Chapter 4
MARTE for CPS and CPSoS

Present and Future, Methodology and Tools

Frédéric Mallet, Eugenio Villar and Fernando Herrera

Abstract Cyber-Physical Systems (CPS) combine discrete computing elements together with physical devices in uncertain environment conditions. There have been many models to capture different aspects of CPS. However, to deal with the increasing complexity of these ubiquitous systems, which invade all the part of our lives, we need an integrated framework able to capture all the different views of such complex systems in a consistent way. We also need to combine tools to analyze their expected properties and guarantee safety issues. Far from handing out a full-fledge solution, we merely explore a possible path that could bring part of the solution. We advocate for relying on UML models as a *unifying* framework to build a single-source modeling environment with design, exploration and analysis tools. We comment on some useful extensions of UML, including MARTE and SysML, and show how they can together capture different views of CPS. We also report on some recent results obtained and discuss possible evolutions in a near future.

Keywords MARTE · Cyber-Physical Systems · Systems of systems · Model-driven methodology · System engineering

4.1 Introduction

4.1.1 CPS and CPSoS

Cyber-Physical Systems combine digital computational systems with surrounding physical processes. Computations are meant to control and monitor the physical environment, which in turn affects the computations.

F. Mallet (✉)
Université Côte d'Azur, CNRS, Inria, I3S, 06900 Sophia Antipolis, France
e-mail: Frederic.Mallet@unice.fr

E. Villar · F. Herrera
TEISA, GESE, Universidad de Cantabria, Av. Castros sn, 39011 Santander, Spain
e-mail: evillar@teisa.unican.es

F. Herrera
e-mail: fherrera@teisa.unican.es

The main characteristics of Cyber Physical Systems and main design challenges have been identified some years ago [22, 23]. CPS are: heterogeneous, in the sense that they combine various models of computations relying on both discrete and continuous time abstractions; platform-aware and resource-constrained, and thus the software depends on various non-functional properties imposed by the platform; time-sensitive and often safety-critical; widely distributed with heterogeneous interconnects.

CPS are first and foremost complex systems and as such designing them requires several models, usually hierarchical, to fully capture the different aspects and views, whether structural or behavioral. Structural models include a description of the components or blocks of the systems and of the communication media involved. Behavioral models include hierarchical state machines and dataflow streaming diagrams. Expected or faulty interactions with the surrounding environment can be captured as a set of use cases or requirements that correspond to positive or negative scenarios. Such models are usually called **heterogeneous** in the sense that they combine different models, each of which may follow a different *model of computation*.

CPS also have the main characteristics of embedded systems, which are usually **platform-aware**. Contrary to standard software engineering, embedded system design depends a lot on the execution platform on which the system should execute, be it a system-on-a-chip (SoC), with multiple computing resources and a complex memory hierarchy, or a wide scale distributed system, with potentially all the variety of interconnects and communication media. This awareness of the platform makes it important to account for how and when the available resources are accessed or 'consumed', considering together both the spatial and temporal dimensions. The spatial dimension is not only about how much resource is available but also about where the resources are physically located in the system relative to each other. How much resource is available is indeed easy since it is usually given by the technology used and the targeted selling price. However, how the resources are used makes all the difference between two a priori equivalent products. The spatial dimension encompasses the interconnect topology, i.e., physical parallelism available, but also and more importantly where the data and programs are allocated. Indeed, the distance between the data memory and the computing resource that executes the program largely impact the fetching time that may potentially largely exceed the computing time. Then, this spatial distribution requires to perform the temporal scheduling of both the execution of programs and the routing of data from memory to computing resources, forth and back. This leads to logical concurrency coming both from the physical parallelism and the inherent data and control dependencies of the application. CPS are therefore **resource-constrained** and **time-sensitive** systems. Even though the resources (memory size, computing power) are not necessarily as scarce as they used to be, nevertheless finding the right tradeoff between the resource usage, the computation speed and the cost makes it a multi-criteria optimization problem difficult to solve. The cost is not only measured in terms of money, but includes all kinds of additional extra functional properties (like power, energy, thermal dissipation), also called **non-functional properties**.

More than being mere embedded systems, CPS are usually made of multiple interconnected embedded subsystems, some of which are computing devices and some other being physical devices. This requires some abilities to describe **heterogeneous interconnects**, while simpler embedded systems usually only rely on homogeneous communication structures. Being made of several computing devices also constitute a big step since it requires to model the whole system as a closed model, with software, devices but also with the environment and the expected **continuous interactions** with this environment. In standard software development, the environment is by definition outside the system to be developed. **Close loop systems** have been modeled for several years with tools and techniques to find approximate solutions to differential equations and are well established. However, the integration with discrete models still causes problems in many tools and each of them proposes ad-hoc solutions. A seamless integration with a generic environment is still to be proposed.

Finally, cyber-physical systems are often big and often interact directly with users that are not even aware of the computer. The size is an aggravating factor since a single system concerns potentially millions of people (smart cities, intelligent transportation systems…). It means some CPS are **safety-critical**, just like embedded systems but at a larger scale. The large scale and the integration of multiple systems gives rise to the notion of System of Systems. It also increases the demand to have sound and scalable models along with verification tools. Sometimes, they also require certification tools to be accredited and allowed to be used in public environments (e.g., unmanned aerial vehicle). However, we do not address at all the certification issue in this chapter.

Moore's Law has dominated the (re-)evolution of electronics during the last quarter of the 20th century. All the electronic products we use today depend directly or indirectly on the increasing integration capability allowed by semiconductor technology. If Moore's Law has changed the world, its end may have a similar effect. For the first time, the underlying technology will be stable with only incremental improvements in time. Cyber-Physical Systems of Systems (CPSoS [9])[1] will dominate the electronics century becoming pervasive in all the aspects of our daily lives. In this new scenario, modeling, analysis and verification of CPSs must evolve. The tendency is toward complex, heterogeneous, distributed networks of many computing nodes. Services will be offered by the interaction of functional components deployed onto many distributed computing resources of many kind, from small motes,[2] embedded systems and smart-phones to large data centers and even High-Performance Computing (HPC) facilities. Electronic design in this new context should address effectively new requirements. Among them, scalability, reusability, human interaction, easy modeling, fast design-space exploration and optimization, powerful functional and extra-functional verification, efficient handling of mixed-criticality and security. An essential aspect will be the availability of powerful, CPS modeling and analysis frameworks able to produce automatically efficient implementations of the system

[1]We use CPS or CPSoS interchangeably, while most considered systems are complex enough to be seen as an integration of systems with more or less explicit interactions.

[2]Motes: embedded devices consisting of sensors, radios, and microprocessors.

model on many different computing resources. The Unified Modeling Language (UML) is a very good candidate to provide the modeling means in this new context.

4.1.2 Role of UML and Its Extensions

The Unified Modeling Language [33] is a general-purpose modeling language specified by the Object Management Group (OMG). It proposes graphical notations to represent all aspects of a system from the early requirements to the deployment of software components, including design and analysis phases, structural and behavioral aspects. As a general-purpose language, it does not focus on a specific domain and relies on a weak, informal semantics to widen its application field. However, when targeting a specific application domain and especially when building trustworthy software components or for critical systems where lives may be at stake, it becomes necessary to extend the UML and attach a formal semantics to its model elements. The simplest and most efficient extension mechanism provided by the UML is through the definition of profiles. A UML profile adapts the UML to a specific domain by adding new concepts, modifying existing ones and defining a new visual representation for others. Each modification is done through the definition of annotations (called stereotypes) that introduce domain-specific terminology and provide additional semantics. However, the semantics of stereotypes must be compatible with the original semantics (if any) of the modified or extended concepts, i.e., the base metaclass.

Domain-specific profiles, such as the UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE [32]) extends the UML with generic features required for real-time and embedded systems. Still, MARTE should be extended [34] to cover the modeling of the complete CPS and in particular the physical models that are usually left outside the scope of classical digital (cyber) models.

SysML [31] is another extension dedicated to systems engineering and has been successfully used to cope with physical models.

To summarize, CPS demand the integration of continuous models, classical state-based or dataflow models, hardware descriptions, and non-functional constraints. UML offers a tool-neutral non-proprietary solution that already contains most of the required notations. However, those notations need to be tailored to capture specific aspects of CPS (time, non-functional properties, continuous models). Both MARTE and SysML offer some extensions dedicated to these goals, and we discuss here some examples of useful features of either MARTE or SysML to model CPS. These notations also need to come with adequate, not tool-specific, explicit semantics if we are to address safety-critical issues.

4.1.3 *Outline*

Focusing on the key characteristics of CPS, we present some selected aspects of MARTE in Sect. 4.2. In Sect. 4.3, we select a simple case study addressed in a recent project and show which aspects can be modeled using UML, MARTE, SysML and some extensions that we thought were lacking in MARTE. More specifically, we explore two extensions, one to allow for the modeling of systems with mixed-criticality issues, another to deal with design space exploration. Section 4.4 describes some developed tools. Section 4.5 discusses the potential future use of UML in the modeling of CPSs, before our partial conclusion in Sect. 4.6.

4.2 Overview of MARTE

4.2.1 *Overview*

The UML profile for MARTE [32] extends the UML with concepts related to the domain of real-time and embedded systems. It supersedes the UML profile for Schedulability, Performance and Time (SPT [30]) that was extending the UML 1.x and that had limited capabilities. UML 2.0 has introduced a simple (or even simplistic) model of time and has proposed several new extensions that made SPT unusable. Therefore MARTE has been defined to be compatible with UML Simple Time model and now supersedes SPT as the official OMG specification.

