same time, OMNeT++ makes more efficient use of the available memory.

The simulation models presented in [86], [87], [88] and [89] and developed in OMNeT++ are independent of the type of communication. These models can be used in simulating distributed applications, as they allow: simulation of interconnected PSoC devices running at the same clock frequency, simulation of interconnected PSoC devices running at different clock frequencies, simulation of devices that have different phase shifts. Also, in [88], a speed-up simulation mechanism intended to improve physical time model simulation is proposed. All these models support both wired and wireless types of communication and are extended in [89].

Interconnection of distributed devices in these models is transparent related to simulation framework and it is left for the developers of the applications to implement. This capability allows for the models in question to be integrated in any derived simulator from the OMNeT++ family.

In [90], Gavrilescu et al. propose an XML-based event driven model specification of the distributed network which allow the developer to describe specific simulation scenarios for the distributed application. The authors also present a reusable event oriented programming model for handling event-driven scenarios. The programming model can be used on any event-driven simulation environment. Also, the presented approach contributes to the reduction of the lines of code required for implementing the distributed applications, with a variable percentage which depends on the application goals.

Another testing methodology, except for running simulation scenarios, is given by formally specifying PIM models and verifying the static and dynamic properties for the UML models. A first attempt in this direction has been made in [91] and was continued with a more detailed study in [102].

Chapter 7. Conclusions and Future Work

7.1 Conclusions

An efficient, intuitive and easy to use design and programming model is desired for CPS applications. It is required that even users without advanced knowledge about sensor networks design methodologies to be able to specify applications of CPSs, at a high level of abstraction. The users command and control such applications by specifying the network topology and component nodes, at hardware level, and by specifying the goals at software level, without taking into consideration the actual limitations imposed by the physical environment.

The users can customize the CPS applications by using predefined artifacts, grouped into two UML profiles, for hardware and software specification of CPS applications, respectively. The customizations can apply at component level, node and network level.

The novelty presented in this thesis allows raising the level of abstraction, first by tailoring the CPS network at logical level and then by requiring that application objectives to be specified only at the highest logical level. The application middleware handles translating the goals at lower logical levels, until the physical level and their accomplishment.

The methodology presented in this thesis uses MDA approach for CPS applications design. The UML profiles for hardware and software specification compose the MDA CIM and are used in defining the MDA PIM. The PIM is defined at both hardware and software level by creating UML models using deployment and component diagrams, respectively. The PIM is validated through simulation and, using code transformations, is later translated into a network deployable PSM. The resulted code represents the final scope of using MDA approach, as it reduces the total amount of code the user must create when programming a CPS application. The objective is to have a PIM as complete as possible and in accordance with the requirements, along with corresponding transformation rules.

7.2 Contributions

The contributions of the author of this thesis regarding CPS design research have been discussed in chapters 2-6. All these proposals regarding CPS applications have been gathered into 15 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 design follow the initial research objectives, described in the proposal for the theme of research, presented in September 2010. These objectives have also been summarized in subchapter 1.3.

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

- At theoretical level
    - A comprehensive study and systematization of publication in CPS design area, MDA approach and goal-oriented programming.
    - The definition of a UML hardware profile for specification of CPS applications. The stereotypes, along with the defined tagged values and OCL constraints are tailored into several groups, depending on their functionality. The profiles presentations are made both by presenting the stereotype hierarchies and by describing the defined stereotypes. The UML hardware profile contains first level stereotypes which map the root node of the other groups of stereotypes, stereotypes for simple and compound modules, stereotypes for PSoC based CPSs, stereotypes for wireless communication, stereotypes for module interfaces.
    - The definition of a UML software profile for specification of CPS applications. The stereotypes for software description, the tagged values and associated OCL constraints, are also tailored into groups and express behavior associated to hardware components, nodes and network, respectively. The UML software profile contains first level stereotypes which are further base stereotypes in the other groups, stereotypes for software part definition, stereotypes for message handling, stereotypes for strategy handling, stereotypes for specific PSoC strategy handling and stereotypes related to communication.
    - The definition of a goal-oriented approach to be used in handling the requirements of CPS applications. The approach implies tailoring the application at logical level into several logical levels. Such an approach allows the user to pose the application goals only at the highest computational level. It is the middleware task to handle goals translation and accomplishment at lower logical levels.
    - The definition of the models implied in the proposed MDA approach, the CIM composed of the defined UML profiles, the PIM obtained starting from CIM and CPS application requirements and the PSM obtained after the PIM is validated and the transformations from one model to the next one.
    - The identification of PIM validation methodologies, by using simulation models, customized for CPS applications and the defined goal-oriented approach.

