eDIANA
Embedded Systems for Energy Efficient Buildings

Grant agreement no.: *100012*

D2.1-A Model Driven Engineering methodology for architecture realisation

Author(s): **Claudio Parrella** **ST**
Contributor(s): **ESI**
MU
ED
UNIBO
ATOS
LABEIN

Issue Date	29 May 2009 (m04)
Deliverable Number	D2.1-A
WP Number	WP2: Design of architecture and middleware for effective system composability
Status	Delivered

Dissemination level	
X	**PU** = Public
	PP = Restricted to other programme participants (including the JU)
	RE = Restricted to a group specified by the consortium (including the JU)
	CO = Confidential, only for members of the consortium (including the JU)

Document history			
V	***Date***	***Author***	***Description***
0	*2009-03-10*	*ESI*	*ToC*
0.1	*2009-04-03*	*ESI*	*ToC*
0.2	*2009-05-08*	*ESI*	*First version of sections 2, 3, 5 & 6*
0.3	*2009-05-15*	*ST* *ATOS* *MU*	*First version of Introduction and Conclusion; updated tables 3.2.3 and 6.3* *First version of section 4.1; updated table 6.1;* *First version of section 5.7; updated tables 6.1, 6.2, 6.3;*
0.4	*2009-05.20*	*ATOS*	*Modified Sect.4.1.3.7*
0.5	*2009-05-24*	*ED*	*Moved section 4.1 in 4.2 and added a new version of the section 4.1*
0.6	*2009-05-28*	*MU*	*Eliminate comments and add new text related to a new phase in the process and some other changes*
0.7	*2009-05-29*	*ESI*	*Updated eDIANA process figure (Figure 3-1) according to the partners' comments.*
0.8	*2009-05-29*	*ATOS*	*Moved section 4.1.3.x embedded into a table in section 6.1*
0.9	*2009-05-29*	*MU*	*Table format changed in section 5.7*
1.0	*2009-05-29*	*ST*	*Final version, comments removed*

Disclaimer

Summary

The "Model Driven Engineering methodology for architecture realisation" is a public document delivered in the context of WP2, Task 2.1. Model Driven Architecture for Components Engineering with regard to methods and techniques that will be adopted and instantiated to the domain of eDIANA in order to ensure composability during system realisation

This document is about the methodology that will support eDIANA platform developers to design, develop and deploy them to scenarios defined in the context of the project.

Contents

Abbreviations

eDIANA	Embedded Systems for Energy Efficient Buildings
MDE	Model-Driven Engineering
GENESYS	GEneric Embedded SYStems platform
LIF	Linking InterFace
PIM	Platform Independent Model
PSM	Platform Specific Model
V&V	Verification and Validation

1. Introduction

The target of this document is to define a Model Driven Engineering practices and Architecture Variability techniques to be applied to the eDIANA Reference Architecture development.

Model Driven Architecture and Engineering methods and techniques will be adopted and instantiated to the domain of eDIANA in order to ensure composability during system realisation. The design and the realisation of the software and hardware elements within the eDIANA architecture will follow model driven and component based design methods and tools, allowing the design and development of collaborating elements in a concurrent fashion, empowering contract-based engineering and reasoning, allowing the replacement, suppression, or inhibition of platform elements while ensuring correct behaviour (functional and non-functional), in line with the robustness and diagnosis challenges for the envisioned eDIANA architecture.

The target of this document is to describe how the methodology proposed in the GENESYS [2] European project could be applied to the eDIANA platform. The current maturity of GENESYS methodology is not fully applicable for the design of complex platform as eDIANA, where different devices, protocols and multi-processor System on Chip are integrated. When applicable, in the next steps of the eDIANA project, this methodology will be the guideline for the hardware and software design. While the state of art will be apply for overcoming the specific methodology limitation.

2. Principles of the eDIANA Methodology

This section will go over the main principles of the eDIANA Model-Driven Engineering (MDE) methodology. The methodology will support eDIANA application developers to design, develop and deploy them to scenarios defined in the context of the project. Therefore, the eDIANA methodology must be tightly coupled with the definitions of the eDIANA architecture, with the devices that will be used in it and also with the scenarios in which the applications will be deployed.

The eDIANA platform is deployed on top of a set of embedded devices which will control energy consumption in apartments and buildings. Therefore, the development targets of the platform are twofold:

- On the one hand, the concrete embedded devices that will be used for establishing the eDIANA environment must be developed.
- On the other hand, the eDIANA Cell and MacroCell applications, which will be built on top of these devices, must be specified, designed and deployed.

Despite the fact that these two targets focus on different integration levels, it would be highly desirable that the two of them were integrated in the same architectural paradigm. Other research projects have also addressed the industrial need of a cross domain multi-level embedded system architecture. The GENESYS project [2] is an example of such an initiative. In this section we will also discuss the similarities between the eDIANA platform needs and the generic platform proposed by GENESYS in order to locate the possible synergies and profit from the results of that project.

2.1 The Principles of the eDIANA Platform

As it has already been stated previously, this document presents an MDE methodology to be applied for the development and deployment of the eDIANA embedded devices and applications (i.e. Cells and MacroCells). It is, therefore, a requirement that the modelling methodology and tools are well aware of the characteristics of the eDIANA platform, as well as aware of its devices and communication needs.

Due to the fact that neither the eDIANA platform nor the eDIANA devices are defined at this phase of the project, the eDIANA methodology will be defined on top of the principles defined for them. These principles are available in the document "eDIANA scenarios descriptions".

According to this document, the eDIANA platform will be built on top of six principles:

1. **Strict component orientation**. Components will be the smaller pieces of the platform, and their functionality will not vary from one scenario to another.

2. **Two level organization**. The eDIANA platform will be organized in two different levels, the MacroCell at the top control level and the Cell at the lower control level. The cardinality between MacroCells and Cells will be of 1-N.

3. **Connection hierarchy**. The devices of the platform will be connected to their local Cell, which will control the devices connected to it. The MacroCell will be connected to the Cells and will be in charge of taking decisions from the whole building or even wider perspective.

4. **Control hierarchy**. The eDIANA platform control for energy efficiency is divided into two levels. At the top level, the MacroCell owns the most sophisticated system wide control algorithms and it sends "not binding" commands to the Cells. The Cells, at the lower level, have control over the devices connected to them (e.g. plug&play services, discovery, activate/deactivate devices, etc.), being this control based on their own perspective and on the general policy and criteria set up by the Macro Cell .

5. **Platform based on logical components**. The devices involved in an eDIANA application are defined by their logical behaviour and functionalities provided by their services. Such a definition enables different implementations to easily interact with each other. The independence between the logical behaviour provided by a component and its implementation is known as technology agnosticism.

6. **Limited number of components**. The eDIANA platform will identify a number of logical components that will be used in eDIANA applications. Despite the fact that this component list can be extended in the future, a few number of component categories will be defined, both at MacroCell and Cell levels.

From the point of view of the eDIANA developers, the design and development methodology should support these principles providing means of checking the compliance of the developed devices with the eDIANA platform principles and requirements.

Regarding the two development paradigms present within eDIANA, the latter principles affect differently the development processes of eDIANA devices and complete eDIANA applications. The following table illustrates these differences.

Development target	Applicable eDIANA platform principles
eDIANA devices	5, 6
eDIANA applications	1, 2, 3, 4

Table 1. eDIANA platform principles and development targets

2.2 Relation with GENESYS

The GENESYS project is an FP7-STREP research project focusing on the development of a cross-domain multi-level architecture for embedded systems and embedded system based applications. The main outcomes of the project are:

GENESYS reference architecture template. This template is composed of a series of core and optional services that will be used by embedded systems and applications designers as support functionalities provided by the architecture by construction. Moreover, the template is defined at different integration levels, namely: Chip level (L1), Device Level (L2) and System level (L3). The architecture is a template, since it has to be instantiated for each application domain (i.e. the optional services have to be selected and, optional and core services implemented) according to its peculiarities.

GENESYS methodology framework. The GENESYS methodology framework consists of a set of model-driven methods, languages, transformations and tools that support designers and developers of GENESYS compliant systems throughout the whole development cycle. Similarly to the architecture template, the concrete methods, languages, transformations and tools used for an instantiation of the architecture may vary from the ones used in another instantiation.

From the point of view of the outcomes, the eDIANA platform can be seen as an instantiation of the GENESYS architecture at both Device Level (L2) for the development of the eDIANA devices, and System Level (L3) for the development of control algorithms and configuration of operations and constraints of MacroCells and Cells in the eDIANA scenarios.

Regarding the principles, GENESYS envisages a large set of principles applicable to the reference architecture template and methodology. Among those, the most important ones are [3]:

Strict Component Orientation. The systems developed under the scope of GENESYS should be component based. The type of components used in the architecture varies from an integration level to another, namely IP-cores at level L1, chips at level L2 and devices at level L3. Moreover, in other to ease the composition of GENESYS systems, each component must provide a Linking InterFace (LIF) to enable interactions with it.

Multicast Unidirectional Communications. In order to provide error containment by construction, the GENESYS architecture foresees a multicast unidirectional message-based communications environment in GENESYS compliant applications.

Hierarchy of Services. The GENESYS reference architecture is based on a hierarchy of services that will be supported by the architecture by construction, therefore easing the designers' task and increasing the portability of the applications.

Openness. The components of a GENESYS application are defined via their functionality description (i.e. semantics & behaviour) and their LIFs (i.e. syntax). Therefore, components are defined in the logical level leaving implementation details to component developers. This enables the easy integration of third party components into GENESYS applications.

Multiple integration levels. As we have already explained the GENESYS architecture is intended to be applicable at a wide range of integration levels, from Chip Level (L1) developments, to Device Level (L2) and System Level (L3) developments; what leads to full integration of the development process of embedded systems too.

It is important to note that the GENESYS architectural principles have been taken from the ARTEMIS Strategic Research Agenda (SRA) [1].

- By comparing the GENESYS architectural principles and the eDIANA platform principles it is immediate to discover similarities between the two sets. Indeed:
- Both GENESYS and eDIANA foresee a component based architecture for their applications.
- Both GENESYS and eDIANA require the logical definition of the components to enable technology agnosticism and enable the integration of third party components into the applications under design.

The eDIANA platform can be seen as an instantiation of the GENESYS reference architecture template.

Taking into accounts all the similarities between GENESYS and eDIANA, we are going to develop the eDIANA methodology taking GENESYS [4] as starting point. This methodology will be further described in the following sections.

3. The Process Model

This section will describe the eDIANA design and development process model, its main phases, artifacts, inputs and outputs. This model will provide the structure on top of which we will define the MDE methodology of eDIANA platform applications.

The process model will also give eDIANA developers a set of guidelines and best practices intended to provide eDIANA devices and applications of higher quality figures.

3.1 Overview of the Process Model

Figure 3-1 represents the main phases of the eDIANA design and development process based on the eDIANA platform principles and the GENESYS methodology.

System engineering starts with the requirements specification phase, which will be based on the target eDIANA scenario and results in the definition of the functional and non-functional properties, quality requirements and constraints of a system, regarding both application and platform components. Moreover, the V&V evaluation criteria take also in account quality requirements, prioritized according to the scope and importance of the requirements.

The application architecture design phase takes as input the application requirements defined in the requirements specification phase. Moreover, application designer may reuse existing eDIANA application components taken from the application components repository. Application architecture design phase results in a platform independent model (PIM) of the application architecture which contains not only the structure of the application, but also the behaviour of the application components involved and a set of non-functional constraints and characteristics applicable to the components and their LIFs [linking interface].

Complete platform architecture design is done by instantiating a set of existing platform components. The components used may vary from a design to another depending on the integration level of the system under design; moreover, these components may be either hardware or software components (i.e. CPUs, IP-cores, chips, operating system, middleware, etc.). The complete platform components repository contains a set of reusable platform components. It is foreseen that eDIANA devices will be introduced into this repository once they have been created.

Platform components provide a set of services to the application components above them. These services must also be specified in the platform component descriptions. If a particular service is missing from the platform components repository and a new component is created, its interaction with the upper layers must be specified at this

level. The platform architecture design phase outputs a platform model created out of instances of the platform components at a specific integration level that is further used as a system-platform model upon which the application PIM is allocated to.

The system allocation phase maps the components in the application architecture model onto the components of the platform model resulting in the complete system architecture model. System models consist of a set of complementary views, namely structure, behaviour, policies and allocation. These models contain the design information of the whole embedded application, which is required for the next phase: early V&V. In the system architecture design phase, the platform architecture is configured for the use of a specific platform. In fact, in this phase the whole system architecture is the first time described as a whole, and therefore, several refinements are typically needed. These refinements may be required before and after performing the early V&V evaluation phase. Architecture modelling and evaluation is a highly iterative and incremental process, and what steps need to be performed depends on the improvements defined as the results of the V&V evaluation.

Depending on the evaluation methods used, specific models may be needed for quality evaluation purposes. These specific models can be derived from the system model via model transformations created for that purpose.

The early V&V process is iterative. It starts from the V&V requirements of the highest priority and goes down to the properties of lower priority. Each quality property is evaluated separately, and thereafter the tradeoffs analysis is conducted. If conflicts are encountered, a new iteration is to be taken (i.e. System Allocation and Early V&V phases). When all the requirements are met the system model becomes the validated design model, which will be taken as input for the realization of the system. Realization includes a set of refinement and testing phases, which are not discussed here. Thereafter, any new application components designed for the current application can be included to the application components repository as a new reusable service and, similarly, the validated platform components are also included into the repository of platform models.