SysML [13] is another extension dedicated to systems engineering. We use some notations from SysML and we introduce those notations when required. The task forces of MARTE and SysML have synchronized their effort to allow for a joint use of both profiles. The remainder of this subsection gives an overview of MARTE, which consists of three parts: Foundations, Design and Analysis. We try to give a general overview of most of it while focusing on the aspects that we use in our examples.

4.2.2 *Foundations*

The foundation part of MARTE is itself divided into five chapters: *CoreElements*, *NonFunctionalProperties* (NFP), *Time*, *GenericResourceModeling* (GRM) and *Allocation*.

4.2.2.1 CoreElements

Define configurations and modes, which are key parameters for analysis. However, their realization is mainly inspired from classical UML State Machines and we do not use specific constructs from this chapter.

4.2.2.2 NFP

Gives a support to describe Non-Functional properties. In real-time systems, preserving the non-functional (or extra-functional) properties (power consumption, area, financial cost, time budget…) is often as important as preserving the functional ones. The UML proposes no mechanism at all to deal with non-functional properties and relies on mere strings of characters for that purpose, while one could expect a richer type-based system to guarantee the well-formedness of expressions.

4.2.2.3 Time

Is often considered as an extra-functional property that comes as a mere annotation after the design. These annotations are fed into analysis tools that check the conformity without any actual impact on the functional model: e.g., whether a deadline is met, whether the end-to-end latency is within the expected range. Sometimes though, time can also be of a functional nature and has a direct impact on what is done and not only when it is done. All these aspects are addressed in the time chapter of MARTE.

4.2.2.4 GRM

Chapter provides annotations to capture the available resources on which the applicative part shall be deployed.

4.2.2.5 Allocation

Chapter gives a SysML-compatible way to make deployments. In MARTE, we use the wording allocation since the UML deployment usually implies (in people's mind) a physical distribution of a software artifact onto a physical node. Allocation in MARTE goes further. It encompasses the physical distribution of software onto hardware, but also of tasks onto operating system processes, and, more importantly, it covers the temporal distribution (or scheduling) of operating parts that need to share a common resource (e.g., several tasks executing on a single core processor, distributed computations communicating through an interconnect).

4.2.3 Non-functional Properties

The *NFP subprofile* offers mechanisms to describe the quantitative as well as the qualitative aspects of properties and to attach a unit and a dimension to quantities. It defines a set of predefined quantities, units and dimensions and supports customization. NFP comes with a companion language called Value Specification Language (VSL) that defines the concrete syntax to be used in expressions of non-functional properties. VSL also recommends syntax for user-defined properties.

There are two levels of support in MARTE. The first level is a set of ready-to-use types that should be imported by users whenever relevant. For instance, the *MeasurementUnits* Library in MARTE provides a set of predefined units and dimensions, most of the ones that are relevant from the International System of Units (SI) [36]. Figure 4.1 (upper part) shows an except of such elements. A symbol is assigned to each dimension. This symbol is used to build new dimensions like *PowerUnitKind* defined as the square of length times a weight divided by a time to the power 3. The second level is a set of mechanisms to create constructs that do not exist in MARTE, for instance to build new units or new dimensions. Figure 4.1 (lower part) shows, on the left side, the creation of a new dimension to represent a Torque (or moment force), and, on the right-hand side, the creation of one new type. NFP_Weight is proposed as part of MARTE libraries and NFP_Torque is created following the same construction mechanisms. We shall use this dimension in subsequent examples to represent the torque of a rotor in a quadrotor system.

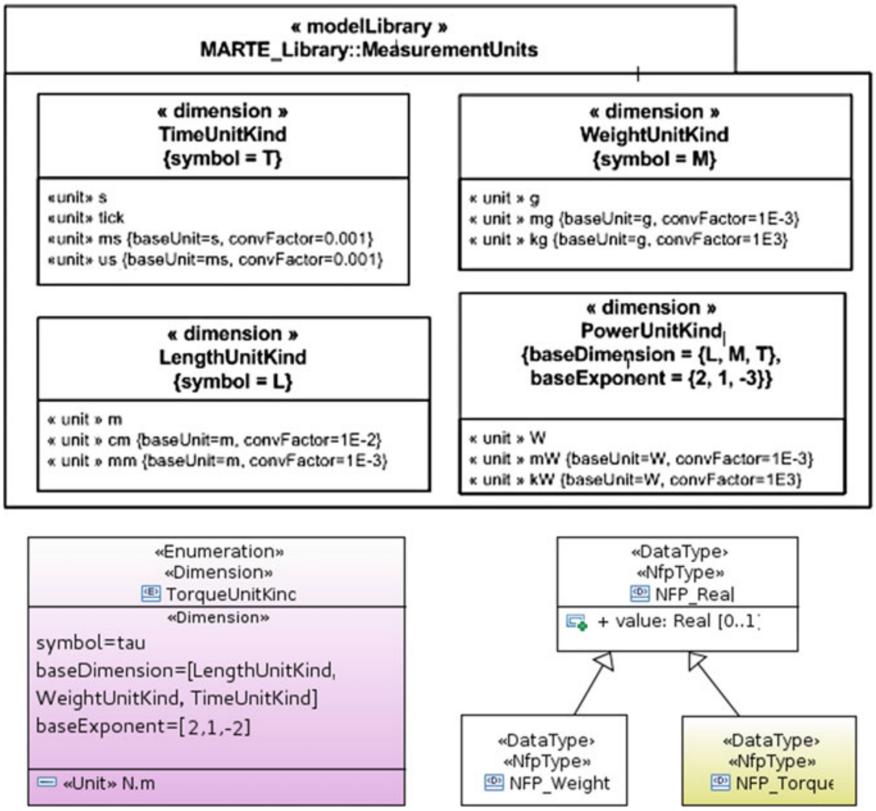
« modelLibrary »
MARTE_Library::MeasurementUnits
« dimension »
TimeUnitKind
{symbol = T}
«unit» s
«unit» tick
«unit» ms {baseUnit=s, convFactor=0.001}
«unit» us {baseUnit=ms, convFactor=0.001}
« dimension »
WeightUnitKind
{symbol = M}
« unit » g
« unit » mg {baseUnit=g, convFactor=1E-3}
« unit » kg {baseUnit=g, convFactor=1E3}
« dimension »
LengthUnitKind
{symbol = L}
« unit » m
« unit » cm {baseUnit=m, convFactor=1E-2}
« unit » mm {baseUnit=m, convFactor=1E-3}
« dimension »
PowerUnitKind
{baseDimension = {L, M, T},
baseExponent = {2, 1, -3}}
« unit » W
« unit » mW {baseUnit=W, convFactor=1E-3}
« unit » kW {baseUnit=W, convFactor=1E3}
«Enumeration»
«Dimension»
TorqueUnitKinc
«Dimension»
symbol=tau
baseDimension=[LengthUnitKind,
WeightUnitKind, TimeUnitKind]
baseExponent=[2,1,-2]
«Unit» N.m
«DataType»
«NfpType»
NFP_Real
+ value: Real [0..1]
«DataType»
«NfpType»
NFP_Weight
«DataType»
«NfpType»
NFP_Torque

Fig. 4.1 NFP subprofile of MARTE: two levels of support

MARTE tools rely on such mechanisms to perform dimension analyses, which is of paramount importance for system engineering.

4.2.4 Time

The time model of MARTE has been extensively described before [1, 2, 26], we give here the minimum to understand its role for modeling CPS.

Time in SPT is a metric time with implicit reference to physical time. As a successor of SPT, MARTE supports this model of time. UML 2, issued after SPT, has introduced a model of time called *SimpleTime*. This model also makes implicit references to physical time, but is too simple for use in real-time applications, and was initially devised to be extended in dedicated profiles.

MARTE goes beyond SPT and UML 2. It adopts a more general time model suitable for system design. In MARTE, Time can be physical, and considered as continuous or discretized, but it can also be logical, and related to user-defined clocks. Time may even be multiform [4], allowing the use of different units (seconds, steps, processor ticks, heartbeats) to refer to temporal phenomena and allowing different time logics with progresses in a non-uniform fashion, and possibly independently to any (direct) reference to physical time. In MARTE, time is represented by a collection of *Clocks*. The use of word *Clock* comes from vocabulary used in the synchronous languages. They may be understood as a specific kind of events on which constraints (temporal, hence the name, but also logical ones) can be applied. Each clock specifies a totally ordered set of instants, i.e., a sequence of event occurrences. There may be dependence relationships between the various occurrences of different events. Thus this model, called the MARTE time structure, is akin to the Tagged Systems [21]. To cover continuous and discrete times, the set of instants associated with a clock can either be dense or discrete.

Figure 4.2 shows the main stereotypes introduced by MARTE Time subprofile.