- At practical level
    - The proposed goal-oriented modeling of CPS applications using MDA approach has been applied to several CPS applications from different domains of activity.
    - 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.
    - Limited code generation in OMNeT++ simulator related to network infrastructure design, presented in the previous detailed case studies.

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 and BDI journals. Also three of them have been presented in workshops relevant to the research topic. The papers are presented below in chronological order:

1. (ISI Proc.) - F. Naghiu, D. Pescaru, G. Magureanu, I. Jian, A. Doboli, “Corrections of sensing errors in Video-based Traffic Surveillance”, 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. 217-224.

2. (IEEE Proc.) - 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.

3. (IEEE Proc.) - 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.

4. (IEEE Proc.) - 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.

5. (IEEE Proc.) - 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.

6. (Workshop) - 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.

7. (BDI Journal) - 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.

8. (IEEE Proc.) - G. Măgureanu, 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.

9. (BDI Journal) - 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.

10. (Workshop) - G. Magureanu, "UML Profile for Wireless Networks Based on Cyber Physical Systems", in Workshop no. 2, "Interdisciplinaritatea si Managementul Cercetarii", Timisoara, Romania, November 2011.

11. (IEEE Proc.) - 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 2011, Timisoara, Romania, pp. 383 – 388, May 2012, doi: 10.1109/SACI.2012.6250034.

12. (Workshop) - G. Magureanu, "Specification And Verification Of UML Models For Cyber Physical Systems", in Workshop no. 3, "Interdisciplinaritatea si Managementul Cercetarii", Oradea, Romania, June 2012.

13. (IEEE Proc.) - 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.

14. (ISI Proc.) - 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.

15. (ISI Journal) - G. Magureanu, M. Gavrilescu, D. Pescaru, "Validation of Static Properties in UML Models for Cyber Physical Systems", Accepted for publication in January 2013 for Journal of Zhejiang University Science C, Impact factor = 0.308 (2011), ISSN 1869-1951.

7.4 Future Research Perspectives

Future work will be focused on developing a tool specific for CPS design, which will be based on the presented MDA approach and will help defining the models involved and the transformations between them. Also, future work will cover a full implementation for a goal-oriented middleware intended for CPS networks constructed on PSoC devices.

References

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

[2] J. Jensen, D. Chang and E. Lee, "Model-Based Design Methodology for Cyber-Physical Systems", in *Proceedings of the 1st IEEE Workshop on Design, Modeling and Evaluation of Cyber-Physical Systems (CYPHY)*, Istambul, Turkey, 2011.

[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 Official Website – http://www.omg.org, 2012.

[5] L. Kuzniarz, M. Staron, and C. Wohlin "An empirical study on using stereotypes to improve understanding of UML models", in *Proceedings of the International Workshop on Program Comprehension*, IEEE Computer Society, 2004, pp. 14–23.

[6] D. Nessett, "Massively Distributed Systems: Design Issues and Challenges", in *Proceedings of the USENIX Embedded Systems Workshop*, Massachusetts, USA, March 1999.

[7] S. Heath, "Embedded Systems Design", publisher Butterworth-Heinemann, ISBN 0-7506-3237-2, 1997.

[8] "Cyber-Physical Systems Summit", Report, Missouri, USA, April 2008.

[9] P.Tabuada, "Cyber-Physical Systems: Position Paper", *in NSF Workshop on Cyber-Physical Systems*, 2006.

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

[11] P. Levis et al., "TinyOS: An Operating System for Sensor Networks", from Book "Ambient Intelligence", Publisher Springer Berlin Heidelberg, ISBN 978-3-540-27139-0, 2005.

