Modeling Hardware/Software Embedded Systems with UML/MARTE: A Single-Source Design Approach

5

Fernando Herrera, Julio Medina, and Eugenio Villar

Abstract

Model-based design has shown to be a powerful approach for embedded software systems. The Unified Modeling Language (UML) provides a standard, graphically based formalism for capturing system models. The standard Modeling and Analysis of Real-Time Embedded Systems (MARTE) profile provides syntactical and semantical extensions required for the modeling and HW/SW codesign of real-time and embedded systems. However, the UML/MARTE standard is not sufficient. In addition, a modeling methodology stating how to build a model capable to support the analysis and HW/SW codesign activities of complex embedded systems is required. This chapter presents a UML/MARTE modeling methodology capable to address such analysis and design activities. A distinguishing aspect of the modeling methodology is that it supports a *single-source* design approach.

Acronyms

BCET	Best-Case Execution Time
BSP	Board Support Package
CPS	Cyber-Physical System
DSE	Design Space Exploration
DSL	Domain-Specific Language
EDF	Earliest Deadline First
EML	Execution Modeling Level
ESL	Electronic System Level

F. Herrera • E. Villar
GESE Group, TEISA Department, ETSIIT, Universidad de Cantabria, Santander, Cantabria, Spain
e-mail: fherrera@teisa.unican.es; evillar@teisa.unican.es

J. Medina (✉)
Software Engineering and Real-Time Group, University of Cantabria, Santander, Cantabria, Spain
e-mail: medinajl@unican.es

S. Ha, J. Teich (eds.), *Handbook of Hardware/Software Codesign*,
DOI 10.1007/978-94-017-7267-9_6

FPGA	Field-Programmable Gate Array
GME	Generic Modeling Environment
HLS	High-Level Synthesis
HRM	Hardware Resource Modeling
HSCD	Hardware/Software Codesign
ISA	Instruction-Set Architecture
M2M	Model-to-Model
MARTE	Modeling and Analysis of Real-Time Embedded Systems
MCS	Mixed-Criticality System
MDA	Model-Driven Architecture
MPSoC	Multi-Processor System-on-Chip
NFP	Non-Functional Property
OMG	Object Management Group
OS	Operating System
PIM	Platform Independent Model
PVT	Programmers View Time
RR	Round Robin
RTOS	Real-Time Operating System
SLS	System-Level Synthesis
TLM	Transaction-Level Model
UML	Unified Modeling Language
UTP	Universal Testing Profile
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VSL	Value Specification Language
WCET	Worst-Case Execution Time

Contents

5.1 Introduction

Model-based design is a powerful approach for the design of complex embedded systems [27]. It can be adapted to different design contexts and domains, being compatible with methodologies like *Agile* [2]. The Unified Modeling Language (UML) supports Model-Driven Architecture (MDA) [40] and provides the following remarkable advantages:

- It is a **widely spread** language, **known and used in different domains**.
- It is an Object Management Group (OMG) **standard** [30].
- It provides a set of **generic modeling elements**, supported by a **graphical syntax** and a closed set of **diagrams**, which enables the capture of architectural and behavioral details.

A key of the success of UML is related to the generality of the provided modeling elements. These elements have a simple semantics that can be easily understood and interpreted by engineers of different domains. For instance, stating that a UML *port* is a mechanism to access to/from a UML *component* which hides component internals. This simple element semantics is encompassed by a simple graphical syntax, which facilitates the comprehension and adoption of UML diagrams.

UML has been also proposed for the modeling of embedded systems. Embedded systems have become complex. The increasing amount of silicon available in a single-chip enables the integration of more hardware and software functions. In close relationship, the specification and modeling tasks have become increasingly complex. Models need to reflect systems integrating multiple applications and diverse software platform components, e.g., embedded RTOS, middleware, drivers, etc. Similarly, current hardware architectures rely on multi-core processors, surrounded by many HW devices for communication, storage, sensing, and actuation. In addition, several types of analysis are applied (e.g., schedulability, timed-simulations, etc.) which require to add additional information to the model, e.g., annotations of extra-functional properties related to timing, memory sizes, energy, etc.

A solution is to build models integrating parts under different Domain-Specific Languages (DSLs). However, fragmentation into DSLs limits the understanding of the overall model (all the engineers handling the model should know all the DSLs involved) and requires an additional effort to integrate the DSLs.

In this scenario, relying on UML has the advantage of providing a common and comprehensive host language. UML lacks the semantics and specific elements required for tackling the Hardware/Software Codesign (HSCD) of complex

embedded systems. However, UML enables to cover this *semantic lack* by means of a UML extension mechanism called profile. A UML profile provides *stereotypes*. Stereotypes are applied to UML modeling elements and add them additional attributes and domain-specific semantics. In fact, the OMG currently provides a rich portfolio of UML profiles oriented to different domains like telecommunication, middleware, and real time.

Among these OMG standard profiles, the Modeling and Analysis of Real-Time Embedded Systems (MARTE) profile [29] provides a rich set of modeling elements, sufficiently broad to cover HSCD of real-time and embedded systems. For instance, it supports the modeling of the hardware and software platform, of the application, of extra-functional properties, and the definition of performance and real-time analyses.

As well as the UML and the MARTE profile, a modeling methodology is required. The modeling methodology states how to use the language in order to build the model of a system. It states which information have to be captured and how it is structured and captured through the available modeling techniques, such that the model can be processed and provides all the information required by the analysis tools and processes related to the HW/SW codesign. All these aspects defining a modeling methodology have to serve to its main purposes. The main purpose of the methodology presented in this chapter is to enable the development of abstract models which can be used as a single-source for the different activities involved in electronic system-level design. The modeling methodology shall not fix a specific Electronic System Level (ESL) design flow, but it has to allow that applicable design flows can progressively enrich the model from early stages, when limited information is available, toward an enriched model which allows the implementation of an efficient, potentially optimal, solution.

Following, Sect. 5.2 presents the goals of the methodology. The section motivates how these goals turn into requirements because of the need of higher design productivity. Moreover, it precises the meaning of some of this requirements (e.g., single-source, incremental modeling) in the modeling context. Section 5.3 provides an overview of related modeling approaches, showing how they partially cover the aforementioned requirements. Then, Sect. 5.4 presents the generic UML-based modeling techniques adopted by the methodology to cover Sect. 5.2 requirements. Section 5.5 introduces, before the conclusions, a single-source design framework exploiting the presented modeling methodology.

5.2 Modeling Requirements

5.2.1 Single-Source Approach

A model-driven design approach helps in analyzing and predicting the behavior of a system from different perspectives and for different purposes. In a *multi-source* approach, several models of the same system have to be developed, each one associated to a perspective, type of analysis, or design activity. However, relying on several

models easily leads to extra modeling efforts, redundancies, inconsistencies, and traceability problems. In contrast, an advanced software development methodology like *Agile* adopts a *single-source* approach, which reduces the maintenance burden and the traceability burden and increases consistency [2].

The same advantages motivate the proposal of a single-source approach for modeling and design of embedded systems [7], which is sketched in Fig. 5.1.

A single model, a UML/MARTE model in this case, captures all the information required by the many tasks involved in a modern embedded system design flow, e.g., reusability, verification, schedulability analysis, architectural mapping, simulation and performance analysis, design space exploration, etc.

The left-hand side of Fig. 5.2 illustrates the multi-source approach, where two independent models, A and B, of an embedded system are developed.

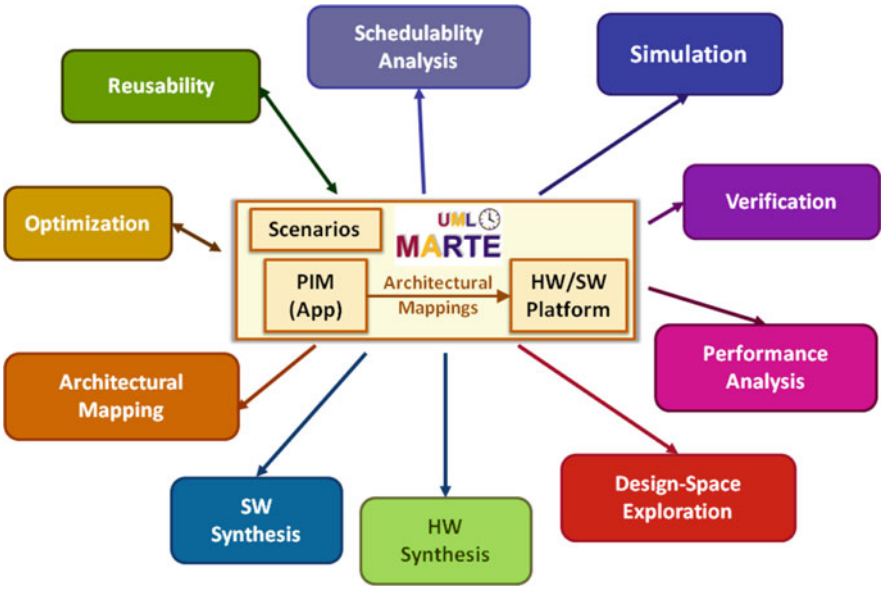

Fig. 5.1 The MARTE model as a single-source model

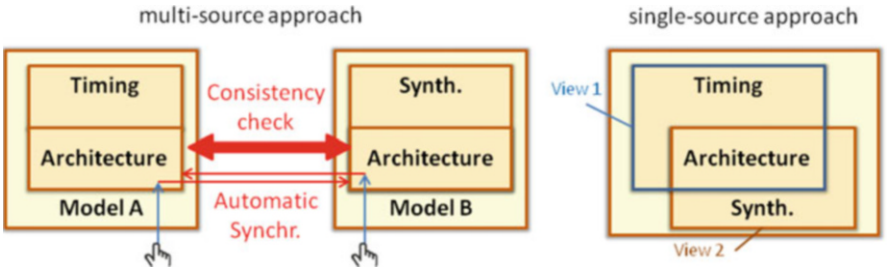

Fig. 5.2 Multi-source vs. single-source approach

Model A serves for analyzing timing performance. Model B is the input for a SW synthesis process. Model A adds some timing annotations, which are not present in model B. Model B adds target information, which is not present in model A. However, both models should reflect the same architecture of the application. Therefore, a double unnecessary modeling effort has been done. A consistency check to ensure that both models A and B reflect the same architecture is required, at least after the first development of the models and specially in a context where model architectures can be edited. These problems exponentially grow with the number of independent models required (in the worst case, one per design activity shown in Fig. 5.1).

In contrast, in the single-source approach (right-hand side of Fig. 5.2), the architectural information is captured once in the model. Then, in order to capture models A and B, the model is extended twice: once for capturing the time annotations required by the A model and once more for capturing the synthesis information required by the B model. The adoption of a single-source approach is a distinctive and remarkable aspect of the shown approach. Moreover, there are other important characteristics which need to be preserved and adopted, as addressed in the following sub-sections.

5.2.2 Separation of Concerns

The single-source approach centralizes the information required for the design tasks in a single model, with the advantages discussed in the previous section. The complex system model will have a big amount of information. In this context, *Separation-of-concerns* helps to provide a structure to such information. Model *Viewpoints* [22] and *Perspectives* [46] become essential for the modeling activity. Viewpoints are about defining the most relevant categories of information and about the structure the model concerning them. Perspectives are related to the model concerns, e.g., model properties or actions performed on the model, that shall be presented and can be accessed in the modeling framework. Combining perspectives and viewpoints facilitates model edition. It enables the cooperation of different modelers, which can focus on specific information of the model. It also simplifies tools and activities around it, e.g., the model navigation performed by code generators.

5.2.3 Incremental Modeling

In software engineering, *incremental modeling* refers to the delivery of a series of releases called *increments* [39]. This approach enables to progressively provide more functionality on each increment. In software modeling, increments can refer to customizations, as well as to extensions [1].

In our single-source modeling context, incremental modeling refers to the possibility to start by building a first model that enables a first set of analysis

and design activities, and enhance it later on. Then, further modeling increments may enable additional design activities and/or improve the results of the previously applicable activities, e.g., enabling more accuracy in the performance assessments. A model increment shall not prevent performing the design activities already enabled by previous versions of the model.

5.2.4 Component-Based Functional Modeling

A software component-based approach [38] has important advantages. First, it enables to build system functionality as a composition of existing and reusable components. These components interact with each other only through well-defined interfaces [32], which declare the functional services they provide and require. Adopting a software-centric approach, i.e., assuming a default software implementation of the functionality, is also an efficient approach according to the increasing and dominant amount of software in current embedded systems.

5.2.5 Support of System-Level Design Activities

Adoption of ESL design [4] is key in order to shorten the *productivity gap* [19]. ESL design tackles the productivity gap by raising the abstraction of the starting model and by introducing automated design activities around that system-level model. Two main design activities are:

- **Design Space Exploration (DSE):** consists in the activity of exploring design alternatives and selecting the optimal ones.
- **System-Level Synthesis (SLS):** consists in the generation of the implementation from the initial model, used as a specification. It requires the decision of the HW/SW partition and the automated generation of software, hardware, and HW/SW interfaces.

DSE can rely on SLS. For instance, [4] highlights that one of the most important benefits for High-Level Synthesis (HLS) is that it enables (hardware) design exploration. The methodology in [37] enables the automated SW synthesis from a UML/MARTE model of the binaries targeted to a multi-core heterogeneous platform. This automation is exploited for the exploration of different software implementation alternatives. The aforementioned approaches are applied after HW/SW partition.

An *explore-then-synthesise* approach relying on performance assessment techniques is also possible. Kang et al. [20] refers to DSE as the activity of exploring design alternatives *prior to implementation*. Therefore, in this approach, the exploration activity is done first on the model. Performance estimations on different model configurations, which can reflect different HW/SW partitions, serve to decide the most convenient one. Thus the system-level synthesis is only applied on that

choice. In a UML/MARTE context, [13] showed how the model can reflect different mappings, comprising different HW/SW partitions. It enables the generation of a performance assessment model which, linked to a DSE tool, enabled an automated search of the Pareto solutions set.

The automation of the DSE and SLS activities is key in order to cope with the exploration of huge design spaces and in order to eliminate human errors both in the exploration and implementation steps. This has motivated the development of automated DSE, SW synthesis, and hardware HLS frameworks. The possibility to exploit these frameworks relies on the fact of enabling an input model which conjugates the abstraction required by ESL, with the information required for performing these activities. This chapter shows how it is done from a UML/MARTE model under a single-source approach.