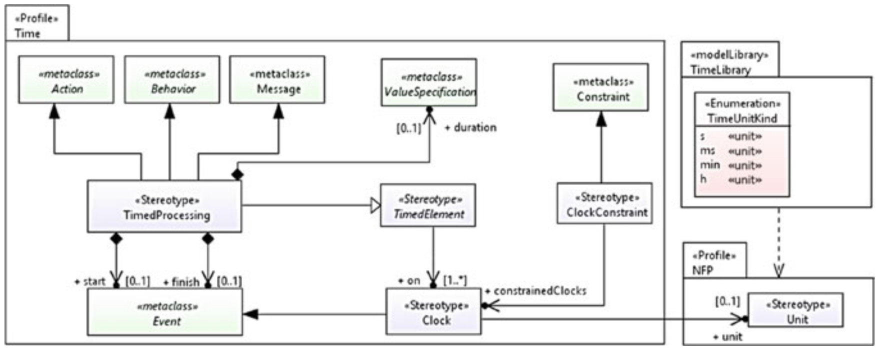

Fig. 4.2 Excerpt of MARTE time subprofile

Stereotype Clock is one important stereotype that extends UML metaclass Event. A Clock carries specific information such as its actual unit, and values of quantitative (resolution, offset…) or qualitative (time standard) properties, if relevant.

TimedElement is another stereotype introduced in MARTE. A timed element is an abstract stereotype that associates at least one clock with a modeling element. TimedProcessing is a specialization of TimedElement, which extends the UML metaclasses Action, Behavior and Message. It defines a start and a finish event for a given action/behavior/message. These events (which are usually clocks) specify when the action starts or when it finishes. TimedProcessing also specifies the duration of an action. Duration is measured on a given logical or physical clock. In a MARTE model of a system, stereotype TimedElement or one of its specializations is applied to model elements which have an influence on the specification of the temporal behavior of this system. The expected behavior of such TimedElements is controlled by a set of ClockConstraints. Those constraints specify dependencies between the various occurrences of events. Dedicated languages, like the Clock Constraint Specification Language (CCSL [25]) can be used to specify those constraints formally.

The MARTE Time subprofile also provides a model library named TimeLibrary. This model library defines the enumeration TimeUnitKind which is the standard type of time units for chronometric clocks. This enumeration contains units like s (second), its submultiples, and other related units (e.g., minute, hour). The library also predefines a clock called IdealClock, which is a dense chronometric clock with the second as time unit. This clock is assumed to be an ideal clock, perfectly reflecting the evolutions of physical time. It should be imported in user's models with references to physical time concepts (e.g., frequency, physical duration).

4.2.5 Allocation

Since embedded systems are platform-aware, one needs a way to map the elements of the application onto the execution platform. This aspect is specifically addressed by the allocation subprofile of MARTE, which is further described in this subsection.

The wording *Allocation* has been retained to distinguish this notion from UML Deployment diagrams. Deployments are reserved to deploy artifacts (e.g., source code, documents, executable, database table) onto deployment targets (e.g., processor, server, database system). The MARTE allocation is much more general than that. For instance, it is meant to represent the allocation of a program onto a system thread, or of a process onto a processor core. More generally, it is used to represent the association of an element (action, message, algorithm) that consumes a resource onto the consumed resource (processing unit, communication media, memory). Wordings 'mapping' or 'map' have also been discarded since they often refer to a function and then map one input from a domain to one single output in the co-domain. The allocation process, however, is an n-to-m association, take for instance a bunch of tasks that need to be scheduled on several cores.

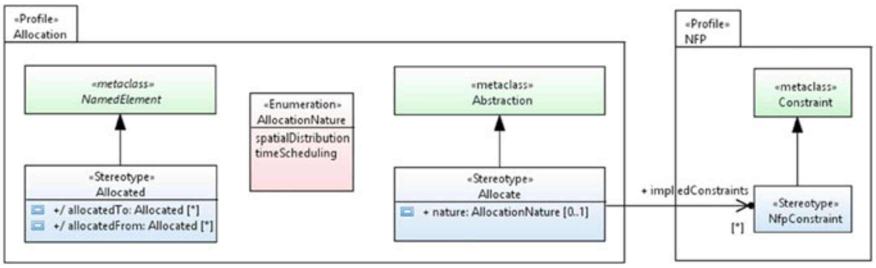

Fig. 4.3 Exerpt of MARTE allocation subprofile

Note that the wording *execution platform* has been preferred to 'architecture' or 'hardware'. Indeed, architecture is a way to describe the structure of a system, while an execution platform contains both structural and behavioral parts. On the other hand, the execution platform is not necessarily a piece of hardware. It can be a piece of software, a virtual machine, a middleware, an operating system or a mixed platform that combines software and hardware intellectual properties (IPs).

Finally, it is also important to note that this notion of allocation is common between MARTE and SysML in a bid to ease the combination of the two profiles. In particular, for CPS both profiles can and shall be used jointly [34].

Figure 4.3 shows the two main stereotypes of the subprofile, Allocate and Allocated. Allocate represents the allocation itself, while Allocated may be used on both sides to mark either the element that is allocated or the resource onto which an element is allocated. The property nature is meant to distinguish two kinds of possible allocations: *spatial* and *temporal*. Typically, when messages are allocated onto a buffer or a memory, this is a spatial allocation. Indeed, the message will consume/use some cells of the memory. However, when two tasks are allocated onto a processing unit, this is a temporal allocation (scheduling); It means those two tasks must be scheduled to avoid resource conflicts. When a program is allocated onto a processor, this can be considered both as spatial and temporal allocations; Spatial because the program consumes disk and memory resources; Temporal because while this program executes, another one cannot execute simultaneously. The allocation usually implies constraints that describe precisely the impact (or cost) of the allocation on the non-functional properties. This is why there is an association to a specific MARTE stereotype called NfpConstraint, i.e., to capture the constraints implied by the allocation in terms of memory consumption, power consumption, or execution time, for instance.

4.2.6 Design and Analysis in MARTE

4.2.6.1 The Design Part

Has four chapters: High Level Application Modeling (HLAM), Generic Component Modeling (GCM), Software Resource Modeling (SRM), and Hardware Resource Modeling (HRM).

The first chapter (HLAM) describes real-time units and active objects. Active objects depart from passive ones by their ability to send spontaneous messages or signals, and react to event occurrences. Normal objects, the passive ones, can only answer to the messages they receive or react on event occurrences. HLAM also introduces the notions of real-time feature and specification. A real-time feature («RtFeature») can be anything on which one wishes to pose some kinds of real-time constraints. The constraints themselves are applied via a real-time specification («RtSpecification»). We give some examples of such usage in Sect. 4.3.

While clocks are precisely sequences of time points where events repetitively occur, a real-time feature makes no assumption on the specific constraint to be applied, i.e., it can refer to time point or time intervals or something much more complex. However, as we show later, some phenomena can be described either as clocks and clock constraints or with real-time features and real-time specifications. We consider that the specification is a compact way to assign several kinds of properties to a generic feature while the semantics of such properties can be described at a finer grain using clocks and clock constraints.

The three other chapters provide a support to describe resources used and in particular execution platforms on which applications may run. A generic description of resources is provided, including stereotypes to describe communication media, storage and computing resources. Then this generic model is refined to describe software and hardware resources along with their non-functional properties.

4.2.6.2 The Analysis Part

Also has a chapter that defines generic elements to perform model-driven analysis on real-time and embedded systems.

This generic chapter is further specialized to address schedulability analysis and performance analysis.

The chapter on schedulability analysis is not specific to a given technique and addresses various formalisms like the classic and generalized Rate Monotonic Analysis (RMA), holistic techniques, or extended timed automata. This chapter provides all the keywords usually required for such analyzes.

Finally, the chapter on performance analysis, even if somewhat independent of a specific analysis technique, emphasizes concepts supported by the queuing theory.

The single-source modeling methodology introduced in Sect. 4.3 gives a way to combine all these mechanisms. However, there is no space here to describe them in length and we rather give methodological information in the following section.

4.3 MARTE for CPS

4.3.1 Case Study: Quadcopter

We consider the example of a quadcopter as an example of cyber-physical system. An efficient design of the flight control part requires coupling to the analysis a model of the dynamics of the quadcopter and of its environment (i.e., physical laws, data received by sensors).

Figure 4.5 gives a UML model of a quadcopter and combines several models to capture different views. The structural view (at the bottom right) is a UML composite structure model. We use MARTE allocation with its two acceptions. One structural allocation to allocate the PIDController on the Zinq board. It is identified as a spatial distribution. We also allocate the QuadRotorController to the PIDController as a time scheduling allocation. Indeed, all the actions have to be scheduled on the controller.

Looking at behavioral views, we use a UML activity to model the controller and its five main concurrent actions. All these actions have to compete for the available resources. «TimedProcessing» refines the description to assign a *clock* to the start instant of three actions. Two of them become synchronized, both start (synchronously) on clock $c2$. The last one starts with clock $c100$ and we use a clock constraint to denote the relative speed. Indeed $c100$ is fifty times slower than $c2$. Later (see Sect. 4.3.2), we can refine such a specification to assign a precise physical periodic behavior to each action.

Another behavioral view describes the operating modes, three while being in-flight. CoreElements stereotypes of MARTE serve to identify the modes. Switching from OnGround to InFlight is done according to events on and off. Both timeEvent become $clock$ and a clock constraint explains their relative behavior.

Each of the three inflight-mode behaves differently. Here we describe one of the modes using SysML parametrics, which are acausal models introduced by SysML. The word acausal is used here in opposition to causal, i.e., there is no causal relationship among the variables of a formula, no assumption on which ones are inputs and which ones are outputs.