[12] 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.

[13] 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", *IFSA Sensors and Transducers Journal*, ISSN 1726-5479, Vol. 5, March 2009, pp. 69-85.

[14] R. Sugihara, R. K. Gupta, "Programming Models for Sensor Networks: A Survey", in *ACM Transactions on Sensor Networks (TOSN)*, March 2008.

[15] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler, "The *nesC* Language: A Holistic Approach to Networked Embedded Systems", in *Proceedings of Programming Language Design and Implementation (PLDI)*, San Diego, USA, June 2003.

[16] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, M. A. Perillo, "Middleware to Support Sensor Network Applications", in *IEEE Network* Journal, Vol. 18, ISSN 0890-8044, Jan/Feb 2004.

[17] K. Whitehouse, F. Zhao, J. Liu, "Semantic Streams: A Framework for Composable Semantic Interpretation of Sensor Data", in *Proceedings of the 3rd European Workshop on Wireless Sensor Networks (EWSN)*, Zurich, Switzerland, February 2006, pp. 5–20.

[18] R. Newton, M. Welsh, "Region Streams: Functional Macroprogramming for Sensor Networks", in *Proceedings of the 1st International Workshop on Data Management for Sensor Networks (DMSN)*, Toronto, Canada, August 2004.

[19] R. Newton, Arvind, M. Welsh, "Building up to Macroprogramming: An Intermediate Language for Sensor Networks", in *Proceedings of the 4th International Conference on Information Processing in Sensor Networks (IPSN)*, Los Angeles, USA, April 2005.

[20] R. Gummadi, O. Gnawali, R. Godivan, "Macro-programming Wireless Sensor Networks using Kairos", in *Proceedings of the 1st International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 126-140, California, USA, 2005.

[21] C. Borcea, C. Intanagonwiwat, P. Kang, U. Kremer, L. Iftode, "Spatial Programming using Smart Messages: Design and Implementation", in *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Tokyo, Japan, March 2004, pp. 690-699.

[22] C. Intanagonwiwat, R. Gupta, A. Vahdat, "Declarative Resource Naming for Macroprogramming Wireless Networks of Embedded Systems", Technical Report CS2005-0827. University of California, San Diego, 2005.

[23] Y. Ni, U. Kremer, A. Stere, L. Iftode, "Programming Ad-hoc Networks of Mobile and Resource-Constrained Devices", in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Chicago, USA, June 2005, pages 249–260.

[24] A. Aguiar, P. R. Pinheiro, A. L.V. Coelho, N. Nepumoceno, A. Neto, R. Cunha, "Scalability Analysis of a Novel Integer Programming Model to Deal with Energy Consumption in Heterogeneous Wireless Sensor Networks", in *Communications in Computer and Information Science Journal*, Vol. 14, 2008, pages 11-20.

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

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] R. van Renesse, "Goal-oriented programming, or composition using events, or threads considered harmful", in *8th ACM SIGOPS European Workshop*, Sintra, Portugal, September 1998.

[33] W. Heaven, A. Finkelstein, "A UML Profile to Support Requirements Engineering with KAOS", in *IEEE Proceedings - Software*, Vol. 151, 1, February 2004, pp. 10-27.

[34] E. Navarro, P. Letelier, I. Ramos, "UML Visualization for an Aspect and Goal-Oriented Approach", in the *5th Aspect-Oriented Modeling Workshop (AOM)*, collocated to UML 2004 Conference, Lisbon, Portugal, October 2004.

[35] U. Saif, H. Pham, J. Mazzola Paluska, J. Waterman, C. Terman, S. Ward, "A Case for Goal-oriented Programming Semantics", in *System Support for Ubiquitous Computing Workshop at the 5th Annual Conference on Ubiquitous Computing, (UbiComp)*, 2003.

[36] J. Mazzola Paluska, H. Pham, U. Saif, C. Terman, S. Ward, "Reducing Configuration Overhead with Goal-Oriented Programming", in *Proceedings of the 4th annual IEEE International Conference on Pervasive Computing and Communications Workshops*, 2006.

