Bachelorarbeit

Dennis Fadljevic

Development of a Unified Modeling Process for the Analysis and Design of Systems, Software and Hardware based on UML 2.0 and SysML 1.0

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Dennis Fadljevic

Development of a Unified Modeling Process for the Analysis and Design of Systems, Software and Hardware based on UML 2.0 and SysML 1.0

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Bettina Buth
Zweitgutachter : Prof. Dr. rer.nat. Stephan Pareigis

Abgegeben am 13. Juni 2007

Dennis Fadljevic

Thema der Bachelorarbeit

Entwicklung eines vereinheitlichten Modellierungsprozesses für die Analyse und Auslegung von Systemen, Software und Hardware basierend auf UML 2.0 und SysML 1.0

Stichworte

Systeme, UML, SysML, Prozess, Modellierung, Systementwicklung

Kurzzusammenfassung

Die zunehmende Komplexität moderner Systeme erfordert den Einsatz neuer Entwicklungstechniken und Methoden. Die vorliegende Arbeit untersucht einen Modellierungsprozess der Firma EADS Deutschland GmbH, mit dem Systeme bestehend aus Hardware und Software mittels der Modellierungssprachen UML 2.0 und SysML 1.0 funktional analysiert und ausgelegt werden können. Basierend auf diesem Prozess werden neue Konzepte erarbeitet, die den Prozess optimieren und erweitern sollen. Hauptgegenstand dabei ist die Analyse der SysML Modellierungsmöglichkeiten, welche im bestehenden Prozess noch nicht berücksichtigt wurden, als auch speziell das Erarbeiten von Konzepten, welche die Transition von den Systementwicklungs-Aktivitäten zu den nachfolgenden Hardware- und Softwareentwicklungs-Aktivitäten verbessern sollen.

Dennis Fadljevic

Title of the paper

Development of a Unified Modeling Process for the Analysis and Design of Systems, Software and Hardware based on UML 2.0 and SysML 1.0

Keywords

Systems, UML, SysML, Process, Modeling, Systems Engineering

Abstract

The growing complexity of modern systems requires the application of new development techniques and methods. This thesis analyzes a modeling process of the company EADS Deuschland GmbH, which supports the functional analysis and design of systems consisting of hardware and software by means of the modeling languages UML 2.0 and SysML 1.0. Based on this process, new concepts will be elaborated with the goal to optimize and enhance the process. The main subjects here are the analysis of SysML modeling capabilities, which were not yet considered in the modeling process as well as the elaboration of concepts that shall improve the transition from the systems to the hardware and software development activities.

Acknowledgements

I want to thank Martin Kress for the help and support during the preparation of this thesis. In addition I want to thank Dr. Uwe Kühne for enabling me to get in contact with this interesting field of work and finally Prof. Dr. Bettina Buth for the support and feedback throughout my work.

Table of Contents

Acknowledgements **1**

Table of Contents **2**

List of Figures **3**

Abbreviations **5**

1 Introduction **6**

2 Scope and Background **8**

2.1 VM-GBV 9

2.2 Scope of Systems Engineering within the VM-GBV 13

2.3 Systems Engineering in the OPES4 Department 14

2.3.1 Tool-based Systems Engineering in OPES4 14

2.3.2 The OPES4 Systems Engineering Process 17

2.4 Functional Modeling Process 21

2.4.1 General Process Phases 21

2.4.2 Modeling Activities 23

2.4.2.1 Analyze System Functions 23

2.4.2.2 Refine BB Model 31

2.5 Systems Modeling Language (SysML) 36

3 Analysis and Concepts **39**

3.1 Problem Definition and Approach 39

3.2 Example Problem 40

3.3 Modeling of System Structure 40

3.4 Modeling of Items and Data 46

3.5 Data and Item Flow 52

3.6 Segment Handover 59

3.7 Modeling the Physical Architecture 63

3.8 Transition to Hardware/Software 74

4 Results **78**

5 Summary and Conclusion **79**

References **81**

List of Figures

Figure 2-1 The V-Model GBV [VMG2004] 10
Figure 2-2 OPES4 Systems Engineering Sub-Model [OP12007] 18
Figure 2-3 The two phases of functional systems engineering with UML/SysML 21
Figure 2-4 The "Analyze System Functions" Process Activity 22
Figure 2-5 The "Refine System Black Box Model" Activity 23
Figure 2-6 Analyze System Functions: Modeling Activities 24
Figure 2-7 Example of a Use Case Diagram 25
Figure 2-8 Two alternative ways to refine a use case 26
Figure 2-9 Example of a Use Case Scenario 27
Figure 2-10 Example of a Use Case Activity Diagram 28
Figure 2-11 Example of a Structure Diagram 29
Figure 2-12 Example of a State Chart 30
Figure 2-13 Refine Black Box Model: Modeling Activities 31
Figure 2-14 Example of a Decomposition Structure 32
Figure 2-15 Example of OpCon Allocation in an Activity Diagram 33
Figure 2-16 Example of OpCon Allocation in a Sequence Diagram 33
Figure 2-17 Example of a System Architecture in a Structure Diagram 34
Figure 2-18 Example of a Segment State-based Behavior 35
Figure 2-19 Relationship between UML and SysML 37
Figure 2-20 SysML Diagram Types 37
Figure 3-1 SysML Block Definition Diagram 42
Figure 3-2 SysML Internal Block Diagram 43
Figure 3-3 Block Definition Diagram showing the DIRCM White Box System Structure 44
Figure 3-4 Internal Block Diagram showing the DIRCM White Box internal System Structure 45
Figure 3-5 SysML ValueType Concept 48
Figure 3-6 Modeling Complex Types as Block 49
Figure 3-7 ValueTypes, Units and Dimensions in Rhapsody 50
Figure 3-8 ValueTypes, Units and Dimensions in the Rhapsody Browser 51
Figure 3-9 Service Oriented Approach to model Data Flow 52
Figure 3-10 Communication between Blocks with Ports and Interfaces 53
Figure 3-11 Continuous Item Flow with UML Ports 54
Figure 3-12 Continuous Item Flow with SysML Flow Ports 56
Figure 3-13 Flow Port Communication in Sequence Diagrams 57
Figure 3-14 Voltage Flow over a Flow Port 58
Figure 3-15 Flow Port Support in State Charts 58
Figure 3-16 Use Cases of the DIRCM System 61

Figure 3-17 Use Case Decomposition and Allocation 62
Figure 3-18 Functional vs. Physical World 64
Figure 3-19 Internal Block Diagram showing Logical Communication Structure 65
Figure 3-20 Logical Communication in a Sequence Diagram 65
Figure 3-21 Internal Block Diagram showing Physical Communication Structure 66
Figure 3-22 Physical Communication in a Sequence Diagram 67
Figure 3-23 Decomposition of the ProcessingComputer Block in the DIRCM Example..68
Figure 3-24 Carrier Hardware Architecture 69
Figure 3-25 CPU Block with Information Tags 70
Figure 3-26 CPU Deployment 71
Figure 3-27 Memory Deployment..72
Figure 3-28 Logical Communication in the Processing Segment (excerpt).......72
Figure 3-29 Interface Deployment...73
Figure 3-30 Functional Sequence on System Level.....................................75
Figure 3-31 Decomposition of a System Block into Classes 76
Figure 3-32 Decomposed Functional Sequence in the Software Model...........77

Abbreviations

SysML	System Modeling Language
UML	Unified Modeling Language
CONOPS	Concept Of Operations
BWB	Bundesamt für Wehrtechnik und Beschaffung
VM-GBV	V-Modell Geschäftsbereich Verteidigung
RAMS	Reliability, Availability, Maintainability, Safety
NASA	National Aeronautics and Space Administration
HW	Hardware
SW	Software
ILS	Integrated Logistic Support

1 Introduction

The complexity of today's system is constantly growing. Beyond doubt, the main drivers are innovation pressure and the ever growing technical advances, which build up each other in a cycle. A car for example could be seen as a predominantly mechanical system only thirty years ago. Today, a car is a highly complex electro-mechanical system, consisting of a large number of subsystems and housing about 100 processors that all act together in technical combination.

To handle this complexity, new approaches, methods and processes for the development of systems are required. One major point is the ability to describe component based systems with a common design language to facilitate the exchange of designs between the different development teams. This is especially important when teams from different engineering disciplines, such as mechanical, electric, software or even chemical engineering are involved in the product development. Another key concept is the ability to simulate and verify system designs in the early development stages. Design faults on the overall system level in most cases have severe impacts on both project cost and schedule when identified too late and may even lead to the complete failure of the project.

The EADS department OPES4 within the business unit *Defence Electronics* develops complex integrated avionics systems, consisting of both hardware and software. For the reasons mentioned above, the department has developed a systems engineering process, which pursues a model based approach for the functional analysis and design of systems based on the *Unified Modeling Language* (UML) and the *Systems Modeling Language* (SysML). The UML/SysML is used together with the model-driven development (MDD) tool *Telelogic Rhapsody*, which allows the execution and thus verification of UML/SysML models based on the semantic UML/SysML metamodel. By following this approach, the department expects a significant improvement in the quality of captured requirements, system specifications and also the communication with the subsequent hardware and software development teams. This seams especially promising for the communication with software development teams, given that the UML has been the de-facto standard modeling language in the software engineering discipline for several years and has also been applied successfully to several projects within the software department in OPES4 in the past.

However, the modeling approach is still in its infancy and certain aspects were not yet considered at the beginning of the bachelor thesis. One of it is the transition from systems engineering to the subsequent development of hardware and software. While the modeling approach was self-contained within the systems engineering activities, no detailed concepts existed to support a seamless development transition between the en-

gineering phases and teams. This is one of the goals of this thesis. The thesis will analyze the current approach and develop possible concepts for the transition process.

The current approach uses a mixed set of UML and SysML model elements. This is primarily because the used modeling tool Telelogic Rhapsody is intentionally a UML tool for software engineering and does only support fragments of the SysML language elements natively. However, especially when exchanging designs between different departments or even companies, it is necessary to develop the designs based on a concise language definition. Therefore, the usage of SysML only as modeling language is to aspire. Telelogic claims that Rhapsody offers extensive customization capability, thus the second goal of this thesis is to analyze if the tool can be adapted to use the SysML language specification. Based on this analysis, another thing to consider is the extensions that the SysML offers for the modeling of systems. The current modeling approach features only the SysML language elements that are also part of the UML. SysML concepts like the support for modeling item and data flows and the ability to model continuous systems were not yet considered. Thus, an additional goal of the thesis is to analyze the possible application of SysML enhancements to the current modeling approach. The benefits that the SysML enhancements promise for the modeling of systems will be analyzed and the current modeling approach will be adapted and extended if applicable.