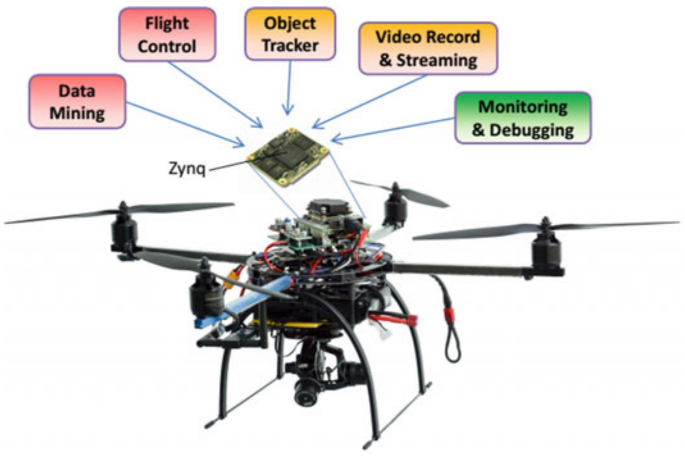

Fig. 4.4 Quadcopter system

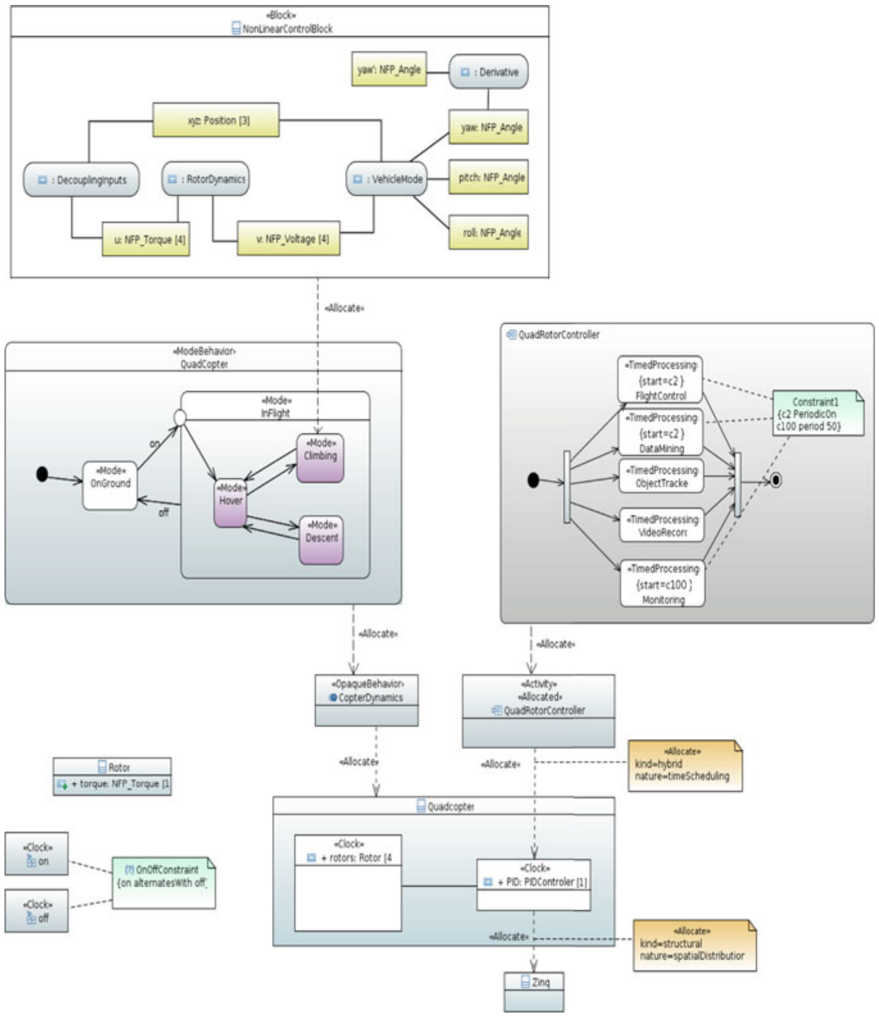
«Block» NonLinearControlBlock
yaw': NFP_Angle
: Derivative
xyz: Position [3]
yaw: NFP_Angle
pitch: NFP_Angle
roll: NFP_Angle
: DecouplingInputs
: RotorDynamics
: VehicleMode
u: NFP_Torque [4]
v: NFP_Voltage [4]
«Allocate»
«ModeBehavior» QuadCopter
«Mode» InFlight
«Mode» OnGround
«Mode» Climbing
«Mode» Hover
«Mode» Descent
on
off
QuadRotorController
«TimedProcessing» {start=c2} FlightControl
«TimedProcessing» {start=c2} DataMining
«TimedProcessing» ObjectTracke
«TimedProcessing» VideoRecorc
«TimedProcessing» {start=c100} Monitoring
Constraint1 {c2 PeriodicOn c100 period 50}
«OpaqueBehavior» CopterDynamics
«Activity» «Allocated» QuadRotorController
«Allocate» kind=hybrid nature=timeScheduling
Rotor
+ torque: NFP_Torque [1
Quadcopter
«Clock» + rotors: Rotor [4
«Clock» + PID: PIDControler [1]
«Clock» on
«Clock» off
OnOffConstraint {on alternatesWith off}
«Allocate» kind=structural nature=spatialDistribution
Zinq

Fig. 4.5 Quadcopter system: UML model

The rectangles denote values or properties connected the pins, some of them use the NFP types defined earlier (like NFP_Torque). The rounded rectangles denote constraint blocks and define a non-linear equations among the values. This model clearly defines a closed loop to control the position (x, y, z) of the quadcopter. The content of the constraint blocks is typically defined as an equation that can feed a simulation in tools like Simulink.

4.3.2 *Proposed Extensions: Mixed-Criticality, Design-Space Exploration*

The growing capabilities and cost effective solutions provided by embedded platforms has increased their application domains. This can be illustrated through the Quadcopter system sketched in Fig. 4.4.

Moreover, the quadcopter system is also an example of Mixed-Criticality system (MCS) [7, 24]. The application software of this system has parts (data mining, flight control) which concern safety, while other parts concern the mission (e.g., object tracking, stream server) or even less important tasks (e.g., data logging). It shall be possible to model and analyze with sufficient accuracy the behavior and performance of these applications on top of a multi-core, costly effective platform. The capture of a sufficiently detailed platform model exposing shared resources, attributes and details concerning performances, and labeling the criticality of the different parts of the systems is required. Design space exploration (DSE) is a crucial activity in Electronic System Level (ESL) design [3]. It consists in the evaluation of alternative solutions to find a tradeoff between the performances and costs. It requires specific modeling capabilities to capture the alternative solutions and their respective cost, i.e., the potential available *design space*.

MARTE offers several mechanisms to cover most of these needs, as shown in the single-source modeling methodology proposed in [16, 38], and sketched in Fig. 4.6.

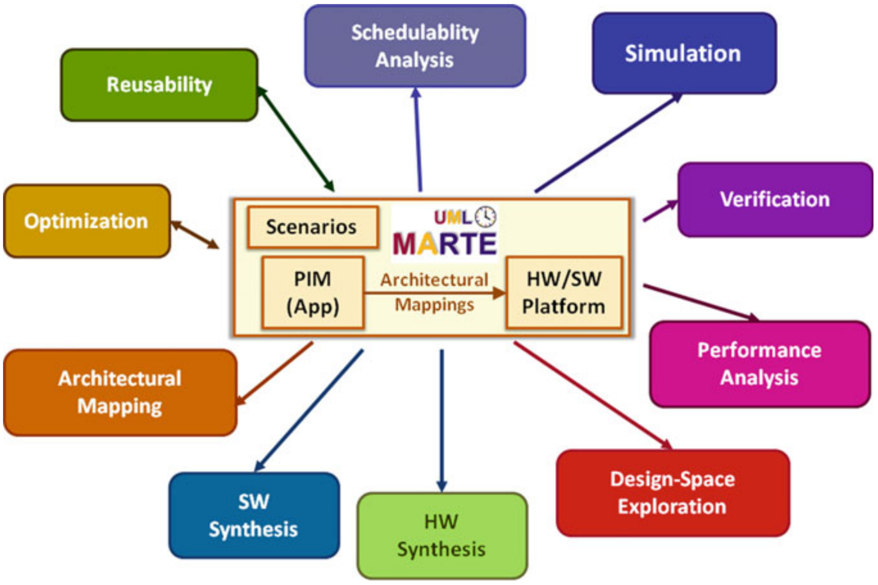

Fig. 4.6 Single-source analysis and design methodology of the University of Cantabria

The main idea is to rely on a common, unique system model, essentially UML and MARTE. Around this model, all the tool infrastructure enables a number of system-level design activities, which include verification, schedulability analysis, simulation, performance analysis, and software synthesis. In [16], the advantages and other features of the single-source methodology are discussed in detail. In the framework of the CONTREX project [29], the single-source design framework is implemented (see details in Sect. 4.4.2). The single-source modeling methodology enables to build up a component-based application model that can be mapped onto a platform. The platform model describes in detail the available HW and SW resources. It includes efficient multi-core platforms with shared resources, a main aspect to deal with in MCS design. The modeling methodology also states how to model a design space by relying on MARTE and on the Value Specification Language (VSL).

The single-source modeling methodology provides fixes to patch the minor deficiencies of the current MARTE specification [32]. They have to do with modeling of MCS and for DSE, which is understandable, as the standard was not initially designed for such purposes.