5.2.6 Support of Mixed-Criticality

Mixed-Criticality System (MCS) have got an increasing interest [23]. MCS integrate applications with constraints whose fulfillment has different levels of criticality and which can share computational, memory, and communication resources. Although there is not an unanimous convey in the meaning of criticality, it has been associated to the impact of the occurrence of a failure [6]: e.g., *safety critical* when the failure can cause injures to humans, *mission critical* when the failure prevents the system to perform its expected behavior, but it does not compromise safety, and *low critical* when the impact is affordable. Safety standards associate criticalities to a set of more or less strict requirements on the development process. There are several reasons for the highest interest in mixed-criticality systems. Reusing existing functionalities and integrating them in the same chip or in the same platform is an efficient way to exploit growing integration capabilities and cost-effective platforms. However, dependability problems arise, as not all the functionalities and performance requirements are equally important. A mixed-criticality aware design methodology considers this differences. For instance, methodologies need to combine real-time analysis for the safety parts with hard real-time constraints, with other techniques employed in the optimization of soft real-time embedded systems, e.g., based on simulation and on average-case optimization. Accordingly, MCS models need to provide support for the emerging mixed-criticality design methodologies. Mixed-criticality has to be reflected in system models.

The modeling scenarios which can be identified in recent MC research and in current industrial practices related to safety standards lead to the need to associate criticalities to different type of elements, i.e., application components, platform resources, constrains, annotations of extra-functional properties. These criticality annotations are used to apply mixed-criticality-specific modeling rules to feed mixed-criticality-specific schedulability analyses, DSE flows, and development and implementation constraints.

5.3 State of the Art

There are several examples of model-based methodologies for modeling, exploration, and implementation of complex embedded systems. For instance, *MILAN* [5] enables a model-based approach to capture the application and the platform (called resource model). MILAN also introduced the idea of constraint model, which distinguishes between semantic constraints, which give composability rules, from design constraints, which capture performance requirements. The framework shows how a model-based approach facilitates the integration of several simulation tools at different levels of abstraction for the estimation of the performance of the design point by relying on a Generic Modeling Environment (GME) [35]. *Koski* [21] is a design flow for Multi-Processor System-on-Chip (MPSoC) covering the design phases from system-level modeling to Field-Programmable Gate Array (FPGA) prototyping. System-level modeling relies on UML and separates the capture of the application from the platform. However, this approach relies on a proprietary profile for capturing the required semantics.

Despite the relative recent release of the standard MARTE profile, there have been already several proposals relying on it. *Gaspard2* [8, 35] is a design environment for data-intensive applications which enables a MARTE description of both the application and the hardware platform, including MPSoC, and regular structures. Gaspard2 uses composite diagrams and the MARTE profile for capturing both application and platform architectures. Gaspard2 tooling supports the chaining of different Model-to-Model (M2M) transformation tools. This facilitates the generation of synthesis flows and also of performance models. Specifically, Gaspard2 supports the generation of SystemC TLM models at the Programmers View Time (PVT) level. It enables fast simulations, which speeds up exploration.

MoPCoM [45] is another design methodology for the design of real-time embedded systems which supports UML and the MARTE profile for system modeling. Specifically, MoPCoM uses the Non-Functional Property (NFP) MARTE profile for the description of real-time properties; the Hardware Resource Modeling (HRM) MARTE profile for platform description; and the Alloc MARTE profile for architectural mapping. Moreover, MoPCoM defines three levels of generation. The second level, called Execution Modeling Level (EML), targets the generation of models for performance analysis, and it is suitable for obtaining performance figures used in DSE iterations. However, work reported in [24] mostly focuses on the Detailed Modeling Level (DML), intended for implementation, by enabling VHDL code generation.

The *PHARAON* methodology [33] provided a solution for automatically synthesizing models combining new communication semantics with standard UML/MARTE real-time management features. This approach provides a flexible and easy-to-use way to specify and explore the system's concurrent architecture.

CoFluent methodology [18] captures application and hardware architecture by means of composite diagrams and SysML blocks. UML activity diagrams are used to specify application execution flows. The MARTE HRM profile is used for capturing the HW platform. CoFluent models can be translated into executable SystemC transaction-level models, which serves to obtain utilization, time, and power performance metrics.

A main limitation of the previous methodologies is that the exploration of architectural alternatives requires the edition of the UML/MARTE model and a re-generation of the executable performance model.

In [24], a UML-MARTE-based methodology relying on activity threads is proposed in order to reduce the effort required to capture the set of architectural mappings. An activity thread is a UML activity diagram where each path reflects a design alternative, that is, an architectural mapping.

In [26], a methodology for supporting designers on the evaluation of the HW/SW partitioning solutions, specifically, to identify design points fulfilling the timing constraints is shown. It proposes a way to depict in one set of diagrams all possible combinations of system configurations. By means of annotation of MARTE non-functional properties and of the application of schedulability analysis, the design space is restricted to the design points fulfilling timing requirements. However, this methodology does not rely on automated technologies for the estimation of performance metrics.

These MARTE-based specification methodologies are still limited for DSE purposes. The exploration of different platform architectures, of different architectural mappings, and even a small change in a design parameter (e.g., a cache size) still requires a manual change of the model. The *COMPLEX* [13] flow proposes a single-source approach to overcome the aforementioned limitations. Moreover, the COMPLEX framework produces a configurable performance model which avoids the re-generation and re-compilation of a performance model for each exploration alternative and thus a significant impact in the exploration time. This framework also supported the capture of the output performance metrics to be used by the objective function(s) of the DSE process within the model and of performance constraints. Enabling the capture of the performance metrics in the model, and so in a tool independent manner, enabled the direct relation of such metrics with the performance constraints also captured in the model.

Modeling complexity has increased with the need to consider the modeling of Cyber-Physical System (CPS) and Mixed-Criticality Systems (MCS). A *UML-Modelica-SysML* integrated modeling environment as a ModelicaML profile integrated in Eclipse is presented in [36]. *Modelica* is an object-oriented mathematical language for component-oriented modeling of complex physical systems containing components of diverse nature, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power, etc. The modeling of mixed-criticality systems in UML/MARTE has been proposed in [15, 16]. This work provides modeling techniques which cover a number of scenarios where mixed-criticality has to be captured.

5.4 Single-Source Modeling Methodology

5.4.1 Introductory Example: Quadcopter System

Along the following sections, the main modeling techniques of the proposed single-source methodology are presented. These modeling techniques will be presented by taking excerpts of a model of a digital electronics system embedded in a quadcopter. This quadcopter system, shown in Fig. 5.3, has been developed by the OFFIS Institute for Information Technology in the context of the CONTREX project [28]. The quadcopter digital system includes:

- The data mining, radio control & telemetry, and flight control functionalities. They are safety critical as a failure on them compromises person's safety.
- A mission functionality, which consists in the detection and tracking through a camera a moving ball, e.g., in a sport action, in order to track, record, and stream that action video to a base station.
- A functionality to log monitoring and debug data.

All this functionality is implemented in a *Xilinx Zynq* platform, which contains a processing system with two *ARM Cortex-A9* processors. In addition, the Zynq platform has an FPGA which allows the integration of custom logic and additional *Microblaze* processors, configured without caches, for enabling more predictable computational resources for safety critical functions. The Zynq board is integrated

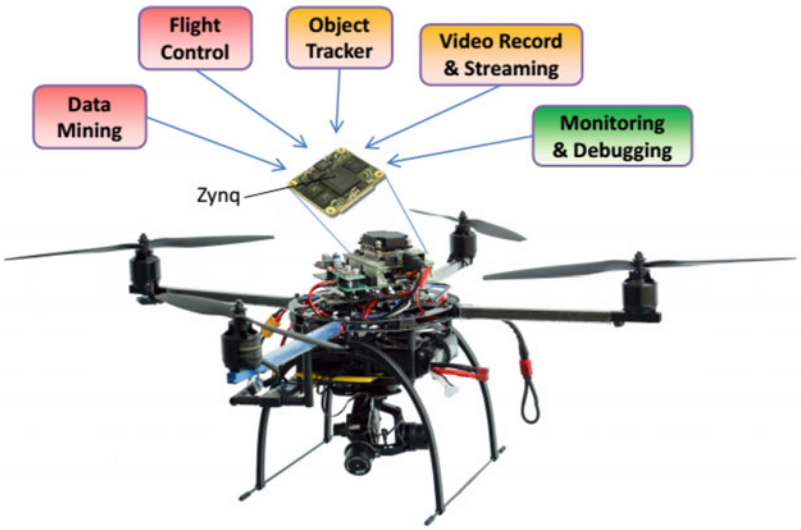

Fig. 5.3 The digital electronic system of a quadcopter is used for introducing the single-source methodology

in a board together with sensor components, motor actuators, and IO devices IO. These components are abstracted as part of the environment in the UML/MARTE model.

5.4.2 Introduction

The modeling methodology supports separation of concerns. At the root of the UML/MARTE model, model information is distributed into *views*. Model views are captured as UML packages decorated with a methodology-specific stereotype which adds the view semantics. Figure 5.4 shows a diagram with the views enclosing all the quadcopter model information.

In this methodology, model views support the separation under different concerns. The model of the verification environment (*verification view*) is separated from the system model (remaining views). As stated in MDA, the Platform Independent Model (PIM) is separated from the platform model. The PIM is captured through the data, functional and application views. Platform-dependent information is added later. The *HW resources view* declares the HW components, which can be later instanced in the *architectural view*. The declaration of software platform resources (in the *SW platform view*) is also separated from the declaration of hardware platform resources. Architectural information is captured through UML composite diagrams, and separated from component declaration, which have associated the behavioral information. The methodology follows a pragmatical and generic approach with respect to the association of behavior to the model. Figure 5.5 shows an excerpt of the quadcopter model where source files are associated to application components (UML artifacts allocated to the PIM component via UML relations decorated with the MARTE «allocated» stereotype). In addition, the methodology allows the association of paths for the sources through UML

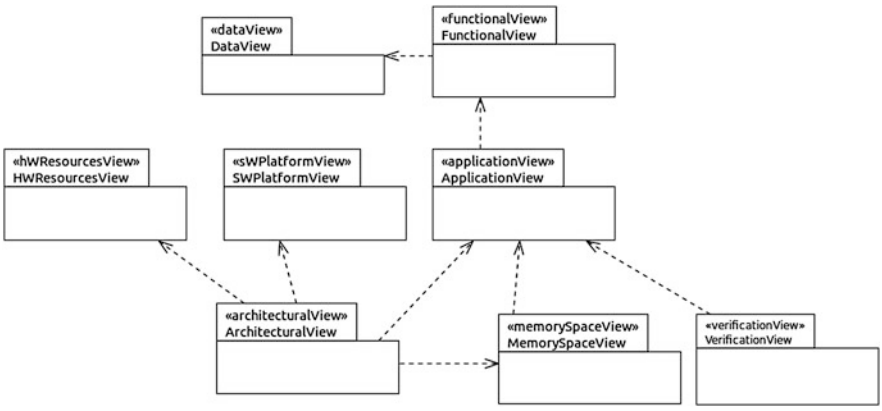

Fig. 5.4 Model views

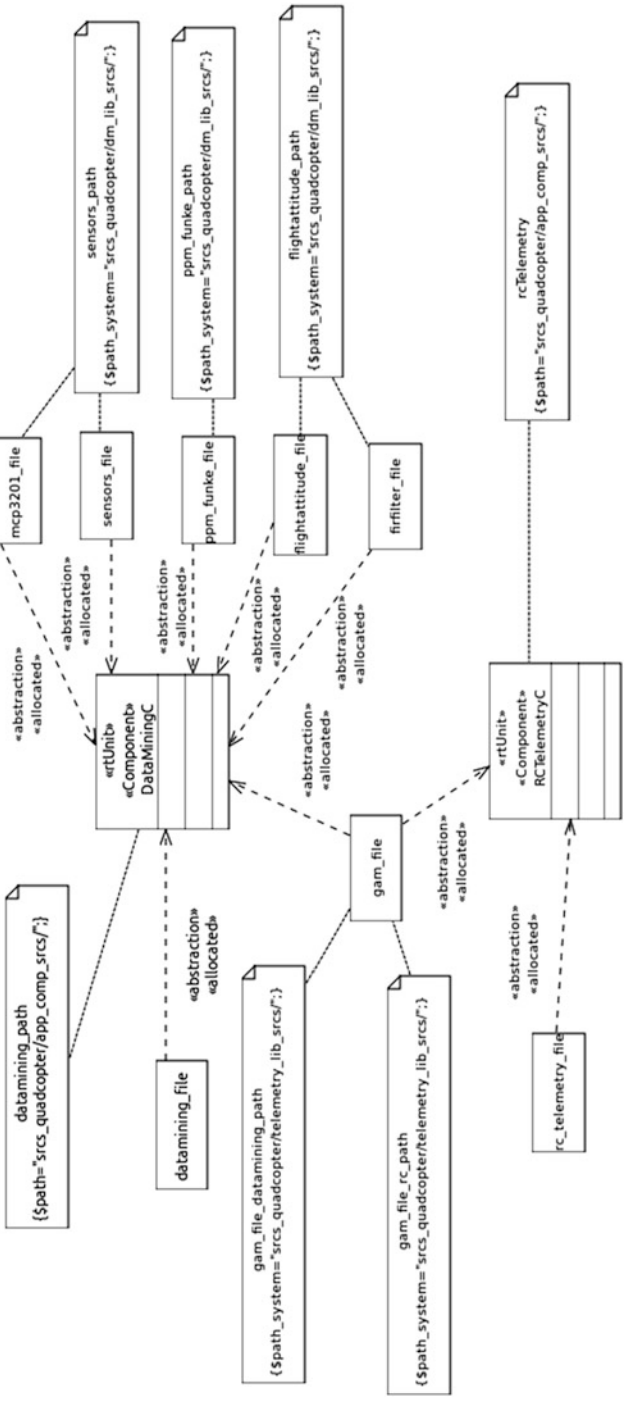

Fig. 5.5 Association of source files containing the behavior to PIM components

constraints associated to the artifacts. This mechanism is language independent and has been exploited in the automated generation of executable performance models and of synthesized binaries.