Chapter 2 will give an overview of the technical background in that the general development process as well as the current modeling approach will be described. After that, a short introduction to the SysML is given in chapter 2.5. Chapter 3 will then analyze the current modeling approach and elaborate concepts that address the problems stated above. In chapter 4 it will then be described, how the elaborated concepts affect the current modeling approach. Chapter 5 summarizes the thesis and provides an outlook over possible future consecutive topics.

2 Scope and Background

The development of complex systems requires the usage of clearly defined development processes to structure the overall development activities and to maintain a certain quality. This is especially important when mission or safety critical systems, which avionics systems in most cases are, are to be developed. The development processes that have to be used are generally mandated by the customer. Development processes in general describe primarily *what* has to be done in terms of documents that have to be generated in certain development phases and reviews that have to be performed before the transition can be taken from one development phase into another. In addition, the processes also often give advice about methods and tools that can be used to aid the development activities. However, these methods and especially tools are often not suited to the existent contractor's development environments or may just not consider state-of-the-art methodology, especially if the process description is several years old. Thus, when wanting to use new sophisticated methods and tools, such as UML/SysML and Rhapsody in the case of OPES4, an adapted development process has to be created that can be mapped to the general processes mandated by the customer.

For the OPES4, the primary customers are the EADS division *Airbus* and the German federal office for procurement, the *Bundesamt für Wehrtechnik und Beschaffung* (BWB). Airbus mandates the company-specific *RBE* (Requirements Based Engineering) process for its contractors while the BWB demands the application of the German *V-Modell*. However, for contracts with the BWB, the EADS *Defence and Security* Division, to which the OPES4 belongs, has created a company-specific process model that is based on the V-Modell. This model is called *VM-GBV* (V-Modell Geschäftsbereich Verteidigung). Because these processes set the main context for the development activities in OPES4, it would generally make sense to describe both. A description of both processes however would go beyond the scope of this thesis, therefore only the VM-GBV as an example development context will be further described in this document. This will be done in chapter 2.1.

The VM-GBV is an overall development process that considers all different aspects and roles within a whole product development lifecycle. However, the modeling approach discussed in this thesis is primarily in the focus of the systems engineering portion of the overall process. Chapter 2.2 therefore points out the scope of systems engineering within the VM-GBV.

The OPES4 has created a department specific process model for the systems engineering activities within it. This process defines the overall systems engineering activity workflow as well as the used methods and tools. This process also sets the scope of the

modeling approach with UML/SysML and Rhapsody. Therefore it is necessary to explain it in more detail, which will be done in chapter 2.3. The actual modeling approach will then be described in chapter 2.4. There the single modeling activities will be described in detail. Finally, an overview of the SysML language will be given in chapter 2.5.

2.1 VM-GBV

As already noted in the introduction to chapter 2, contractors of the BWB have to follow the German V-Modell for the development of products. The V-Modell currently exists in three different versions [VM2007]:

- **V-Modell**: This is the initial version of the development process. It was developed in 1986 for the German military and also was introduced to civil institutions in 1993.
- **V-Modell 97**: This is a revision of the original V-Modell, which among other things considers object-oriented approaches to software development. It was released in 1997.
- **V-Modell XT** (XT = Extreme Tailoring): This is the most recent version from 2005. It offers broad support to tailor the process to the individual needs of different projects and also provides process modules for the customer. The process is oriented more towards an agile and incremental approach.

Because the V-Modells have a strong focus on software development, the EADS Defence and Security company division felt the need to define an integrated development process that unifies the development of software, hardware and logistic aspects of the system into one single process. The result was a company-specific process model, called *VM-GBV* (V-Modell Geschäftsbereich Verteidigung). This process is based on the original V-Modell 97 and thus maintains compliance for the development of military products for the BWB. Since completion and introduction, the VM-GBV is the standard process for product development in the whole Defence and Security division for contracts with the BWB.

This chapter will give an overview of the VM-GBV and describe the various aspects of it that are relevant for the scope of this thesis. However, because of the extensive nature of the process, only the very basic concepts of the process can be shown. The information about the VM-GBV is based on the documents [VMG2004].

The process is structured into several phases, as shown in Figure 2-1.

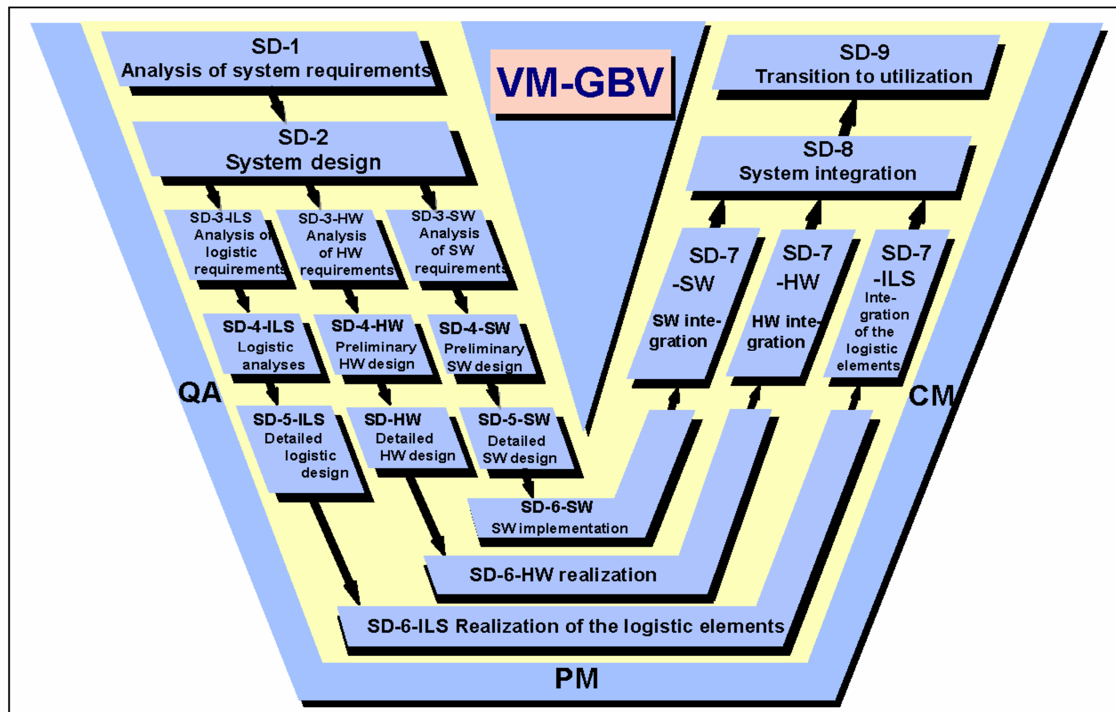

Figure 2-1 The V-Model GBV [VMG2004]

The VM-GBV can be divided into four sub models, System Development (SD), Quality Assurance (QA), Project Management (PM) and Configuration Management (CM). However, for the scope of systems engineering, only the system development sub-model is of primarily interest and thus described in more detail.

The system development sub-model again comprises four sub-models, the sub-model *System Development (SD)*, *Software Development (SD-SW)*, *Hardware Development (SD-HW)* and *Integrated Logistic Support (SD-ILS)*. The sub-models are defined by several development phases. The VM-GBV specifies the generation of certain output documents in each development phase; however, these are not further described in this thesis because they are not relevant for the overall context.

It has to be remarked that although in Figure 2-1 it seems that the phases follow a fixed timely sequence, the VM-GBV doesn't mandate it. The sequence can be rather considered as a rough logical order.

A brief description of the sub-models and its various phases various phases is given below.

System Development (SD)

The sub-model SD describes the activities which are necessary for the system development at the levels of the overall system and the segments.

- **Analysis of system requirements (SD-1)**

  The requirements of the system to be constructed and of its environment are captured. Possible threats and risks are analyzed and a security plan is built. A technical model for functions, data and objects is elaborated. The requirements of the logistic support system are described.

- **System design (SD-2)**

  The system is structured into segments, SW units and HW units. The technical requirements of the overall system, its segments, its SW units, its HW units and on the logistic support system and its logistic elements are described. All external and internal interfaces are described and the draft of a logistics plan is created.

- **System integration (SD-8)**

  The various SW units and HW units are integrated into segments. The segments and the logistic support system are integrated into the overall system.

- **Transition to utilization (SD-9)**

  The system is put into operation at the desired location.

Software Development (SD-SW)

The sub-model SD-SW describes the activities which are necessary for the development of SW portions of the overall system.

- **Analysis of SW requirements (SD-3-SW)**

  The requirements on a SW unit and environmental constraints, defined at SD-2 or at the system architecture, are described.

- **Preliminary SW design (SD-4-SW)**

  The SW units are structured into SW components, modules and databases. The interfaces and the interaction between components, modules and (if applicable) databases are specified.

- **Detailed SW design (SD-5-SW)**

  The components, modules and databases with regards to the software-technical realization of their functions, of the data management, of the error handling and of the programming rules are described.

- **SW implementation (SD-6-SW)**

  The objectives for programming in statements of the (given) programming language are realized. The obtained code is inspected informally and databases are realized, if applicable.

- **SW integration (SD-7-SW)**

  The modules are integrated into components and the components into SW units.

Hardware Development (SD-HW)

The sub-model SD-HW describes the activities which are necessary for the development of HW portions of the overall system.

- **Analysis of HW requirements (SD-3-HW)**

  The requirements of a HW unit and its environment, defined at SD-2 or at the system architecture are described. In addition the requirements of the components with regards to qualifying, environmental conditions, authorized manufacturers and families of components are described.

- **Preliminary HW design (SD-4-HW)**

  The HW units are structured into modules and components. The interfaces and the interaction between modules and components as well as the component-related interfaces are specified.

- **Detailed HW design (SD-5-HW)**

  The HW is technically developed and technical documents are elaborated.

- **HW realization (SD-6-HW)**

  Resulting from SD-5-HW, material or samples of equipment are procured or manufactured.

- **HW integration (SD-7-HW)**

  Modules and components are integrated into HW substructures and HW substructures into HW units.