4.3.2.1 Extensions for Design Space Exploration

A basic modeling construct for DSE is the DSE parameter. A DSE parameter states that the value of an attribute associated with a modeling element can adopt one among a set of values during the DSE activity. Along the DSE, several solutions are assessed. A solution is defined by the set of values assigned to each of the DSE parameters. Figure 4.7 illustrates the modeling construct employed for describing a DSE parameter in the single-source methodology. A MARTE NFP constraint plus the *ExpressionContext* stereotype is linked to the model element (a HW resource component in Fig. 4.7) containing the attribute. The expression is captured in MARTE VSL with the following syntax:

$DSEParameterName = DSERangeSpecification

The *"$"* symbol prefixes a VSL variable, and thus the DSE parameter name. The *DSERangeSpecification* expresses the range of the DSE parameter, that is, all the values that the *DSEParameterName* variable can have during the exploration. The DSE parameter range can be annotated either as a collection or as an interval. Collections are captured with the syntax *DSERangeSpecification = (v1, v2, v3, unit)*, where *"v1, v2, v3"* are the numerical values of the parameter and *unit* expresses the physical unit associated with the values. MARTE provides a rich set of unit kinds (see Sect. 4.2.3), to support the different extra-functional properties characterising systems components, e.g. frequencies, bandwidth, data size. Intervals follow the syntax *"DSERangeSpecification = ([$v_{min}..v_{max}$], unit)"*. For a complete determination of the exploration range, this style obliges to assume an implicit step. For an explicit and complete determination of the DSE range, the support of the style *"DSERangeSpecification = ([$v_{min}..v_{max}$, step], unit)"* is proposed, which means a minor extension of VSL. The definition of non-linear ranges is possible. For instance, *step* can take

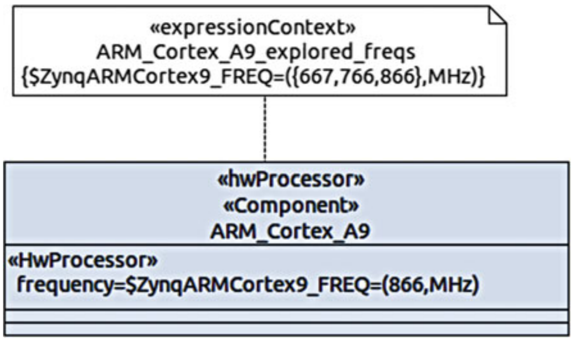
«expressionContext»
ARM_Cortex_A9_explored_freqs
{$ZynqARMCortex9_FREQ=({667,766,866},MHz)}
«hwProcessor»
«Component»
ARM_Cortex_A9
«HwProcessor»
frequency=$ZynqARMCortex9_FREQ=(866,MHz)

Fig. 4.7 DSE parameter associated with the frequency of a HW processor component

the value *exp2*, which enables the definition of a geometrical progression, i.e., the second value is “$v_{min}x2$”, and so on.

Figure 4.7 illustrates the specification of a design space on the frequencies of the ARM processing cores of the quadcopter HW platform. This way, the exploration of the impact on performance depending on the selection of a Z-7020 device (which works at 667 MHz), a Z-7015 device (at 766 MHz) or a Z-7020 device (at 866 MHz) can be explored.

In Fig. 4.7, the DSE has been associated with the processor component declaration (in the HW platform resources view). Therefore, once the DSE parameter value is fixed, it is fixed for all its instances. The single-source methodology also allows the association of the DSE parameter with the instance properties. Moreover, the methodology enables the definition of DSE parameters at the application level, and the specification of parameterizable mappings. Further details can be found in [16, 38]. In any cases, as shown, MARTE (and specifically VSL) is exploited for most of the required constructs for DSE, while a minor extension is required (for explicitly stating the steps of an interval-based specification of a DSE parameter).

4.3.2.2 Extensions for Mixed-Criticality

The single-source modeling methodology [16, 38] was specifically extended in CONTREX [29] for supporting the MCS modeling and design. This requires a minor extension of MARTE, sketched in Fig. 4.8, and proposed to the OMG. The first extension adds a criticality attribute to a NFP constraint (Fig. 4.8, left hand side). The criticality attribute is an integer to denote an abstract criticality level. The NFP constraint can be associated with different types of modeling elements, e.g. UML components and UML constraints. The second extension (Fig. 4.8, right hand side) consists in adding an attribute criticality to the $NFP\_CommonType$ so that it can be used in VSL values.

These two small extensions impacts lots of modeling constructs since constraints and values are used in many different contexts. More precisely, the first extension adds a criticality level, through UML constraints, to application and platform components,

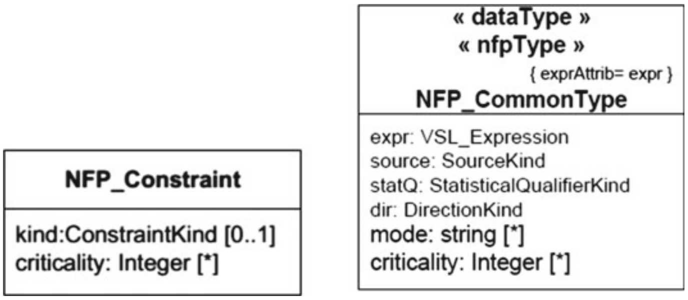

Fig. 4.8 Extensions of MARTE for mixed-criticality

as well as performance requirements. The second extension adds a criticality level to value annotations, e.g. a WCET, and also to performance constraints captured. These modeling constructs provide solutions to cover different mixed-criticality modeling scenarios which have been identified and described in [16, 38].

Figure 4.9 shows an illustrative and novel MCS modeling scenario, where it is possible to add criticality levels along with the performance requirements of the system. Specifically, in Fig. 4.9, a criticality level 3 is assigned to a power performance requirement for the quadcopter system, i.e., the first kind of extension proposed.

Figure 4.10 completes the association of criticalities to other time-related performance requirements imposed to the quadcopter system. Specifically, the quadcopter model captures a number of deadline requirements on components of the quadcopter Platform-Independent Model (PIM). Those deadlines are captured as VSL value annotation through the attribute *relDl* of stereotype *RtSpecification*. The annotation of these performance requirements relies on our second proposed extension.

As mentioned, the single-source methodology covers other scenarios, e.g. modeling for mixed-criticality aware schedulability analysis, and for validation of architectural mapping according the criticality associated with application components and to the platform resources the former are mapped to (this is illustrated in Sect. 4.4.2).

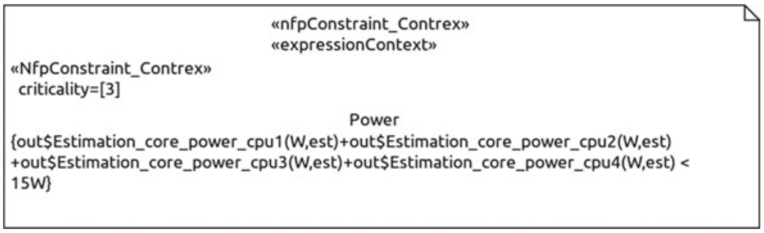
«nfpConstraint_Contrex»
«expressionContext»
«NfpConstraint_Contrex»
criticality=[3]
Power
{out$Estimation_core_power_cpu1(W,est)+out$Estimation_core_power_cpu2(W,est)
+out$Estimation_core_power_cpu3(W,est)+out$Estimation_core_power_cpu4(W,est) <
15W}

Fig. 4.9 Criticality level assigned to a power-related requirement

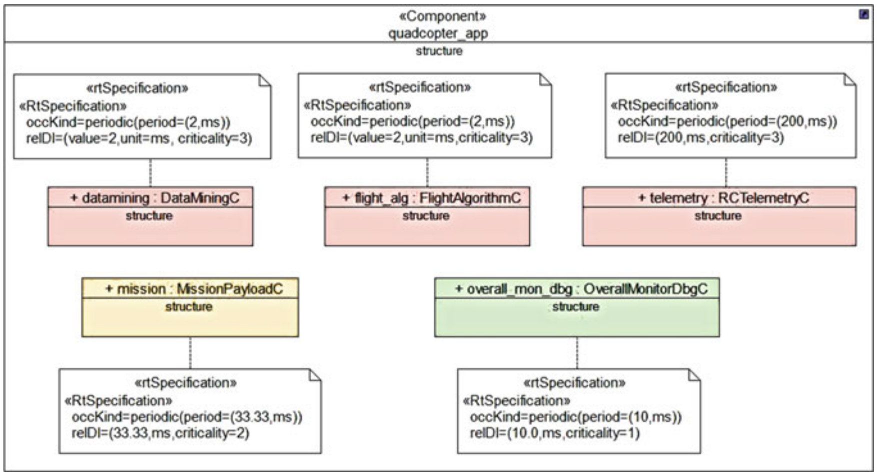
«Component»
quadcopter_app
structure
«rtSpecification»
«RtSpecification»
occKind=periodic(period=(2,ms))
relDl=(value=2,unit=ms, criticality=3)
+ datamining : DataMiningC
structure
«rtSpecification»
«RtSpecification»
occKind=periodic(period=(2,ms))
relDl=(value=2,unit=ms,criticality=3)
+ flight_alg : FlightAlgorithmC
structure
«rtSpecification»
«RtSpecification»
occKind=periodic(period=(200,ms))
relDl=(200,ms,criticality=3)
+ telemetry : RCTelemetryC
structure
+ mission : MissionPayloadC
structure
«rtSpecification»
«RtSpecification»
occKind=periodic(period=(33.33,ms))
relDl=(33.33,ms,criticality=2)
+ overall_mon_dbg : OverallMonitorDbgC
structure
«rtSpecification»
«RtSpecification»
occKind=periodic(period=(10,ms))
relDl=(10.0,ms,criticality=1)

Fig. 4.10 Criticality associated with deadline requirements in the PIM

4.4 Tooling

4.4.1 State-of-the-Art

All the commercial UML editors support MARTE to diverse degrees. The simplest support consists in defining all the stereotypes. Others go further by usually allowing transformations of subsets into ad-hoc external tools. For instance, when using the *analysis* part of MARTE, one can performance schedulability analysis by transformations to RapidRMA or Cheddar (see http://www.omgmarte.org). Also another transformation from another subset of MARTE allows for performance evaluation [11].