As Fig. 5.4 shows, building a platform-specific model depends on the PIM model, but not the opposite. Then, incremental model development is possible. Thus, for instance, the methodology makes possible the generation of an executable functional model. After capturing the platform-dependent model, the generation of a performance model or the synthesis phase is enabled. Similarly, time or energy annotations can be added, e.g., to a hardware component, to add accuracy to the automatically generated performance model. However, if these annotations are not present, the generation of the performance model is still possible (default values are used).

5.4.3 Platform-Independent Model

The methodology enables the description of a platform-independent model (PIM) under a component-based approach.

The methodology enables a clean separation of the PIM information. Figure 5.6 shows the components of the quadcopter PIM as they are seen in the Eclipse model explorer view.

Among the six components, the *quadcopter_app* component is decorated as *«system»* component (it can be seen in Fig. 5.7). The PIM system component is the top component of the PIM model hierarchy, which contains the PIM architecture, shown in Fig. 5.7. The remaining components are PIM components, to be eventually instanced in the PIM system component and which shall be stereotyped to be either an active component, a passive component, or a shared variable. The MARTE *«RtUnit»*, *«PpUnit»*, and *«SharedComResource»* stereotypes are respectively used for that purpose. Figure 5.8 shows the application of the *«RtUnit»* stereotype to the datamining (DataMiningC) and radio-control and telemetry (RCTelemetryC) components of the quadcopter. As can be observed in Fig. 5.6, non-system components have been enclosed in an additional UML package called *AppComponents*. This

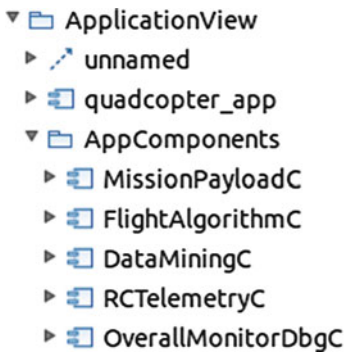

Fig. 5.6 Components of the quadcopter PIM model

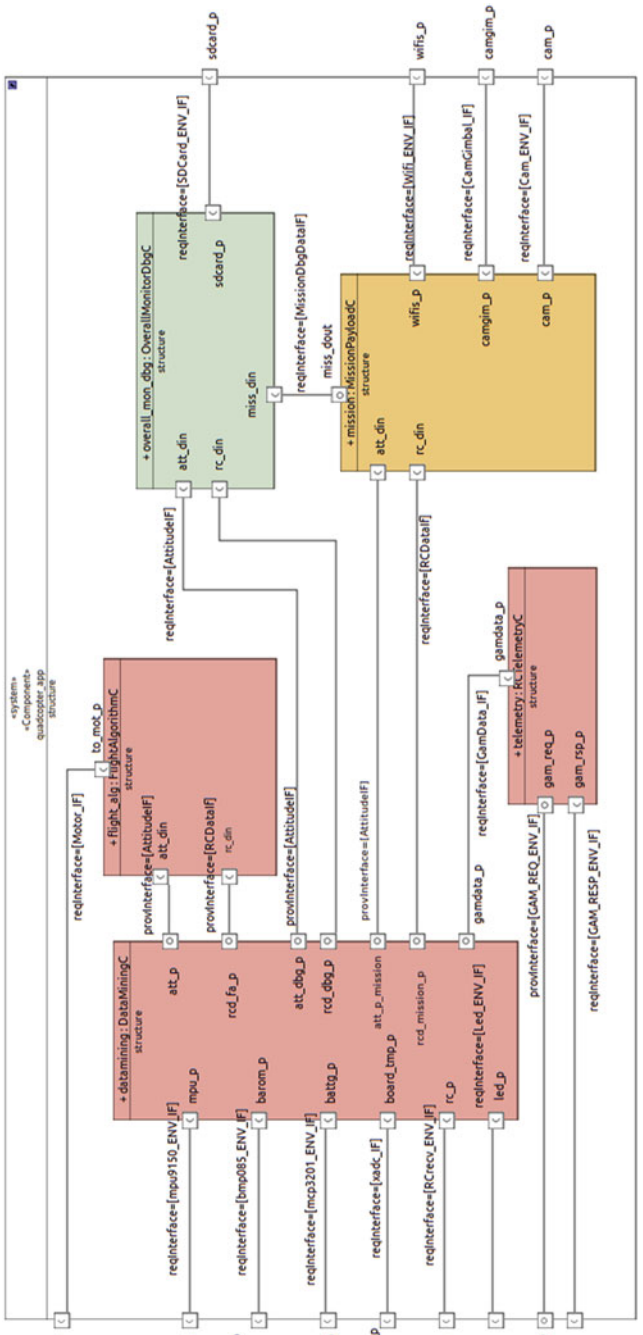

Fig. 5.7 Quadcopter PIM architecture

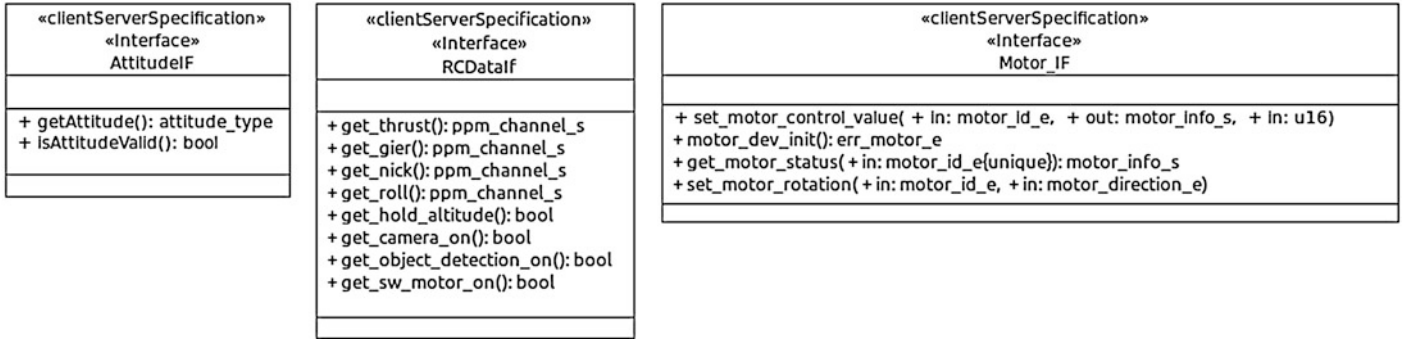

Fig. 5.8 Quadcopter functional view excerpt

is not required, but supported by the methodology, to structure more PIM model information.

The architecture of the platform-independent model is captured within the PIM system component by means of a UML composite diagram, as shown in Fig. 5.7.

The PIM system component makes instances of the PIM components as UML properties, either active components or passive components, that is either of *«RtUnit»* or *«PpUnit»* components. The quadcopter model only instances active components, since all the components have internal periodic tasks. For instance, *datamining* is an instance of the *DataMiningC* component. PIM component instances are connected via channels. Channels are captured as UML port-to-port connectors. While these connectors reflect to *internal connections*, the PIM architecture also contains UML connectors. In this case, they connect a port of the top component and a port of an internal component, reflecting the delegation of the function services or requirements across one hierarchy level.

Each port is stereotyped as a client-server port via the MARTE *«ClientServerPort»* stereotype. This stereotype has the attribute *kind*, of the MARTE *ClientServerKind* type, which allows the methodology to state that the port has either a *provided* or a *required* interface. In addition, the attributes *provInterface* and *reqInterface*, both of the UML interface type, enable the specification of the specific interface associated to the port. Such an interface has to be previously captured as a client-server interface in the functional view. Figure 5.8 shows an excerpt of the functional view of the quadcopter, with three interfaces.

All of them have been applied to the MARTE *«ClientServerSpecification»* stereotype to identify the interfaces that can be exported by PIM components. Each interface declares the methods that are exported at that interface. For instance, the AttitudeIF interface declares two methods: the *getAttitude* method for obtaining the attitude information from the provider component and the *isAttituteValid* method for obtaining a flag stating if the attitude value that can be currently retrieved from the component is valid. In turn, each of those methods are specified by their input and output parameters. Both of them have to have a precisely specified type. The data view enables the user to precise all the types to be employed in the interfaces. Figure 5.9 shows an excerpt of the data view of the quadcopter model. This excerpt shows the capability of the methodology to capture complex structured types, e.g., the *attitude_type* returned by the *getAttitude* method.

Concerning the capture of communication semantics, Fig. 5.7 reflects the simplest case, where a default semantics is associated to channels. For instance, the default semantics states that the call to the service is blocking at the initiation and end of the service. That is, the component requiring the service, i.e., making the call, waits for the provider component, i.e., the one implementing the called function, to be ready for executing it, and also waits for its completion.

In addition, the methodology enables a more detailed specification of the channel semantics. Figure 5.10 illustrates a case where the semantics of the channel used by the *overall_mon_dbg* component to retrieve attitude data from the *datamining* component has a user-defined semantics. For it, the UML port-to-port connector representing the channel is stereotyped with the *«channel»* methodology-specific

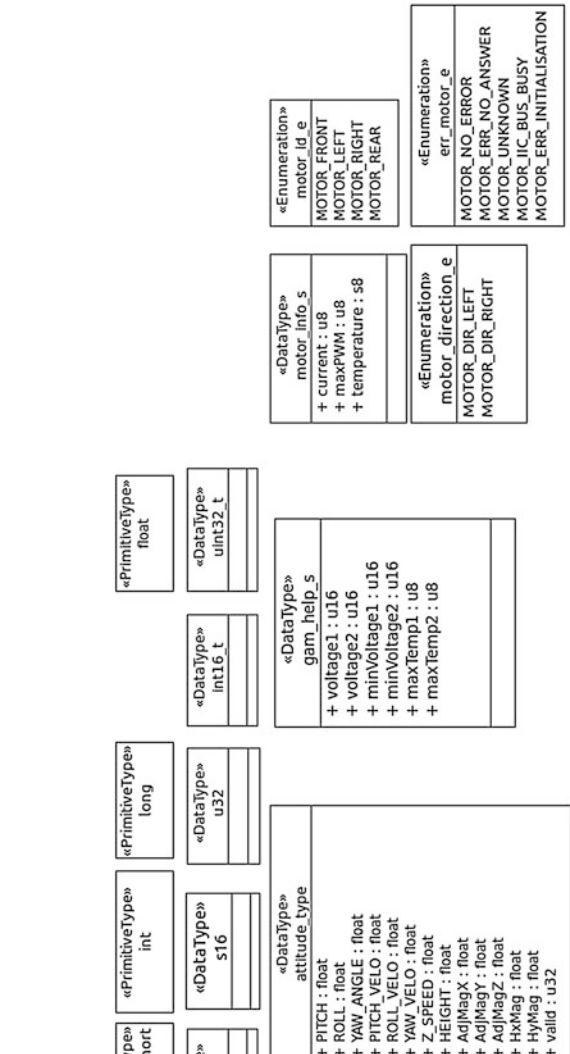

Fig. 5.9 Quadcopter data view excerpt

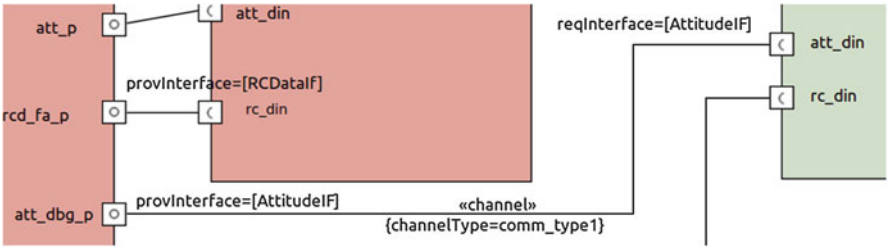

Fig. 5.10 Channel instance with custom semantics

Fig. 5.11 Specification of channel with custom semantics

stereotype. The *«channel»* stereotype provides the attribute *channelType*. This attribute can be assigned a UML component decorated with the MARTE *«CommunicationMedia»* stereotype. This component has to be included in an additional view, the communication view (see Fig. 5.11), and represents a channel whose semantics can be customized by the user. In order to customize the communication semantics, the MARTE *«StorageResource»* stereotype and the *«ChannelTypeSpecification»* stereotype are used. The former stereotype makes possible to specify the channel buffering capability, i.e., how many function calls can be buffered by the channel. The methodology-specific *«ChannelTypeSpecification»* stereotype contributes additional attributes for configuring the channel semantics, for instance, if the communicating components shall synchronize at function call and at function return.

5.4.4 Platform Resources

As was mentioned, the methodology enables the specification of platform resources in two separated views. The *SW platform view* is used to declare the software resources of the platform, i.e., operative systems and drivers. Figure 5.12 shows the software platform resources in the software platform view of the quapcopter model. The basic SW platform resource is the Operating System (OS), which is captured as a UML component decorated by the methodology-specific *«OS»* stereotype. The only OS semantics injected by this means is used for producing performance models based on generic RTOS models. However, more specific information is required in other contexts. The *«OS»* stereotype provides the *type* property, which serves as a string descriptor which uniquely identifies the target OS or RTOS in SW synthesis. The methodology also supports a more detailed specification of the OS behavior. This is necessary for safety critical cases, where an accurate performance analysis also relies on an accurate modeling of the OS scheduler behavior. For it, the *«OS»*

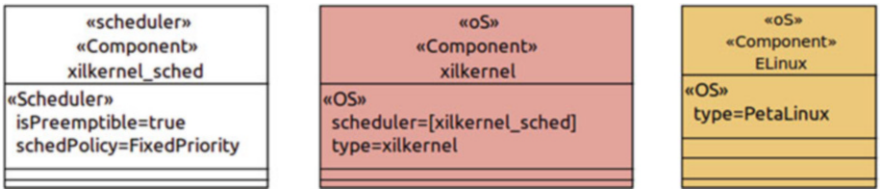

Fig. 5.12 SW resources of the quadcopter platform

stereotype provides the *scheduler* attribute. The scheduler attribute specifies one scheduler component, i.e., a component stereotyped with the MARTE *«Scheduler»* stereotype. In turn, the *«Scheduler»* stereotype enables the attributes *isPreemptible*, *schedPolicy*, *schedule*, and *otherSchedPolicy*, which enable quick and versatile specification of scheduling policy. The *schedPolicy* attribute enables a synthetic capture of the most usual scheduling policies (static scheduler, fixed priorities, Earliest Deadline First (EDF), Round Robin (RR), etc.). The attribute *isPreemptible* states that the RTOS re-schedules on release events of other application tasks. The *schedule* attribute is used to configure and complete the description of the scheduling policy when *schedPolicy*=*TimeTableDriven*. The*TimeTableDriven* value can be used in MARTE to specify both order-based schedules and time-triggered schedules. In the former case, the *schedule* attribute serves to capture the execution order, i.e., the schedule of the tasks allocated to the OS. The *otherSchedPolicy* attribute is used to support other scheduling policies not covered by MARTE. In the methodology, it is also exploited for enabling a more synthetic description capable to preserve the single-source approach in a DSE context (an example is given in [16]). In the quadcopter case, the SW resource view shown in Fig. 5.12 states that the xilkernel OS has been configured with a priority-based scheduling policy.