Integrated Logistic Support (SD-ILS)

The sub-model SD-ILS describes the activities to ensure the fulfillment of the logistic requirements during system planning and system development, as well as identification and preparation of the logistic resources for use by the product. The logistic resources for keeping a system running during its usage are combined in the logistic support system. It includes e.g. spare parts, measuring and test equipment, consumption material, instruction aids, user documentation, technical and logistic data, instruction and measures for the infrastructure and on the organizational aspects.

Note: The ILS activities, which are relevant for the overall system, are described in the main activities SD-1, SD-2, SD-8, and SD-9.

- **Analysis of logistic requirements (SD-3-ILS)**

  The requirements of the logistic support system are stated more precisely. The ILS plan is continued. A first life cycle cost (LCC) analysis is performed.

- **Logistic analyses (SD-4-ILS)**

  The logistic product structure is defined. A logistic data base is implemented. The ILS plan and the LCC analysis are continued and refined.

- **Detailed logistic design (SD-5-ILS)**

  The logistic product structure definition is finalized and the LCC analysis is completed.

- **Realization of the logistic elements (SD-6-ILS)**

  Instruction material and the user documents are created. Spare parts are manufactured or procured.

- **Integration of the logistic elements (SD-7-ILS)**

  The logistic elements are integrated into the logistic support system. The availability of the logistic support system is proven by logistic inspections.

2.2 Scope of Systems Engineering within the VM-GBV

The scope of systems engineering within the VM-GBV is defined by the phases SD-1, SD-2, SD-8 and SD-9. On the left-side of the V-Modell, systems engineering captures user requirements in SD-1 and defines technical requirements for logistics, HW and SW as output of SD-2. On the right-side, systems engineering is responsible for the integration (SD-8) and the transition to utilization (SD-9) of the system.

However, for this thesis, only a portion of the overall systems engineering scope is of relevance. The model based approach is part of the analysis and design phases of systems engineering and thus only the phases SD-1 and SD-2 will be described. In addition, logistic aspects of systems are also out of the scope of the modeling approach and will therefore not be covered; only functional aspects that can be developed in hardware or software are in the scope of the modeling approach.

2.3 Systems Engineering in the OPES4 Department

As mentioned in chapter 2, development processes generally describe the general process activities, the document that have to be generated and the reviews that have to be performed before transitions from one development into another can be taken. They also indeed give advice about method and tools but these are often not suited to the special contractors environments and also often don't consider modern methodologies. This is also the case with the Airbus RBE and VM-GBV process models [VMG2004].

To address this issue, the OPES4 has developed a department-specific systems engineering process. This process can be mapped to both the VM-GBV as well as the Airbus RBE but defines a detailed workflow as well as the tool-context that is tailored to the department needs. This chapter will give an overview of the general process activities and workflow, and will describe the tools that are used to aid the process.

2.3.1 Tool-based Systems Engineering in OPES4

The OPES4 makes use of modern tools to aid requirements engineering and to model aspects of systems. The modeling tools used in systems engineering can be roughly categorized into two departments, *functional* and *non-functional* modeling tools. To gain a better understanding of these categories, a brief description is given below.

Functional versus Non-Functional

Aspects of systems are driven by requirements on the system. Requirements can be classified into two categories, *functional* and *non-functional*.

Functional requirements are requirements that define the internal working and behavior of a system such as requirements on system services or the reaction of the system to inputs.

Non-functional requirements are requirements that define constraints on the system in various aspects as well as requirements on quality or performance.

A classification system that was devised by Robert Grady from Hewlett-Packard [Gra1992] goes by the acronym FURPS+ and divides requirements into the following:

- Functionality
- Usability
- Reliability
- Performance
- Supportability

The "+" in FURPS+ comprises requirements such as:

- Design Requirements
- Implementation Requirements
- Interface Requirements
- Physical Requirements

For embedded systems in the avionics and space domain, also the classification of non-functional requirements into *RAMS* requirements is prominent. The acronym RAMS stands for:

- Reliability
- Availability
- Maintainability
- Security

For the successful creation of systems, it is essential that all functional and non-functional requirements are considered equally. The current supporting toolset of the OPES4 systems engineering development activities are described below.

Requirements Engineering

For the management of requirements, the Tool *Telelogic DOORS* [Tel2007] is used. DOORS is a requirements management tool that allows storing requirements in textual and also graphical form. Requirements are held in repositories, called modules in DOORS. The requirements can be linked to other requirements, design artifacts – such as design models – and also test cases and test setups. This allows the complete trace-ability of requirements, e.g. from the high-level to the low-level requirement, from the requirement to the design, from the requirement to the test case etc. In addition, re-quirements can be attributized, for example to specify requirement priorities or to cate-gorize requirements in functional and non-functional requirements. DOORS provides a

scripting mechanism that for example allows the automated generation of documents from the repositories, consistency checks and the generation of metrics.

Functional Systems Engineering

Currently, two tools are used to aid the functional systems engineering activities, *MathWorks Matlab* and *Simulink* and *Telelogic Rhapsody*.

MathWorks Matlab [Mat2007] allows complex numerical computations and the design and analysis of algorithms based on a proprietary programming language. Typical fields of application in the systems engineering domain are the analysis of algorithms for signal and image processing or statistical computations. Basically any kind of mathematical tasks can be accomplished with this tool.

MathWorks Simulink [Mat2007] allows the modeling and simulation of dynamic systems and functions, such as control loops for actuators or dynamic image processing environments. The design language is graphical and many model libraries are available for different fields of engineering such as aerospace, mechanics, hydraulics or power systems. The designs can be simulated within the tool and computer code in the languages *C* and *C++* can be generated from it.

Telelogic Rhapsody [Tel2007] is a UML 2.1/SysML 1.0 modeling tool. It is primarily a software engineering tool with the focus on the development of real-time software for embedded systems. It features the automatic generation of code from the model and full roundtrip-functionality. A feature which lets Rhapsody stand-out from other UML modeling tools is that it provides an own production-quality execution framework which supports many different operating systems and target platforms. The framework is operating system and target platform independent, thus the same operational code can be run on both host and target platforms. Code is automatically generated from UML state machines and can be directly run on the application framework. This feature also allows the simulation of the UML model: Simulation code can be generated that allows stimulating the model and viewing the internal behavior in form of animated sequence diagrams and state charts. This feature is commonly called *model execution*. Rhapsody supports the programming languages C, C++, Java and ADA.

As one of the leading UML tool vendors and also supporters of the SysML language development, Telelogic are also marketing Rhapsody as the tool-of-choice for systems engineering applications. As a fact, the built-in simulation framework is a key feature for the acceptance of Rhapsody for systems engineers: The state chart syntax is clearly defined by a so-called *action language*. Systems engineers – who often come from mixed engineering fields and are *not* software experts – can use the tool to model systems without having a reasonable proficiency in a computer-language.

Another feature of Rhapsody is the support of exporting models into a DOORS repository. With this feature, textual requirements in DOORS can be linked to UML design artifacts. This feature significantly enhances the support of requirements traceability.

Rhapsody provides various add-ons, for example for automatic test generation, the customization of code generation and the automated generation of documents from the model. In addition, Rhapsody features a broad interface for the extension of the tool functionality with custom scripts or applications. Last but not least, various configuration management tools are supported to allow the collaborative work on one model by different developers. This includes the visual state-awareness (checked out/not checked out etc.) of model elements as well as the ability to graphically diff/merge UML diagrams.

Telelogic Rhapsody builds the core of the current UML-based functional systems engineering approach in OPES4 and therefore is the basis of the further analyses in this thesis.

Non-Functional Systems Engineering

To the current point, the OPES4 is still in the process of evaluating tools that support the modeling and analysis of non-functional aspects of systems. However, the department has not decided on specific tools yet. For example, the department is in contact with a vendor who offers a tool that allows the modeling of performance aspects of systems. The tool offers the possibility to import system architectures from different sources such as UML models, Matlab/Simulink models or computer code and perform a performance analysis on virtual hardware platforms that can be chosen from a model database that comprises a lot of common parts such as CPUs, RAM and bus systems.

2.3.2 The OPES4 Systems Engineering Process

The information about the OPES4 systems engineering process that is described in this chapter is based on [OP12007].

Figure 2-2 shows the OPES4 systems engineering development process as process submodel. The left side of the V shows the various process activities and the mapping of them on the phases SD-1 and SD-2 of the VM-GBV. The phases are described below.