Another possible set of analyses is available through the transformation [12] of a MARTE specification into AADL, which is a standard from the Society of Automotive Engineers and that gains interests for the design of safety-critical systems. There again, a specific subset of MARTE is retained.

TimeSquare [10] proposes a different family of tools. One side, when using clocks and clock constraints, TimeSquare attempts to build one behavior consistent with all the constraints. If this is possible, then it proposes one simulation model and animate the UML model. It can also use the constraints to detect possible inconsistencies and potential deadlocks [27].

Another approach [5] is to use a model-based approach to verify a given implementation through the analysis of execution traces.

A more detailed survey [6] gives an overview of other approaches based on UML, including MARTE. However, most solutions use ad-hoc transformations from a relatively small subset of UML and MARTE to ad-hoc external tools. In the following subsection, we promote integrated approaches with a single-source design for MARTE.

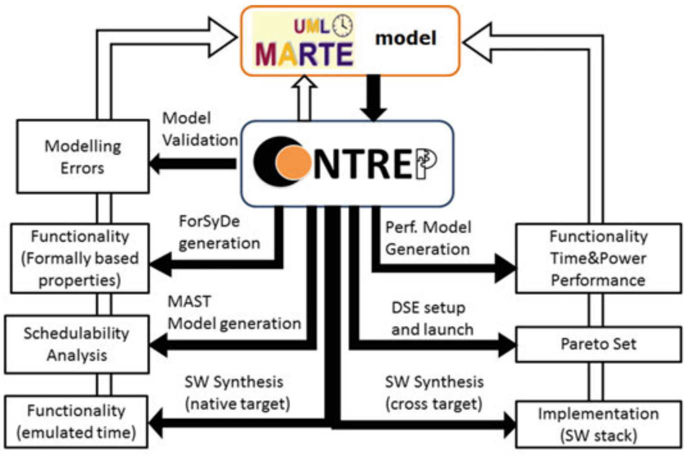
UML MARTE model
CONTREP
Model Validation
Modelling Errors
ForSyDe generation
Functionality (Formally based properties)
MAST Model generation
Schedulability Analysis
SW Synthesis (native target)
Functionality (emulated time)
Perf. Model Generation
Functionality Time&Power Performance
DSE setup and launch
Pareto Set
SW Synthesis (cross target)
Implementation (SW stack)

Fig. 4.11 The CONTREX Eclipse-Plugin provides a front-end and integrates tool infrastructure for single-source design from UML/MARTE

It is an attempt to bring consistencies among different views and integrate several tools within the same framework.

4.4.2 Single-Source Design from MARTE

The generic single-source design approach shown in Fig. 4.6 has been implemented by the University of Cantabria into the framework sketched in Fig. 4.11. In this framework, the CONTREP Eclipse Plugin (CONTREP) enables an unified front-end where the single-source model is developed, and where system-level design activities (boxes in Fig. 4.11) are applied from.

The architecture of the single design framework, revealing how and which profiles, libraries and tools are stacked is sketched in Fig. 4.12. Following, the tools integrated in the implemented framework and their role is explained under the perspective of the design tasks they enable. *Eclipse* MDT is at the base of the unifying front-end.

For the *modelling task*, *Papyrus* and the *MARTE profile* are at the base of the UML/MARTE modelling. CONTREP complements Eclipse/Papyrus with a profile with the MARTE extensions introduced in Sect. 4.3.2. CONTREP also provides a tool and a related configuration tab for the static validation of the model.

For enabling the different system-level design activities from the UML/MARTE model, CONTREP provides code generators, configuration and launching facilities. This performs an effective integration of state-of-the art and novel system-level design tools.

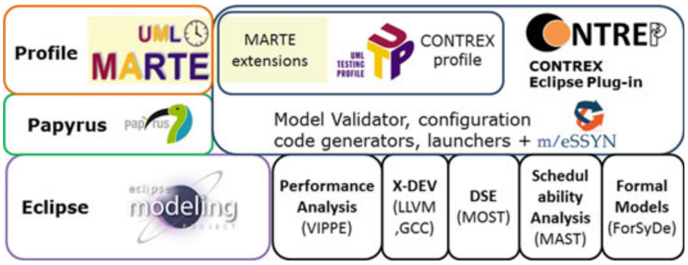

Fig. 4.12 Architecture of the single-source design environment

For *formally based functional validation*, CONTREP enables the generation of a ForSyDe model-SystemC model. By relying on ForSyDe-SystemC library [19, 28], CONTREP enables the generation of an executable model abiding the rules of the Synchronous Dataflow (SDF) Model-of-Computation (MoC) [20]. This ensures functional determinism by construction and makes analisable if the application is free from deadlock. CONTREP enables *schedulability analysis* as it integrates the *Marte2mast* code generator [14], able to produce a model that can be processed by the *MAST* [15] schedulabiltiy analysis tool.

CONTREP enables *software synthesis* for heterogeneous multi-core targets. For a given specific solution (once a platform and the architectural mapping is fixed), the platform dependent code is automatically generated, and the target binaries generated. In CONTREP, the code generators developed for eSSYN [37] have been integrated, extended and made available through its user menu.

The single-source design framework also enables the *automatic generation and the execution of an emulation model*. An emulation model enables the functional validation of the program, and the emulation of the timing specified on the application on top of the host machine. For instance, if the application states that a tasks will execute periodically, with period 1s, the produced emulated model will rely on the host RTOS services for emulating that timing behaviour. However, this type of model does not mimic the timing due to the mapping of the application to a given platform. In fact, the production of the emulation model considers the PIM, but neither the platform model nor the mapping information. The emulation model integrates platform independent functionality (referred by the PIM components of the model), and platform dependent functionality. The latter is generated through a specific code generator, the *m/eSSYN code generator*. This generator is invoked from CONTREP and produces all the code which implements the communication, concurrency and time services required by the semantics of the PIM components. In this sense, this platform dependent code *wraps* the platform independent code. Because of that, this code is also called *wrappers* code, and the code generator, *wrappers generator*.

CONTREP automates the generation of a fast executable performance model. The generation of the performance model relies on the VIPPE tool [39], which implements advanced techniques for fast simulation and performance assessment. The generation also relies on m/eSSYN, an extension of the eSSYN code generators, to support also a configurable simulation target. The m/eSSYN code generator allows to integrate DSE variables in the automatically produced code. This way, the performance model enables fast exploration (without trigerring new SW synthesis) of several design solutions considering also the variation of some aspect of the target dependent code, i.e. a task period. The final outcome is a more holistic exploration, in the sense that DSE parameters can be associated to platform attributes and also to the application. Simulation is convenient for getting accuracy and considering the dynamism of the application and of the input stimuli of the system environment, that is for an scenario-aware assessment of the system. VIPPE provides a rich set of metrics related to the time performance of the system. Thus the user is not only able to explore if the time requirements are fulfilled, but also facilitates the assessement of the main causes of eventual violations of those time requirements. Moreover, VIPPE provides energy and power consumption metrics. VIPPE relies on host-compiled simulation, and it is capable to parallelize the simulation to exploit multi-core host platforms. This is a main reason foor the suitability of VIPPE to be very suitable for design space exploration. Another one is that VIPPE supports the Multicube XML interface [35]. VIPPE is also capable to export power traces readable by tools able to perform dynamic temperature analysis, like *ThermalProfiler* [17].

CONTREP automates also the validation of a performance solution by automatically comparing the performance of a simulated solution versus the performance requirements captured in the UML/MARTE model. When the exploration is done in an interactive way, the comparison relies on two XML files. CONTREP code generators translate the performance requirements captured in UML/MARTE to an intermediate XML file. The run of the (VIPPE) executable performance model for a given combination of DSE parameter values (configuration or solution) produces an XML file reporting the performance metrics required to evaluate the performance requirements.

CONTREP also automates the generation of a complete simulation-based DSE infrastructure. As well as the performance model, files describing information like the design space, the performance constraints, the cost functions, and the exploration strategy, a basic input for the exploration tool coupled to the simulatable performance model is generated. The framework is flexible and allows the user to select the specific exploration tool to employ in the DSE (i.e. *MOST* [8] or *Multicube explorer*[40] so far), among the different exploration strategies available for the selected exploration tool, and among different report options, e.g., if intermediate results are preserved, of whether an HTML report is produced. Moreover, CONTREP allows to control the generation of a cost function which takes into account the criticality levels associated to the performance requirements. Thus the framework allows to give preference to solutions with larger safety margins in the more critical performance requirements, provided a minimum or equivalent performance on other metrics.