Figure 5.13 shows the resources declared in the hardware platform view of the quapcopter model. In this view, all the hardware platform resources to be instanced in the hardware platform architecture shall be declared. Each platform resource is declared through a component with a specific MARTE stereotype adding the hardware resource semantics. For this, the HRM MARTE profile is intensively used. As shown in Fig. 5.13, the methodology supports the modeling of computational resources (e.g., HW processors), communication resources (e.g., buses), memory resources (e.g. cache memories), main memories, and I/O devices. Depending on the type of hardware component, and thus of the stereotype, different attributes are available. None of the platform views contain any architectural information. There is one exception in the HW resources view, which allows to directly link a set of cache components to a processor component. The set is passed as the value of the *caches* attribute of the MARTE *«HwProcessor»* stereotype. Each element of the set passed to the *caches* attribute is a component decorated with the *«HwCache»* MARTE stereotype and also declared in the HW resources view. This mechanism has been used in the quadcopter model to simplify the capture of level 1 caches associated to the *ARM_Cortex_A9* processor components. For the same example,

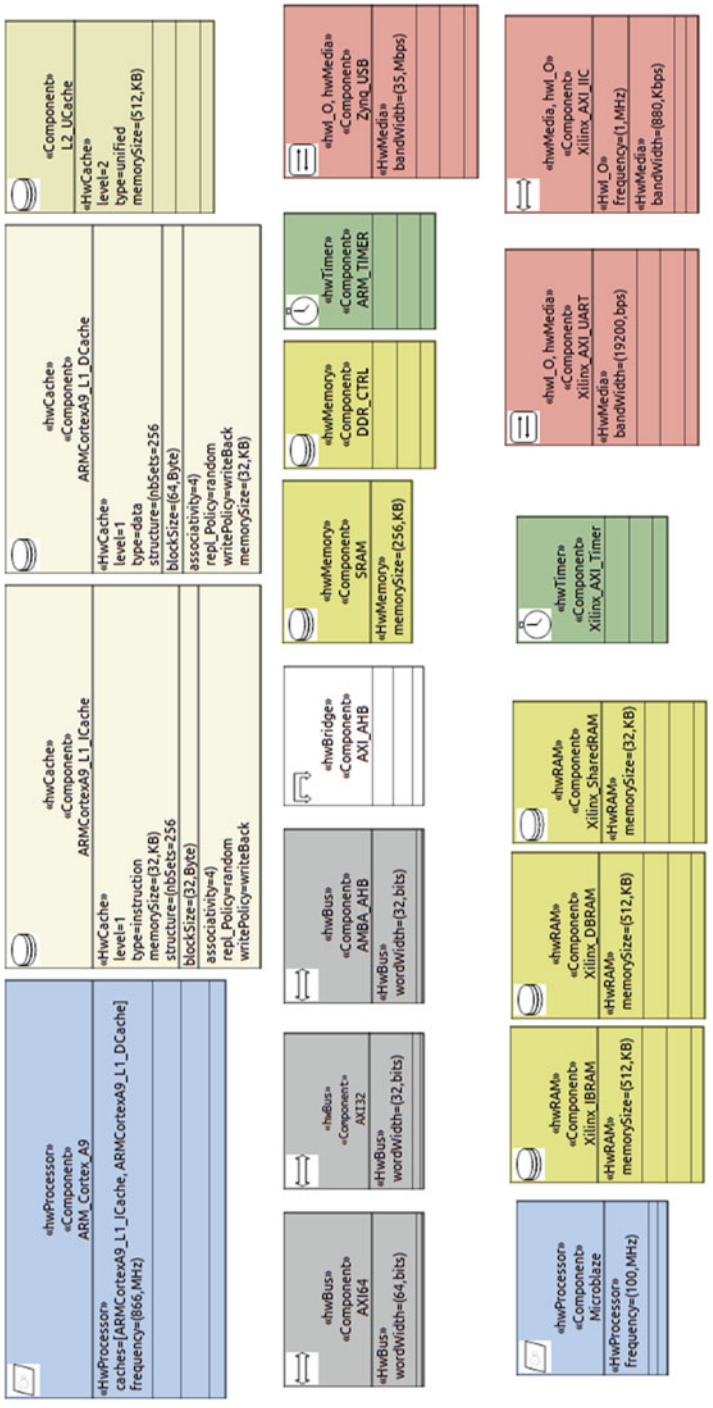

Fig. 5.13 HW resources of the quadcopter platform

the *microblaze* processor components declared in the HW resources view have an empty *cache* attribute. The microblaze processors in the quadcopter system have no cache since they are used to run the datamining and control functionalities, which require more predictability due to their criticality.

5.4.5 Platform-Specific Model

As was shown in Sect. 5.4.4, the SW platform and the HW resources views declare platform resources. They do not contain architectural information, apart from cache resources associated to processors. The methodology supports the specification of the SW/HW platform architecture and of the mapping of the PIM model to the SW/HW platform through the memory space and architectural views.

The memory space view is a non-mandatory view which can be used to specify memory spaces and the mapping of the component instances to the declared memory spaces. This is a first mapping level, which is relevant in SW implementation. A software process is inferred for each memory space. By default, if the user does not specify a memory space view, an implicit one with a single memory space is inferred, and it will be assumed that all the component instances are mapped to the implicit memory space. Figure 5.14 reflects a case where four memory spaces have been specified. The specification is done again within a composite diagram associated to a system top component, called *quadcopter_memspaces*, captured within the memory space view. This component is captured as a specialization of the PIM top component. Therefore, the references to the component instances, i.e., *telemetry*, *datamining*, *flight_alg*, *mission*, and *overall_mon_dbg*, are visible and can be used as source of the allocations.

Figure 5.15 shows the architectural view of the quadcopter system. The architectural view captures the mapping of memory spaces to OS instances, which effectively closes the mapping of the PIM to the SW/HW platform.

The SW/HW architecture captures the mapping of the OS instances onto the computing elements, i.e., HW processors, and the interconnection of the different HW elements.

As can be noticed, the mapping of memory spaces to OS instances and the mapping of OS instances to computing elements are captured again by means of UML abstractions with the *«allocate»* stereotype. Regardless of the type of source or destination, a static mapping is captured with the same modeling technique. For the connection of PIM component instances, hardware platform components are linked also through UML port-to-port connectors. Summing up, the methodology employs the same modeling techniques, to solve the same type of modeling needs, for yielding more understandable models and an easier to learn methodology.

5.4.6 Extra-Functional Properties and Performance Constraints

The methodology supports the annotation of several types of extra-functional properties (EFPs). These annotations are used by performance and schedulability

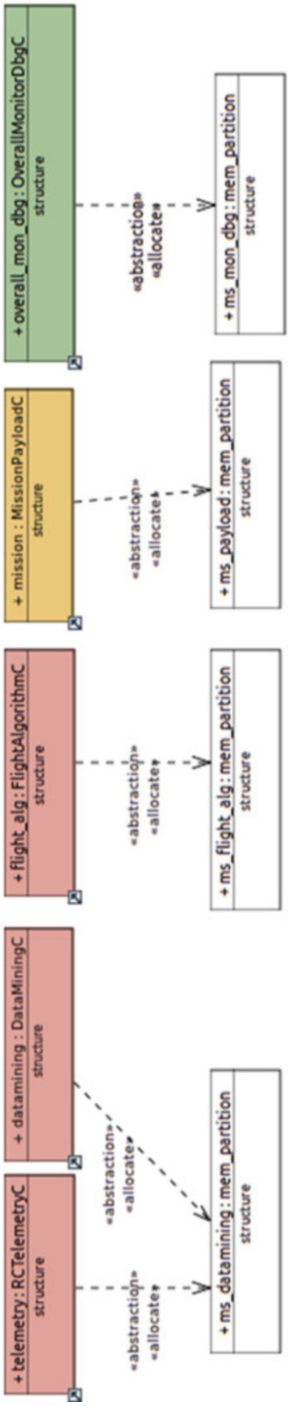

Fig. 5.14 Mapping of PIM component instances to memory partitions

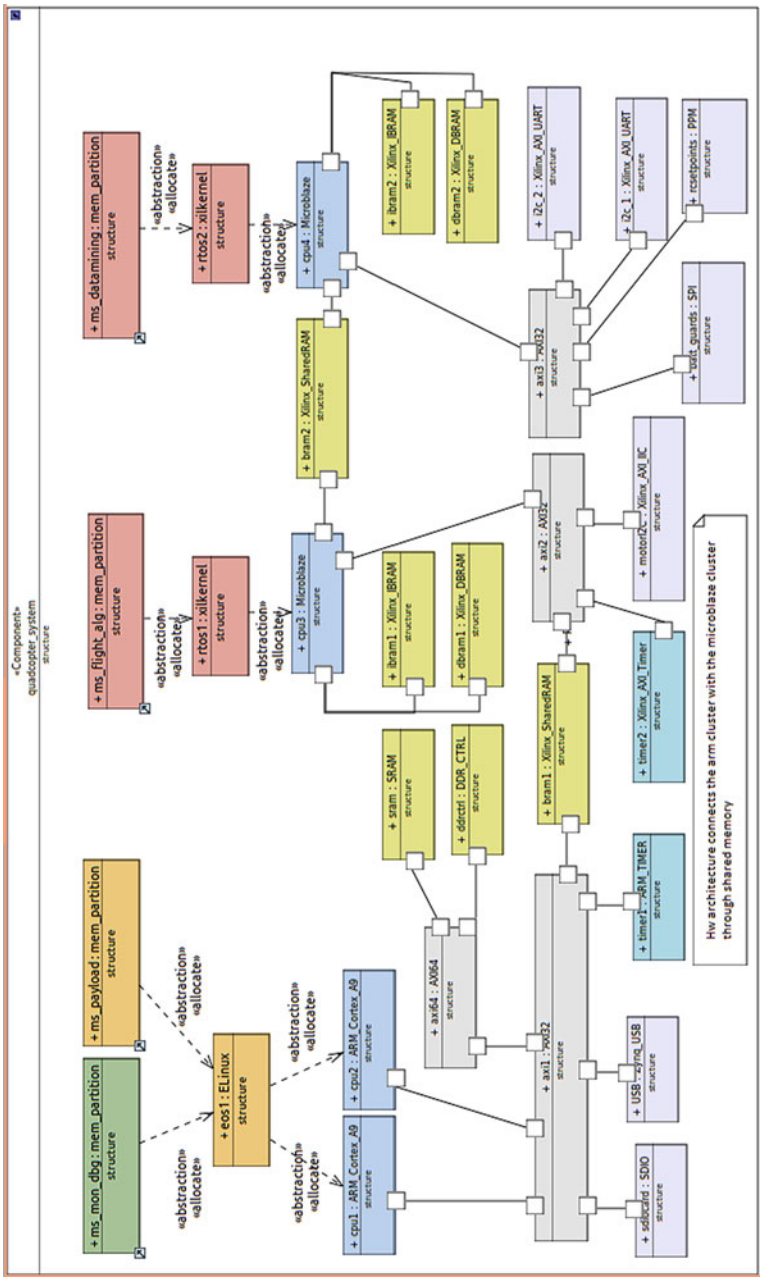

Fig. 5.15 Quadcopter architectural view

analyses. Most of them are performed in the HW resources view. Figure 5.12 illustrates several types of EFPs. For instance, the sizes of the different types of memories and structural information of caches, i.e., number of sets, cache policies, bus widths, frequencies for processors (and other types of hardware resources), and I/O device bandwidths, can be stated by only relying on the different MARTE stereotypes from the HRM profile.

Moreover, the methodology also supports the annotation of power and energy consumption associated to HW resources. For annotating static power consumption, the HW component is decorated with the MARTE *«HW_Component»* stereotype, which provides the *staticConsumption* property. Moreover, the methodology supports the modeling of power state machines. The HW component can have different functional modes defined by the operating frequency, source voltage, dynamic power, and average leakage. A UML state diagram associated to the HW component and the MARTE *«mode»* and *«ModeTransition»* stereotypes are employed for that purpose. The *«HwPowerState»* stereotype introduced in [3] and the MARTE *«ResourceUsage»* stereotypes are used for characterizing the static and dynamic power consumptions of each mode. The modeling methodology also supports the annotation of energy consumption. The annotations depend on the modeling element. For instance, in order to annotate the energy consumed per cycle in a processor, a *cycle* attribute of MARTE *NFP_Energy* type, and decorated with the MARTE *«NFP»* stereotype, is added to the processor component. An analog technique is used for associating energy consumptions to other components. For buses and memories, the energy associated to an access is annotated. For caches, two energy consumption figures are annotated, for distinguishing the *hit* consumption from the *miss* consumption.

The methodology also allows the annotation of workloads, e.g., Worst-Case Execution Time (WCET), average times, and Best-Case Execution Time (BCET), to application components. Figure 5.30 in Sect. 5.4.10 illustrates such annotations in the mixed-criticality context and how they depend on the allocation to platform resources.

The methodology enables the specification of performance requirements. Performance requirements are used in performance analysis and design space exploration to check if the assessed solutions are acceptable. Similarly, implementation and test phases should consider this information for the validation of the chosen design alternative. Figure 5.16 illustrates a compact mechanism for performance constraint specification, possible whenever the performance constraint is captured through a UML property. In the case of Fig. 5.16, through a UML comment decorated with the MARTE, *«RtSpecification»* stereotype is linked to the PIM component instances of the quadcopter. The *«RtSpecification»* contributes the *relDl* stereotype, a UML property which enables a synthetic capture of the deadline through an expression under the MARTE Value Specification Language (VSL).