[37] G. de Sousa, J. de Castro, "Towards a Goal-Oriented Requirements Methodology Based on the Separation of Concerns Principle", in *Workshop on Requirements Engineering (WER)*, Brasil, November 2003.

[38] M. Kim, M. Stehr, C. Talcott, "A distributed logic for networked cyber-physical systems", in *Proceedings of the 4th IPM International Conference on Fundamentals of Software Engineering (FSEN)*, Springer-Verlag, Berlin, pp. 190-205.

[39] M. Hause, F. Thom, "Building Bridges between Systems and Software with SysML and UML", in *Proceedings of the 18th INCOSE International Symposium*, Utrecht, Nederland, June 2008.

[40] L. Rioux, T. Saunier, S. Gerard, A. Radermacher, R. de Simone et al., "MARTE: A New Profile RFP for the Modeling and Analysis of Real-time Embedded Systems", in *DAC 2005 Workshop UML for SoC Design (UML-SoC)*, Anaheim, CA, USA, June 2005.

[41] G. Martin, W. Mueller, "UML for SoC Design", Springer, 2005.

[42] T. Grötker, S. Liao, G. Martin, S. Swan, "System Design with SystemC", Springer, ISBN 1-4020-7072-1, 2002.

[43] Papyrus Tool– http://www.eclipse.org/modeling/mdt/papyrus/, 2012.

[44] UML2 Tools – http://wiki.eclipse.org/MDT-UML2Tools, 2012.

[45] Smart Development Environment – http://www.visual-paradigm.com/, 2012.

[46] I. Podnar, B. Mikac, A. Caric, "SDL based approach to software process modeling", in *Lecture Notes in Computer Science*, Volume 1780, 2000, pp. 190-202.

[47] 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.

[48] S. Abdelrahman, M. Badawy, "HASoC for developing a software system", in *International Journal of Computer Science Issues,* Vol. 8, Issue 6, No. 1, November 2011.

[49] D. Harel and R. Marelly, "Specifying and Executing Behavioral Requirements: The Play In/Play-Out Approach", in *Software and System Modeling (SoSyM)*, 2003, pp. 82-107.

[50] J. Zimmerman, O. Bringmann, J. Gerlach, F. Schaefer, U. Nageldinger, "Holistic system modeling and refinement of intern-connected micro-electronic systems", in *Proceedings of the Conference on Design, Automation and Test in Europe IEEE*, Los Alamitos, CA, 2008.

[51] MDA with Executable UML, http://www.kc.com/XUML/, 2012.

[52] A. Koudri, J. Champeau, J. Le Lann, V. Leilde, "MoPCoM Methodology: Focus on Models of Computation", in Lecture Notes in Computer Science, Vol. 6138, 2010, pp. 189-200.

[53] 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.

[54] 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.

[55] 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)*, Lisbon, Portugal, 2004.

[56] K. 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.

[57] P. Kukkala, J. Riihimaki, M. Hannikainem, T. Hamalainen, K. Kronlof, "UML 2.0 profile for embedded system design", in *Proceedings of the Design, Automation and Test in Europe*, Vol. 2, 2005, pp. 710–715.

[58] 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.

[59] 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.

[60] Simulink, Simulation and Model-Based Design – http://www.mathworks.com/products/simulink/, 2012.

[61] Electronic Design Automation Consortium, http://www.edac.org, 2012.

[62] TTTech Computertechnik AG, http://www.tttech.com/, 2012.

[63] T. A. Henzinger, C. M. Kirsch, M. A.A. Sanvido, W. Pree. "From control models to real-time code using Giotto", in *IEEE Control Systems Magazine,* 23(1):50-64, 2003.

[64] D. Bacon, P.Cheng, V. T. Rajan, "The Metronome: A Simpler Approach to Garbage Collection in Real-Time Systems", in *Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES)*, OTM Workshops, 2003.

[65] H. Neema, H. Nine, G. Hemingway, J. Sztipanovits, G. Karsai, "Rapid Synthesis of Multi-Model Simulations for Computational Experiments in C2", in AFCEA-GMU Symposium, May 2009.