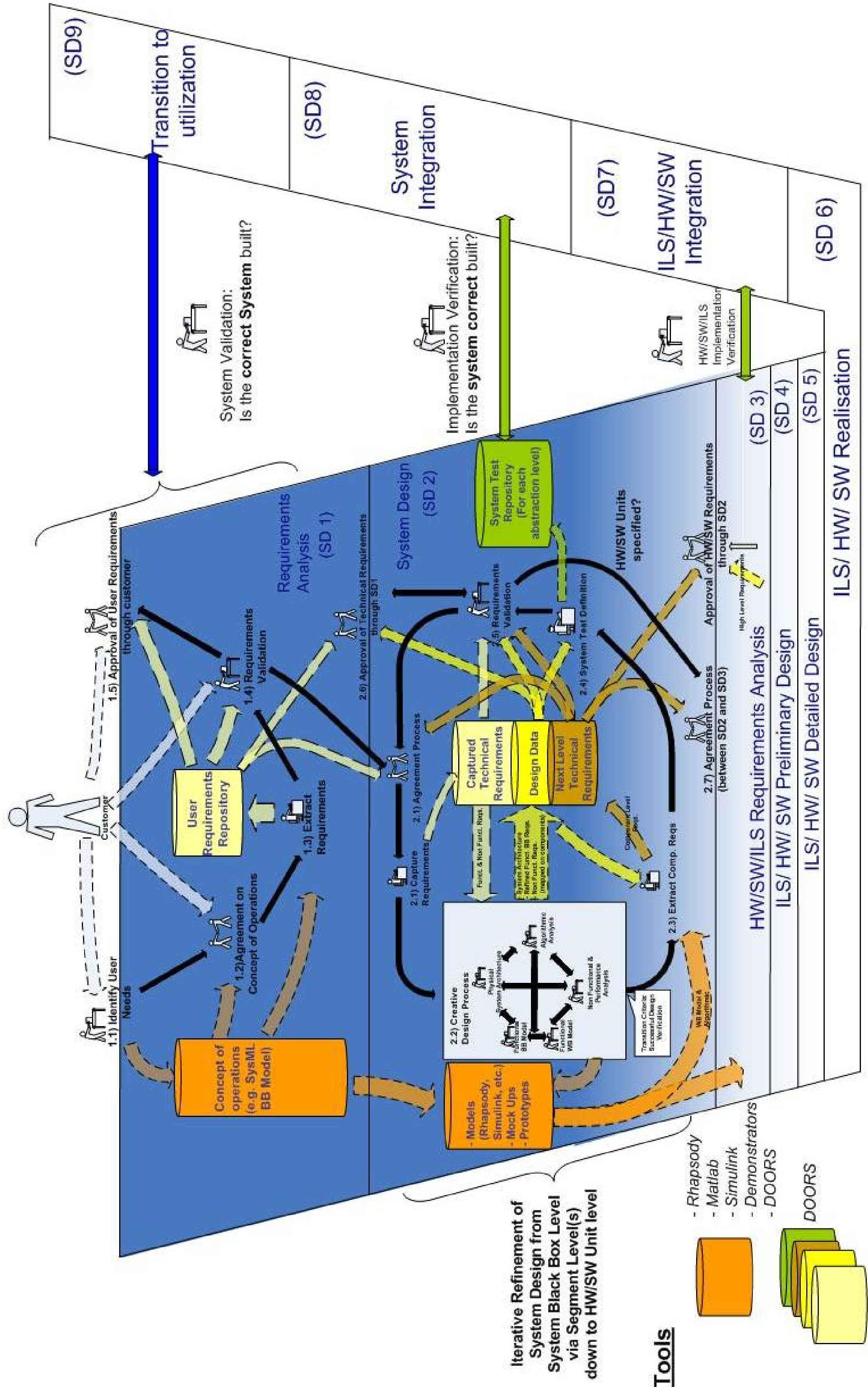

Figure 2-2 OPES4 Systems Engineering Sub-Model [OP12007]

Requirements Analysis (SD-1)

In the Requirements Analysis phase, the user requirements are captured and technical system requirements are created. The user requirements are elaborated by means of a Concept Of Operations (CONOPS). The concept of operations contains all aspects of the system that are relevant to the customer and describes in particular

- *what the system shall do* in its operational environment
- *who or what* will use the system
- *how the system will be used* from users point of view
- *how well the system performs*
- *how the system will be maintained.*

The concept of operations may be refined with the help of different tools, such as UML/SysML models, Matlab/Simulink models, or the development of demonstrators. Demonstrators are prototypes of the system to develop that usually feature most of the system functionality but don't realize all non-functional aspects of the system. They are used to analyze the technical realizability of the system and to elaborate the technical requirements.

When the CONOPS is agreed on with the customer, the user requirements are extracted, validated and approved by the customer. Because the terms verification and validation are used several times within this thesis, the difference between them has to be clarified in short. It can be distinguished between two different types of validation and verification:

- **Requirements Validation** answers the question:

  Are the product requirements correct and complete at each level of abstraction?
- **Design Verification** answers the question:

  Meets the product design in each development phase the requirements on which it is based?
- **Product Verification** answers the question:

  Meets the product at each integration stage the according requirements?
- **Product Validation** answers the question:

  Meets the final product the user needs?

Within the process activities in the scope of this thesis, validation always refers to requirements validation and verification to design verification.

The requirements are managed and structured in a *User Requirements Repository* with help of the tool Telelogic DOORS. Finally, the user requirements are handed over to the System Design (SD-2) phase of the process.

System Design (SD-2)

The System Design phase takes the user requirements as input and generates technical requirements for the subsequent Hardware and Software development activities.

The first step of the design phase is to agree on the user requirements that were captured in the requirements analysis phase. In this *Agreement Process*, the user requirements are checked in respect to the general (= legal, cost and time) and technical realizability by the design team. The agreement process is generally only needed, if the requirement analysis and systems design activities are performed by different teams. After that, the requirements that are relevant for the system design are captured from the input requirements and stored in another DOORS repository, the *Captured Technical Requirements* module. Based on the captured requirements, the C*reative Design Process* begins. The creative design process is an incremental, iterative engineering process, where functional and non-functional aspects of the system are analyzed and realized in form of different analysis and design models. An analysis model captures functional and non-functional aspects of a system from the black-box perspective while a design model considers the possible decompositions of the system into different segments. These analysis and design models may consist of algorithmic Matlab/Simulink models, UML/SysML models, Prototypes, Mock Ups and basically any kind of tools that are appropriate to capture and realize all technical aspects of the system. The various design activities performed in the creative design process are bilateral, that is, outputs of every containing activity may be input of the other containing activities. Output of the process is a system architecture which satisfies all functional and non-functional requirements on the system. The system architecture as well as the resulting refined functional and non-functional system requirements are stored in the *Design Data* DOORS repository.

When the design is successfully verified, technical requirements on the architectural segments are extracted from the design data and stored in the *Next Level Technical Requirements* repository. Based on the requirements on the overall system and its different units, test cases and setups are defined and stored in a separate *System Test Repository*, which is also a DOORS module.

This process is performed iteratively in a loop as often, as the system segments can be decomposed into concrete hardware and software units. When the hardware and software units are identified, the requirements on the units are finally handed over to the subsequent hardware and software development teams.

2.4 Functional Modeling Process

For the modeling approach with UML/SysML and Rhapsody, the OPES4 has developed a functional modeling process document [OP22007] that defines how the modeling language can be applied to various analysis and design activities. This modeling process document is the basis of the process analysis in this thesis. The process description in this chapter is based on this document.

An overview of the general phases of the modeling process is given in chapter 2.4.1. The detailed modeling activities are then described in chapter 2.4.2.

2.4.1 General Process Phases

The process is divided into two main activities, as shown in Figure 2-3.

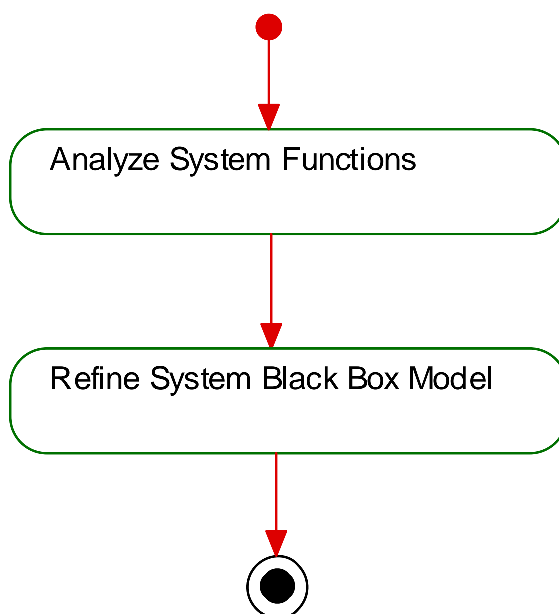

Figure 2-3 The two phases of functional systems engineering with UML/SysML

Analyze System Functions

In the first activity *Analyze System Functions*, the system functions are analyzed by means of a Rhapsody UML/SysML model that describes the system from the black-box perspective. The black box model captures and describes the system context, functions and behavior in form of UML/SysML model elements and diagrams. As the name suggests, the black box model doesn't reveal anything about the internal structure of the system, the system is indeed seen as a *black box* that provides overall system functions and behavior. Input for the analysis of the system functions are primarily the user requirements. However, there are additional inputs that have to be considered such as de-

velopment and document standards, additional support services and the modeling process itself (Figure 2-4).

When the whole system functionality is captured in the black box model, the model is verified against the user requirements by means of a model execution. After successful verification, the model is then handed over to be refined in the *Refine System Black Box Model* process activity.

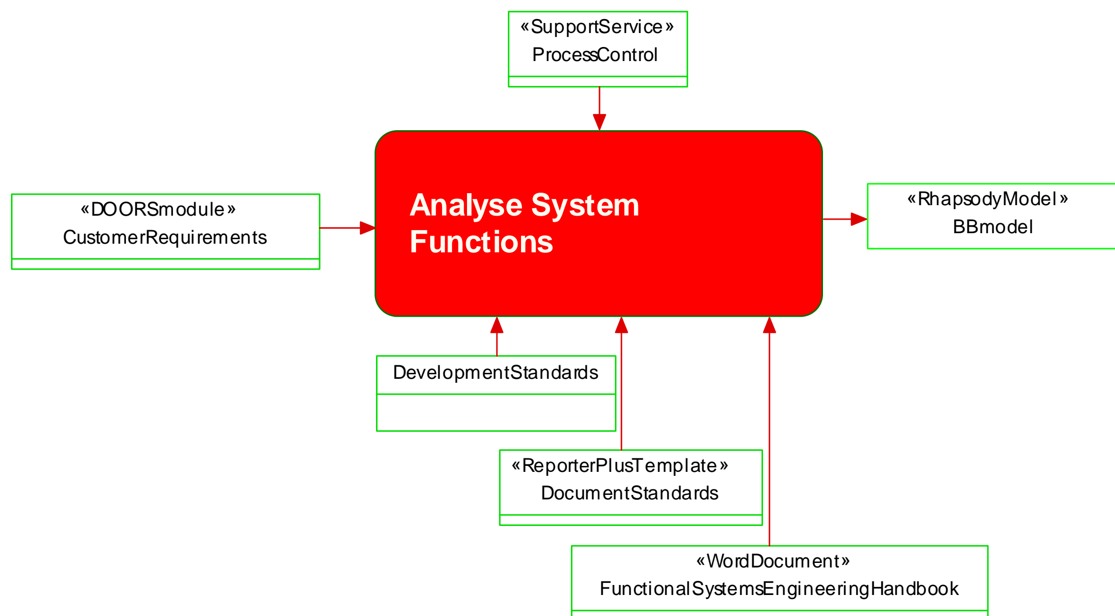

Figure 2-4 The "Analyze System Functions" Process Activity

Refine System Black Box Model

In this activity, the black box system model is taken and decomposed into system segments and HW/SW units. The *system architecture* is created. The creation of the architecture is driven by both functional and non-functional aspects of the system. For example, special RAMS requirements on the system usually have also a huge impact on the functional architecture of a system. The non-functional requirements and the functional requirements are taken from the *TechnicalSystemReq* DOORS module. The functional requirements in the *TechnicalSystemReq* consist of requirements that have not been modeled as an UML/SysML model before, such as requirements that were captured in Matlab/Simulink models. In addition, as with the preceding process activity, development and document standards as well as supply services and the modeling process are also considered (Figure 2-5).