The methodology provides a mode general mechanism to express performance constraints. It is done by means of a UML constraint decorated with the MARTE *«NFP_constraint»* and «ExpressionContext» stereotypes. The latter enables the capture of a Boolean expression written in VSL. The Boolean expression refers

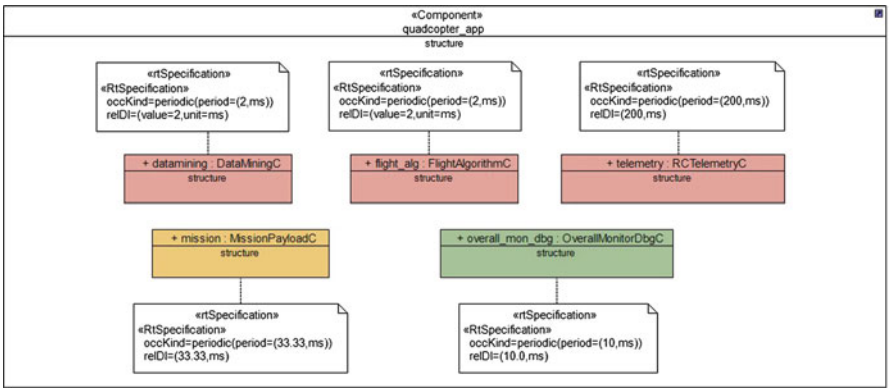

Fig. 5.16 System and application performance constraints on the quadcopter

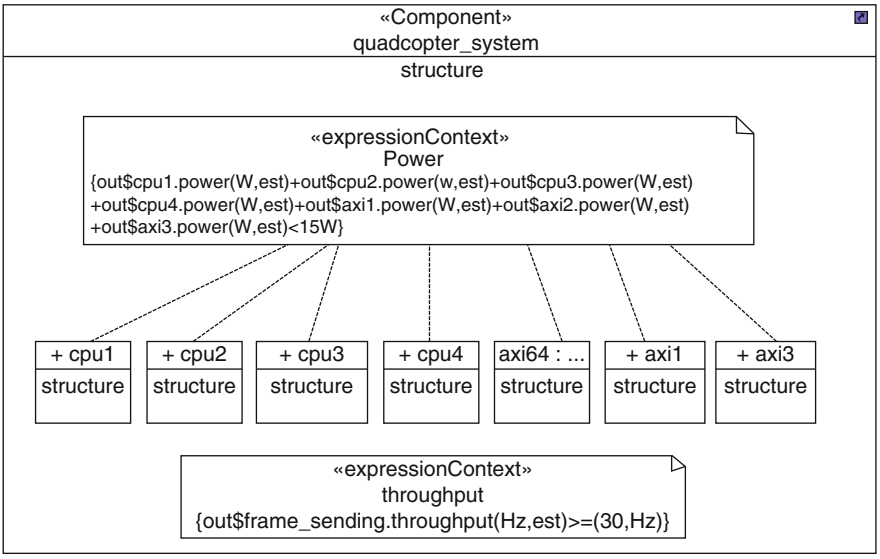

Fig. 5.17 System and application performance constraints on the quadcopter

to at least one output performance metric. An output performance metric is a value of an extra-functional property which should be estimated by an analysis tool after taking the own UML/MARTE model as an input. The output performance metric is expressed as a VSL variable with the *out* prefix. The output performance metric can be general (independent on the model), e.g., overall power consumption, or refer to a model element, e.g., a processor consumption. Figure 5.17 provides an example on the quadcopter. The NFP constraint is linked to the processor and bus instances referenced in the expression and whose power dissipation contributes to the overall power consumption. Several output performance metrics are required to

be calculated, i.e., the dissipation of each CPU and of the *axi64*, *axi1*, and *axi3* bus instances. Notice that the performance requirements refer to both application and platform elements.

5.4.7 Design Space

The proposed methodology enables the description of a design space. That is, instead of a design solution, the modeler can capture several or many design solutions, e.g., several PSMs, in a single model, thus effectively enabling the single-source approach for DSE. As was mentioned, this is crucial for avoiding model re-factoring and thus enabling fast iterations during the design space exploration.

The modeling of the design space refers to two basic modeling elements: property values annotations and architectural mappings.

The methodology supports DSE parameters. A DSE parameter is a property value annotation which, instead of specifying a single fixed value, specifies a range of possible values. The capture of a DSE parameter relies on VSL. The syntax of the VSL expression capturing the DSE parameters is the following one:

$DSEParameterName = DSERangeSpecification

The *"$"* symbol prefixes a VSL variable and thus the DSE parameter name. The *DSERangeSpecification* expresses the range of the DSE parameter, that is, all the values that the *DSEParameterName* variable can have during the exploration. The DSE parameter range can be annotated either as a collection or as an interval. Collections are captured with the syntax *DSERangeSpecification=(v1, v2, v3, unit)*, where *"v1, v2, v3"* are the numerical values of the parameter and *unit* expresses the physical unit associated the values. MARTE provides a rich set of unit kinds to support the different extra-functional properties characterizing systems components, e.g., frequencies, bandwidth, data size, etc. Intervals follow the syntax *"DSERangeSpecification=([v_{min} … v_{max}], unit)"*. For a complete determination of the exploration range, this style obliges to assume an implicit step. For an explicit and complete determination of the DSE range, the support of the style *"DSERangeSpecification=([v_{min} … v_{max}, step], unit)"* is proposed, which means a minor extension of VSL. The definition of non-linear ranges is possible. For instance, *step* can take the value *exp2*, which enables the definition of a geometrical progression, i.e., the second value is *"$v_{min}x2$"*, and so on.

Figure 5.18 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 (works at 766 MHz), or a Z-7020 device (at 866 MHz) can be explored.

In Fig. 5.18, the DSE has been associated to 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 methodology also allows the association of the DSE parameter to the instance properties. Therefore, if the user wants to explore the variations on the frequency of a single ARM core, then the constraint has to be

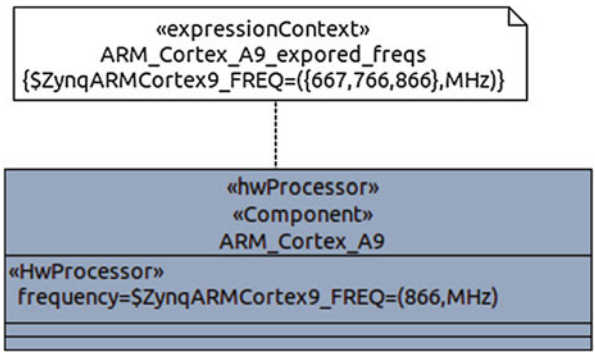

Fig. 5.18 A DSE parameter associated to the ARM Cortex-A9 processor component declaration

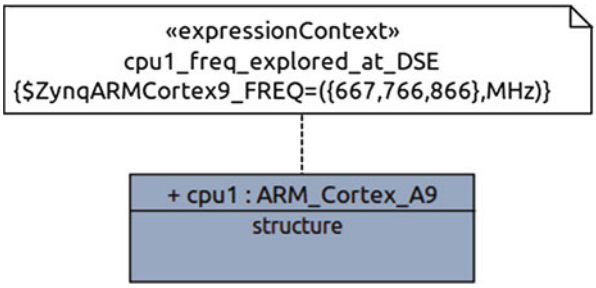

Fig. 5.19 A DSE parameter associated to the ARM Cortex-A9 processor component declaration

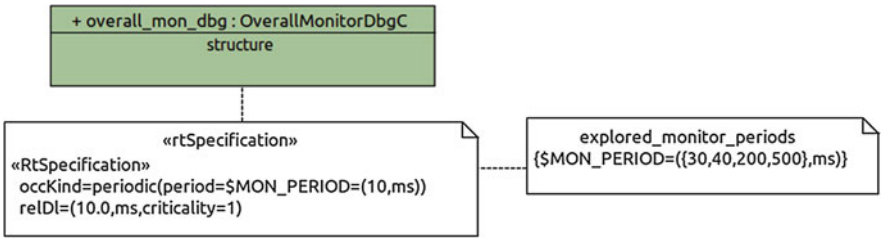

Fig. 5.20 A DSE parameter associated to the period of the monitor functionality in the quadcopter PIM

linked to that processor instance in the suitable view, i.e., in the architectural view in this case, as illustrated in Fig. 5.19.

The DSE parameter is a powerful mechanism since it is applicable to any property of the model, and therefore, it enables to specify the exploration of its impact on the performance of the system. This includes also elements of the platform-independent model. Figure 5.20 shows an example where a DSE parameter is associated to the period of the monitor and debugging component of the quadcopter PIM. In this example, this DSE parameter is useful to explore how a relaxation on its refresh frequency helps in the fulfillment of time performance constraints.

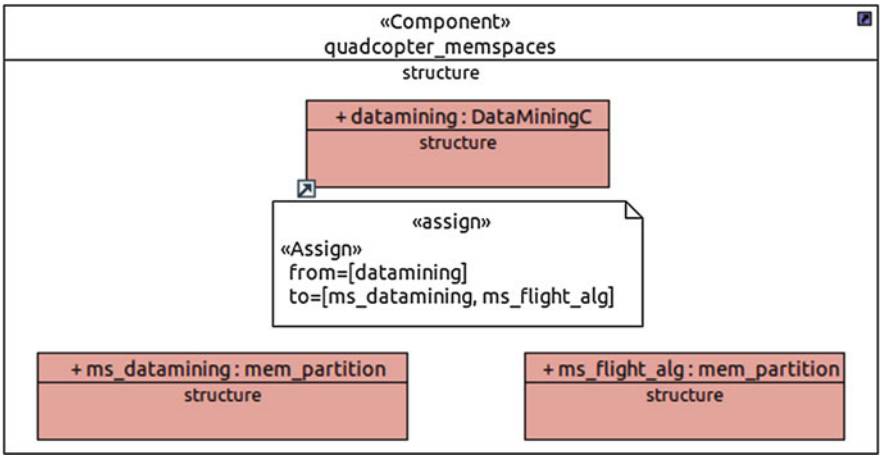

Fig. 5.21 Specification of a mapping design space in the memory space view of the quadcopter

The methodology supports the modeling of configurable mappings and, for each configurable mapping, the expression of which is the set of mappings to be explored. This way, the single-source approach is kept, as the same model serves to express all the mapping alternatives to be explored. A configurable mapping is expressed through a UML comment decorated with the MARTE *«assign»* stereotype. Specifically, its *from* and *to* attributes reflect a range of possible source and destination elements, respectively. Therefore, the *from* and *to* values are DSE parameters. Figure 5.21 shows an example of the quadcopter memory space view which specifies the exploration of two possible mappings of the *datamining* component. The *«assign»* comments can be present in the model together with the *«allocate»* fixed mappings. In a DSE context, the *«assign»* comment overrides the fixed allocation, stating the range of sources and destinations. Since the *«assign»* comment in Fig. 5.21 states nothing about the mapping of the *flight_alg*, *telemetry*, *mission*, and *overall_mon_dbg* component instances, the fixed mapping information stated in Fig. 5.14 is used in a DSE context. In an implementation context, the fixed allocations are used as a statement of the selected implementation.

Figure 5.21 illustrated the specification of a space of mappings in the memory space view (from PIM components to memory spaces). The methodology enables the modeling of configurable mappings in other levels, e.g., for the mapping of memory spaces to platform resources in the architectural view.

Several *«assign»* comments can be present in the same model. The cross product of the mapping spaces defined by each *«assign»* comment states the overall mapping space of the model.

The modeling methodology supports *DSE rules*. DSE rules are constraints embedding Boolean expressions which impose dependencies among the values of the DSE parameters. DSE rules can also refer to the *from* and *to* properties of the *«assign»* comments, and thus to configurable mappings. DSE rules provide an

additional expressiveness to the modeler to customize and prune a potentially huge and redundant design space that can result out of the cross product of all the DSE parameters and configurable mappings.

5.4.8 Modeling for Software Synthesis

The functionality associated to the model is platform independent and has no call to a specific OS API or communication middleware. In the proposed methodology, such type of platform-dependent code is automatically generated by a SW synthesis tool called eSSYN [44]. As well as the associated functionality, eSSYN needs to read PIM information, i.e., retrieved from all PIM views (data, functional and application view). In general, the PIM information read refers to the component partition, to the configuration of static and dynamic concurrency, to the semantics of the communication stated among components and to timing. Taking this information at its input, eSSYN generates all the target-dependent code to implement in SW the specified concurrency, communication, and time semantics. Most of the pieces of target-dependent code are component wrappers, named *wrappers* in short.

The partition into components determines the structure of the generated code and the ambits of visibility. Moreover, the mapping of the application component instances to memory partitions is also read at software synthesis time. A software process is inferred for each memory partition, and all functionality of PIM components mapped to the memory partition will be integrated and in the ambit of the inferred process.

Component attributes stating the concurrency semantics include the *«RtUnit»* attributes *isDynamic* and *main*. The former states if dynamic threads are created for attending the service calls. The latter captures the functionality to be statically triggered, i.e., as an autonomous thread at the beginning of the execution. The communication semantics and the mappings specified by the model have also involvements on the inference of the dynamic concurrency of the system. While some services can consist in a simple procedure call, the mapping to different memory spaces (and thus to different processes) necessarily involves the inference of dynamic threads to service the calls.

By default, a default semantics for the communications is inferred for the port-to-port connectors, unless an explicit semantics is captured as was shown in Sect. 5.4.3. The methodology supports the capture and annotation of additional information which is specifically required and exploited by implementation tasks. Figure 5.22 illustrates how a channel instance can be annotated with information relevant for the synthesis of communications. Specifically, two additional attributes of the *«channel»* stereotype introduced in Sect. 5.4.3 enable to specify the communication stack and specific communication service employed. This specification capabilities were exploited in [37] to show how the different communication alternatives could be rapidly implemented and its impact in performance explored.