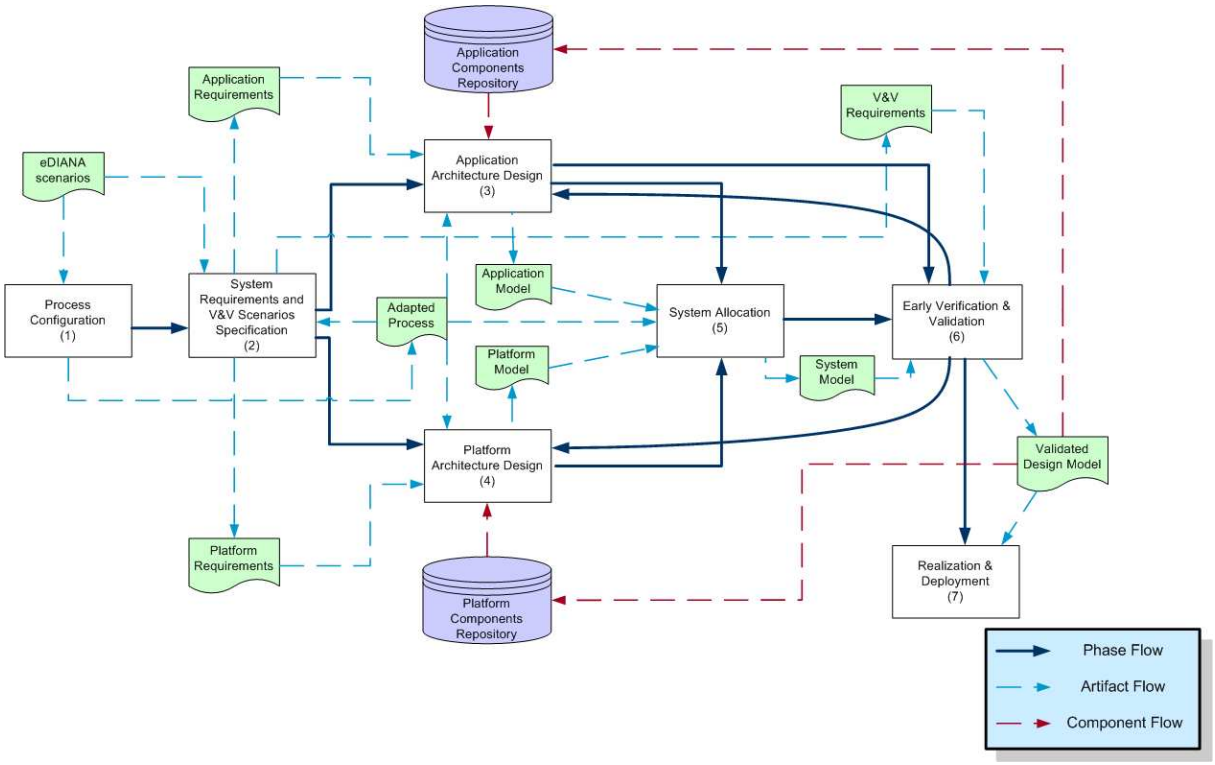

Figure 3-1. The eDIANA development process model

3.2 eDIANA Process Phases

In this section we will describe in detail the process model phases (1-6) defined in Figure 3-1, as well as their inputs, outputs, triggers and existing tool support.

3.2.1 Process Configuration phase

	Process Configuration
Description	The eDiana process is configured in this phase to adapt it to the new system's characteristics. The models and methods that will be used during the process will be selected in this phase.
Start conditions	The decision to develop a new eDIANA product or system.
Triggers	V&V Scenarios

Inputs	Usage of models, system characteristics and modelling objective
Outputs	Adapted Process: Selected set of models, methods and tools
Specification method and language	Textual language and process specification
Tool support	-

3.2.2 System Requirements and V&V Scenarios Specification phase

	System Requirements and V&V scenarios Specification
Description	Two steps or subphases are distinguished in this phase: The System Requirements Specification phase will produce the requirements documents for the development of applications and platforms of the eDIANA systems. And the V&V scenarios specification will specify the evaluation requirements to be applied in the V&V phase (5). Among other documents, this phase will always take as an input the concrete eDIANA scenario(s) in which the designed product is to be applied.
Start conditions	The decision to develop a new eDIANA product or system. And process configured.
Triggers	Application Architecture Design phase and Platform Architecture Design phase
Inputs	Targeted eDIANA scenario(s). Customer requirements, market forecasts, standards, product poRtFeatureolio.
Outputs	Application requirements, platform requirements and requirements for early V&V evaluation.
Specification method and language	SysML + MARTE Non-Functional Properties description.
Tool support	Papyrus, Rational Software Architect.

3.2.3 Application Architecture Design phase

	Application Architecture Design

Description	The goal of this phase is to obtain a PIM of the application that will be designed. The phase will use both existing and new models to obtain an application model that meets the requirements described in the application requirements document provided as input for this phase. It is important to state that in order for the models to be fully eDIANA compatible, the models used and generated during this stage must follow the eDIANA modelling style, described in section 5.5.
Start conditions	This phase will start whenever the application requirements are available. Once this phase has ended for the first time, it will restart if the Early V&V phase detects a quality error in the application model.
Triggers	System Allocation phase. And early V&V phase in some cases.
Inputs	Application requirements document. Application components repository.
Outputs	Application Model (PIM).
Specification method and language	UML2 + MARTE (GCM, HLAM and NPFs subprofiles).
Tool support	Papyrus, Rational Software Architect.

3.2.4 Platform Architecture Design phase

	Platform Architecture Design
Description	The goal of this phase is to obtain first an abstract model of the platform architecture that supports the execution of the embedded application. The phase will use both existing models and new models to obtain a platform model that meets the requirements described in the platform requirements document provided as input for this phase. In order for the models to be fully eDIANA compatible, the models used and generated during this stage must follow the eDIANA modelling style described in section 5.4.
Start conditions	This phase will start whenever the platform requirements are available. Once this phase has ended for the first time, it will restart if the Early V&V phase detects a quality error in the platform model.
Triggers	System Allocation phase.
Inputs	Platform requirements document

	Platform components repository.
Outputs	Platform model of the system under development.
Specification method and language	UML2 + MARTE (GRM, HRM, SRM and NFPs subprofiles) SystemC / pseudo code/Verilog/VHDL
Tool support	Papyrus, Rational Software Architect. CADENCE IUS

3.2.5 System Allocation phase

	System Allocation
Description	The goal of this phase is to map the application model obtained from the Application Architecture Design phase onto the platform model obtained from the Platform Architecture Design phase. As a result, a full system architecture model will be obtained.
Start conditions	This phase will start for the first time when both the Application Architecture Design and the Platform Architecture Design phases have finished. After the first execution, this phase is executed once again if the any of these phases is executed again.
Triggers	Early V&V phase.
Inputs	Application model. Platform model.
Outputs	Plaform Specific (PSM) System model.
Specification method and language	MARTE Alloc + all other languages used in previous phases.
Tool support	Papyrus, Rational Software Architect, MOFScript, Open ArchitectureWare (OAW), ATL.

3.2.6 Early Verification & Validation phase

	Early Verification & Validation
Description	In this phase the system architecture model obtained from the System Allocation phase is evaluated against the V&V requirements defined in the requirements specification phase. The V&V evaluation

	results may lead to a redesign of the application model, the platform model or both. The resulting model will contain a validated system design. Verification & Validation can also be performed in an early phase before system Allocation, after phase.
Start conditions	System Allocation phase is completed or in some cases where phase is completed.
Triggers	Upon completion, this phase may trigger different phases depending on the results: - If one or more defects are detected the System model regarding the system application, the Application Architecture Design phase is triggered. - If one or more defects are detected the System model regarding the system platform, the Platform Architecture Design phase is triggered. - If no defects are detected the Realization & Deployment phase is triggered and the validated model is produced. Note that as a result of the V&V tests more than one phase can be triggered.
Inputs	V&V requirements System model
Outputs	Validated System model.
Specification method and language	Different languages are used depending on the V&V methods selected for the application.
Tool support	Specific analysis and V&V tools. Model transformation tools are used for support (MOFScript, Open ArchitectureWare, ATL)

3.2.7 Realization & Deployment phase

	Realization & Deployment
Description	The goal of this phase is to realize the validated system model obtained from the Early V&V phase. The realization can include design of HW components, source code and deployment. This phase

	also involves the tasks of testing the realized system against the design models and system requirements.
Start conditions	This phase starts whenever the Quality Evaluation triggers it.
Triggers	None
Inputs	Validated System model
Outputs	Finalized eDIANA device or application.
Specification method and language	Different languages used depending on the final device or application.
Tool support	System specific compilers. Model transformation tools are used for support (MOFScript, Open ArchitectureWare, ATL)

3.3 Artefacts

This section will provide detailed descriptions of the artefacts involved in the eDIANA development process.

3.3.1 Documents and Models

	eDIANA Scenario
Description	The eDIANA scenario holds a description of the scenario targeted by the device or application currently under design. The selected scenario has implications on the system requirements and, therefore, must be provided by the client before the System Requirements and V&V Scenario Specification phase starts.
Produced by	Client
Used by	System Requirements and V&V Scenario Specification phase

	Adapted Process
Description	This document contents the eDiana process adapted to be used in the development of a specific product with the models and methods to be used selected

Produced by	Process Configuration phase
Used by	Developers

	Application Requirements
Description	This document contents the requirements that the application have to meet. It is the main input for the Application Architecture Design phase.
Produced by	System Requirements and V&V Scenario Specification phase
Used by	Application Architecture Design phase

	Platform Requirements
Description	This document contents the requirements that have been defined for the embedded platform. It is the main input for the Platform Architecture Design task.
Produced by	System Requirements and V&V Scenario Specification phase
Used by	Platform Architecture Design phase

	V&V Requirements
Description	This document contents the verification and validation tests and correctness criteria to be followed in the Early V&V phase. Those V&V requirements will be defined at different abstraction level: for V&V of the system Allocation.
Produced by	System Requirements and V&V Scenario Specification phase
Used by	Early Verification & Validation phase

	Application Model
Description	This model contains the application components that will be used in the current eDIANA device or application design. Application models

	must contain information regarding the structure, behaviour and non-functional characteristics of the application under design. Lastly, application models are always platform independent; however, it is possible for application components to access platform services through the use of model proxies.
Produced by	Application Architecture Design phase
Used by	System Allocation phase

	Platform Model
Description	This model contains the platform components that will be used in the current eDIANA device or application design. Plaform models must contain information regarding the structure, behaviour and non-functional characteristics of the platform components used, as well as information about their interfaces.
Produced by	Platform Architecture Design phase
Used by	System Allocation phase

	System Model
Description	This model is created from the application and platform models. The system model combines coherently these two models to provide a full system description that can be used for early V&V purposes.
Produced by	System Allocation phase
Used by	Early Verification & Validation phase

	Validated System Model
Description	The validated system model is a system model that fulfils all the V&V requirements defined in the System Requirements and V&V Scenario Specification phase. This model contains a verified design model that will be used for realization and deployment purposes in the last phase of the process model.
Produced by	Early Verification & Validation phase

Used by	Realization & Deployment phase

3.3.2 Repositories

	Application Components Repository
Description	The application components repository is an organized database with validated eDIANA application components. These components might have been developed in other development contexts. The repository enables the designer to reuse these components in future eDIANA developments.
Used by	Application Architecture Design phase

	Platform Components Repository
Description	The platform components repository is an organized database with validated eDIANA platform components. These components might have been developed in other development contexts. The repository enables the designer to reuse these components in future eDIANA developments.
Used by	Platform Architecture Design phase

4. System Requirements Specification

4.1 Overview on the problems

The biggest problems to be solved lie in the field of coordination of a huge number of controlling devices (Cells) that operate in different environment, having the need to cope with the different profiles of each owner without losing the objective to achieve energy efficiency and select the best decision of buy-sell of energy produced by local distributed micro-generation. The problem is worsened even more by the need to establish a real time cooperation and exchange of information among the Cells and between the set of Cells and the Macro cells. By using or mixing the tools that are described herein below, it is crucial to devise a formal set of requisites that:

- Supports the categorization of different requisites in a small number of set containing a number of homogeneous requisites
- Can be transformed in a set of strong and weak constraints for the optimization algorithms
- Allows formal validation of the transformed requisites vs. the original ones and permits early evaluation of feasibility vs control policies
- Allow easy manipulation of requisutes to allow modification that confirm feasibility

Another set of information to be taken into account and fromalyy described is related to communications issues. In this case what is necessary to correct specify in terms of constraints and requisites is:

- Requisites that define the concept of real time exchange of information for this specific application
- Value of information and cost of its loss in terms of the quality of performance of control algorithms and decision support systems
- Requirements about the need to recall (or reconstruct) the correct information

The final set of requisites to be set via a formal support tools concern the interaction with the users either as controllers (At Macro cell level or even higher) or at Cell Level (typically the owner). Since these requisites (subjective in nature) are likely to have a difficult transformation in a set of objective requites, the methodology is required to:

- Devise a proper methodology to formalize the issues from users
- Identify the mechanism to transform these issues in one or more set of requisites
- Provide an objective tool to verify and validate these sets of requisites

4.2 Requirements engineering

One of the main problems in Software and Systems Engineering is to bridge the gap between customer and analyst: to get the analysts to have an in-depth understanding of the problem and business needs, and to get the users to understand how the solution that the analysts propose will solve those needs.

In order to deal with these problems, Requirements Engineering has been developed as the branch of Systems Engineering that covers all of the techniques, methods, and procedures applied to the definition and management of the user needs that the system under study in this case eDIANA project must satisfy.

Requirements are vital throughout the whole lifecycle of a system. The process of building a system begins with the identification of high level user requirements, that get completed and refined in later stages of the project lifecycle, and eventually evolve into technical specifications that define the system to be developed.

4.2.1 Requirements development and Requirements management

We can split the entire domain of software requirements engineering into requirements development and requirements management .

We can further subdivide requirements development into four engineering activities, elicitation, analysis, specification, and validation, as illustrated in Figure 1 (Abran and Moore 2001).

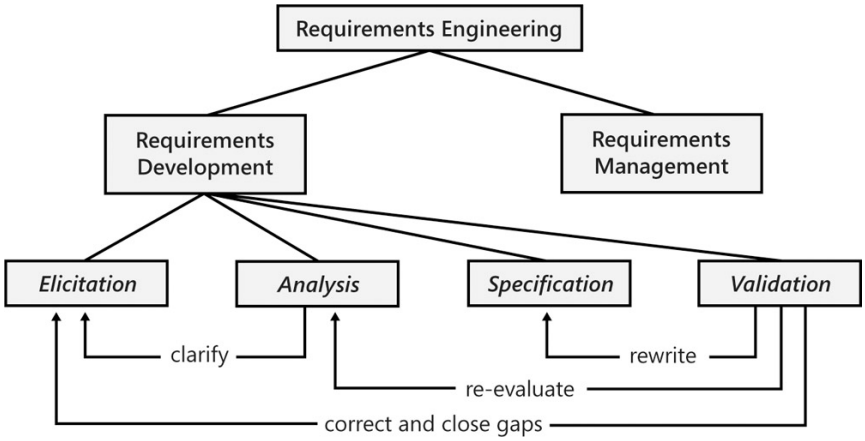

Figure 1: Subcomponents of requirements engineering.

Requirements Elicitation – Deals with discovery, review and understanding of requirements between the customer(s) and developer(s). For the elicitation phase eDIANA project will use Structured Natural Language templates for capturing the requirements for the customers. These templates will be developed based on the taxonomy or ontology created for the classification of the requirements in a formal way.

Requirements Analysis – Reasoning and analyzing the needs of customers and users to arrive at a definition of software requirements. A major requirements analysis activity is to derive more detailed requirements from higher-level requirements. Analysis also involves creating multiple views of the requirements, such as prototypes, graphical analysis models, and tests. Other aspects of requirements analysis include negotiating priorities, searching for missing requirements, and evaluating technical feasibility, risk, and failure modes. Analysis provides a feedback loop that refines the understanding that the analyst developed during an elicitation activity.

Requirements Specifications – Process of producing a record of each of the requirements clearly and precisely. Traditionally these records are documents containing natural language text. As we'll see in this book, though, other representation techniques also are valuable, such as graphical analysis models, tables, and mathematical expressions. The "specification" could consist of requirements information stored in a database, as in a commercial requirements management tool, rather than being a traditional document.