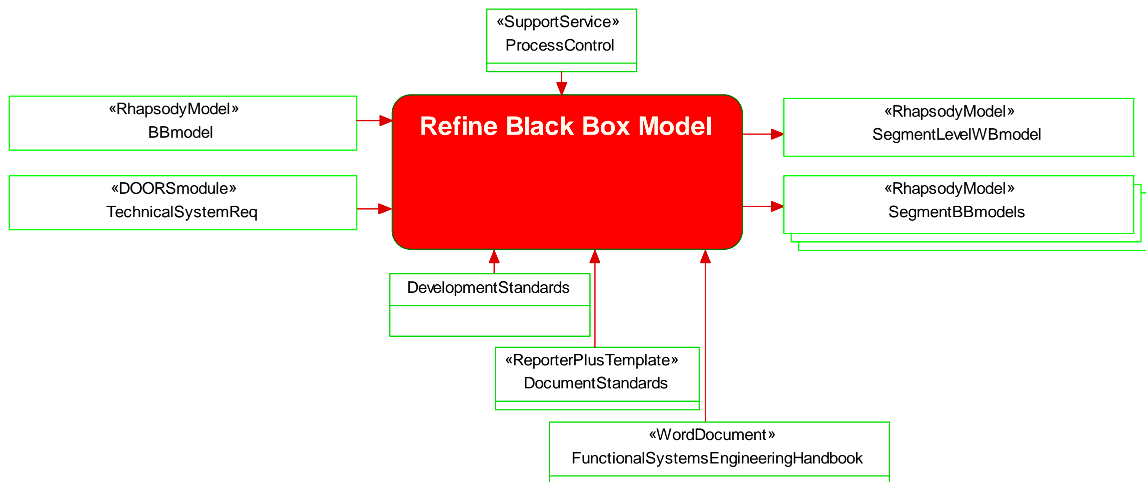

Figure 2-5 The "Refine System Black Box Model" Activity

The system architecture is created in a top-down fashion. The Black-Box model is decomposed into segments in a *white box model*. The white box model describes the system segments, the interaction among them and the allocation of functions and behavior to it. After the white box model is verified against the functional requirements that were defined in the preceding black box model and the *TechnicalSystemReq* requirements repository, a new model is created for each segment. The new segment model describes the segment as a black box with functions, behavior and interfaces to other internal segments or external systems. A segment may consist of both hardware and software and even may be further decomposed in additional segments that also consist of both hardware and software. For each segment identified, a new segment-level black box model is created and the *Refine System Black Box Model* activity reentered. The decomposition is performed until concrete HW/SW units can be identified. Handoff models are then generated for each HW/SW unit, describing the unit as a black box.

2.4.2 Modeling Activities

The OPES4 functional modeling process defines how the UML/SysML can be applied to the two engineering activities described in chapter 2.4.1.

2.4.2.1 Analyze System Functions

Figure 2-6 shows an overview of the different modeling activities of the system functional analysis. The analysis starts with the creation of a new Rhapsody project for the black box system model. This Rhapsody model is the container of all artifacts created within this process activity.

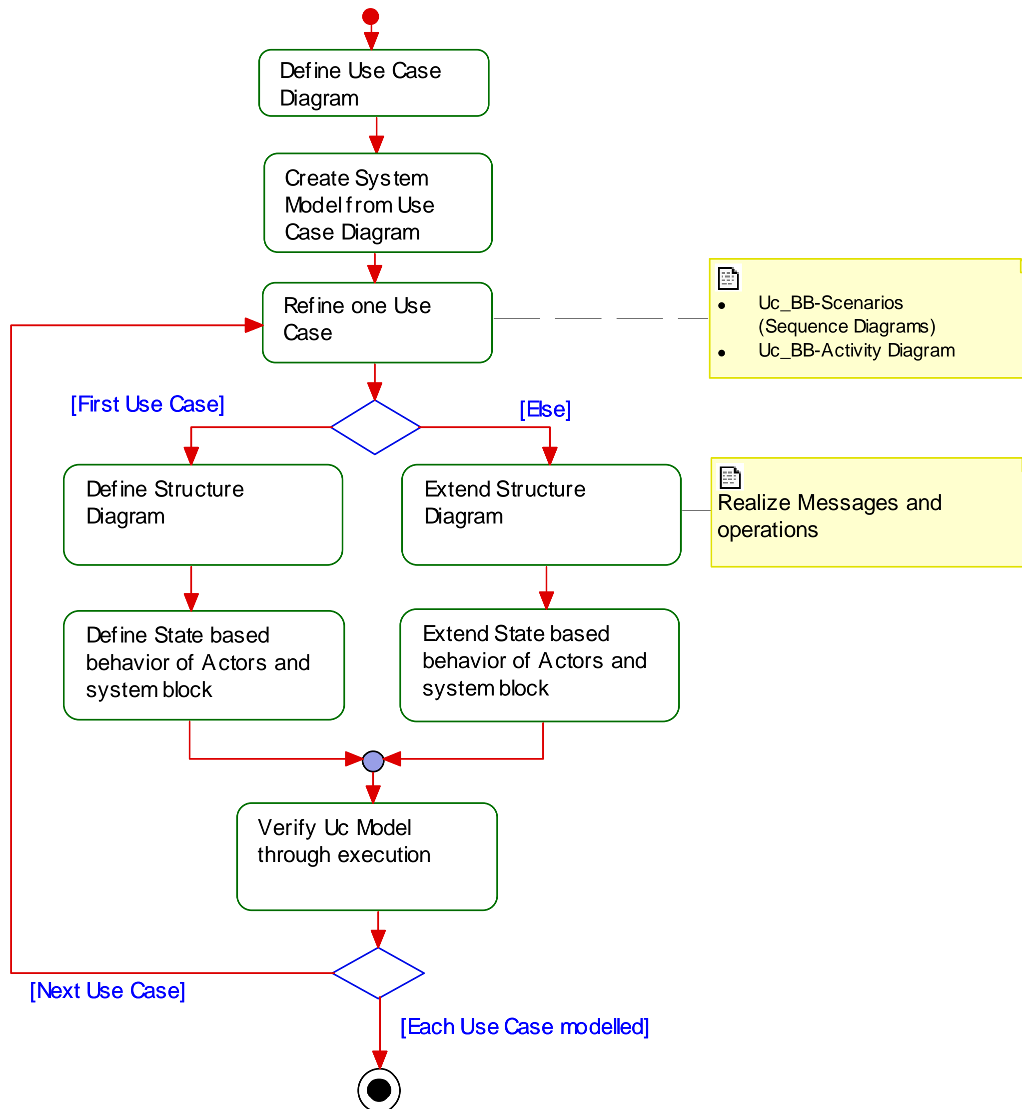

Figure 2-6 Analyze System Functions: Modeling Activities

Define Use Case Diagram

The system analysis starts with the definition of the system boundary and the system main tasks by means of a *use case diagram*. In it, the services of the system are brought into the model in form of use cases. External persons and systems that interact with the system under design are modeled as actors.

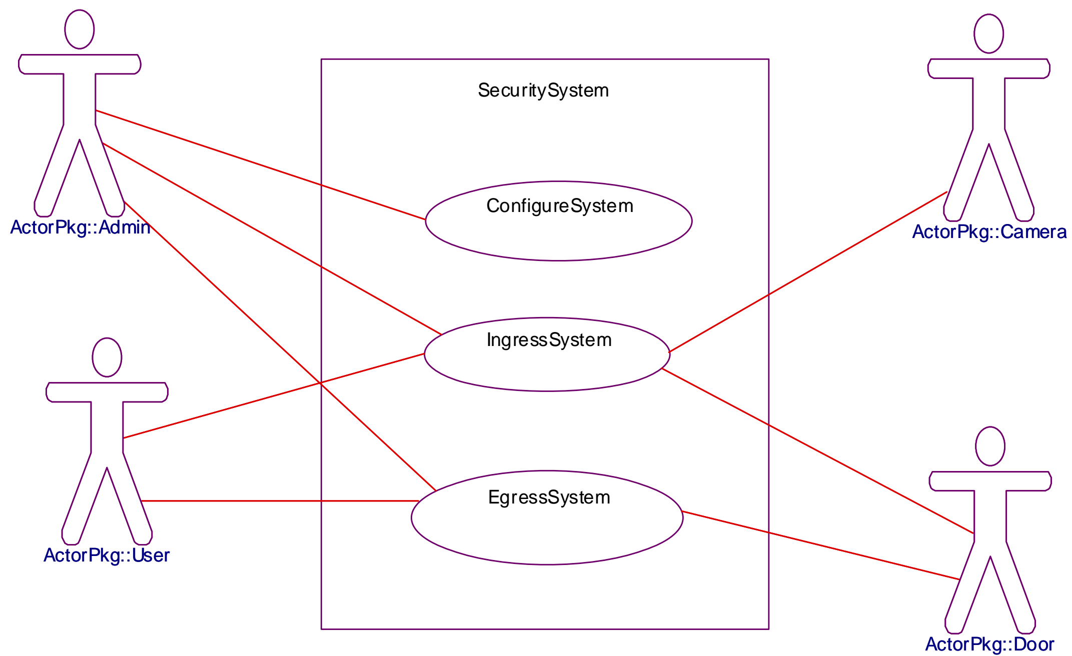

Figure 2-7 Example of a Use Case Diagram

Create System Model from Use Case Diagram

Based on the use case diagram, the system model is created by means of a wizard in Rhapsody. The wizard automatically restructures the model, creates a SysML *block* representing the system under design and a separate model package per use case. In the use case model packages, the individual use cases will then be refined by means of sequence and activity diagrams in the next step.

Refine One Use Case

The process is use-case driven and so the system analysis is performed incrementally per use case. For the refinement of a single use case, the process defines two alternative ways, as shown in Figure 2-8. The first alternative is to first define use case scenarios by means of *sequence diagrams* and after that define the functional flow with an *activity diagram*. The second alternative is opposite way around; first define the functional flow and then the scenarios.