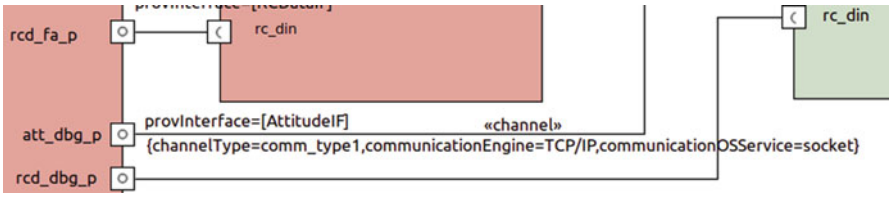

Fig. 5.22 Channel attributes exploited in software synthesis

Specific information can be also captured at lower levels for the SW synthesis phase. It was also mentioned that the *type* attribute of the *«OS»* stereotype is used to identify a specific target OS or RTOS distribution. In addition, the *«OS»* stereotype supports the capture of a set of drivers. This information is used to build up the so-called Board Support Package (BSP). At the HW platform modeling level, the processor Instruction-Set Architecture (ISA) is specified through the MARTE *«HwISA»* stereotype. ISA information is used for the selection of the suitable target at the compilation phase of the synthesis flow.

5.4.9 Verification Environment

The modeling methodology enables the capture of an environment model. This environment model is required by simulation-based performance analysis in order to describe the external stimuli and collect the functional outputs of the system.

The environment model is separately captured in the verification view, i.e., enclosed in a UML package decorated with the *«VerificationView»* stereotype. The environment model has a top component which instances the system components and an arbitrary number of environment components connected to the system component. Figure 5.23 shows the top environment component (*TopEnvQuadcopter*) and the environment components (in gray). The environment model is at a functional level, thus at the same abstraction level as the PIM. Therefore, it has to have client-server ports compatible with the system component, as illustrated in Fig. 5.23. Similarly as for the PIM, functionality is associated to the environment by means of artifacts and path constraints, thus using modeling techniques familiar to the system modeler. The functionality of environment components can invoke libraries and tools which facilitate the environment modeling. In performance model generation context, this functionality is not annotated. In a SW synthesis context, this functionality embeds code dealing with environment, i.e., target-dependent code accessing drivers.

For the identification of the top environment component and of the environment components, the methodology relies on the Universal Testing Profile (UTP), an OMG standard. Figure 5.24 shows the components of the verification view of the quadcopter.

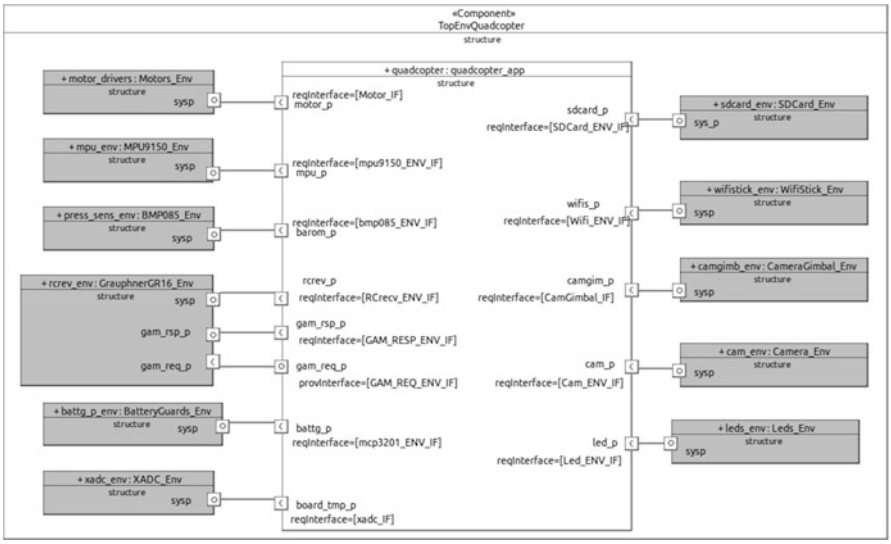

Fig. 5.23 The top environment component of the quadcopter instantiates environment components and connects them to the instance of the system component

5.4.10 Mixed-Criticality

The proposed modeling methodology enables the association of different criticalities to several types of modeling elements, including value annotations of non-functional types, constraints on extra-functional properties, application components, and platform resource components. These techniques rely on two minor extensions of the MARTE profile shown in Fig. 5.25. These extensions were required to support the CONTREX metamodel [25] (▶ Chap. 32, "Metamodeling and Code Generation in the Hardware/Software Interface Domain" introduces metamodeling), capable to cover the MCS modeling. The first extension adds a criticality attribute to a NFP constraint (left-hand side of Fig. 5.25). The criticality attribute is of integer type, which denotes an abstract criticality level. The NFP constraint can be associated then directly to different types of modeling elements, e.g., UML components and UML constraints. Therefore, this extension enables the association of criticalities to components, e.g., application and platform components, and also to constraints, employed to capture performance requirements and contracts. The second extension consists in enabling the annotation of a criticality value on a value annotation (right-hand side of Fig. 5.25). Again, the criticality value is an integer, denoting the criticality level.

These extensions support several modeling scenarios requiring mixed-criticality modeling. Figure 5.26 illustrates the direct association of criticalities to PIM component instances of the quadcopter. The association is performed through a *«NfpConstraint»* with its criticality value annotated in the *criticality* attribute. Please

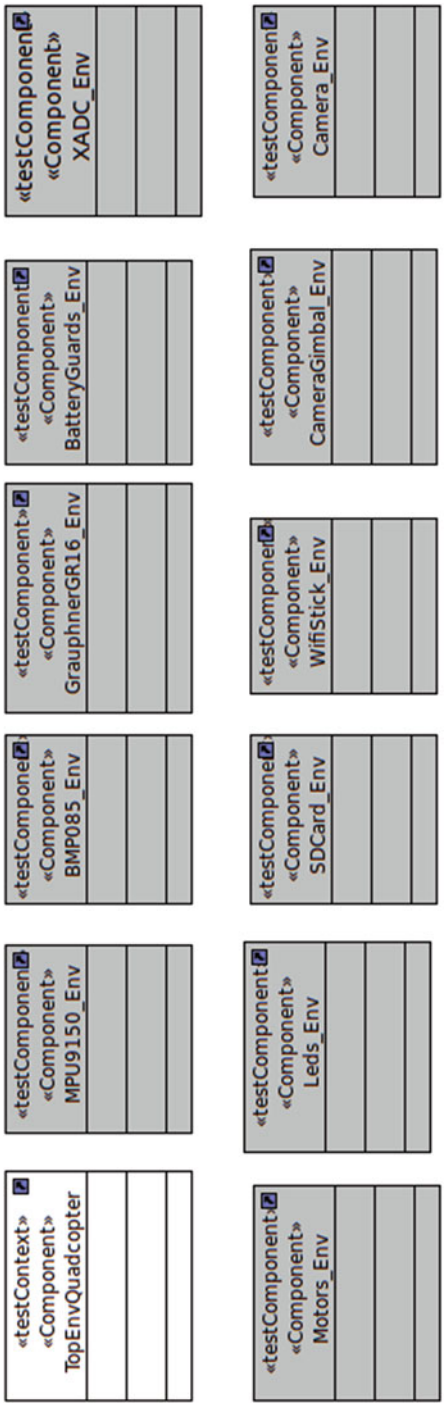

Fig. 5.24 Components of the environment model of the quadcopter

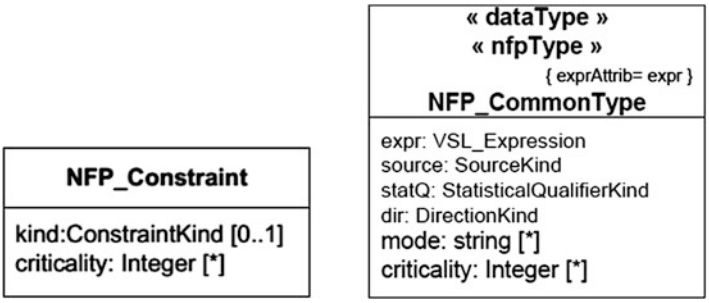

Fig. 5.25 Proposed MARTE extensions for mixed-criticality modeling

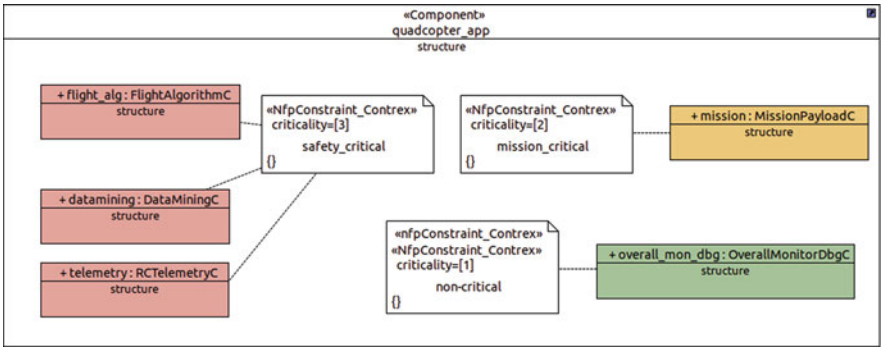

Fig. 5.26 Criticalities directly associated to PIM component instances

observe that Figure 5.26 actually shows the *«NfpConstraint_Contrex»* stereotype. This is provided by the methodology-specific eSSYN profile, and it is used as long as the aforementioned MARTE extension remains as a proposal. Similarly, criticalities can be also directly associated to other application components, e.g., memory spaces, and to software and platform components. Figure 5.27 shows the association of a criticality level to the computational resources of the quadcopter platform. The criticality associated to the application and platform component instances can then be used according to the design context. A main application of this direct association of criticality to PIM and platform components is the application of mapping rules at different levels (from application component to memory spaces, from application level to platform level), oriented to ensure a given degree of separation of resources. Other scenarios covered by this technique is when the development process of the components, either application or platform components, is conditioned by the criticality level, e.g., higher criticality components require more testbenches and more strict coding rules.

The methodology enables the association of criticalities to requirements and specifically to performance requirements. The methodology supports the association of criticalities to extra-functional requirements in several ways. An implicit

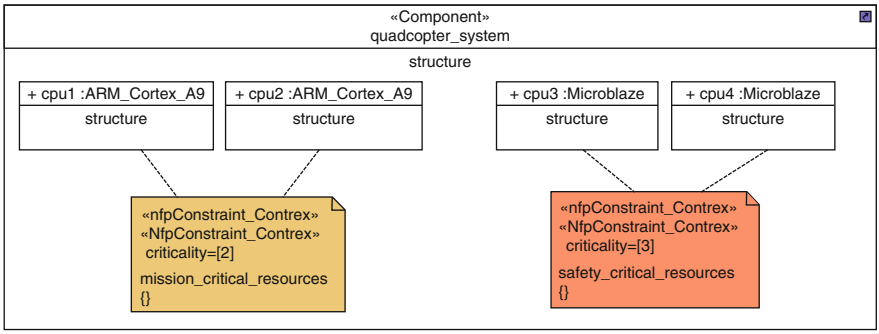

Fig. 5.27 Criticalities directly associated to HW computational elements

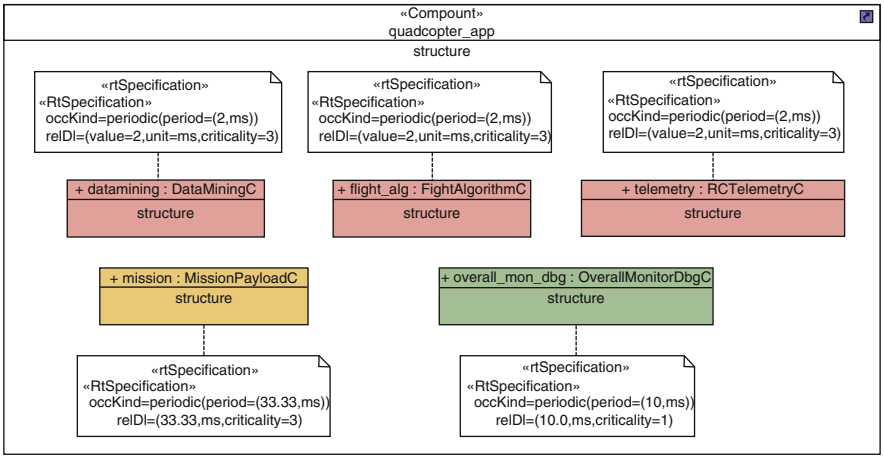

Fig. 5.28 Criticalities associated to deadline constraints

association is supported through the previously shown direct association of criticalities to components. In such a case, the methodology assumes that all the performance constraints associated to a component with an associated criticality inherit such a criticality. Additional techniques for associating criticalities to specific requirements are available in case a component has more than one associated requirement. In the constructs where the requirement on the extra-functional property is captured as a value annotation by means of VSL expression, now it is possible to annotate a criticality associated to the value annotation. Figure 5.28 reflects the association of the deadline requirements on the periodic tasks of the quadcopter and the criticalities annotations on the VSL expression; such a criticality value is associated to each specific deadline requirement.

When the performance requirement is expressed by means of a *«NfpConstraint»*, e.g, as it was the case in Fig. 5.17, the extension of the *«NfpConstraint»* with the criticality attribute can be used. Figure 5.29 shows the extension of the model

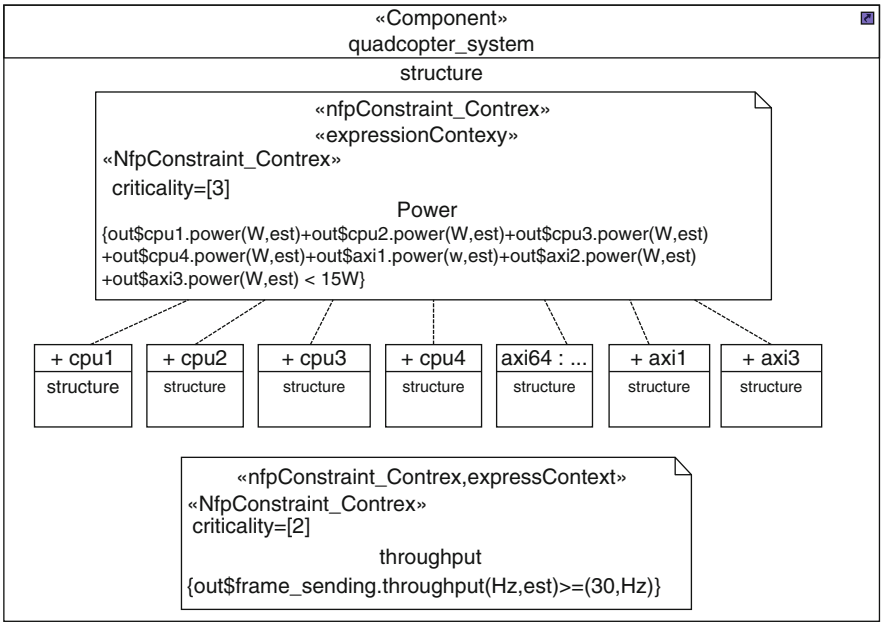

Fig. 5.29 Criticalities associated to system performance constraints

shown in Fig. 5.17, in order to specify the power dissipation constraint as safety critical (criticality = 3). This is because the heating can have side effects on the hardware executing the flight algorithm. The same technique is used to state that the throughput requirement associated to the PIM *mission* component (in charge to capture video, to detect and track an object, and to streaming the recorded action) is mission critical (criticality = 2).