Requirements Verification and Validation – Assurance that requirements specifications are an adequate basis for the preliminary design phase. Ensures that the product fulfils its specific intended use. will satisfy customer needs, and have all the characteristics of high-quality requirements. Validation might lead the analyst to rewrite some requirements specifications, to reassess the initial analysis, or to correct and refine the set of documented requirements.

Requirements Management – Overall process of planning and controlling the requirements elicitation, analysis, and verification activities listed above. In the other Requirements management commences when the team says they believe their requirements are good enough to serve as the foundation for design and construction of some portion of the product. At this point, the analyst defines a requirements baseline, a snapshot in time that represents the current reviewed, agreed-upon, and approved set of requirements for a specific product release. Project stakeholders make schedule and cost commitments based on the

requirements baseline. Because changes in the baseline can affect those commitments, formal change control begins at the time the baseline is established.

4.2.2 Requirement traceability

Customers say what they want in requirements. They can only be sure they get it by verifying that each requirement has been met. To do this, the acceptance tests must trace back to the requirements, covering all of them appropriately. Incidentally, scenarios of interest to users are good candidates for acceptance test scripts.

Similarly, the developers can only be sure they are implementing all the requirements if they can ultimately - though not necessarily directly - trace each design element back to the requirements concerned, and check that each requirement is fully covered. They can also use traces in the other direction to show that each design element is actually called for in the requirements. The management of traces between engineering objects such as requirements, tests, and design elements is called traceability. It is a vital tool in managing system development through requirements.

Advantages of using a requirement tool

A requirements tool can help you check that there is at least one trace from each requirement to the design: if there are any untraced requirements, there is work to be done. But it can't check that the traced parts of the design are sufficient or correct - that's your job. There may be any number of links between requirements and system or test specifications:

Handling traceability and change without a requirements tool is tedious, and it is easy to make mistakes. The design changes quite often and requirements need to be updated as well. On any but the smallest project, tracing requires reliable, industrial-strength tool support. To keep track of changes by hand means recording in a table each change to each requirement, each design element, and each test, and checking each time via a traceability matrix for any possible impact on other items. If you need to trace directly to design, a requirements tool that can interface directly with your design tool is virtually essential.

4.2.3 Requirements management tools

To manage all of the before mentioned activities, the market offer different solutions to carry out all the lifecycle that we need to cover. These kind of tools allows to map all the requirements in Use cases and modelling these in the system to achieve the UML models for transformation process in MDA.

For example, Rational RequisitePro is an easy to use requirements management tool that lets a team:

- Author and share their requirements using familiar document-based methods while leveraging database-enabled capabilities such as requirements traceability and impact analysis.
- Apply requirements management using the Use Case technique which should help the eDIANA project to manage individual requirement artifacts and fit requirements within the Rational Unified Process (RUP).
- Make customizations to the requirements process specific to the eDIANA project and work with guidelines and techniques for capturing functional and system requirements.
- Use traceability and tools to automate time-consuming processes . Specify, validate and manage evolving requirements.

At the end, Use Cases provide the basis for the whole object-oriented, software life cycle including architecture, design (including GUI design) and development. At the same time Use cases help testing efforts by facilitating the creation of test cases. All tests must contain a sequence of events, which will be followed to test a particular area of the eDIANA system.

5. Architecture Design

This section is targeted to to describe the modelling methodology applicable to the three phases of an eDIANA platform architecture design, namely, the Application Architecture Design phase, the Platform Architecture Design phase and the System Allocation phase.

The models involved in each of these phases will be explained in terms of views, modelling languages, etc. and a set of modelling guidelines and constraints will be given. In order to develop a completely coherent model-driven development framework, it is necessary to constrain the modelling languages to a subset in order to enable the reusability of the model transformation engines, e.g. to connect system designs with analysis tools.

Lastly, eDIANA specific components will be addressed in the methodology framework in order to ease the designers' job to use this modelling methodology.

5.1 Modelling Languages

The eDIANA platform will be an integration of different systems that cooperate with each other towards a common goal. One of the main characteristics of an eDIANA application is heterogeneity. Any eDIANA platform will be typically composed of devices from different vendors, each running different applications and operating systems and they will be probably built on top of different hardware platforms. The eDIANA model-based design and development methodology has to be aware of this extra complexity and provide an integrated modelling framework. Moreover, this modelling framework should be easily integrated with the modelling abstractions currently used in the embedded systems industry to enable a smooth transition from legacy methodology toward the eDIANA approach.

On the other hand, we have already discussed the similarities between eDIANA and GENESYS. The development of GENESYS systems is supported by a generic model-driven methodology intended to be applicable to the whole design and development cycle of the embedded products. Taking into account the latter similarities, it is reasonable to think that the GENESYS methodology could be adapted to the eDIANA requirements.

Taking into account the latter considerations we present a UML centric modelling approach enriched with the profile for Modelling and Analysis of Real-Time Embedded systems (MARTE). MARTE is a UML profile, standardised by the OMG, which provides a set of subprofiles and stereotypes for embedded systems modelling and analysis. Since UML+MARTE is intended for cross domain embedded applications, it is a suitable language for the eDIANA platform modelling and design.

Moreover, due to its broad modelling scope, MARTE can be adapted and transformed to domain specific and tools specific languages to enable integration of the whole development process of the eDIANA platform. Lastly, it is important to note that MARTE is a profile constructed on top of UML. UML already provides designers of rich and very low level constructs to describe the behaviour of the systems and components. Such a low level description of the behaviour enables transformations from models to implementation code.

Therefore, the proposed methodology framework establishes a MARTE centric approach for modelling that integrates other modelling languages, analysis tools and which provides code generation.

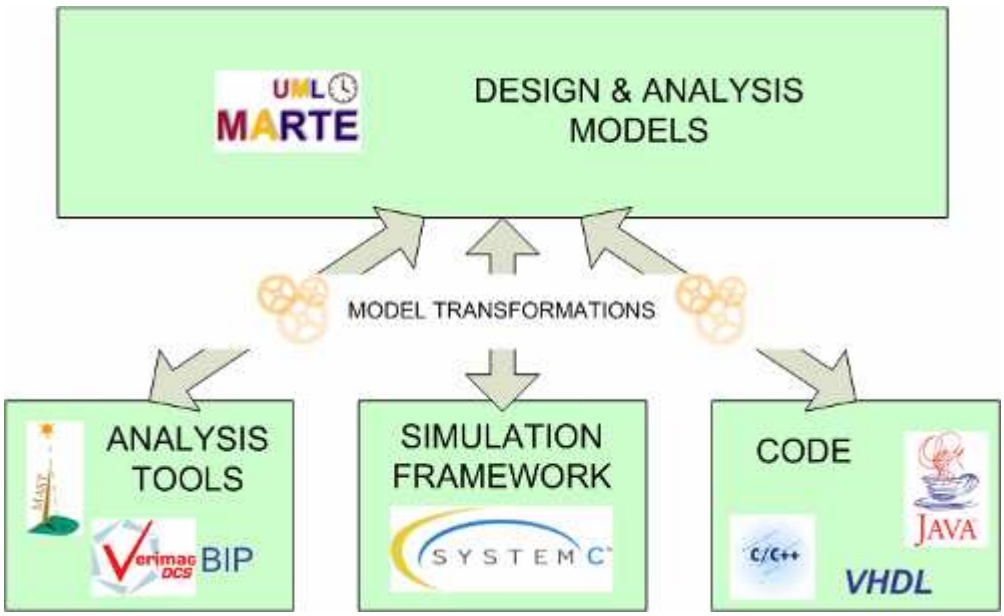

Figure 5-1. The MARTE centric modelling approach

As we already stated, in order for the methodology to be successful it is mandatory that modelling languages already in use in the embedded systems industry are integrated into the modelling framework. In the following lines we will describe those relevant to the eDIANA platform.

5.2 eDIANA Architectural elements

The eDIANA platform principles state that any eDIANA deployment must be strictly component oriented. Following this criterion, the eDIANA MDE methodology proposes a component oriented framework for designing eDIANA devices and applications.

In order to create the component-based design framework it is a requirement to fully characterize the whole range of eDIANA platform and application components, including all the different scenarios and hierarchical levels (i.e. Cell and MacroCell). The component repositories described in the process model will store the eDIANA components collection. Yet, to use these components in an MDE environment, it is also necessary to create eDIANA compliant models that can be reused during model-driven design stages.

Depending on their nature, application or platform components, the modelling constructs required for them vary. Application components are basically composed of:

Job. The job is the concrete functionality provided by the component. A job may be further split into tasks; however, from the component based designer's point of view, the job of a component implies certain behaviour and a set of non-functional properties.

LIF (Linking InterFace). The LIF specifies the messages and signals consumed and provided by the job.

On the other hand, platform components provide the physical entity that hosts/contains the logical application component. It is also possible that some platform components (e.g. software libraries, middlewares...) provide services used by the applications (i.e. APIs).

In the eDIANA platform the following component types are foreseen:

- Cell Level
  - Local control component: algorithms
  - Local control component : local acquisition of data and processing
  - Local control component : communication (peer to peer) and hierarchical
  - Local control component : simplified HMI
  - Local control component : diagnostic
  - Local control component : performance evaluation
- Macro cell level
  - Control component : scenario evaluation
  - Control component: optimization policy

- Control component: decision support
- Control component : iterative optimization and policy setting
- Local control component : communication (peer to peer) and hierarchical
- Local control component : simplified HMI
- Local control component : diagnostic
- Local control component : archival and data retrieval

5.3 Transformation framework

The design and development framework and methodology proposed for eDIANA is intended to be flexible and dynamic. In terms of flexibility, the eDIANA methodology aims at enabling a smooth transition for the legacy development tools to the modern MARTE based approach as was already stated before. Moreover, it is also important to provide some level of automation between the development process phases.

The methodology described in this document proposes a set of transformations that integrate the different development tools into the eDIANA process and also transformations that integrate a set of required analysis tools.

5.4 Platform Architecture Design

The Platform Architecture Design phase deals with the modelling of the platform architecture that supports the applications designed during the Application Architecture Design phase.

Platform requirements can be business requirements, system requirements and technical constraints. Business requirements scope the platform architecture design. System requirements define which kinds of properties are required from the platform; they can be technical features and/or restrictions that affect the platform like pricing, weight, communications. Technical constraints are based on earlier decisions or standards the platform has to support.

The eDIANA platform designs may address the development of new eDIANA devices (equivalent to GENESYS Device Level, L2) or full eDIANA applications (GENESYS

System Level, L3). Each of these levels addresses different application description challenges through the use of modelling techniques.

Device level applications focus on creating complete embedded devices. They make use of platforms that can, for example, be composed of a set of chip level platforms providing services to device level applications. Additional middleware and services may be used on top to provide a powerful interface to application designers.

System level applications are composed of a set of distributed devices that interact with each other. At this integration level only software platform elements may need to be considered (e.g. communications middleware) since the devices composing the application already have their hardware/software architecture defined.

Despite their nature, hardware or software, platform elements have to be considered using two different points of view: a structural view and a behavioural view. Both views are defined at the logical level. The first view provides a designer an understanding of which kinds of building blocks the platform is composed of, and which kinds of services are provided for the applications. The behaviour view describes how the defined building blocks of the platform structure work together and depend on each others. It also enables system simulation that allows early detection of design errors, increases the quality of the final system and reduces maintenance costs.

The platform architecture modelling produces the models of the following views:

Structural view. The view describes the platform architecture from its structural point of view. The platform architecture model is composed of resources (both SW and HW) at different granularity levels (e.g. processor, computing node, multiple interconnected computing nodes).

Behavioural view. The behavioural view describes the behaviour of the services included in the platform structural view. The behavioural view can be provided using different mechanisms, such as pseudo-code, UML models, etc.

The structural and behavioural views form a skeleton of the whole platform architecture and it is used for allocating/mapping application models onto it. The following sections describe the definition of the structural and behavioural views with UML-MARTE. Defining non-functional and quality properties is also possible in these views.

5.4.1 Structural view

The structural view of a system's platform is meant to describe which elements the execution platform consists of. The execution platforms are composed of hardware and software elements. The structural views represent real elements that have

critical influence on the non-functional properties of a system. These non-functionalities have to be captured in the platform models in order to be able to achieve useful simulation and evaluation results in future phases of the embedded systems development process.

The structural view of a system's platform provides information about the real elements of the execution environment intended for the application/service under construction. Those model elements refer to real resources in the final systems.

MARTE provides platform modellers the Generic Resource Modelling (GRM) sub-profile which allows describing embedded platforms at the high level including both SW and HW in a generic way, without going into details of the actual platforms (i.e. which processor and/or operating system). In the following subsections we will try to describe how to use this sub-profile to model the resources of the embedded platforms.

There is a necessity to differentiate between application and platform elements. The former were discussed in section 5.5, the latter in this section.

5.4.1.1 MARTE GRM concepts for execution platform modelling

An embedded system platform model is composed of models of SW and HW elements and their interaction relationships. The GRM sub-profile gives concepts Resource, ResourceService, and their corresponding instances ResourceInstance and ResourceServiceExecution. Resources are used to model the execution platform from a structural point of view, while the resource services supply the behavioural point of view.

As it occurs with classifiers, the execution platform may be represented as a hierarchical structure of resources.

- Resource types:
- Storage resources
- Timing resources
- Synchronization resources
- Computing resources
- Concurrency resources
- Generic device resources
- Communication resources: end-points and media

These concepts have to be addressed by the modelling language; for example in MARTE, a Scheduler is defined as a kind of ResourceBroker that brings access to its broked ProcessingResource or resources following a certain scheduling policy.

SchedulableResource is defined as a kind of ConcurrencyResource with logical concurrency.

When the executionBehavirours of concurrencyResources need to access common protected resources, the underlying scheduling mechanisms are typically implemented using some form of synchronization resource, (semaphore, mutex, etc.) with a protecting protocol to avoid priority inversions.

ResourceUsage links resources with concrete demands of usage over them. A few concrete forms of usage are defined at this level of specification under the concept of UsageTypedAmount; those are aimed to represent the consumption or temporary usage of memory, the time taken from a CPU, the energy from a power supply and the number of bytes to be sent through a network.

As we can see, each construct needs its own representation in the eDIANA platform models and, therefore, they must be specified following a

Platform architecture model – structural view

When modelling execution platform it is usually presented as a hierarchical layered model, e.g. platform layer consisting of a set of (different) computing nodes linked with (internal) network communication, computing node (also called processing node or sub-system) layer consisting of components and component layer, all layers containing appropriate software implementing system services.

Examples at the component layer are processing (e.g. programmable processors), storage (e.g. volatile memories) and interconnection (e.g. bus) elements as well as OS/scheduler and device drivers. Their services are of basic type, like read, write, etc. Examples of NFP property types include e.g. clock frequency, cycles-per-instruction, pipelining, read-latency, write-latency and burst-latency.