Both, for SW synthesis and for the production of an accurate VIPPE performance model, set ups of the cross-compiler tool-chains (X-DEV in Fig. 4.12) supporting the targeted platforms are required.

The final result is an joint user front-end to give access to triggering the different design activities from the same menu, shown in the capture of Fig. 4.13.

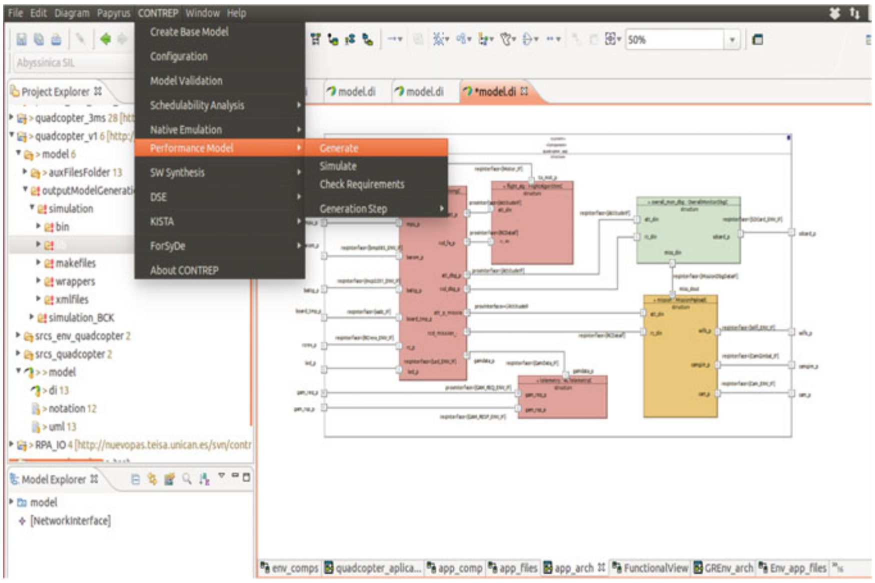

Fig. 4.13 Menu enabled by the CONTREX Eclipse Plug-in

4.5 Forecast About the Role MARTE May Have in Designing CPS

4.5.1 *The End of Moore's Law*

In its latest edition of 2015, the International Technology Roadmap for Semiconductors (ITRS) roadmap provided by the semiconductor companies changed its format from previous editions [18]. On the one hand, it confirms the end of Moore's Law as it has been performing until the first decade of the century. In fact, for several years now, the scaling has not been geometric as in the past. Although manufacturers still associate technology nodes with increasingly smaller dimensions, the size reduction is no longer the main cause of technological improvement. This improvement depends mostly on new structures (FinFET and FD-SoI transistors), new materials, etc. The roadmap proposes to achieve 10 nm technology in 2021. However, stagnation is observed from that date on. Only through technological improvements in processes with the same scale factor i.e. devices with the same size, will it be possible to manufacture integrated circuits with better performance. This is what is called equivalent scaling. A clear finale. An electronic technology that reaches maturity and is not going to be able to offer the exponential growth maintained to date. The end of Moore's Law will cause a radical change in the technological evolution as we have been enjoying it so far. On the one hand, the amortization of the billion investments to be made in each new line of semiconductor manufacturing can be spread on longer periods of time so that manufacturing costs can be significantly reduced. Today, advanced technologies are available only for those designs with high enough volumes of production. The aim is to ensure that the investment pays off in the short period of time in which that technology node is active before being replaced by more advanced technologies in new products. If this is no longer the case, semiconductor technology could become much more accessible to companies with smaller volumes. A similar evolution would follow design tools and libraries. On the other hand, for the designer, the pressure of time-to-market' is reduced, which will reduce design costs. Thus, it is quite possible that in the near future, the design of integrated circuits is accessible to any company in which the value added by silicon compensates the higher non-recurrent engineering cots. Between a product and its successor there will be only incremental improvements; no substantial improvements based on a significantly higher computing power, as it happens now. Innovation would come from the new applications and services based on new electronic products making use of application-specific integrated circuits fabricated with stable technologies. The price/performance evolution of electronic products will follow a similar trend to many other products. No longer fueled by the exponential improvement predicted by Moores Law. Some large companies are already preparing for this scenario. In some cases, this is an even desirable scenario as it could free up resources to invest in other technologies to offer new services to the marketplace.

4.5.2 The Rise of Connected Ubiquitous Smart Objects

Nowadays, we have just started to realize the enormous potential of an interconnected world of billions of smart devices providing new services to people. The end of Moore's Law commented above, might facilitate the proliferation of new electronic systems supporting these new services. Currently, most systems involve just a small number of computing resources, such as a data-center processing the voice from a smartphone and providing a voice-to-text service, or the distance sensors in a car connected to an Electronic Control Unit providing an automatic parking service to the driver. In these examples, specifying the complete service, deciding which functionality to execute in each node, and programming the corresponding application, although reasonably complex, are affordable tasks. However, services in a fully interconnected world will be composed of many SW components deployed on multiple devices of many kinds, from small sensing motes, embedded systems and smartphones to data-centers, and even, High-Performance Computing (HPC) facilities. All of them may be implemented with integrated systems containing heterogeneous processing elements (i.e. CPUs of different kinds, GPUs, DSPs and HW co-processors). And in all cases, the systems will need to satisfy functional and extra-functional critical constraints, including safety, security, power efficiency, performance, size, and cost. The global characteristics of the system as a whole will depend on the characteristics of their independent components, but also on the interactions with the physical environment and among them through the different communication networks. Therefore, the main innovation in the time to come shall be to jump from the design of cyber-physical systems (CPS) to cyber-physical systems of systems (CPSoS). These complex, heterogeneous, distributed systems require an interdisciplinary approach where the knowledge about the physical side of the systems is essential to arrive at solutions that are taken up in the real world. To integrate these diverse research and development communities is the most crucial aspect for a successful future development of CPSoS [9]. Current domain-specific methods are becoming obsolete; hence new predictive, engineering and programming methods and tools are required ensuring the satisfaction of the functional and extra-functional constraints imposed to the system while taking into account its interaction with the physical world. An additional important aspect to consider is the interaction between the system and the humans both as users or involved directly or indirectly in its operation (humans in the loop). The main reaction to the continuous evolution of computing platforms has been to decouple the application SW from the underlying HW. To achieve this goal many abstraction layers of middleware, communication protocols, operating systems, hypervisors and HW abstraction layers are being used. This approach is powerful enough for general purpose systems, for which extra-functional constraints such as execution times, energy efficiency, dependability, etc. are not strict. But the technological evolution towards CPSoS based on heterogeneous devices composed of CPUs of different kind, GPUs, DSPs, HW co-processors etc., added to the need to satisfy stricter non-functional properties makes this goal unrealizable. Although the software development on each of these platforms shares many commonalities, the current situation is that specific programming approaches, using different languages and tools are applied in each case. As a consequence, SW development for a supercomputer, for instance, becomes completely different to a smartphone or an embedded system. This was not a major problem when the supercomputer and the embedded system were functionally decoupled. Nevertheless, this is no longer valid in the fully connected world commented above. An initial functional partition followed by separate programming approaches leads to inefficient distribution of loads and communication traffic causing performance, energy, dependability, data movement, and cost overheads. Model-Driven Engineering (MDE) is a mature system engineering approach which is being successfully applied in many different domains. UML has proven its support to MDE in all these domains based on domain-specific profiles. Nevertheless, none of the currently available domain-specific approaches is applicable to develop SW for CPSoS or, the other way around, their applicability to different domains would require interoperability with other different languages, design methodologies and tools. Another important aspect is that in each domain the key concerns are also different. This is no longer valid. When dealing with a cross-domain application involving interacting SW components to be deployed and executed in different execution platforms, the functional and extra-functional properties (i.e. power consumption, real-time properties, performance, security, dependability, data movement) in one of the components may be affected by the rest of components. As a consequence, there is a need for a holistic modeling framework, across SW and HW layers, applications and domains. This modeling framework should be able to capture the complete high-abstraction model, integrating projects with different constraints (i.e. commercial or critical SW) and domains (i.e. from High Performance Computing, to embedded SW). The framework should integrate in a seamless and almost transparent way any abstraction layer (i.e. middleware, communication protocols, operating systems, hypervisors) required in a particular domain, on a particular HW platform. Currently fragmented but mature MDE techniques should be extended to the emergent, distributed, heterogeneous, CPSoS. Beyond the MDE underlying system engineering technology, the solution should allow an average skilled software developer to build an application for the advanced systems discussed previously. The goal is to avoid the system engineer a detailed knowledge of the characteristics of the underlying HW/SW platforms, letting him/her to focus on the Platform-Independent functionality of the complete system he/she wants to implement. The design framework should provide him/her with an accurate knowledge of the implications that the final implementation of the functionality on the concrete (distributed) platforms under a specific functional mapping will have in terms of extra-functional constraints. Beyond performance; energy consumption, safety, data traffic, security, adaptability, scalability, complexity management and cost-effectiveness have to be taken into account. This information about the complete system characteristics can be used in efficient optimization and design-space exploration for the complete system. As commented above, an essential aspect to be taken into account is the interaction of the system with humans all along the life-cycle, since the specification and design of the system until its deployment, field and obsolescence. UML has the potential to be the central modeling language in this new context. To achieve this goal, a consensus on a profile, powerful enough to capture all the relevant concepts required in CPSoS engineering while, at the same time, simple enough to find wide acceptance by the design community, is required. MARTE is a good starting point for two main reasons. Firstly, it captures most of the concepts required in system engineering on heterogeneous platforms under strict design constraints. Secondly, there is clear convergence among computing platforms and today, it is possible to find the same computing resources (i.e. CPUs, GPUs, and application-specific HW) in platforms apparently as different as an embedded system, a smartphone and a supercomputer. As the technology evolution will be incremental and no longer, exponential, this profile will be stable in time with only incremental improvements. If this is achieved, UML could become the system design language for the electronic century.