Figure 5.30 shows another modeling scenario requiring mixed-criticality annotation. Schedulability theory on mixed-criticality systems rely on input models where a set of WCETs, instead of a single one, are associated to a task. Each WCET of the WCET set is associated to a criticality level. The proposed methodology covers this modeling need. Figure 5.30 illustrates the annotation of different worst-case execution times to the main application component instance. In the methodology, the implicit semantics associates the WCET to the main functionality of the component. The annotation relies on the *execTime* property of the *«ResourceUsage»* MARTE stereotype. The methodology assumes that a time annotation is associated not only to the application functionality but also to the computational resource which will run the workload. That is, the time annotation reflects the time taken by the execution of the application functionality in isolation conditions. Therefore, the annotation is performed through an association between the application component and the used computational component of the HW resource view. Specifically, such an association is a UML use dependency decorated with the MARTE *«ResourceUsage»* stereotype. The *execTime* property of the *«ResourceUsage»* is typed as a set of

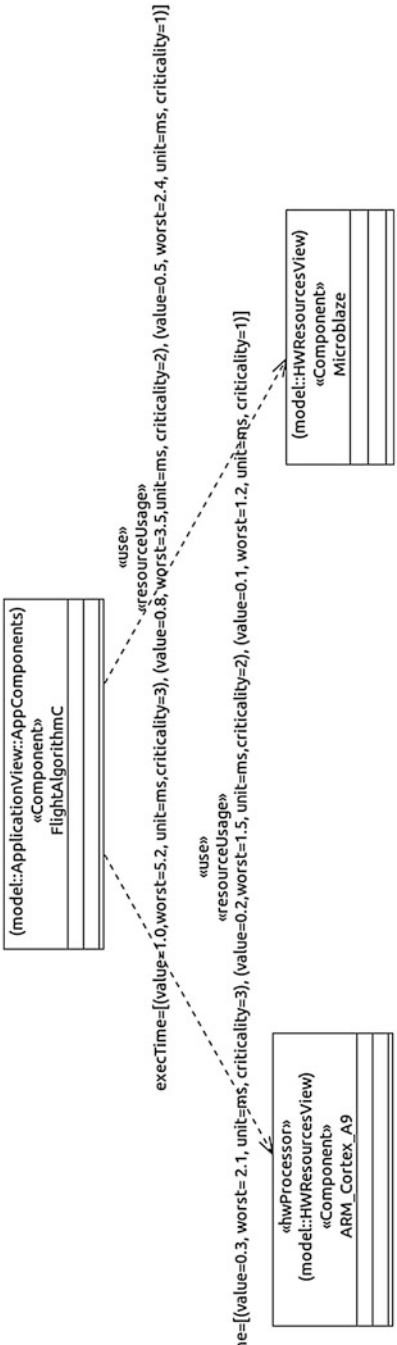

Fig. 5.30 Criticalities associated to different WCET value annotations

values of MARTE NFP_Duration type. Therefore, the *execTime* property can be and is used for annotating several values of execution time.

In addition, the methodology supports several *«ResourceUsage»* associations. This way, a single-source model provides information to trigger several schedulability analyses, e.g., for different mapping alternatives. Specifically, Fig. 5.30 states that if the flight algorithm component is mapped to a *microblaze* processor instance, then the WCETs to be considered are 5.2, 3.5, and 2.4 ms for criticalities 3, 2, and 1, respectively. Notice that this technique can be used only if a single functionality is associated to the component.

5.4.11 Modeling for Schedulability Analysis

This section shows the modeling elements and techniques used to do the validation of timing properties by means of schedulability analysis. The goal is to ensure real-time constraints (hence timing predictability) for critical tasks and evaluate the unused processing capacity that can be used for other non-critical activities in a simple way. This kind of real-time analysis is necessary when the systems are highly complex and have critical time constraints. The modeling for schedulability analysis is integrated in this component-based modeling methodology. The functionality is broken down into the internal and provided functions of communicating components, which in turn are mapped to the processing platform and where the analysis of many real-time situations is possible. A real-time situation corresponds to a workload and a platform. A workload is a set of end-to-end flows and their related stimuli (each end-to-end flow has its associated stimuli). An end-to-end flow describes the causal flow of actions that is triggered by external events and for which a deadline is specified. In such an end-to-end flow or execution path, one or more functions of one or more components can be executed. The objective of the analysis is to ensure that the response time of all the sequences of functions executed on each flow is smaller than the deadline associated.

For each individual function, the annotation of the worst-case and best-case execution times (WCET/BCET) is required. WCETs are needed to get upper bounds for response times and so to verify that the real-time requirements are met. The knowledge of BCETs is useful to reduce jitter and minimize pessimism in upper bounds for distributed systems [31]. The WCET and BCET of a given piece of functionality depends on the type of processing resource executing it. The mode of operation, and specifically, the operation frequency has also direct impact on the execution time of the same code segment. Section 5.4.10 showed how to annotate in the UML model different WCETs for different processor types. The operation modes are captured by means of UML state machines, where the operation modes are represented as UML states specified by the *«Mode»* MARTE stereotype. Transitions are represented as "UML transitions", specified by the *«ModeTransition»* MARTE stereotype.

In general, WCET annotations for independent functions are needed for performing schedulability analysis. However, the schedulability analysis introduced

before requires more information, in particular, the workload, the stimuli patterns that triggers the workload, the platform where it runs, and the hard real-time requirements. Moreover, the modeling methodology shall enable the specification of the ambit and elements of the model involved in the real-time analysis, i.e., the functions and resources of which are really involved out of the overall model.

The schedulability analysis model is contained in a specific view package, stereotyped as *«SchedulabilityView»*. A real-time situation corresponding to an execution model is included in this package and consists in one or more end-to-end flows. The end-to-end flow is modeled as a UML activity decorated with the *«SaEndtoEndFlow»* MARTE stereotype. An activity diagram is used to describe the causal flow as a sequence of *steps*. In general, a *step* represents the usage of resources needed at that point in the flow. Here, it is used to express the execution time taken from the processing resource associated and the messages sizes that are to be sent through network channels. The characteristics of the event that triggers the end-to-end flow are annotated in the initial node of the activity of the end-to-end flow by means of the *«GaWorkLoadEvent»* MARTE stereotype. This stereotype allows to indicate the pattern of activation (e.g., periodic or sporadic) and its parameters (e.g., period). Each step in the flow is modeled in the activity diagram as a UML opaque action annotated with the *«SaStep»* MARTE stereotype. The attribute *execTime* of *«SaStep»* is used to capture the worst and the best execution times of the function involved. The steps may be specified at different granularity levels depending on the knowledge available for the underlying execution elements. At the finest granularity, steps are used on functions with known WCET/BCET. A step can also be used to model operations that invoke other steps. The *concurRes* attribute of a *step* is used to indicate the task it runs on, in the case of a computational step, or the channel through which the message is sent, in the case of a communication step. In the former case, a computation step may also model a sequence of operations executed by the task by means of the *subusage* attribute. Each operation in the subusage sequence is modeled statically by means of UML operations stereotyped with *«SaStep»*. Figure 5.31 shows (on the left-hand side) the modeling of two end-to-end flows for the quadcopter case: one for the data miner task and one for the flight algorithm task. The right-hand side of Fig. 5.31 shows the activity diagram of one of the end-to-end flows, which illustrates the simplest way to describe it.

Schedulability analysis requires to explicitly model the concurrent *tasks* the aforementioned *steps* belong to, i.e., *steps* are mapped to those tasks through the *concurRes* attribute. These tasks are specified through UML properties with the *«SchedulableResource»* MARTE stereotype in the concurrency view. Figure 5.32 shows the modeling of the quadcopter tasks (one for the datamining and another one for the flight algorithm) involved in the schedulability analysis.

In turn, these tasks are associated to the HW/SW platform resources that will execute them by means of the *host* attribute of the *«SchedulableResource»* stereotype. The *«SaExecHost»* and *«SaCommHost»* MARTE stereotypes are employed for indicating the platform resources involved in the schedulability analysis. Specifically, the *«SaExecHost»* stereotype indicates platform component instances where

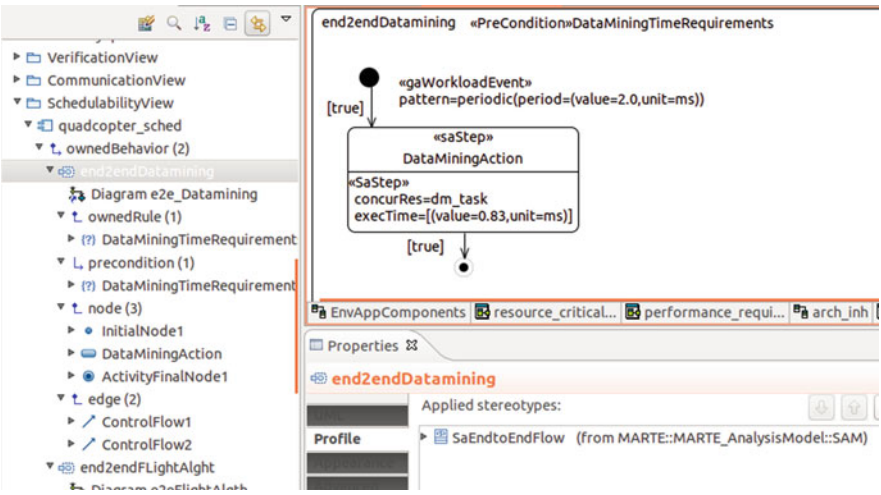

Fig. 5.31 Specification of end-to-end flows in the quadcopter

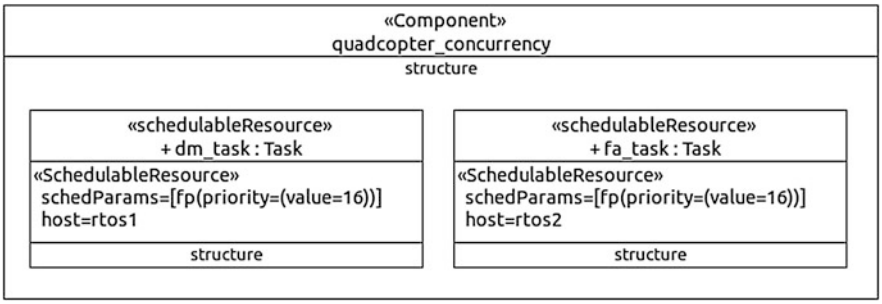

Fig. 5.32 Modeling of quadcopter tasks and their mapping to the SW/HW platform for schedulability analysis

tasks can be scheduled and thus mapped and enables capturing properties such as the range of priorities and context switching times. This way, each *host* attribute of the quadcopter tasks involved in schedulability analysis, shown in Fig. 5.32, can only point to any of the platform resources stereotyped as *«SaExecHost»* in the architectural view of the quadcopter, that is, *rtos1* and *rtos2*, shown in Fig. 5.33.

5.5 Single-Source Design Framework

The single-source modeling methodology introduced in this chapter is documented in detail in [42]. The modeling methodology enables the production of models which serve as an input to a tool infrastructure supporting a single-source design approach. The *CONTREX Eclipse plug-in* (*CONTREP*) [41] is the unified, graphical front-end of that infrastructure. As shown in Fig. 5.34, for the modeling activity,

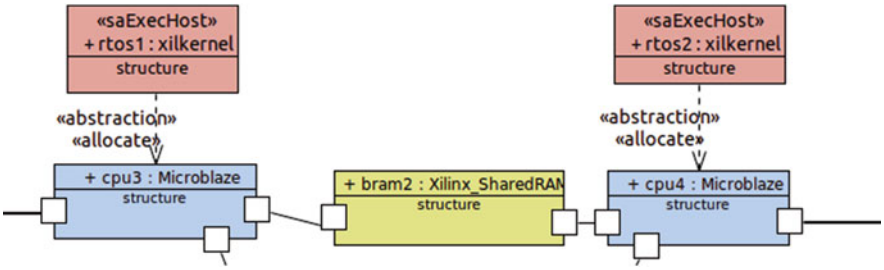

Fig. 5.33 Indication of the execution resources in the SW/HW platform architecture of the quadcopter for schedulability analysis

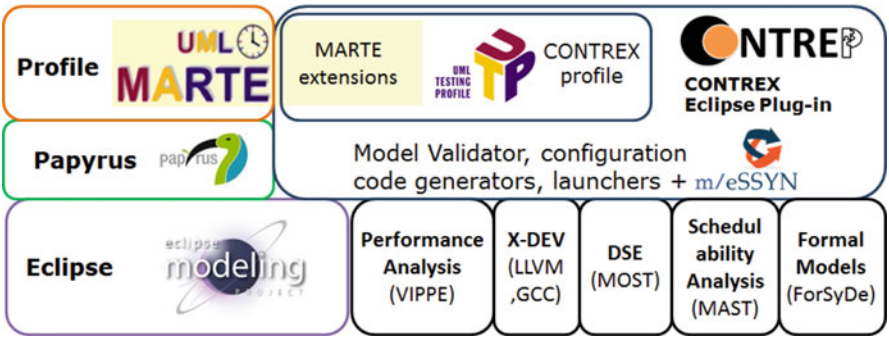

Fig. 5.34 The CONTREX Eclipse plug-in takes the UML/MARTE model as an input for the design activities which rely on different tools