Example at the computing node layer is composed of bus, different processor types each with private memory, OS and device drivers, shared memory, shared I/O component and shared network interface component (to connect to network communication at the platform layer). A processing node is capable of providing generic multi-tasking (multi-processing, multi-threading) and device driver services. Examples of NFP property types include e.g. task processing time, service processing time, number of task switches, number of processor cycles used, communication latency/delay, etc.

Example at the platform layer is instances of different types of computing nodes connected with (internal) network communication. Examples of NFP property types are similar to the computing node layer, but now possibly aggregated if services span several computing nodes.

In the following subsections we will provide some UML+MARTE modelling guidelines related to platform components that might be useful to eDIANA developers.

5.4.1.2 Modelling processing units and tasks

The most common layout of an embedded application is that of concurrent execution threads competing for the processing core/s of the embedded device. These threads are abstracted by the underlying operating system and they are scheduled following the criteria of a certain scheduling policy.

To model this layout, MARTE GRM subprofile provides the designers of three stereotypes: <<ComputingResource>>, <<Scheduler>> and <<SchedulableResource>>. Figure 5-2 depicts an example layout with a single processor scheduled via a fixed priority policy. The figure shows three tasks that compete for the processor.

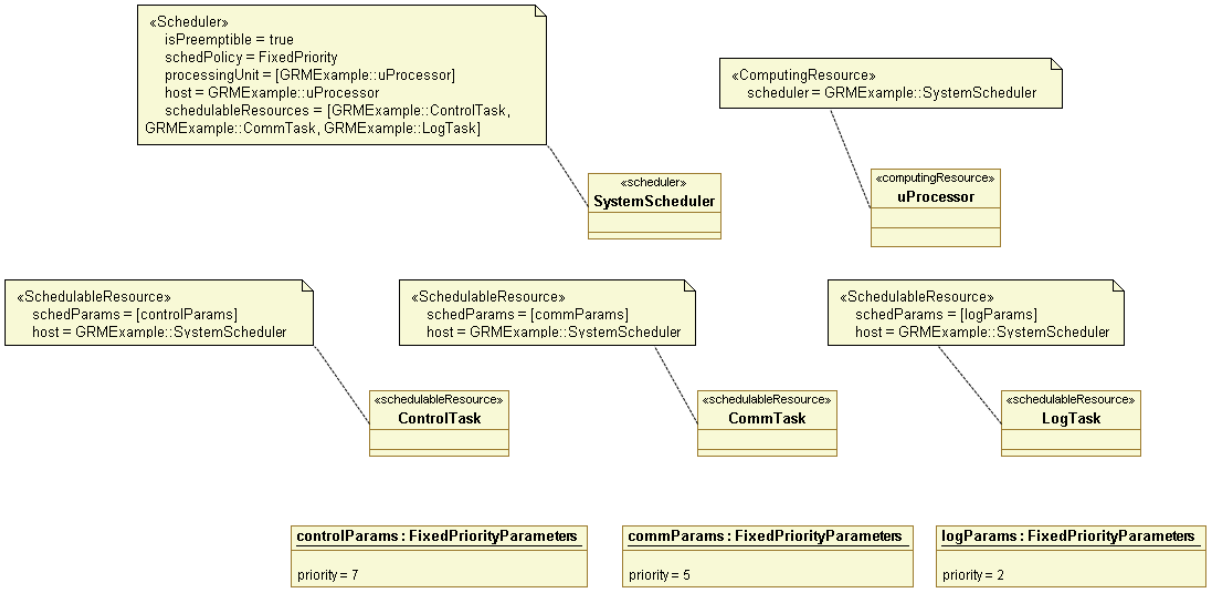

Figure 5-2. Modelling a processor and three tasks.

As shown in the latter diagram, the relations between processors, schedulers and tasks are clearly defined by using the MARTE stereotypes.

Schedulers are a fairly important element in embedded applications running on top of Real-Time Operating Systems (RTOS) and the profile allows describing it with a great level of detail. The latter diagram shows a fixed priority scheduler that is hosted (executed) by the system processor. As shown in the example, the scheduler is scheduling the access of a list of schedulable resources (i.e. processes or threads) to the processor's computing resources. In order to do so, the scheduler will follow a fixed priority policy with pre-emption.

Threads can also be further described using the properties defined for the <<SchedulableResource>> stereotype. In this case, the three tasks defined are described by "FixedPriorityParameters" instances which only contain one parameter: priority. The subprofile also defines parameter types for other scheduling policies. In order for the model to be consistent it is necessary that the parameters used to describe system threads match the scheduling policy used by its scheduler.

5.4.1.3 Modelling shared resources

A problem related with multithread programming is handling the access of the different tasks to the shared resources. MARTE GRM sub profile provides the users of two stereotypes to model shared resources depending on access protocols. These stereotypes are <<SynchronizationResource>> and <<MutualExclusionResource>>. The main difference between these two resources is that the synchronization resources refer to unmanaged elements like semaphores and mutexes while mutual exclusion resources refer to elements handled by an access protocol. Figure 5-3 shows the three tasks defined before and two shared resources of different types.

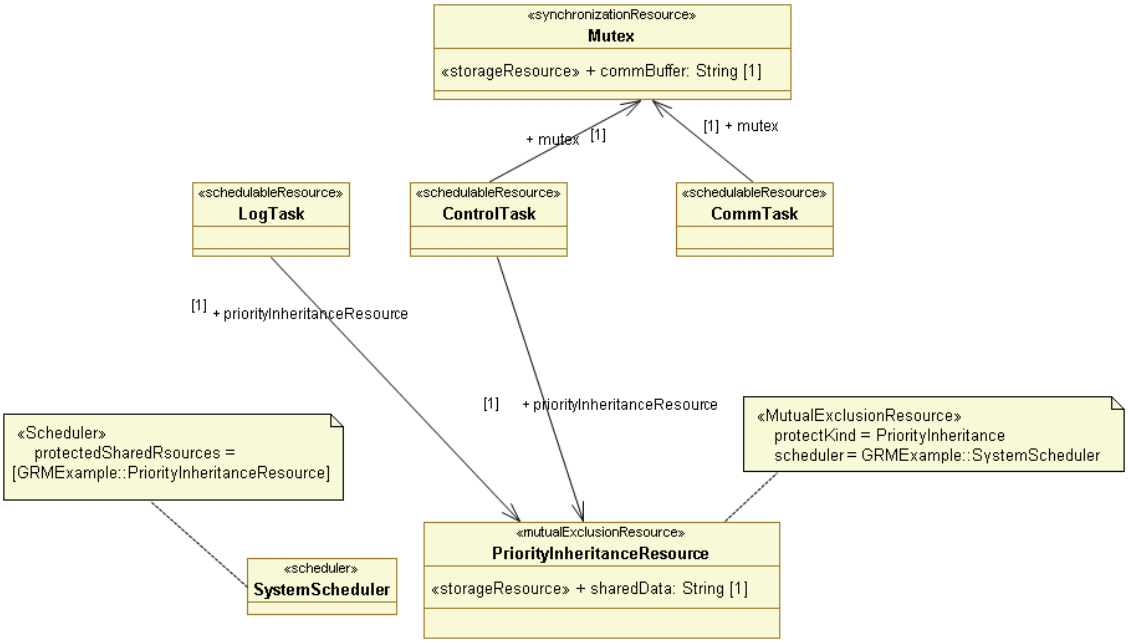

Figure 5-3. Modelling shared resources with MARTE.

The diagram shows how a shared variable can be modelled following both approaches. In this case the mutual exclusion resource depicted follows a priority inheritance protocol managed by the system scheduler that was presented in Figure 5-2.The MARTE GRM subprofile allows describing this relationship through the use of the stereotype properties defined. The shared elements have been modelled using <<StorageResource>> stereotypes. These stereotypes will be covered in the following section.

5.4.1.4 Modelling variables and shared memory

One of the most important characteristics of embedded devices is resource limitation. Among all, the most critical resource in embedded applications is memory. Therefore, in order to successfully describe embedded applications it is necessary to precisely model memory resources and requirements. The MARTE GRM subprofile uses the <<StorageResource>> stereotype (presented in the latter section) to model data containers. Figure 5-4 adds a finer grain description of the example in Figure 5-3.

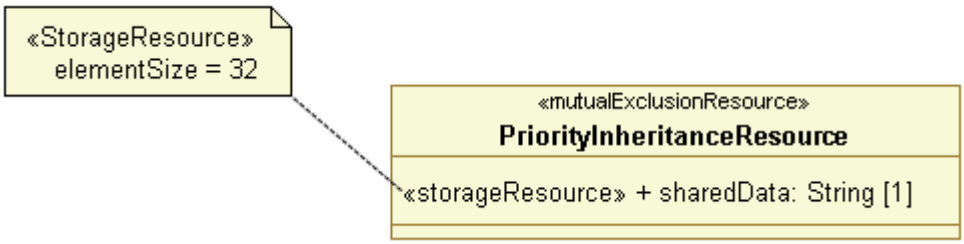

Figure 5-4. Detailing variables and memory in MARTE.

In this example the size of the shared information in the priority inheritance resource defined before is specified. Although these aspects might not be strictly platform issues it is important to describe them in order to know whether a specific platform is well suited for a certain application.

5.4.1.5 Modelling communication resources

Another important aspect of embedded systems is the capability of interacting with other devices. In order to do so, system must access communication media and use communication resources. The MARTE GRM sub-profile provides two stereotypes to model communication resources, both resources internal to the operating systems (i.e. pipes, IPC...) and network resources (i.e. Bluetooth, IP networks...). The following example depicts how a TCP socket connection is modelled using MARTE GRM.

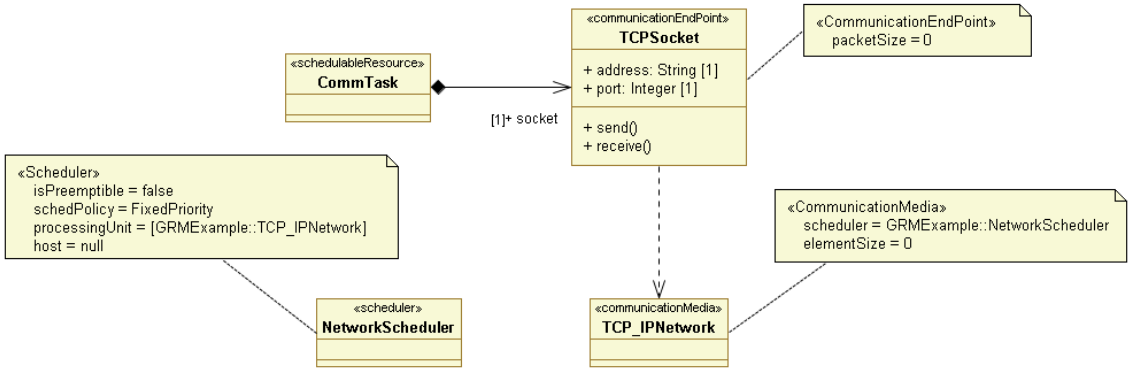

Figure 5-5. Modelling communication resources in MARTE.

As shown in Figure 5-5, the <<CommunicationEndPoint>> stereotype is used to model the platform element in charge of transmitting the messages to a <<CommunicationMedia>> and/or receiving incoming messages from remote peers. Many real-time applications need predictable communication resources (e.g. industrial SCADA systems). These applications use real-time networks to achieve a predictable communication between network devices. MARTE provides support to model this kind of networks. In the example a TCP/IP network is modelled. IP packets can be prioritized according to a fixed priority policy. In order to model this kind of behaviour, we have used a virtual network scheduler which is not related to a physical component, but yet affects the communication behaviour. Again, in order to keep the model consistency, the element sizes for communication media and end points must be the same/compatible.

5.4.1.6 Modelling platform black-boxes

It is common that embedded applications use both dedicated hardware and/or special software libraries to help developers perform a certain action (e.g. driver to access a sensor or an MP3 hardware coder). These pieces of hardware/software are treated as black boxed by the application designers and developers who will use the specific devices without caring for its implementation details. MARTE GRM allows introducing such an element in our platform models by using the <<DeviceResource>> stereotype. Figure 5-6 shows an FFT accelerating hardware piece that interacts with the control task modelled before.

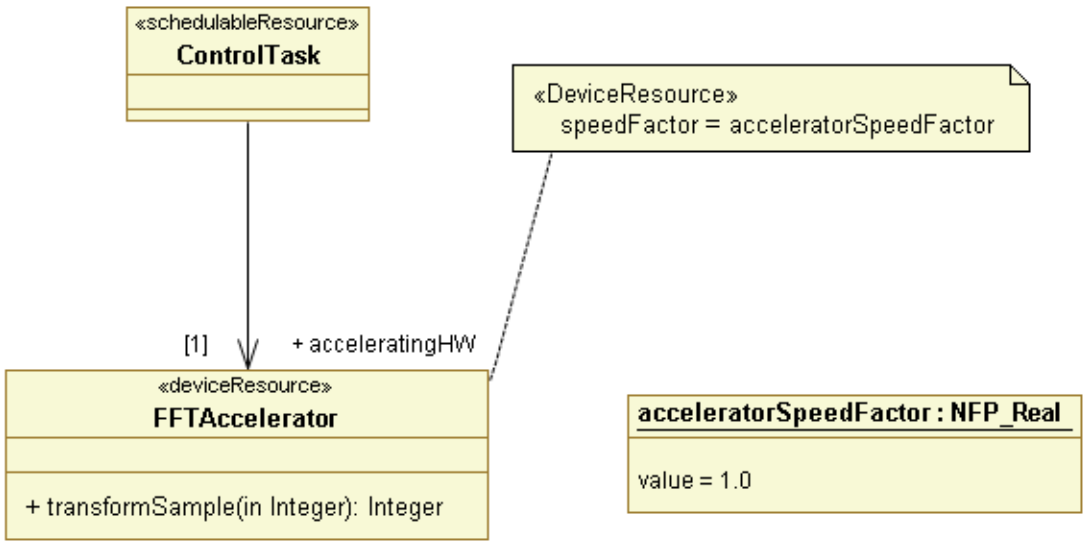

Figure 5-6. Modelling support hardware as device resources.

The stereotype allows the distinction of hardware and software resources by using its properties. A hardware device resource will use the speedFactor property to specify its processing speed with relation with the main computing resource in the system.

On the other hand, software libraries won't specify a speed factor. A device resource may also use a scheduler to prioritize elements accessing it.

5.4.1.7 Modelling timing resources

It is common to find embedded devices that rely on different timing resources which they use for different purposes. MARTE GRM provides two stereotypes to model clocks and timers. Figure 5-7 shows a timer resource included in our example.