4.6 Conclusion

MARTE has the capability to describe several parts necessary to model CPS. Since MARTE is just a language, it does not come with a recommended or unique methodology. Here, we draw a map of what can be done with MARTE and what remains to be done. We advocate for a single-source modeling methodology where a central model is used to feed several analysis tools. This model relies on UML, MARTE, SysML along with very few extensions that we propose. The results are used to extend and refine the models following an iterative process.

To illustrate the process, we use the example of a quadcopter. We manage to put together several chapters (CoreElements, NFP, Time, Allocation, HLAM, VSL) of MARTE to propose a consistent usage in an unprecedented manner.

The question of whether UML is ill-founded is irrelevant, UML is there, it is widely used and will continue to be used. This is the only modeling language widely accepted by industry and that is expressive enough to cover so many aspects of complex heterogeneous systems. So, the good question, is rather to decide how to use it efficiently to gain an even bigger adoption and sound usage. We hope this chapter contributes to this objective.

Acknowledgements This chapter has been partially funded by the European FP7 611146 (CONTREX) project and by the Spanish TEC 2014-58036-C4-3-R (REBECCA) project. UC thanks the OFFIS team in CONTREX their support, documentation and material on their quadcopter implementation, which includes the quadcopter picture integrated in Fig. 4.4.

References

1. C. André, F. Mallet, R. de Simone, Modeling time(s), in *MODELS'07:10th International Conference on Model Driven Engineering Languages and Systems* Nashville, TN, USA, September 2007, Lecture Notes in Computer Science, vol. 4735 (Springer, ACM-IEEE), pp. 559–573
2. C. André, J. DeAntoni, F. Mallet, R. de Simone, *The Time Model of Logical Clocks Availablein the OMG MARTE Profile*, Chap. 7. (Springer Science+Business Media LLC, 2010), pp. 201–227, http://hal.inria.fr/inria-00495664
3. B. Bailey, G. Martin, A. Piziali, *ESL Design and Verification: A Prescription for Electronic System Level Methodology* (Morgan Kaufmann/Elsevier, San Francisco, 2007)
4. G. Berry, The Informatics of Time and Events. Collège de France, inaugural Lecture, March 2013
5. M. Bourdellès, S. Li, I. Quadri, E. Brosse, A. Sadovykh, E. Gaudin, F. Mallet, A. Goknil, D. George, J. Kreku, *Fostering Analysis from Industrial Embedded Systemis Modeling*, Chap. 11 (IGI-Global, Hershey, 2014), pp. 283–300
6. F. Boutekkouk, M. Benmohammed, S. Bilavarn, M. Auguin, UML2.0 profiles for embedded systems and systems on a chip (SOCS). J. Object Technol. **8**(1), 135–157 (2009), http://dx.doi.org/10.5381/jot.2009.8.1.a1
7. A. Burns, R. Davis, Mixed-criticality systems: a review, Technical report of Computer Science, University of York, 6th edn., August 2015
8. F. Castro, G. Palermo, C. Silvano, V. Zaccaria, MOST: multi-objective system tuner design space exploration for system architects, in *Proceedings of the Designing for Embedded Parallel Computing Platforms: Architectures, Design Tools, and Applications Workshop*, March 2011
9. CPSoS Working Group Members: Cyber-physical systems of systems: Research and innovation priorities book (2016), http://www.cpsos.eu/wp-content/uploads/2015/02/CPSoS-Provisional-Roadmap-Paper-for-public-consultation_web.pdf
10. J. Deantoni, F. Mallet, Timesquare: treat your models with logical time, in *TOOLS (50)*, vol. 7304, ed. by C.A. Furia, S. Nanz, Lecture Notes in Computer Science (Springer 2012), pp. 34–41

11. H. Espinoza, H. Dubois, S. Gérard, J.L.M. Pasaje, D.C. Petriu, C.M. Woodside, Annotating UML models with non-functional properties for quantitative analysis, in *Workshops of MoDELS 2005 Conference*. Lecture Notes in Computer Science, vol. 3844 (Springer, 2005), pp. 79–90, http://dx.doi.org/10.1007/11663430_9
12. M. Faugère, T. Bourbeau, R. de Simone, S. Gérard, MARTE: also an UML profile for modeling AADL applications, in *ICECCS* (2007), pp. 359–364
13. S. Friedenthal, A. Moore, R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language* (MK/OMG, Burlington, 2014)
14. A. Garcia, J. Medina, MARTE2MAST, http://mast.unican.es/umlmast/marte2mast/
15. M. Gonzalez, J.J. Gutierrez, J.C. Palencia, J.M. Drake, Mast: modeling and analysis suite for real time applications, in *13th Euromicro Conference on Real-Time Systems* (2001), pp. 125–134
16. F. Herrera, J. Medina, E. Villar, Modelling hardware/software embedded systems with uml/marte: a single-source design approach, in *Handbook of Hardware/Software Codesign*, Chap. 5, ed. by S. Ha, J. Teich (Springer), pp. 125–159. The address of the publisher, 1 edn. (2 2017), printed version scheduled for Feb. 2017
17. Intel DOCEA: Thermal Profiler website, http://www.doceapower.com/index.php?option=com_content&view=article&id=237&Itemid=145
18. International roadmap for semiconductors. Technical report (2015), http://www.itrs2.net/
19. KTH Royal Institute of Technology: ForSyDe website (2016), https://forsyde.ict.kth.se/trac
20. E.A. Lee, D. Messerschmitt, Synchronous data flow (1987)
21. E.A. Lee, A.L. Sangiovanni-Vincentelli, A framework for comparing models of computation. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **17**(12), 1217–1229 (1998)
22. E.A. Lee, Cyber physical systems: design challenges, in *11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (ISORC 2008) (IEEE Computer Society, May 2008), pp. 363–369, http://dx.doi.org/10.1109/ISORC.2008.25
23. E.A. Lee, S.A. Seshia, Introduction to Embedded Systems - A Cyber-Physical Systems Approach (LeeSeshia.org, 2014), ISBN 978-0-557-70857-4
24. M. Lemke, Mixed criticality systems, report from the workshop on mixed criticality systems, Technical report Information Society and Media Directorate-General, February 2012
25. F. Mallet, C. André, R. de Simone, CCSL: specifying clock constraints with UML/Marte. Innov. Syst. Softw. Eng. **4**(3), 309–314 (2008)
26. F. Mallet, Logical Time @ Work for the Modeling and Analysis of Embedded Systems (LAMBERT Academic Publishing, January 2011), ISBN: 978-3-8433-9388-1
27. F. Mallet, R. de Simone, Correctness issues on MARTE/CCSL constraints. Sci. Comput. Program. (2015). doi:10.1016/j.scico.2015.03.001
28. S.H.A. Niaki, I. Sander, An automated parallel simulation flow for heterogeneous embedded systems, in *Proceedings of the Conference on Design, Automation and Test in Europe* (DATE'2013), EDA Consortium, San Jose, CA, USA (2013), http://dl.acm.org/citation.cfm?id=2485288.2485297, pp. 27–30
29. OFFIS: CONTREX FP7 project website (2015), https://contrex.offis.de/home/
30. OMG: UML Profile for Schedulability, Performance, and Time Specification, v1.1. Object Management Group, January 2005. Accessed 02 Jan 2005
31. OMG: Systems Modeling Language (SysML) Specification, v1.1. Object Management Group, November 2008. Accessed 02 Nov 2008
32. OMG: UML Profile for MARTE, v1.1. Object Management Group, June 2011. Accessed 02 June 2011
33. OMG: UML Superstructure, v2.4.1. Object Management Group, May 2012. Accessed 07 May 2012
34. B. Selic, S. Gerard, *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE* (Elsevier, Amsterdam, 2013)
35. C. Silvano, W. Fornaciari, G. Palermo, V. Zaccaria, F. Castro, M. Martinez, S. Bocchio, R. Zafalon, P. Avasare, G. Vanmeerbeeck et al., Multicube: multi-objective design space exploration of multi-core architectures. in *VLSI 2010 Annual Symposium* (Springer 2011), pp. 47–63

36. B.N, Taylor, A. Thompson, International System of Units, v1.1. National Institute of Standards and Technology, March 2008
37. University of Cantabria. TEISA Department GESE group: essyn website (2016), http://www.eSSYN.com
38. University of Cantabria. TEISA Department GESE group: UC single-source modelling and design website (2016), https://umlmarte.teisa.unican.es
39. University of Cantabria. TEISA Department. GESE group: Vippe website (2016), https://vippe.teisa.unican.es
40. V. Zaccaria, G. Palermo, F. Castro, C. Silvano, G. Mariani, Multicube explorer: an open source framework for design space exploration of chip multi-processors. in *Proceedings of the Interenational Conference on Architecture of Computing Systems (ARCS)*, Febraury 2010, pp. 1–7