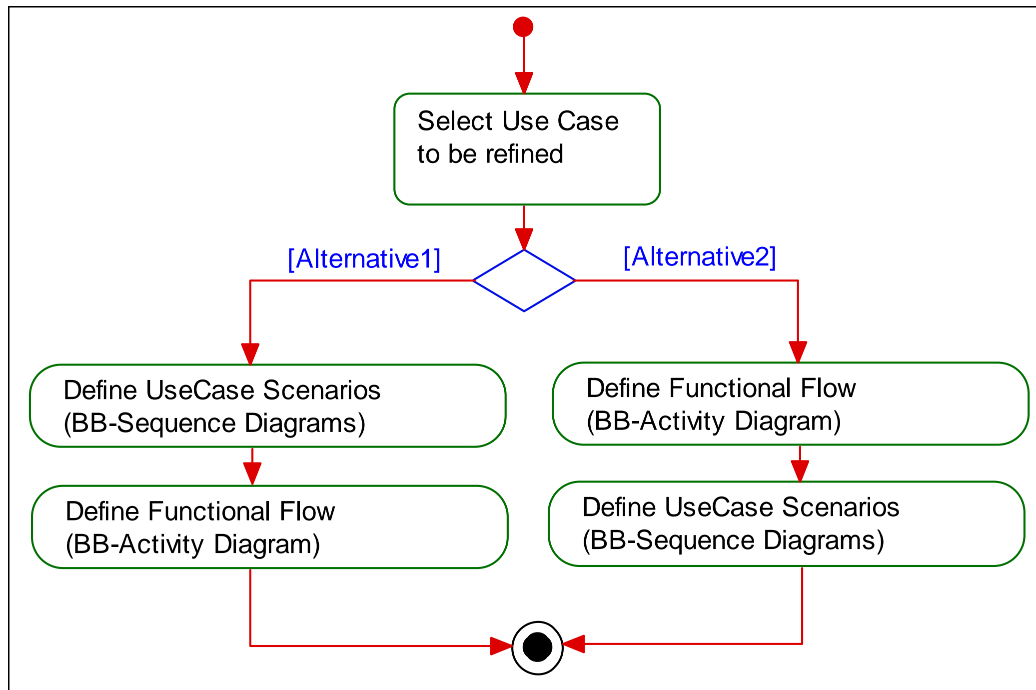

Figure 2-8 Two alternative ways to refine a use case

A use case scenario shows a specific operational flow through the system with the focus on interaction between the system and its actors on a sequence diagram. Usually several scenarios are required to describe a use case sufficiently. Scenarios can be categorized in *sunny day* scenarios and *rainy day* scenarios. A sunny day scenario shows the operational flow under normal systems condition while a rainy day scenario describes the system reaction in case of exceptions.

The communication between the system and the actors is described by means of asynchronous messages. These will later be realized as events in Rhapsody. System functions are shown as self-messages in the lifeline of the system block. The self-messages will be realized as operations of the system block in Rhapsody. Within the process, the system functions are referred to as *Operational Contracts* (OpCons). A proprietary concept of Rhapsody sequence diagrams are the so-called *Condition Marks*. Condition marks are hexagonal description fields on the system block life line that can be used to show system states and property values. These can be seen as both pre and post-conditions within a scenario.

Figure 2-9 shows an example scenario. The scenario depicts the common operational sequence to create a user account in the context of a security system. The communication is shown between the system administrator, represented as the *Admin* block in the diagram and the system under design. The condition mark *ReadyForConfiguration* shows the state that the system must be in as precondition for the sequence. The Admin then sends a sequence of request messages to the system block that is followed by a functional reaction (OpCon) in the system.

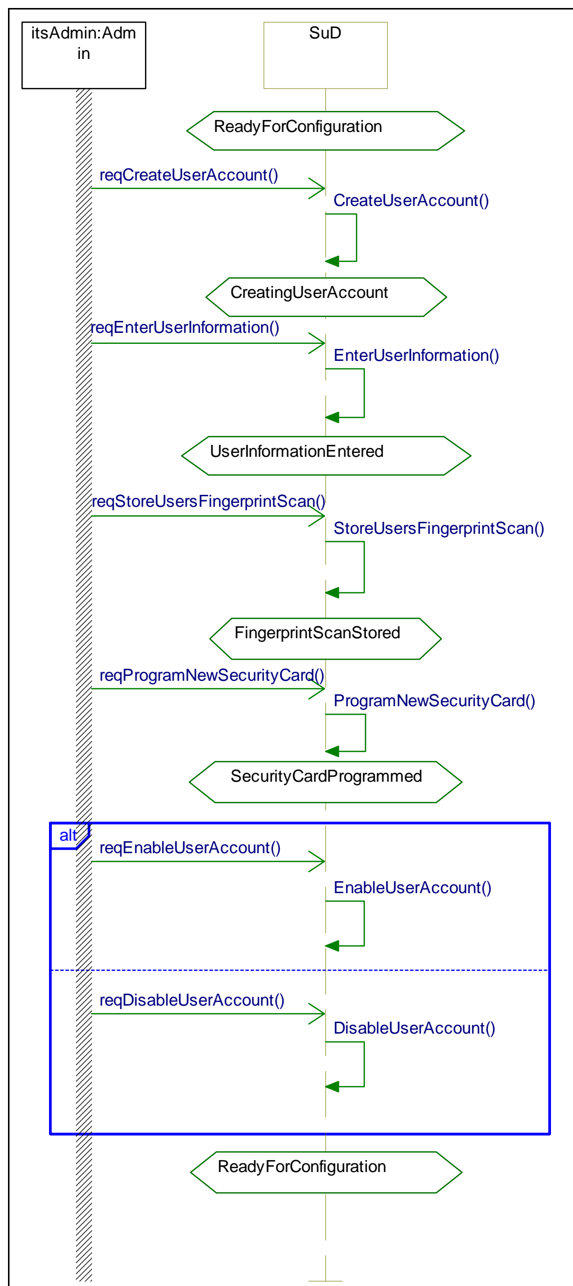

Figure 2-9 Example of a Use Case Scenario

A use case activity diagram describes the complete functional flow through the use case, including all possible functional branches, parallelism and exception handling. The process specifies that every action in an activity diagram should reference an operational contract in a sequence diagram. Figure 2-10 shows an example use case activity diagram for the use case *ConfigureSystem* in a security system. In this diagram, the possible flows through the use case are shown as branches: In the within the use case, a user account can be either created or modified. When the functional flow reaches the termination connector, the use case is ended. The activity diagram corresponds to the operational sequence shown in Figure 2-9.

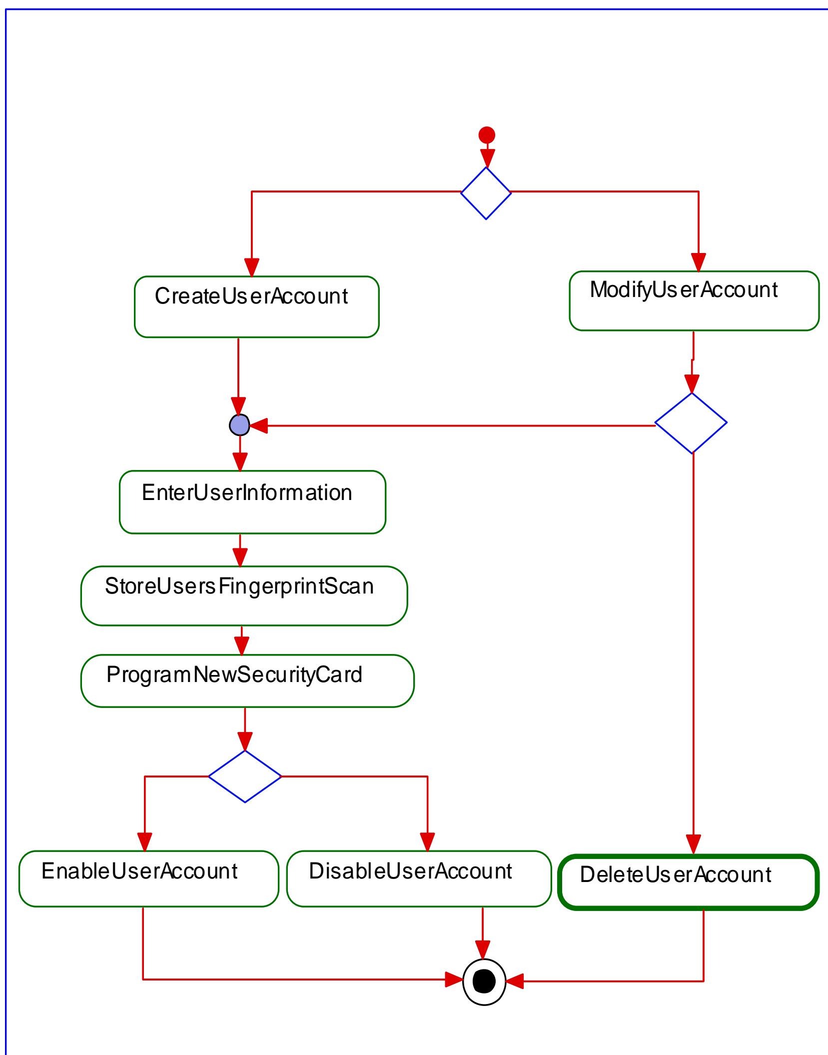

Figure 2-10 Example of a Use Case Activity Diagram

Define Structure Diagram

When the use-case is described by means of scenarios and an activity diagram, the system structure has to be defined in a UML composite structure diagram. The first step is to implement the operational contracts and messages that were defined in the use case scenarios as operations and events in the Rhapsody model. This is performed by means a Rhapsody feature that automatically creates operations, events and event receptions from sequence diagrams. After that, ports are created on the system block and the actors as well as interfaces that define the message exchanges. This is done by a Rhapsody tool wizard automatically. When the ports are created, the system block and the actors are brought into a composite structure diagram and the ports are linked. Figure 2-11 shows an example composite structure diagram for the security system. The communication between the *Admin* and the system block is realized with the ports *pAdmin* and *pSuD*. The interfaces that specify the communication are not shown on the diagram but

are present in the model and can be viewed on in the properties dialog of the ports. The body of the system block element in the diagram shows the OpCons of the system.