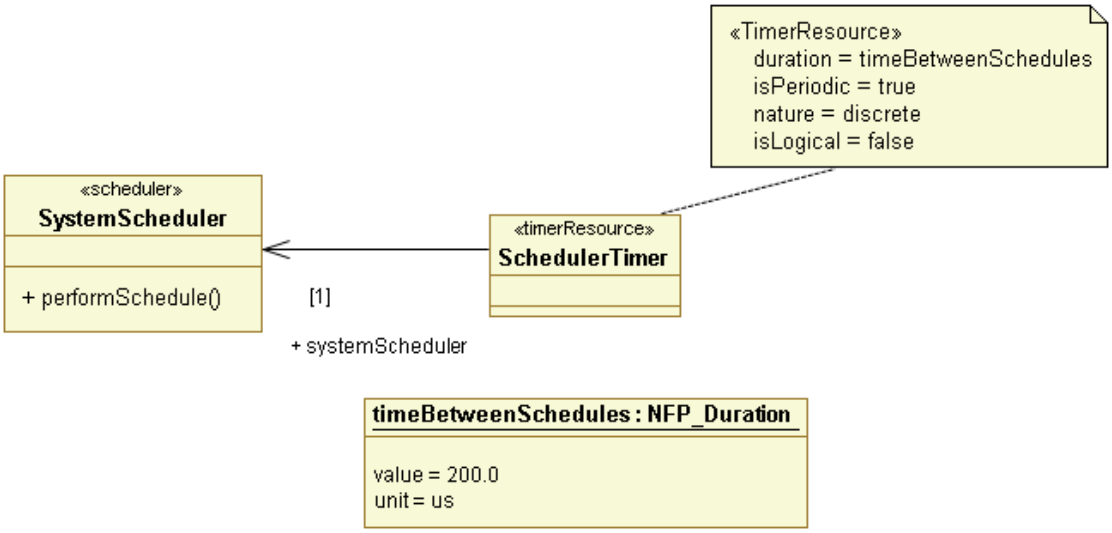

Figure 5-7. Modelling a timer.

As depicted in the diagram, we have now added a timer that will periodically inform the system scheduler that it is time to reschedule the resources managed by it. Clocks are defined in a very similar way. To obtain further information on modelling timing resources please refer to the MARTE GRM specification.

5.4.1.8 Further refining platform structural models

In many cases the MARTE GRM sub-profile is expressive enough to describe platform architectures; however, in certain cases it is possible that the platforms models may be too generic for the application under construction or regarding further phases of the eDIANA development process. If this should be the case MARTE provides two specific and more concrete subprofiles for software and hardware description: the MARTE Software Resources Modelling (SRM) subprofile and the MARTE Hardware Resources Modelling (HRM) subprofile respectively.

5.4.2 Behavioural view

The behavioural view is presented as the interfaces and their state machine descriptions (i.e. protocol) of the above mentioned services.

Regarding the methodology framework, in order to model the behaviour of the platforms, UML behaviours (i.e. activity, sequence and state machine diagrams) could be useful to model interactions within the platforms.

In order to reflect how platform behaviour can be modelled, an operating system round-robin scheduling pattern is modelled. The example, which is based on [5], describes a round-robin scheduling algorithm by a class diagram (structural view) and a sequence diagram (behavioural view). The model elements that complete the pattern have been stereotype according to the GRM.

The round-robin pattern schedules a set of ordered processes with static priorities by assigning time-slots to each of them. Once each processes completes the processing time it is pre-empted and the processor is assigned to the next process in the list.

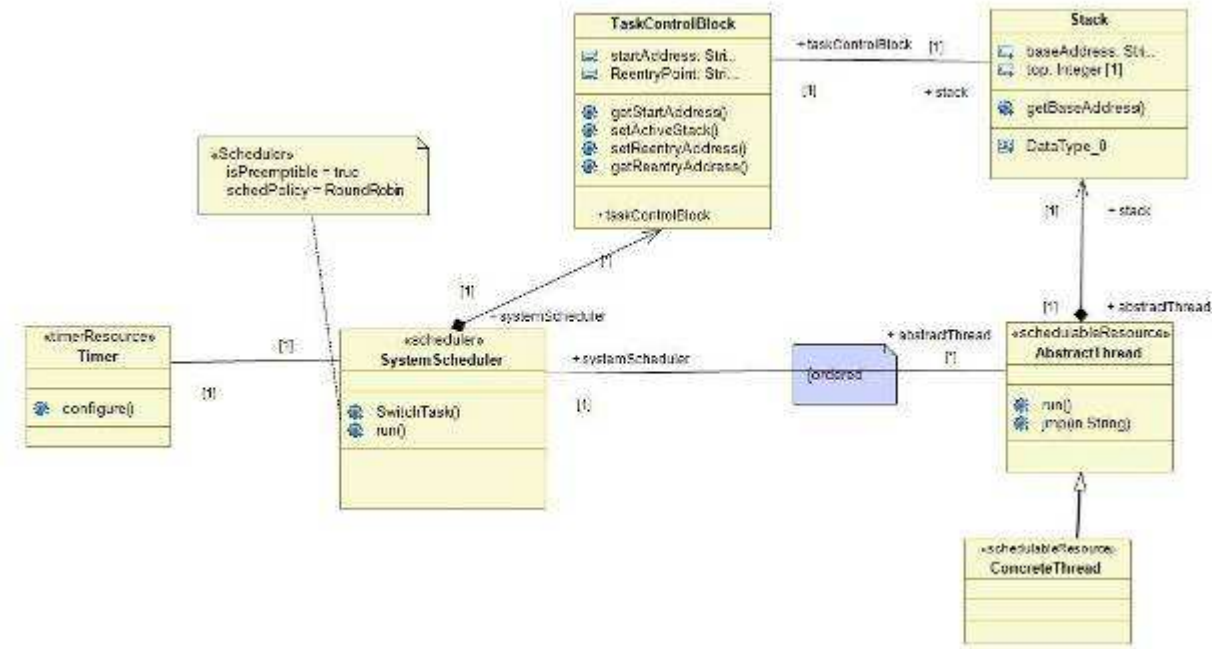

Figure 5-8. Structural view for a scheduling pattern.

The class diagram above describes the main elements involved on the round-robin concurrent pattern. As shown, a system scheduler is interrupted by a timer that is has been previously configured at initialization time. The scheduler assigns time-slots to execution threads which are characterized by their control blocks and stacks. Task control blocks describe the initial addresses of each task and stacks refer to the memory segments assigned to each task for storing temporal variables or parameter or return values for system calls.

In order to describe the behaviour of the platform, a sequence diagram showing a possible scenario is provided below.

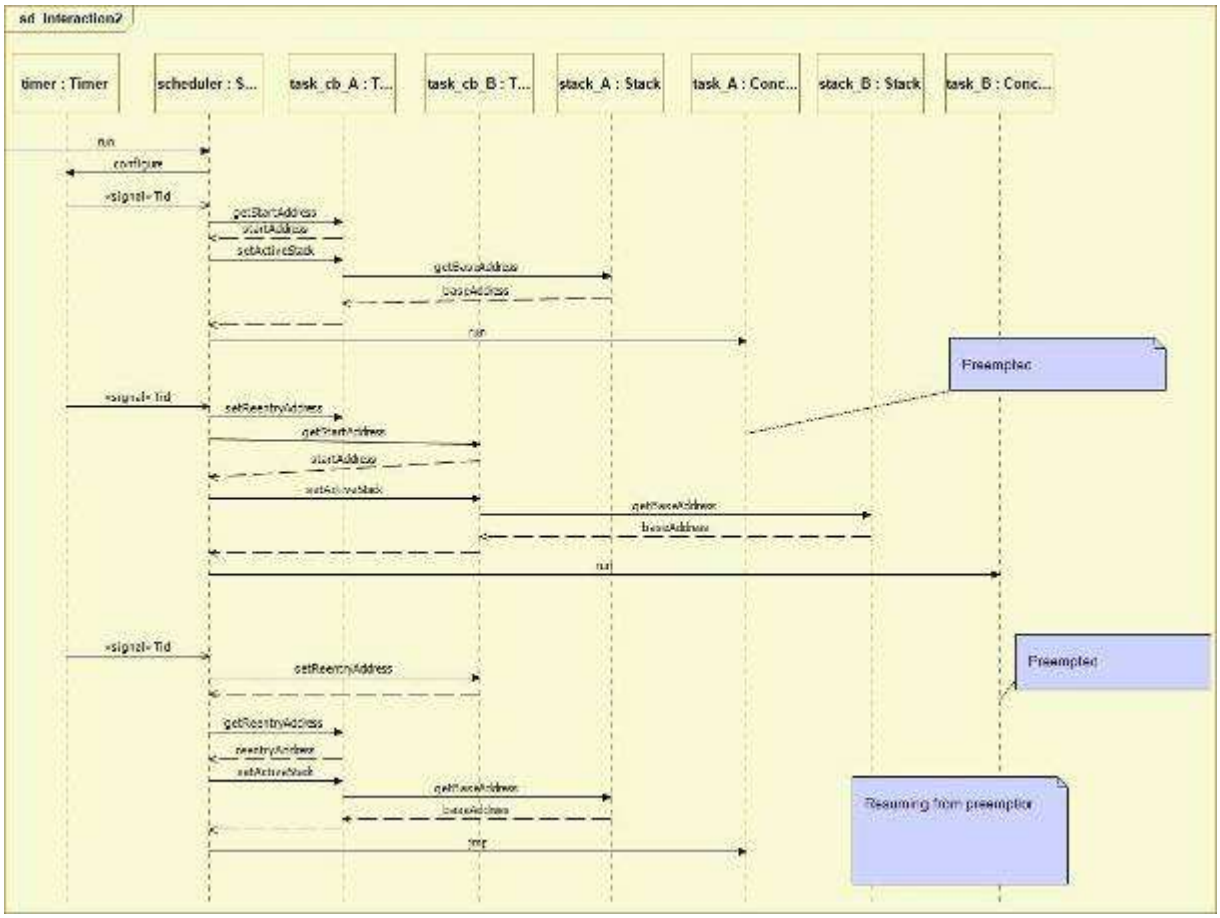

Figure 5-9. Behaviour as a sequence diagram.

In order to ease a link between the models and the analysis/simulation tools UML provides the option of using opaque behaviour UML model elements. Opaque behaviours are defined by pieces of code or pseudo-code regarding it specification; therefore, using these approach, it is very simple to establish a link between the UML2+MARTE models and other modelling languages like SystemC or BIP.

Each element in an architectural view represents real hardware and software parts. Each element has, therefore, a behaviour that is implicit on that component's nature. Behaviroural views capture these behaviours and enable simulation and testing tools to draw early conclusions from system design models.

Due to their link with reality, behavioural views are heavily constrained by non-functional properties regarding timing, power consumption, weight, length, etc. The more complete this kind of views are the greater will be the number of early test that we will be able to perform of the system designs and the greater the quality of our final products will be.

5.5 Application Architecture Design

The Application Architecture Design phase is concerned with the design of the applications from both functional and non-functional points of view. Moreover, in order to enable composition, the services provided by the different eDIANA devices have to be defined, not only at syntactical level, but also at behavioural and semantic levels.

The goal of the Application Architecture Design phase is to produce a platform independent model of an eDIANA device or application. The phase produces the following views:

Structural view. The structural view contains the definition of DAS (i.e. interaction between jobs), the jobs, LIFs and messages that take part in the application under design.

Syntactical view. The syntactical view contains the description of the protocols that manage the access to a certain service. (The interface description is partly defined by the structural view and the syntactical view).

Behavioural view. The behavioural view defines the behaviour of the application at two levels: as behaviour of the application and as behaviour of the jobs involved in the application.

Semantic view. The semantic view provides information regarding the semantics of the application service. The semantic view is related to a service ontology that will be an enabler for service composition engines.

Again, these views can be provided by different languages; however, they must be integrable with the UML+MARTE modelling approach. It is possible that many diagrams are included in a single view. It is also possible to describe two or more views in the same diagram but it is against the separation of concerns, one of the architecture design laws and breaching that law will lead to serious problems in architecture evolution.

In order to provide developers with models that are expressive enough, it is important to present the structures of the services and jobs in application architecture. The next sub-sections will cover the modelling of each of these views using UML2 and the MARTE profile. MARTE is a UML2 profile and, therefore, it cannot be used without it. It is important to note that UML provides several ways to describe the same aspects of the system models. This fact makes it difficult to provide a unique method to create the models as many different diagrams can be used to specify the same aspects in the models. For example, in many cases state machines and activity diagrams can address the same behaviour. Therefore these sections provide a best practices guide for using UML+MARTE.

5.5.1 Structural view

The structural view describes the application as a whole and the building blocks, i.e. jobs and interfaces, which it is combined of. The structural view of an application provides information regarding the construction of the service. Services are defined by their interfaces. Therefore, the structural view is described as follows:

Describe the jobs involved in the application under design.

Describe the service interfaces of each job and messages passed through each interface.

Describe the application as a composite of jobs. Reuse the available application service descriptions.

Describe the resources (e.g. variables, communication channel, etc.) shared between jobs.

Map non-functional and quality requirements and constraints defined for the application in the system specification phase to the appropriate diagrams of the application.

The structural view has to describe the applications in terms of jobs, i.e. different tasks that must be executed. Moreover, the different services involved in the application must be defined in terms of their interfaces and the kind of messages they request/provide. Lastly, structural descriptions also include passive elements that help different jobs to communicate. The MARTE profile provides two specific sub-profiles for this kind of view:

High-Level Application Modelling (HLAM) sub-profile and

Generic Component Model (GCM) sub-profile.

These two sub-profiles along with the UML2 constructs allow a rich description of applications and services.

A cruise control system (CCS) is used as an example to illustrate the structural view. The controller receives two input messages containing the current speed of a car and the desired speed value selected by a car driver and computes an output signal that affects the engine of the car. Thus, the controller provides three interfaces: two input interfaces each of which reads a speed signal, and an output interface which provides the control signal for the engine actuator. To model this controller we will use a UML active class stereotyped with <<RtUnit>> from the MARTE HLAM sub-profile. The stereotype gives a class for the semantics of a task or a set of tasks that will be executed in some computing resource of the underlying platform. The stereotype includes many properties that may increase expressivity of a class.

The final structural model of the CCS controller is depicted in Figure 5-10. In the figure, the real-time unit has three interfaces, each of which is modelled as a UML interface with the <<BFeatureSpecification>> stereotype from the MARTE GCM sub-profile. This stereotype gives interfaces the types of a signal/message provider/consumer. It just adds a single property to the interfaces regarding the direction of the signals/messages defined in them. The protected containers are added for the speed signal values that the controller will receive. The stereotype <<PpUnit>> gives a classifier the semantics of a protected passive element of the system.