[66] J. Sztipanovits, "Frameworks and Tools for High-Confidence Design of Adaptive, Distributed Embedded Control Systems: Project Overview", Presentation for *Frameworks and Tools for High-Confidence Design of Adaptive, Distributed Embedded Control Systems*, December 2009.

[67] PhyNet Project, http://www.cmu.edu/silicon-valley/research/cps/index.html, 2012.

[68] 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.

[69] G. A. Lewis, B. Craig Meyers, K. Wallnau, "Workshop on Model-Driven Architecture and Program Generation", Technical Note, Pittsburgh, Pennsylvania, USA, June 2006.

[70] I. Sacevski, J. Veseli, "Introduction to Model Driven Architecture (MDA)", University of Salzburg, Seminar Paper, June 2007.

[71] G. Nunes Rodrigues, G. Roberts, W. Emmerich, J. Skene, "Reliability Support for the Model Driven Architecture", in *Workshop on Software Architecture for Dependable Systems (ICSE/WADS)*, Portland, USA, May 2003, pages 7-12.

[72] C. K. Fong, "Successful Implementation of Model Driven Architecture", White paper, June 2007.

[73] A. Mos, J. Murphy, "Performance Management in Component-Oriented Systems Using a Model Driven Architecture™ Approach", in *Proceedings of the 6th International Enterprise Distributed Object Computing Conference (EDOC)*, Lausanne, Switzerland, September 2002.

[74] S. Deelstra, M. Sinnema, J. van Gurp, J. Bosch, "Model Driven Architecture as Approach to Manage Variability in Software Product Families", in *Proceedings of the Workshop on Model Driven Architectures: Foundations and Applications*, Netherlands, June 2003.

[75] 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 International Symposium on Intelligent Systems and Informatics (SISY)*, Subotica, Serbia, November 2010, pp. 471–476.

[76] 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.

[77] 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.

[78] F. Naghiu, D. Pescaru, G. Magureanu, I. Jian, A. Doboli, "Corrections of sensing errors in Video-based Traffic Surveillance", 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. 217-224.

[79] D. Jansen et al., "A Probabilistic Extension of UML Statecharts", in *Formal Techniques in Real-Time and Fault-Tolerant Systems*, in *Lecture Notes in Computer Science*, Vol. 2469, Springer, 2002, pp. 355-374.

[80] 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),* Timisoara, Romania, May 2011.

[81] 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)*, Timisoara, Romania, May 2012, pp. 383 – 388, doi: 10.1109/SACI.2012.6250034.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

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

[89] M. Gavrilescu, G. Magureanu, D. Pescaru, A. Doboli, "Time Models for PSoC Based Cyber Physical Systems Simulation", in *BS-UPT TACCS* Volume 56 (70) No. 1, March 2011, pp. 27-34.

[90] 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.

[91] G. Magureanu, "Specification and Verification of UML Models for Cyber Physical Systems", in *Workshop no. 3, "Interdisciplinaritatea si Managementul Cercetarii"*, Oradea, Romania, June 2012.

[92] 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.

[93] Cypress Semiconductor Corporation – http://www.cypress.com, 2012.

[94] MIXIM Project – http://mixim.sourceforge.net/, 2012.

[95] G. Magureanu, "UML Profile for Wireless Networks Based on Cyber Physical Systems", in *Workshop no. 2, "Interdisciplinaritatea si Managementul Cercetarii"*, Timisoara, Romania, November 2011.

[96] 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.

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

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

[99] http://www.omnetpp.org/doc/omnetpp41/manual/usman.html – OMNeT++ User Manual ver. 4.1, 2011.

[100] 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.

[101] 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.

[102] G. Magureanu, M. Gavrilescu, D. Pescaru, "Validation of Static Properties in UML Models for Cyber Physical Systems", Accepted for publication in January 2013 for Journal of Zhejiang University Science C, Impact factor = 0.308 (2011), ISSN 1869-1951.

This work was partially supported by the strategic grant POSDRU/88/1.5/S/50783 Project ID50783 (2009), co-financed by the European Social Fund – Investing in People, within the Sectoral Operational Programme Human Resources Development 2007 - 2013.