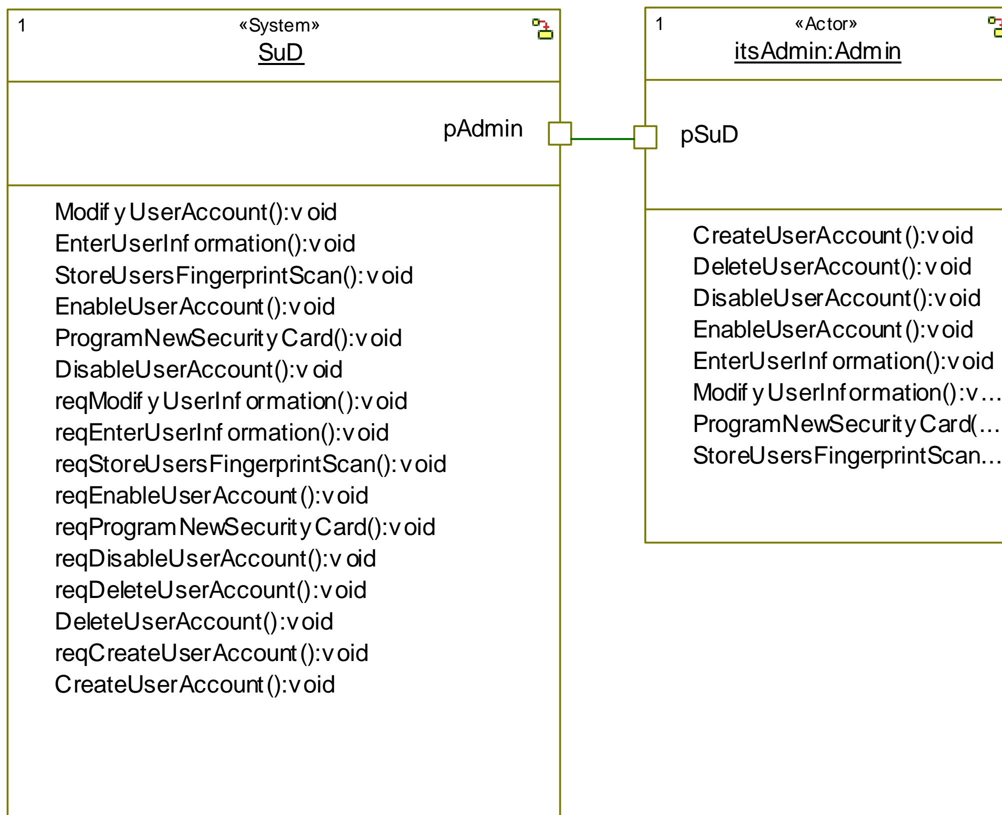

Figure 2-11 Example of a Structure Diagram

Define State Based Behavior of Actors and System Block

When the system structure has been created for the chosen use-case, the state based behavior of the system block and the actors that are involved in the use case has to be defined. This is done by means of a state chart for the system block and each actor. For the actors, only a simple state chart with one entry state has to be created. This is required for the simulation engine to function properly.

The state chart of the system block specifies the internal states, the possible transitions between them, the reaction to inputs and the output to produce. The state chart realizes *all* scenarios that were captured during the use case refinement and with them the complete functional flow into one overall system state based behavior. The state machine is the formal behavioral specification of the system and is the basis for the verification of the model. Figure 2-12 shows the system state machine that implements the use case *ConfigureSystem* for the security system. The condition marks of the sequence diagram in Figure 2-9 are used as system states. The reaction to external messages and the response by means of OpCons are realizes with the transition between the states.

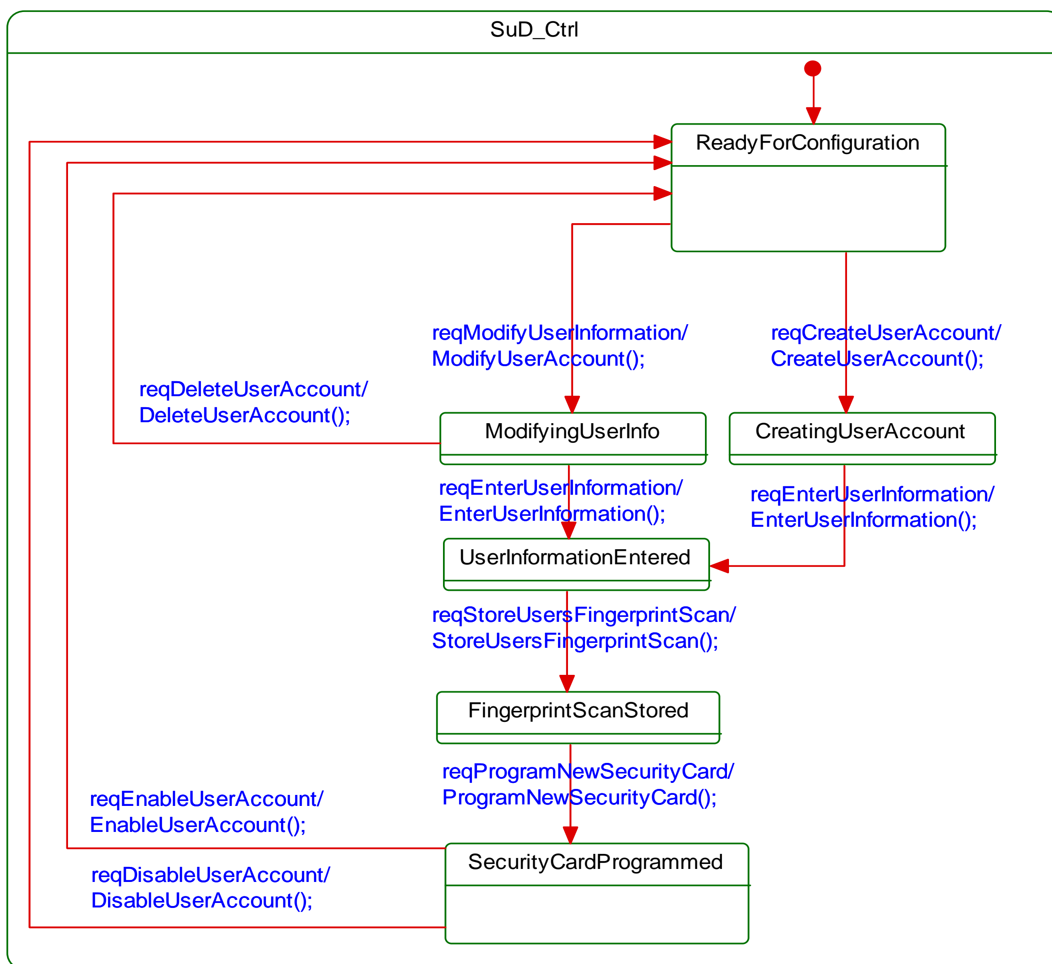

Figure 2-12 Example of a State Chart

Verify UC Model through Execution

When the state based behavior of the system block and the actors have been defined, the model has to be verified against the use case scenarios. An executable model is generated in Rhapsody and the model is simulated. In the simulation, the system block is stimulated with messages from the external actors and the executed functional flow of the system block is recorded in a sequence diagram. The recorded sequence diagram is then compared with the specified scenarios. If the functional flow of the recorded sequence diagram matches the scenarios, the verification was successful.

Extend Structure Diagram / Extend State Based Behavior

State charts and structure diagrams specify the overall system structure and behavior and unify the specifications of all system use cases. Thus, these two diagrams have to be updated incrementally for each use case that is refined and no new diagrams have to be generated.

2.4.2.2 Refine BB Model

When the functional system aspects have been captured in the system black box model, the next step is to refine the model and create the system architecture by means of a system *white box model*. Figure 2-13 shows the activity flow for the system refinement.

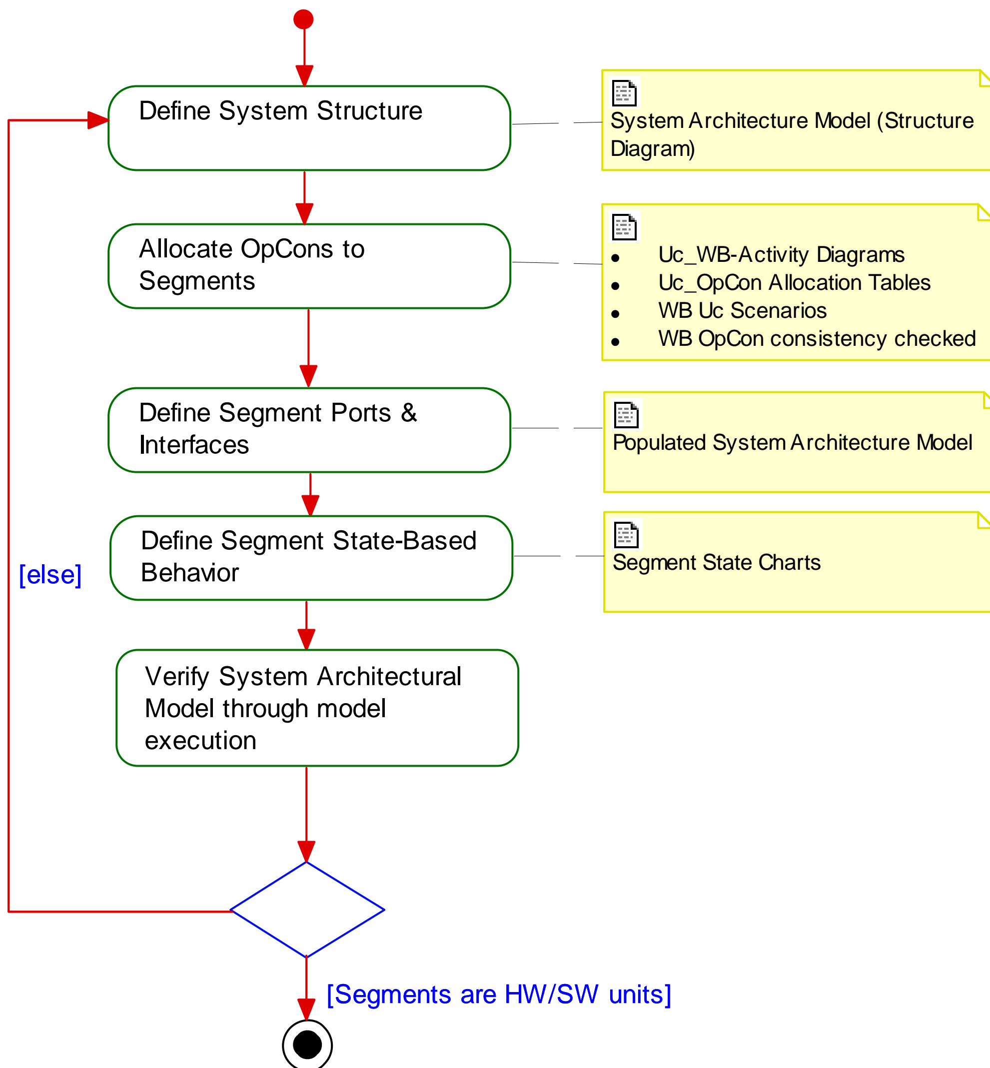

Figure 2-13 Refine Black Box Model: Modeling Activities

The refinement of the black box model is performed iteratively until concrete HW/SW units can be identified. That means, if the resulting architecture of the first black box refinement cannot be directly handed over to HW/SW engineering, the architecture has to be further decomposed in another iteration of the *Refine Black Box Model* activity per system segment. If the segments are refined by other development teams, a black box model describing the segment has to be created after the last gate review of the activity. This model is then handed over to the segment development team. The same goes for concrete HW/SW units: For each unit, a separate model that describes the unit as black box has to be created and handed over to the development team.