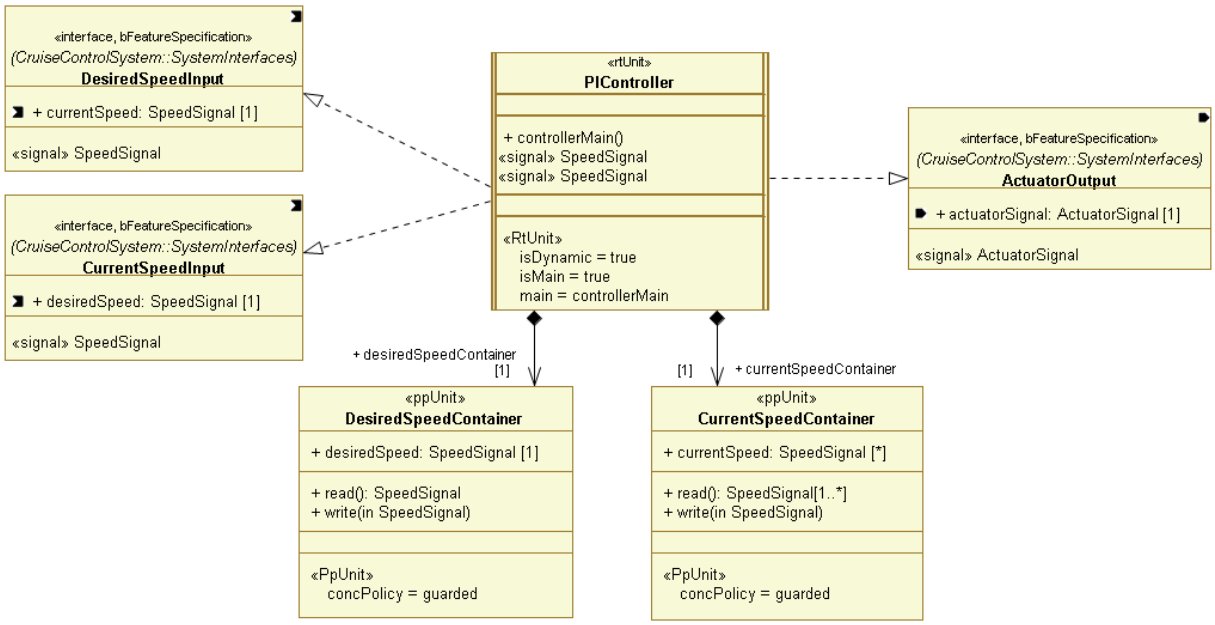

Figure 5-10. Structural view of a CCS controller.

Applications are compositions of jobs that further use application and platform services for achieving the desired functionality/capability of a system. The MARTE GCM sub-profile provides a composite diagram for defining applications by components and connectors. The cruise control application will consist of four components (i.e. instances of the clients and servers): two speed inputs provided by sensors, the controller we defined in the earlier section and the engine actuator. The components interact via UML2 ports (i.e. instances of their interfaces defined in their structural views). To model these interactions a UML composite diagram is used. Composite diagrams of applications are especially needed when application and platform architectures are integrated together.

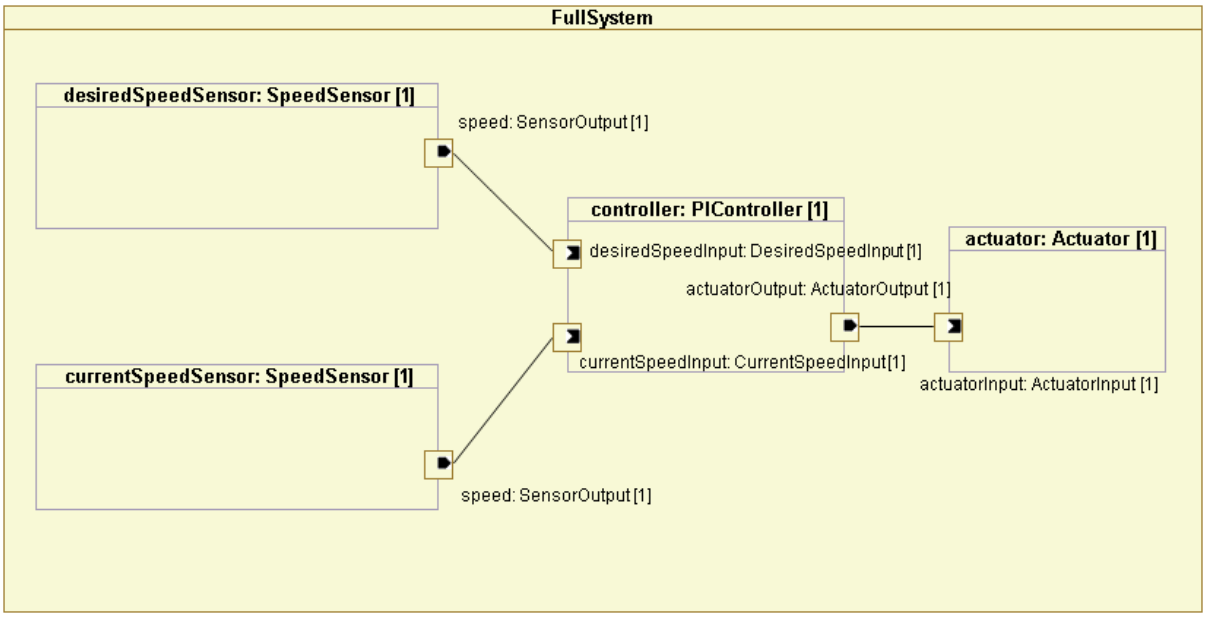

Figure 5-11. The cruise control system composite.

Figure 5-11 shows the cruise control system using UML2 and MARTE. The ports have been stereotyped with <<MessagePort>> stereotypes from GCM sub-profile that give the ports the semantics of being message based communications. The CGM sub-profile also allows modelling data streams. In order to do so we would use the <<FlowPort>> stereotype instead.

5.5.2 Syntactical view

The syntactical view of the application describes how the services are accessed. A syntactical description includes

a description of the messages involved in the access of a certain service,

a description of the communication protocols used,

the operational modes of the applications (e.g. different QoS, emergency modes, etc.), and

non-functional and quality properties related to message, communication protocols and operational modes.

The syntactical view of a service is often mixed with its structural view since service syntaxes are always related to structural elements. For example, messages are related to service interfaces and operational modes are related to the jobs that execute the service.

In this kind of applications the syntax of a service is defined by the signals/messages that are exchanged by service-users and services and by the order in which these signals and messages are sent from service-users to services and vice versa.

To show an example of modelling the syntax of a service we will use the example of an application server. The structure of this server is depicted in Figure 5-12. As depicted in the figure, the server admits three different kinds of messages: a StartConnection message, a Data message and a CloseConnection message. The server also uses two messages to answer the clients: Ack and Nack. The protocol is defined using a UML2 state machine diagram that is pointed by the "operationalMode" property of the <<RtUnit>> stereotype (Figure 5-13). We also add a property to the server in other to keep track of its current state.

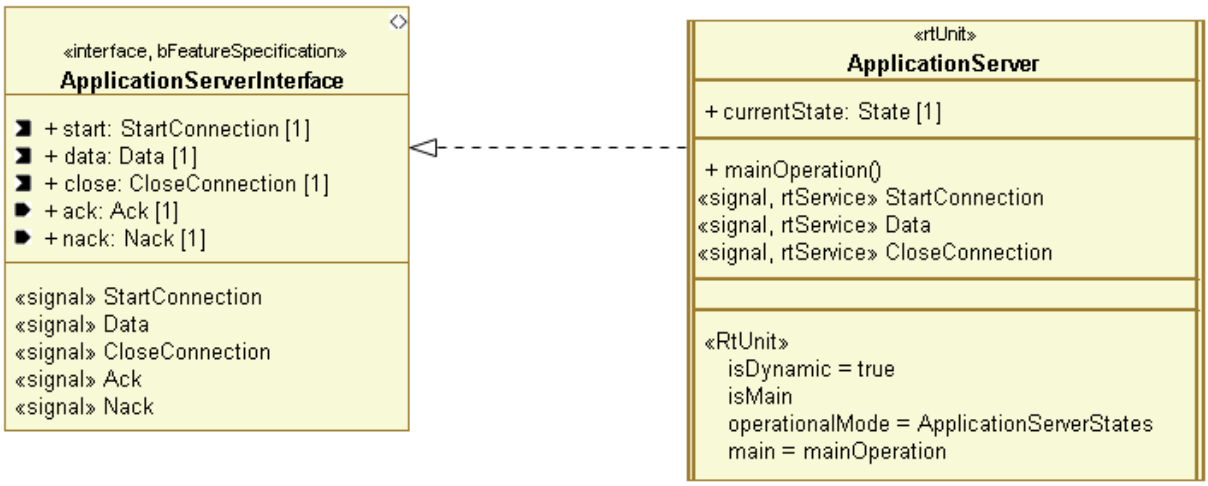

Figure 5-12. Structure of an application server.

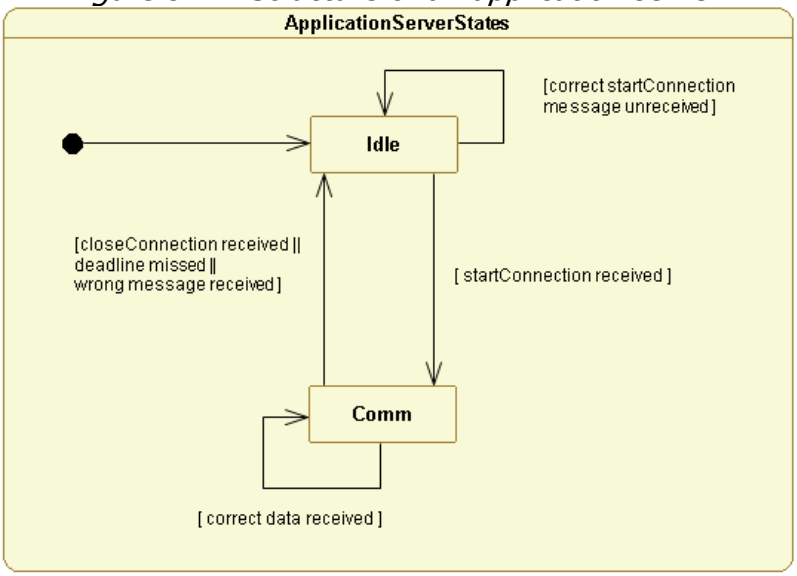

Figure 5-13. State machine diagram of the application server protocol.

From a syntactical point of view, clients have to be aware of the structure of the messages they must send to the server in order to interact with it. Messages, as

already has been shown, are modelled using UML2 signal elements. Figure 5-14 shows the messages involved in the current example.

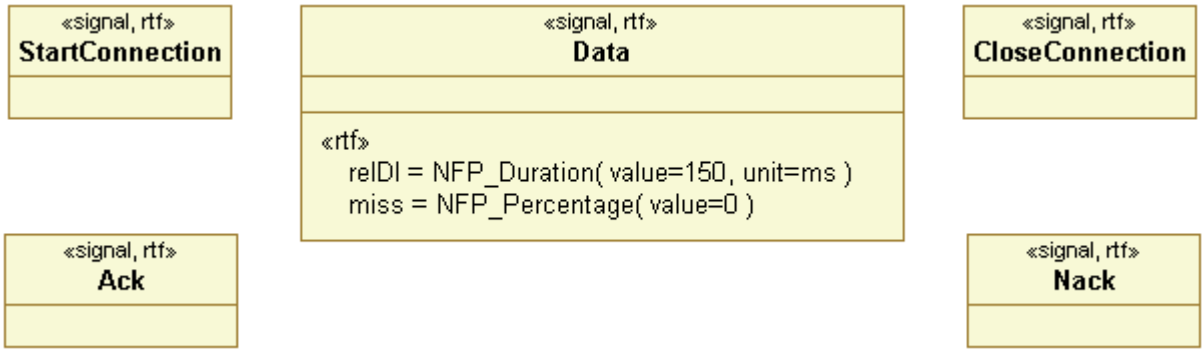

Figure 5-14. Messages involved in the application server example.

In Figure 5-14, UML signal had been stereotyped using <<RtFeature>> from the MARTE HLAM sub-profile. The <<RtFeature>> stereotype adds timing constraints that must be accomplished at the interfaces that produce/consume each of the messages (i.e. different deadline descriptions, deadline miss ratios, priorities and arrival patterns). Table 5-1 shows all the properties of the <<RtFeature>> stereotype in the HLAM sub-profile.

Property	Type	Multiplicity	Description
utility	MARTE_Library:: UtilityType	[0..1]	An abstract type. It must defined by the user. This type enables MARTE to include a semantic description of this service.
occKind	MARTE_Library:: BasicNFP_Types:: ArrivalPattern	[0..1]	This property describes the occurrence pattern for the arrival of this element.
tRef	MARTE_Library:: TimedObservations:: TimedInstantObservation	[0..1]	This property describes a reference time that will be used for relative time measures.
relDl	NFP_Duration	[0..1]	Relative deadline.
absDl	NFP_DateTime	[0..1]	Absolute deadline.
boundDl	NFP_BoundedDurati	[0..1]	Bounded deadline.

	on		
rdTime	NFP_Duration	[0..1]	Time used by the current element to perform its work.
miss	NFP_Percentage	[0..1]	Maximum admissible deadline miss percentage.
priority	NFP_Integer	[0..1]	The priority of this communication.

Table 5-1. Properties of <<RtFeature>> stereotype

As can be seen from the table, no property defined by the <<RtFeature>> stereotype is mandatory. The properties can be used depending on the designer's need for expressivity.

When the <<RtFeature>> stereotype is applied to signals it enables us to specify the frequency at which a service has to be accessed. The property occKind is typed as ArrivalPattern. ArrivalPattern is a MARTE <<choice_type>> which means that it can be assigned any element typed with:

PeriodicPattern. This datatype describes the parameters of a periodic occurrence (i.e. period, jitter and phase).

AperiodicPattern. This abstract datatype describes an aperiodic arrival pattern defined by a statistical distribution.

SporadicPattern. This datatype describes a special aperiodic pattern where the time between occurrences has some kind of bound.

BurstPattern. This datatype describes a special aperiodic pattern where occurrences happen in bursts. The time interval between bursts as well as the time interval between occurrences in a burst is bounded.

IrregularPattern. This datatype describes a special aperiodic pattern where occurrences don't follow any kind of periodicity. The occurrences are described as an array of inter-arrival times.

Specifying these patterns in a message (i.e. in the interface of a service) gives accessing clients information about the timing constraints needed to access the service. Despite this kind of information is not needed in the application server example; it is very useful to describe control or multimedia systems which are much more coupled with time.

5.5.3 Behavirour view

The behavirourl view of an application describes the control flow between jobs and applications. It is possible that many (implementation) constraints appear in

behavirouralviews, since it is common to use variables, function-calls, etc. in them. The behaviour view is very important in the early validation phase since it provides a means to test the system's functionality and evaluate that the system fulfils its quality requirements related to applications. The behaviour view is also crucial in the system realization phase, since the behaviour described in this view is what the developers will implement in the final product.

The HLAM sub-profile of MARTE provides the designer with a series of stereotypes to make some behavioural aspects present when describing the services and applications. The behavioural aspects supported by the HLAM stereotypes include quality of service (QoS) specification and execution, and concurrency and synchronization aspects description.