CONTREP provides the eSSYN profile, which provides an implementation of the UTP stereotypes and of a minimum set of methodology-specific extensions employed by the modeling methodology. In addition, CONTREP also integrates a validation tool for the detection of the violation of the modeling rules. As Fig. 5.35 reflects, as well as modeling, CONTREP enables further design activities. For room reasons, it is not possible to illustrate here the application of a complete use case. However, related publications have reported how the UML model served for the generation of a functional model and for SW synthesis [37] relying on eSSYN. For schedulability analysis, the MARTE2MAST generator [10] has been adapted for its integration in CONTREP, and so to enable the automatic generation of the input for the MAST tool [11]. Schedulability analysis requires also tools for obtaining the WCETs and BCETs. In [34], simulation was used to obtain maximum observed times as an early approach. The framework is also capable to generate a ForSyDe executable model [14] (see ▸ Chap. 4, "ForSyDe: System Design Using a Functional Language and Models of Computation") for formal functional validation. CONTREP also automates the generation of a fast performance model. The performance model relies on the VIPPE tool [43], which implements advanced techniques for fast simulation and performance assessment. Simulation is convenient for getting accuracy and considering the dynamism of the application and of the input stimuli

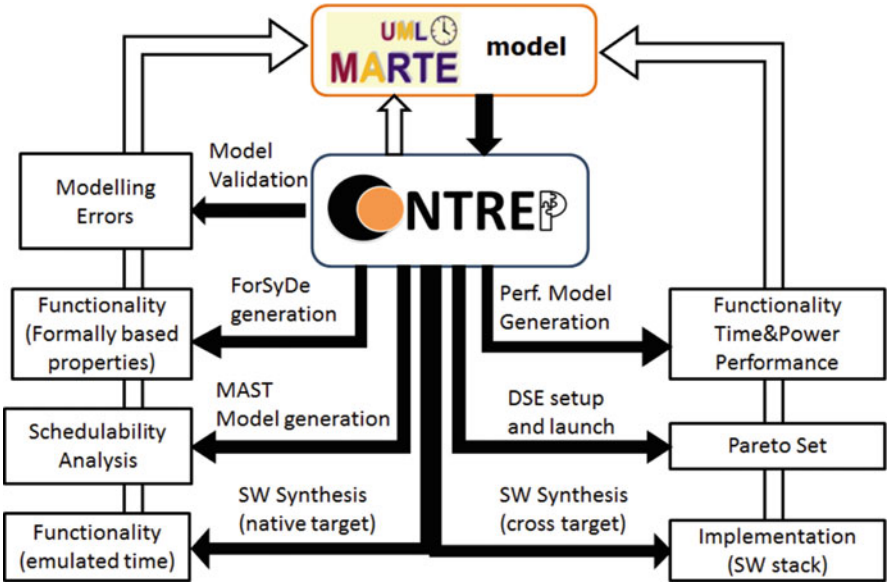

Fig. 5.35 Design activities that can be launched with the CONTREX Eclipse plug-in

of the system environment, that is, for an scenario-aware assessment of the system (▸ Chap. 9, "Scenario-Based Design Space Exploration"). Specifically, VIPPE relies on host-compiled simulation (see ▸ Chap. 19, "Host-Compiled Simulation") and is capable to parallelize the simulation (▸ Chap. 17, "Parallel Simulation" introduces advanced parallelization techniques in a SystemC context) to exploit multi-core host platforms.

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 from the CONTREP front-end, at a high abstraction level, among different exploration strategies (see ▸ Chap. 6, "Optimization Strategies in Design Space Exploration"), exploration tools, and report options. Moreover, the framework enables also the launch of the automated DSE process and the visualization of results from the own graphical environment.

5.6 Conclusions

This chapter has presented the main modeling techniques of a single-source modeling methodology relying on the UML language and the MARTE profile, both OMG standards. These techniques enable the methodology to support separation of

concerns, incremental modeling, and functional component modeling and to feed ESL key design tasks such as design space exploration and software synthesis. All these capabilities are mandatory for the productivity boost required by the current complexities of modern embedded systems.

More information on the introduced methodology can be found in related publications [12–16, 34, 37], in the methodology related website [44], and in the CONTREX website [28].

The presented methodology maximizes the exploitation of UML and specifically of the MARTE and UTP profiles for complex embedded system modeling. This does not necessarily mean that these profiles are currently capable to cover all the required concepts. In most of the cases, the methodology covers the lacks by proposing minimal extensions to MARTE, for instance, for the annotation of criticalities, for the description of a design space in VSL, or for the capture of the system views. In other cases, more important extensions are required. This is the case of embedded distributed systems description, for which CONTREX has made also a proposal [9, 17].

Acknowledgments 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. We are thankful to the OFFIS team in CONTREX for their support and for all the documentation and material on their quadcopter implementation. This includes the quadcopter picture of the chapter.

References

1. Alam O, Kienzle J (2013) Incremental software design modelling. In: Proceedings of the 2013 conference of the center for advanced studies on collaborative research, CASCON '13. IBM Corp., Riverton, pp 325–339
2. Ambler SW (2015) Single source information: an Agile best practice for effective documentation. http://agilemodeling.com/essays/singleSourceInformation.htm
3. Arpinen T, Salminen E, Hämäläinen TD, Hännikäinen M (2012) {MARTE} profile extension for modeling dynamic power management of embedded systems. J Syst Archit 58(5):209–219. doi:10.1016/j.sysarc.2011.01.003. Model Based Engineering for Embedded Systems Design
4. Bailey B, Martin G, Piziali A (2007) ESL design and verification: a prescription for electronic system level methodology. Morgan Kaufmann/Elsevier, Amsterdam/Boston
5. Bakshi A, Prasanna VK, Ledeczi A (2001) Milan: a model based integrated simulation framework for design of embedded systems. In: Proceedings of the 2001 ACM SIGPLAN workshop on optimization of middleware and distributed systems, OM '01. ACM, New York, pp 82–93. doi:10.1145/384198.384210
6. Burns A, Davis R (2015) Mixed-criticality systems: a review, 6th edn. Technical report, Department of Computer Science, University of York
7. Cabot J (2014) Single-source modeling for embedded systems with UML/MARTE. http://modeling-languages.com/modeling-embedded-systems-uml-marte
8. Dekeyser J, Gamatie A, Atitallah R, Boulet P (2008) Using the UML profile for MARTE to MPSoC co-design. In: 1st international conference on embedded systems and critical applications (ICESCA'08)
9. Ebeid E, Medina J, Quaglia D, Fummi F (2015) Extensions to the UML profile for MARTE for distributed embedded systems. In: 2015 forum on specification and design languages (FDL), pp 1–8

10. Garcia A, Medina J: MARTE2MAST. http://mast.unican.es/umlmast/marte2mast/
11. Gonzalez M, Gutierrez JJ, Palencia JC, Drake JM (2001) Mast: modeling and analysis suite for real time applications. In: 2001 13th Euromicro conference on real-time systems, pp 125–134. doi:10.1109/EMRTS.2001.934015
12. Herrera F, Penil P, Posadas H, Villar E (2014) Model-driven methodology for the development of multi-level executable environments. In: Haase J (ed) Models, methods, and tools for complex chip design. Lecture notes in electrical engineering, vol 265. Springer International Publishing, pp 145–164. doi:10.1007/978-3-319-01418-0_9
13. Herrera F, Posadas H, Peñil P, Villar E, Ferrero F, Valencia R, Palermo G (2014) The COMPLEX methodology for UML/MARTE modeling and design space exploration of embedded systems. J Syst Archit 60(1):55–78. doi:10.1016/j.sysarc.2013.10.003
14. Herrera F, Peñil P, Villar E (2015) Enhancing analyzability and time predictability in UML/MARTE component-based application models. In: Proceedings of the 18th international workshop on software and compilers for embedded systems, FDL '15. IEEE
15. Herrera F, Peñil P, Villar E (2015) A model-based, single-source approach to design-space exploration and synthesis of mixed-criticality systems. In: Proceedings of the 18th international workshop on software and compilers for embedded systems, SCOPES '15. ACM, New York, pp 88–91. doi:10.1145/2764967.2784777
16. Herrera F, Peñil P, Villar E (2015) UML/MARTE modelling for design space exploration of mixed-criticality systems on top of predictable platforms. In: Jornadas de Computación Empotrada (JCE'15)
17. Herrera F, Peñil P, Villar E (2015) Extension of the UML/MARTE network modelling methodology in CONTREX. Technical report, University of Cantabria. http://umlmarte.teisa.unican.es/wp-content/uploads/2016/05/ExtendedUML_MARTE_Network_Modelling_Methodology.pdf
18. Intel (2014) Intel cofluent methodology for SysML: UML*SysML*MARTE flow for Intel CoFluent studio. http://www.intel.com/content/www/us/en/cofluent/cofluent-methodology-for-sysml-white-paper.html
19. ITRS: International roadmap of semiconductors. http://www.itrs.net/
20. Kang E, Jackson E, Schulte W (2011) An approach for effective design space exploration. In: Calinescu R, Jackson E (eds) Foundations of computer software. Modeling, development, and verification of adaptive systems. Lecture notes in computer science, vol 6662. Springer, Berlin/Heidelberg, pp 33–54. doi:10.1007/978-3-642-21292-5_3
21. Kangas T, Kukkala P, Orsila H, Salminen E, Hännikäinen M, Hämäläinen TD, Riihimäki J, Kuusilinna K (2006) UML-based multiprocessor soc design framework. ACM Trans Embed Comput Syst 5(2):281–320. doi:10.1145/1151074.1151077
22. Kruchten P (1995) Architecture blueprints—the "4+1" view model of software architecture. In: Tutorial proceedings on TRI-Ada '91: Ada's role in global markets: solutions for a changing complex world, TRI-Ada '95. ACM, New York, pp 540–555. doi:10.1145/216591.216611
23. Lemke M (2012) Mixed criticality systems. Report from the workshop on mixed criticality systems. Technical report, Information Society and Media Directorate-General
24. Liehr AE (2009) Languages for embedded systems and their applications. Lecture notes in electrical engineering, vol 36. Springer Netherlands, pp 43–56. doi:10.1007/978-1-4020-9714-0_3
25. Medina J et al (2015) CONTREX system metamodel. Technical report. https://contrex.offis.de/home/images/publicdeliverables/Deliverable%20D2.1.1%20v1.0.pdf
26. Mura M, Murillo L, Prevostini M (2008) Model-based design space exploration for RTES with SysML and MARTE. In: Forum on specification, verification and design languages, FDL 2008, pp 203–208. doi:10.1109/FDL.2008.4641446
27. Nicolescu G, Mosterman P (2009) Model-based design for embedded systems. Computational analysis, synthesis, and design of dynamic systems. CRC Press, Boca Raton
28. OFFIS (2015) CONTREX FP7 project website. https://contrex.offis.de/home/
29. OMG (2011) OMG UML profile for MARTE, modelling and analysis of real-time embedded systems, Version 1.1. Available at www.omg.org

30. OMG (2015) OMG unified modeling language. Available at www.omg.org
31. Palencia JC, Gutierrez JJ, Gonzalez Harbour M (1998) Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems. In: Proceedings of the 10th Euromicro workshop on real-time systems, pp 35–44. doi:10.1109/EMWRTS.1998.684945
32. Panunzio M, Vardanega T (2009) On component-based development and high-integrity real-time systems. In: IEEE 19th international conference on embedded and real-time computing systems and applications
33. Peñil P, Posadas H, Nicolás A, Villar E (2012) Automatic synthesis from UML/MARTE models using channel semantics. In: Proceedings of the 5th international workshop on model based architecting and construction of embedded systems, ACES-MB '12. ACM, New York, pp 49–54. doi:10.1145/2432631.2432640
34. Peñil P, Posadas H, Medina J, Villar E (2015) UML-based single-source approach for evaluation and optimization of mixed-critical embedded systems. In: DCIS'15
35. Piel E, Atitallah RB, Marquet P, Meftali S, Niar S, Etien A, Dekeyser J, Boulet P (2008) Gaspard2: from MARTE to SystemC simulation. In: Design, automation and test in Europe (DATE 08)
36. Pop A, Akhvlediani D, Fritzson P (2007) Integrated UML and modelica system modeling with modelicaml in Eclipse. In: Proceedings of the 11th IASTED international conference on software engineering and applications, SEA '07. ACTA Press, Anaheim, pp 557–563
37. Posadas H, Peñil P, Nicolás A, Villar E (2014) Automatic synthesis of embedded {SW} for evaluating physical implementation alternatives from UML/MARTE models supporting memory space separation. Microelectron J 45(10):1281–1291. doi:10.1016/j.mejo.2013.11.003. DCIS'12 Special Issue
38. Szyperski C (2002) Component software: beyond object-oriented programming. Addison Wesley, London
39. TILLOO R (2015) What is incremental model in software engineering? It's advantages and disadvantages. Available in http://www.technotrice.com/incremental-model-in-software-engineering
40. Truyen F (2006) The fast guide to model driven architecture – the basics of model driven architecture. http://www.omg.org/mda/mda_files/Cephas_MDA_Fast_Guide.pdf
41. University of Cantabria: CONTREX Eclipse plug-in website. http://contrep.teisa.unican.es. Accessed 04 Oct 2016
42. University of Cantabria: UML/MARTE single-source methodology website. http://umlmarte.teisa.unican.es. Accessed: 04 Oct 2016
43. University of Cantabria: VIPPE website. http://vippe.teisa.unican.es. Accessed 04 Oct 2016
44. University of Cantabria (2016) eSSYN website. http://eSSYN.com
45. Vidal J, de Lamotte F, Gogniat G, Soulard P, Diguet JP (2009) A co-design approach for embedded system modeling and code generation with UML and MARTE. In: Design, automation test in Europe conference exhibition. DATE '09, pp 226–231. doi:10.1109/DATE.2009.5090662
46. Woods E, Rozanski N (2005) Using architectural perspectives. In: 2014 IEEE/IFIP conference on software architecture, vol 0, pp 25–35. doi:10.1109/WICSA.2005.74