The operations which support the services inside RtUnits can be given information about their behaviour. The HLAM sub-profile provides designer of the <<RtService>> stereotype to model how the servers react to incoming invocations. Figure 5-15 shows the way this stereotype is used in the context of the application server example.

The RtService stereotype is applied to the signal receptions in the interfaces of the server RtUnit. In order to add a finer grain description of the behaviour of the RtUnits or their interfaces it is necessary to use activity diagrams, sequence diagrams or state machine diagrams. Figure 5-15 shows the activity diagram of the main operation of the controller RtUnit of the CCS example. The diagram shows that the controller starts up the system and then enters an endless loop in which it only performs the control algorithm on the speed samples provided by the sensors and then it sends a message to the engine actuator through the actuator output interface.

Using the MARTE stereotypes in this kind of diagrams increases their expressivity including extra information. The <<RtFeature>> stereotype, described before, adds timing information to the actions. On the other hand, the <<RtAction>> introduces information regarding signal sending and reception.

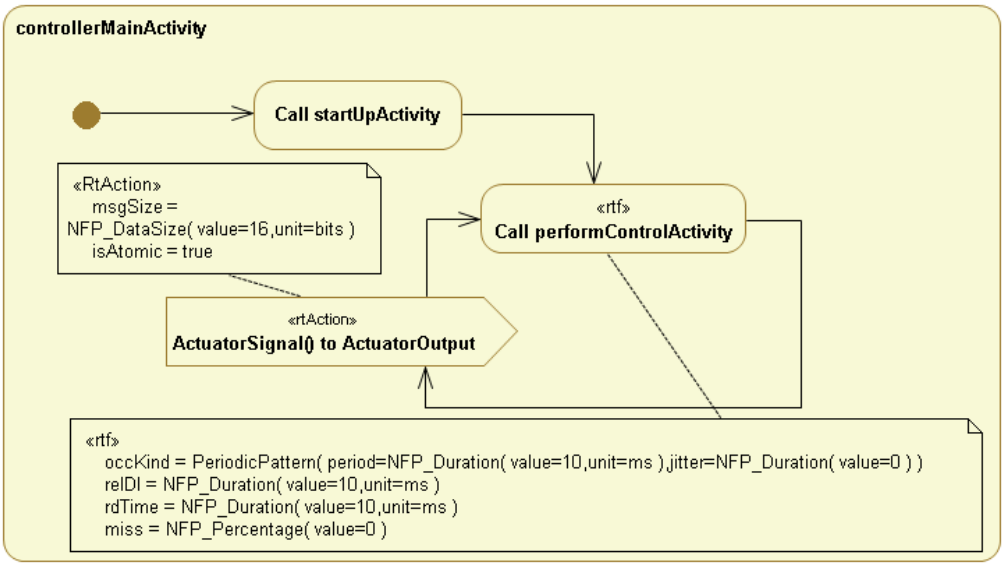

Figure 5-15. Activity diagram of the main operation of the cruise control system controller.

The <<RtBehavirour>> stereotype is used in behavioural UML diagrams modelling how an RtService behaves regarding to invocation queues. It can be used in any kind of UML behaviour diagrams (state machines, activities and interactions). Figure 5-16 shows the activity diagram of the current speed signal reception.

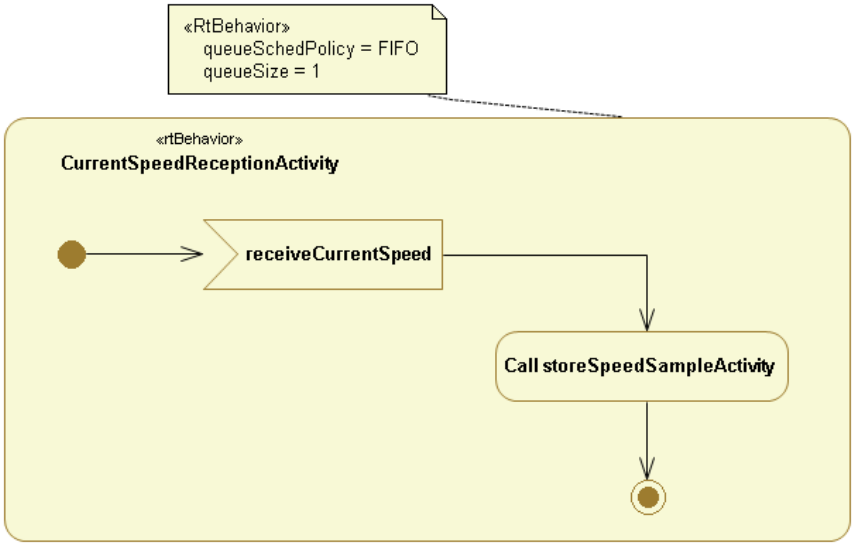

Figure 5-16. Current speed reception activity of the CCS controller.

The RtBehaviour stereotype is very useful to model services that prioritize some invocations from others regarding their real-time parameters and QoS.

5.5.4 Semantic view

The semantic view describes the meaning of the services used in application design. Semantic information is required of:

Functionality provided by the service,

Quality properties of the service,

Meaning of information/data provided by the service,

Usage constraints of the service, and

Context of a service, if it functional or quality properties can change according to the context.

Although MARTE profile has great expressivity to describe real-time embedded applications, it does not include immediate mechanisms to distinguish a service from another from a semantic point of view apart from the plain service name.

Semantic information is often very close to ontologies and taxonomies. In order to use MARTE to fully describe the services of the eDIANA platform it is necessary to define an ontology of the services that an embedded application can request/provide at the different integration levels. Once the ontology is defined, MARTE can be extended or adapted to support the inclusion of this ontological information.

The <<RtFeature>> stereotype, defined in the HLAM sub-profile of MARTE, has been widely used throughout this document to describe the eDIANA devices and applications. <<RtFeature>> includes a property called "utility" that may enable to add the semantics to service interfaces by specifying it in the <<RtFeature>> stereotypes applied to the signals.

The UtilityType is defined as an abstract type in MARTE so that it can be refined into user defined types. By extending this stereotype it is possible to include information regarding reference ontologies, categories, etc. into MARTE compliant models.

5.6 System Allocation

The System Allocation phase of the eDIANA process model is related to the mapping of the applications to the platform architecture elements that will support their execution.

This phase includes an allocation view, the platform architecture configuration view and additional information, e.g. probabilities of state transactions, needed for quality evaluation purposes.

The allocation view defines how applications and services are deployed on the computing and communication resources provided by the execution platform. Typically, platform architecture needs to be configured that is made by parameters. Additional information required for specific evaluation methods is provided by adding the required information to the models provided by the earlier design phases. An

allocated system contains all the necessary information to implement the final product. If the vertical model transformation is supported, simulation and target code can be generated from the validated system architecture models.

The MARTE profile includes a specific sub-profile Alloc that allows a designer to specify which application elements will be associated to which platform resources. In this section we will use again the cruise control system (CCS) example to illustrate allocation modelling in MARTE. Figure 5-17 depicts the platform model that will support the execution of the controller of the system.

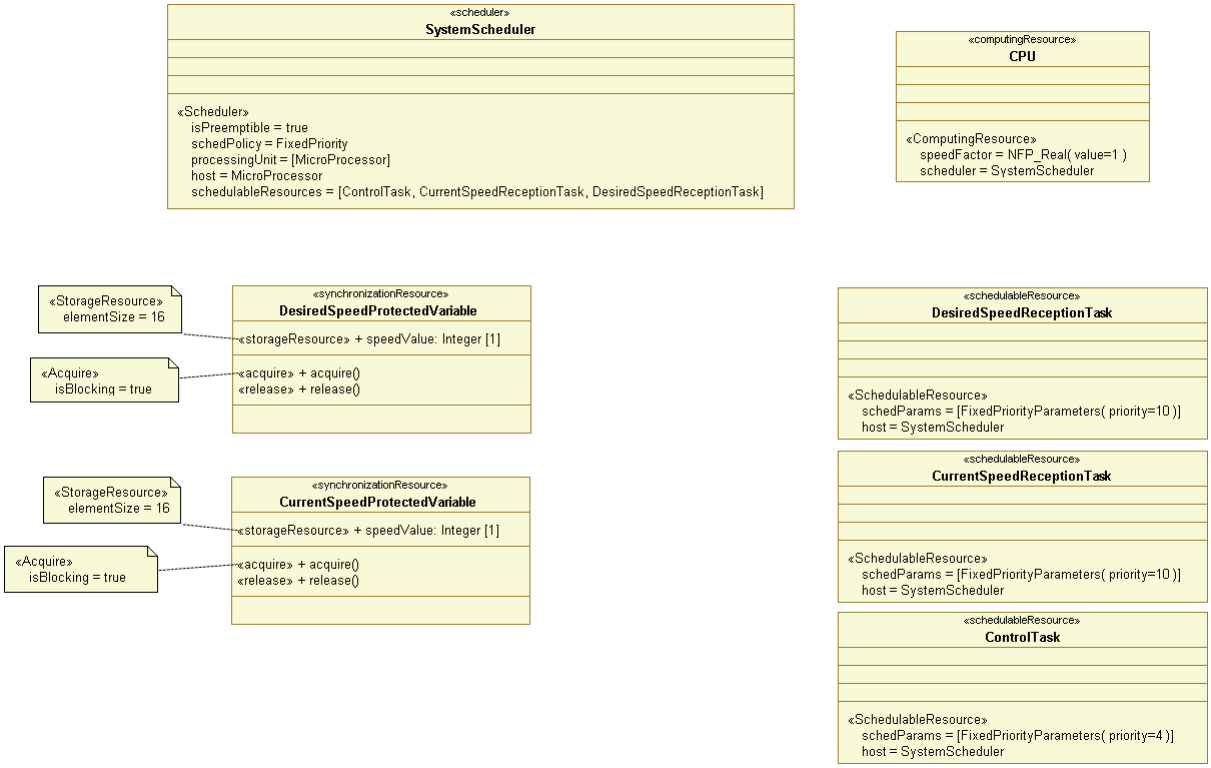

Figure 5-17. Platform model of the CCS controller.

The platform model of the cruise control system consists of a CPU managed by a system fixed priority scheduler. Three threads have been defined, all of them hosted by the system scheduler. Lastly, two shared protected variables have been defined, each of them with a blocking call for acquiring and releasing the variable lock (i.e. a mutex).

The allocation is performed using the structural views of both application and platform models and using the <<Allocate>> stereotype on UML abstraction dependencies. The <<Allocate>> stereotype allows further describing the nature and kind of the allocation as well as any constraints to be applied during the allocation process. Additionally both application and platform elements are stereotyped with <<Allocated>>.

Figure 5-18 shows the structural view of the application model allocated on top of the structural view of the platform model. Each of the operations and receptions in the controller has been allocated on the three threads and the passive protected units have been mapped to mutex-protected variables.

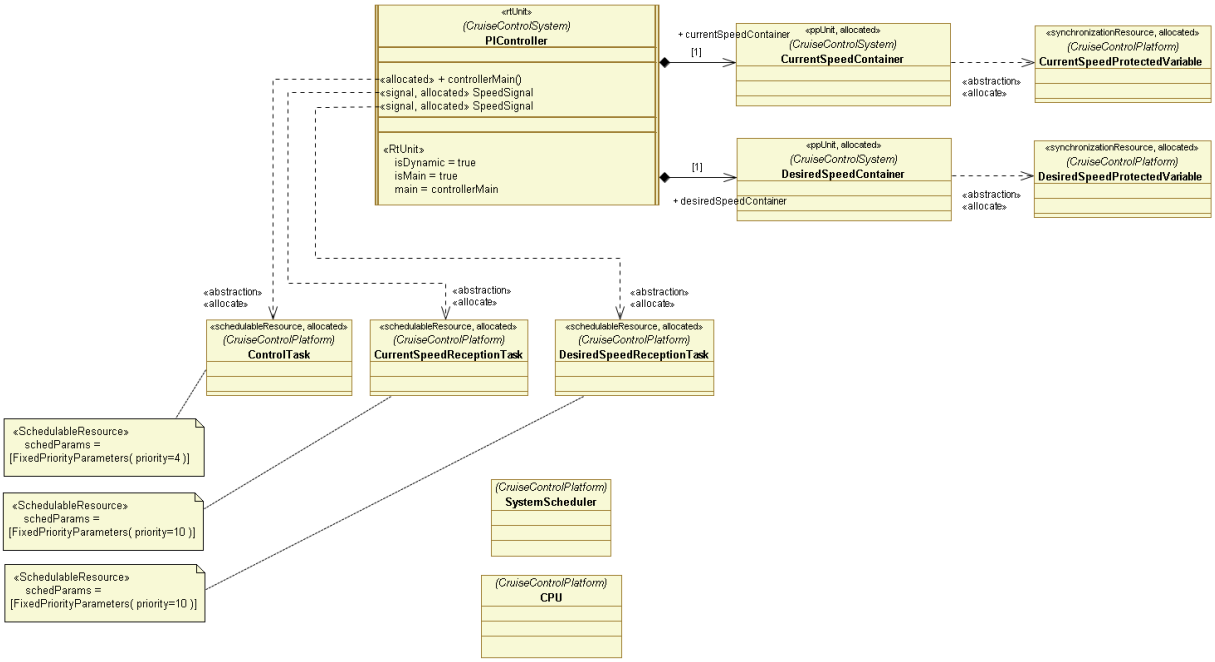

Figure 5-18. Allocated model of the CCS controller.

5.7 Method selection and adaptation (Process configuration)

One of the challenges of software engineering practice is to provide "configurable methodologies and process standards" [6]. This way is possible to configure methodologies to provide support for agile methods and processes for small development teams.

Software development is usually performed in small development groups, usually with less than four participants and sometimes even is in charge of one single person. There is a need for providing guidance to select and adapt methods and models in the eDiana methodology.

MARTE is very complex and have a lot of sub-profiles and models. In the eDiana methodology, some models and diagrams are proposed and this method will help to understand better the purpose of each model and to know when to use which one. For instance, for validating scheduling using a specific tool we need some specific models.

The adaptation method is a set of guided steps to select and adapt the eDiana methodology. The adaptation should not be time-consuming for the developers, and provide value to the development. The adaptation is based on the following aspects:

Usage: the models to be documented are those that are necessary in the lifecycle of the system. Each stakeholder describes his interest and which information requires. Among the stakeholders theoretically interested in models in eDiana system development are the Domain expert, Software architect, Platform architect, Model analyst, Tester and MDE expert.

System characteristics: Both the *software/hardware nature* and the *quality attributes* that the system must fulfil are determinant in the selection of the models to be documented and also for selecting the validation method and tool. The system to develop can have only software if there is no hardware part or the execution platform is provided or both software and hardware. It is important to consider if there are relevant quality attributes and which are the high-priority quality attributes because selected models and validation method must support the validation of those quality attributes.

Modelling objective: The reason of modelling and the use of the models can be different: for documenting the system for new employees, as a mean of communication among stakeholders, with code generation purposes or in order to validate quality attributes.

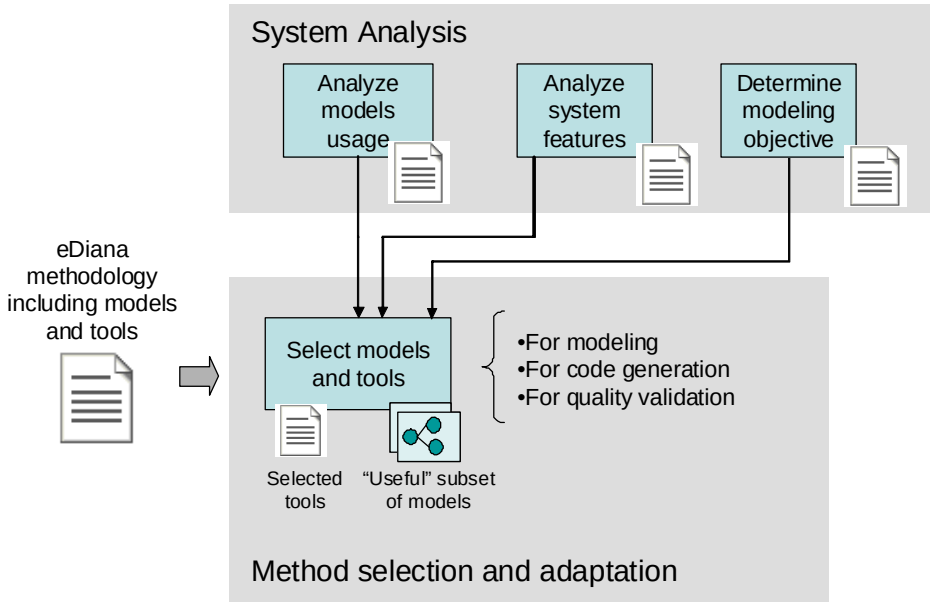

Figure 19: Steps of the adaptation method

5.7.1 . Analyze models usage

In the development of a software application different stakeholders or roles are involved and each one can have different level of interest in the models. In eDiana system development the following main roles are identified:

Domain expert: A domain expert is a person who has specific domain knowledge about the problem that is to be solved by the system and who model this knowledge.

Software architect: A software architect is the person in charge of designing the software architecture of the application.

Platform architect: A platform or hardware architect is the person in charge of modelling the platform architecture that supports the software applications

Model analyst: A model analyst is the person who is concerned with quality validation, who is in charge of annotating system models and performing model based validation.

Tester: A tester is the person who is in charge of testing the system.

MDE expert: A MDE expert is a person who has knowledge about model driven engineering practices: definition of transformations, definition of Domain Specific Languages (DSL), selection of modelling languages, views and methodologies, etc. The MDE expert will select which models to use in each case depending on the application to develop and actors involved.

In the Table 2, the interest of each of the models of the eDiana methodology for the stakeholders is specified. This interest can be high, medium or low.

The analysis model is not mentioned in the eDiana methodology but this model will be used for V&V and the way of describing will be defined in WP6.

Table 2: Interest of each role in the models

Models / Stakeholders	Platform Architecture model	Application Architecture model	Allocation model	Analysis model
Domain expert	-	High	-	-
Software architect	Low	High	High	High

Platform architect	High	Medium	High	Medium
Model analyst	High	High	High	High
Tester	Low	High	Medium	High
MDE expert	High	High	High	High

5.7.2 Determine modelling purpose

The reason of modelling and the use of the models can be different. Four different goals are distinguished:

Document the system: The main use of this documentation will be as a means of education. New employees or external stakeholders will be able to understand the system.

Communicate: The models are used as a vehicle of communication among stakeholders, they are the basis for implementers, testers, maintainers, analyst…

Generation: In Model Driven Development models are not only used for documentation or as a mean of communication, are the central point and drivers of the development. Using the models code generation can be performed.

Analyze / validate: Models are uses as the basis for system analysis. The models must be annotated with the information necessary for the particular analysis that will be performed.

5.7.3 Analyze system features

Both the *software/hardware nature* and the *quality attributes* of the applications must be considered.

Not all the applications to develop in eDiana will be the same. In same cases, only software will be developed whereas in other cases, all the system (SW + HW) will be the target:

Only Software: Applications where only software is designed and there is no hardware part or the execution platform is provided and use as it is.

System (SW + HW): Applications where both software and hardware (the execution platform) are designed and modelled.

Moreover, in some systems quality attributes and their assurance will be critical such as in safety critical or real time systems, whereas in other systems quality attributes are not relevant. This way, two classifications have been made:

With relevant quality attributes: In many systems such as safety critical or real time systems, quality attributes are a key aspect to consider. In those systems to validate those quality attributes is almost always required.

With non relevant quality attributes: In this case, systems have not relevant quality attributes to take into account. And to annotate and analyze models is not necessary.

In the next table, information is provided about the modelling goal and system's features in order to help to select the most appropriate models of eDiana methodology in each case.

Table 3: The eDiana methodology models and their purpose, target system and required experience

	Platform Architecture model	Application Architecture model	Allocation model	Analysis model
Modelling purpose				
Document the system	X	X	X	-
Communicate	X	X	X	X
Generation	X	X	X	-
Analyze / validate	-	-	-	X
System's features				
Only Software	-	X	-	
System (SW + HW)	X	X	X	
With relevant quality attributes				X
With non relevant quality attributes				-

6. Tooling support and integration

This section will provide an overview of existing languages, methods and tools applicable in the context of eDIANA. These tools are provided in tables separated by categories.

6.1 Requirements management tools

Tool	Description and Features	License
StarUML	StarUML in an open source project that has as its goal be a powerful modeling software and serve as an alternative to commercial UML tools. Main features: • Support for multiple languages. • Generates documents supported by the Microsoft offices. • Supports technology MDA. • Optimizes the generated code. • Is repeating given their status as open source. • Supports certain patterns (course, EFB patterns).	Open Source. GNU General Public License (GPL)
CASE Spec	Software developed by Goda software which allows to specify, analyze, verify and validate systems. Main features: • Allows you to sort the data through hierarchical structures. • Allows any artifact use cases (cases of evidence, etc.) modeling. • Establishes relations two to two between artifacts. • Automatic version control. • Automatic generation of documents specification with diagrams and objects. • Establishes links between data and physical files stored in the system. • Allows the concurrency of users as well as establish groups. • Allows exports and imports information. • Generate history (log) reports. • Manages user access control.	Open Source. GNU General Public License (GPL).

Tool	Description and Features	License
Open Source Requirements Management Tool	Tool designed to cover all the the software development lifecycle (SDLC). In this life cycle is include the Analysis of requirements, design, implementation and testing. Main features: • Has versioning. • Allows to define derived requirements. • Allows to define attributes for requirements such as the risk, effort, etc. • Allows to represents both use cases and test cases.	Open Source. GNU General Public License (GPL).
IRQA4	Tool developed by Visure and has the goal of serve as application to provide a comprehensive support in a project software requirements engineering. In addition to include most basic tasks (capture, analysis, modeling, organization and follow-up), requirements engineering this application you have the following features: • Requirements reuse: allows that the requirements defined in a project can be used in other projects have been made by the Organization through the use. This is achieved offering a small advantage of perform product lines. • Documentary view: this new option offers a pooling of requirements that allows the user to see a clear distinction between them and facilitating all work related to these. • Requirements engineering: Besides the management requirements, this application provides functionality related to the engineering requirements, allowing a single tool centralize all activities related to the requirements (including validation and acceptance tests).	Commercial
Telelogic Doors	Telelogic doors is a cross-platform system designed for the management of requirements by capture, traceability, bound, analysis and management of	Commercial

Tool	Description and Features	License
	changes that they occur. Main features: • Provides a collaborative environment management requirements. • Analysis of traceability to identify risk areas. • Easy management of changes in requirements. • Allows to manage the traceability requirements easily by drag-and-drop between screens items. • Allows to manage a large number of efficiently through a simple (high scalability) database requirements.	
GatherSpace	It's a web application to work in a collaborative manner. The most important features are: • Traceability Matrix Reports: Provides a view of dependencies between features, requirements use cases and test cases. • Issue Management: Includes the capability to manage issues/bugs and associate them back to features and requirements. • Test Case Management: A highly requested feature was to manage and associate test cases to requirements and use cases. • Use Case Modeling: Provides a unique way of depicting the use case model in simple HTML view. This report produces a high level use-case model on the fly. • File Attachments: When defining requirements and features, you can associate images, spreadsheets and other docs and have them printable in the reports.	Commercial
IBM Rational RequisitePro	Is considered one of the more comprehensive and powerful analysis and requirements management tool. One of the advantages is that can easily be integrated with popular programs ie. Word, as well as with most used database systems allowing have a central data repository. It also allows work for Web access can be accessed in a distributed manner. Have an array for the follow-up to the requirements that can represent both graphically and textual form.	Commercial

6.2 Modelling languages and tools

Tool	Description	Supported Languages	Open Source?
Papyrus	Papyrus is a UML2 open source modelling tools with very good support for UML profiles. Among others, Papyrus implements the SysML, MARTE, CCM and LwCCM profiles natively.	UML2 + Profiles	Yes
TOPCASED	TOPCASED is a huge modelling project over Eclipse and the EMF. TOPCASED provides the tools for creating editors for DSL and also provides native editors for a set of modelling languages (e.g. UML, SysML, AADL, SAM, etc.). UML profiles are supported but not natively. No UML profiles are provided with the tool.	Many	Yes
Visual Paradigm for UML	Visual Paradigm for UML is a UML CASE Tool supporting UML 2.1 and the Business Process Modeling Notation (BPMN). In addition to UML modeling support, it provides business process modelling, an object-relational mapping (ORM) generator for Java, .NET and PHP.	UML2	Yes
Poseidon for UML	Poseidon for UML is a UML CASE Tool supporting UML 2.1 The Embedded Enterprise Edition is specifically designed for embedded systems development.	UML2	No
IBM Rational Software Architect	IBM Rational Software Architect is a modelling and development environment that leverages the UML for designing architecture for C++ and J2EE applications and web services.	UML2	No
Rational Rose	Rational Rose is a commercial tool for analysis,	UML2 + MAST	No

	modelling, design and construction.	Profile	
MDDi	The Eclipse MDDi project is dedicated to the realization of a platform offering the integration facilities needed for applying a MDD approach. The MDDi platform will provide the ability to integrate modelling tools, as well as tools supporting other technological spaces, to create a fully customizable MDD environment	Various modelling languages (UML, Domain-Specific Languages)	Yes

6.3 Model transformation tools

Tool	Description	Supported Transformations	Open Source?
MOFScript	MOFScript is model transformation plugin for Eclipse and the EMF developed and supported by SINTEF. It supports both model-to-text and model-to-model transformation specification.	M2T, M2M	Yes
OAW	Open ArchitectureWare is a very powerful model transformation plugin. It is built on top of Eclipse and provides the widest range of model transformations possibilities.	M2T, M2M, T2M	Yes
ATL	ATL is a M2M technology and is part of the Eclipse M2M project. It can be combined with the Acceleo M2T technology.	M2M	Yes
Acceleo	Acceleo is an open-source code generation tool that has great integration with the Eclipse IDE and EMF-based metamodels. The tool has a strong emphasis on simplicity and ease of use.	M2T	Yes

6.4 Early Verification & Validation tools

Tool	Description	Test Kind	Open Source?
MAST	MAST is a model-based schedulability analysis tool for real-time systems. MAST is written in Ada and it provides a user environment for the creation of real-time models of the system under analysis.	Schedulability	Yes
Cheddar	Cheddar is a schedulability analyzer and simulation engine for real-time systems. It has been developed in Ada and it is integrated with the AADL modelling language through OCARINA.	Schedulability	Yes
BIP	BIP is a modelling language that enables the detection of deadlocks in application designs.	Deadlock detection	Yes
VERSA	The VERSA schedulability analyzer implements a process-algebraic approach to schedulability analysis for system threads under a wide range of scheduling disciplines and inter- thread dependencies for AADL models.	Schedulability	Yes
IUS	tool of CADENCE for cycle and/or event based simulation. This tool is used for the validation and verification of complex Multi Processor SoC.	Functional verification	No
IFx	IFx works on timed UML models written in the OMEGA profile with widely used commercial CASE tools like Rational Rose or I-Logix Rhapsody.	Deadlocks, timelocks, satisfaction of state invariants, timing constraints, control of scheduling policy, etc.	No
Furness Toolset	The Furness™ toolset integrates and enhances several open-source tools to create a single unified environment for design, analysis and	VERSA Schedulability Analyzer	Open Source Edition and also

	implementation of embedded systems.	Conformance test	Licenses are available
KRONOS	Kronos verifies timed automatons (UML models)	Schedulability	Yes (for academia)
STOOD	STOOD verifies UML 2.0 models and has also support for AADL 1.0 models	Real-time schedulability analysis	No

Conclusions

The GENESYS [2] methodology is proposed for the MDE modelling of the eDIANA project. GENESYS is an European research project (FP7-STREP) focusing on the development of a cross domain multi-level for embedded system.

The implementation of this methodology is supported by a language that allows modelling, designing and integrating the whole systems and applications of the eDIANA platform. The proposal language is the UML and a standardise profile of it for the real time embedded systems: MARTE (Modelling and Analysis of Real Time Embedded systems).

Actually the GENESYS methodology and UML+MARTE language are not a state-of-art for developing a complex system of systems that integrates MPSoC (Multi Processor System on Chip) as the eDIANA platform.

Moreover both the scope and the short timeline of the project, don't allow to allocate specific effort for improving the methodology and the language

For these considerations, GENESYS and UML+MARTE approach will be the guideline for the hardware and software development of the eDIANA platform but also specific methodologies and tools will be apply for overcoming the actual limitation of the proposal method.

Acknowledgements

The eDIANA Consortium would like to acknowledge the financial support of the European Commission and National Public Authorities from Spain, Netherlands, Germany, Finland and Italy under the ARTEMIS Joint Technology Initiative.

References

[1] ARTEMIS SRA Working Group. "ARTEMIS SRA Reference Designs & Architectures", 2006.
[2] The GENESYS project website, http://www.genesys-platform.eu/
[3] Hermann Kopetz. "Overview of the Architectural Style", GENESYS workshop, Munich, February 2009.
[4] Eila Ovaska, András Balogh, Sergio Campos, Adrian Noguero, András Pataricza, Kari Tiensyrjä & Josetxo Vicedo. "Model and Quality Driven Embedded Systems Engineering", VTT publications 705, 2009.
[5] Bruce Powell Douglas "Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems". Addison Wesley, 2002. ISBN: 0-201-69956-7.
[6] ITEA Information Technology for European Advancement, Technology Roadmap for Software-Intensive systems, 2nd edition, 2004