Define System Structure

The first step in refining the black box model is to define the system structure. The system is decomposed into segments, which together build the system architecture. The decomposition is a collaborative process, driven by the domain-knowledge of the various engineering disciplines involved in the project. An architecture is created that satisfies both functional and non-functional requirements. The different segments are brought as blocks into the model, as shown in the structure diagram in Figure 2-14. The system under design is decomposed into the three segments *ConfigurationInterface*, *UserInterface* and *DevideControl*.

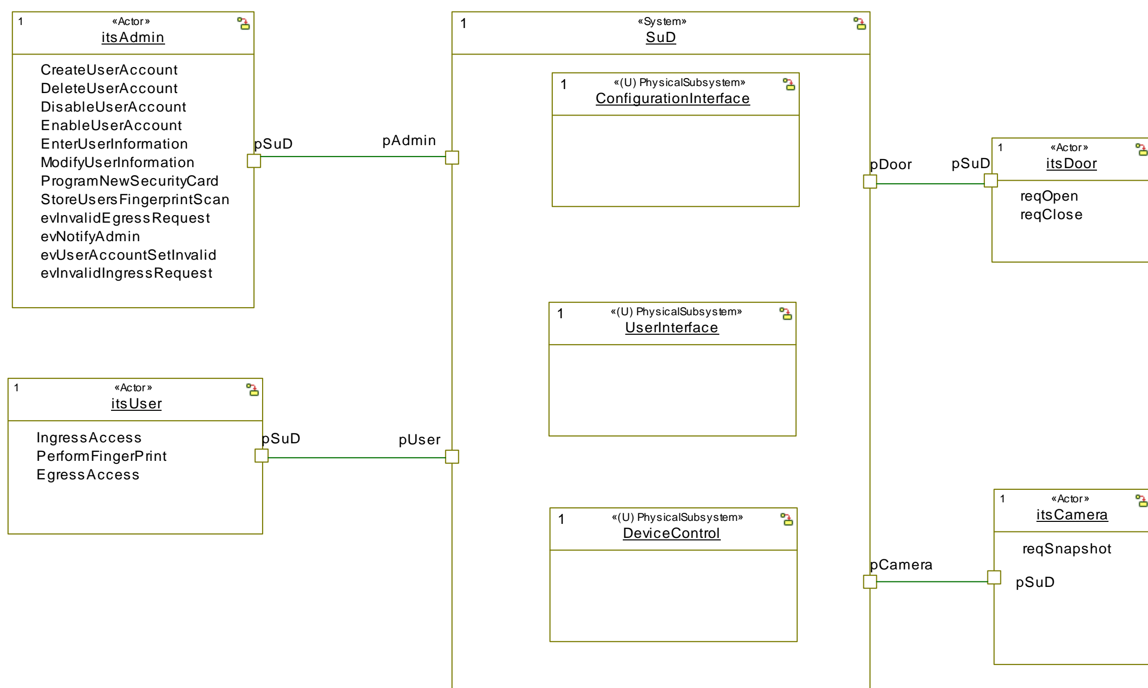

Figure 2-14 Example of a Decomposition Structure

Allocate OpCons to Segments

When the system structure is built, the operational contracts are allocated to the identified segments. This has to be done in both activity and sequence diagrams per system use case. For activity diagrams, the black box activity diagram is partitioned into *swim lanes*, each of them representing the allocation of functions to segments. The actions of the black box diagram are then placed on the swim lane of the segment that shall realize the function (Figure 2-15). In the diagram, the allocation of operational contracts of the *ControlDevice* to its two segments *DMGstimulator* and *ViewGenerator* is shown. It can be seen that the major functionality is allocated to the DMGstimulator. The *SetCurrent-FlightData* OpCon is needed in both segments so it is duplicated.

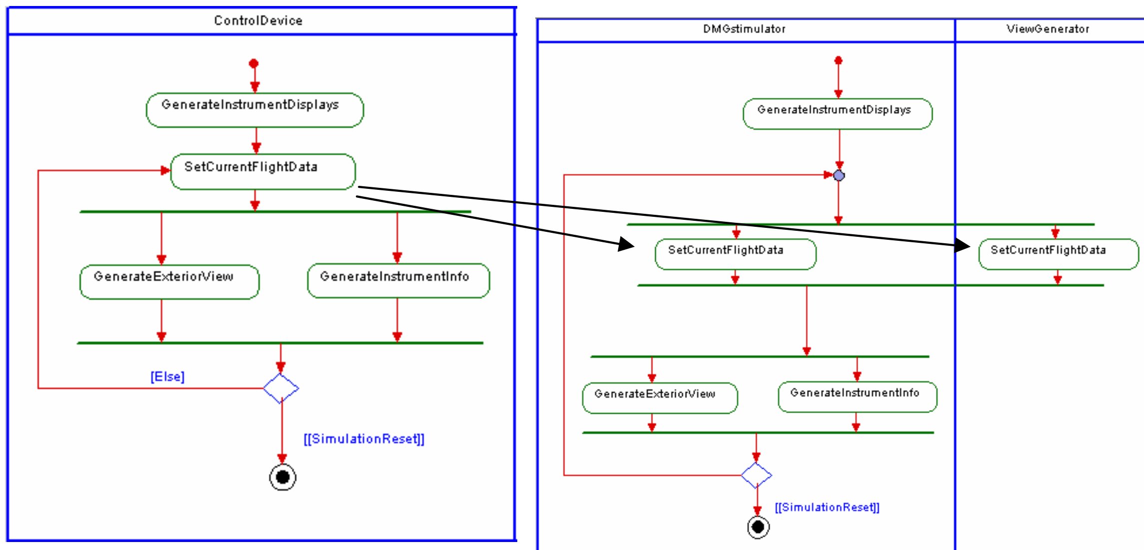

Figure 2-15 Example of OpCon Allocation in an Activity Diagram

The allocation in sequence diagrams is performed by adding new lifelines that represent the segments into the black box diagrams and then placing the messages of the system block to the lifelines of the appropriate segments. For communication between the segments, new messages have to be created. Figure 2-16 shows the allocation of OpCons to segments. The example represents the allocation that is shown in the activity diagram in Figure 2-15.

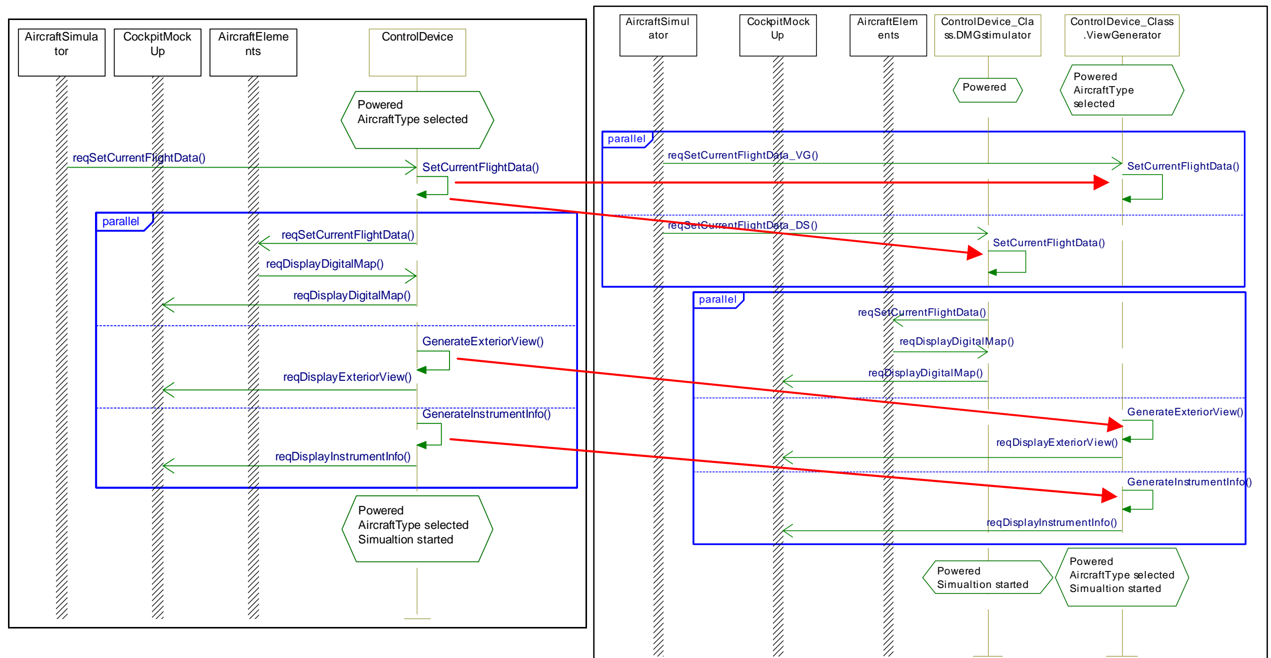

Figure 2-16 Example of OpCon Allocation in a Sequence Diagram

Define Segment Ports & Interfaces

When the OpCons have been allocated to the segments and new messages between the segments are created, the interfaces and ports are created by means of a Rhapsody wizard. The ports of the surrounding black box system block are then switched from "behavior" to "delegation", which means that messages that arrive at the system port from the environment are delegated to the internal structure and vice versa. Finally, the ports between the system and the segments as well as between the segments themselves are linked. Figure 2-17 shows an example of a complete system architecture with linked ports representing the *ControlDevice* segment that was described before.

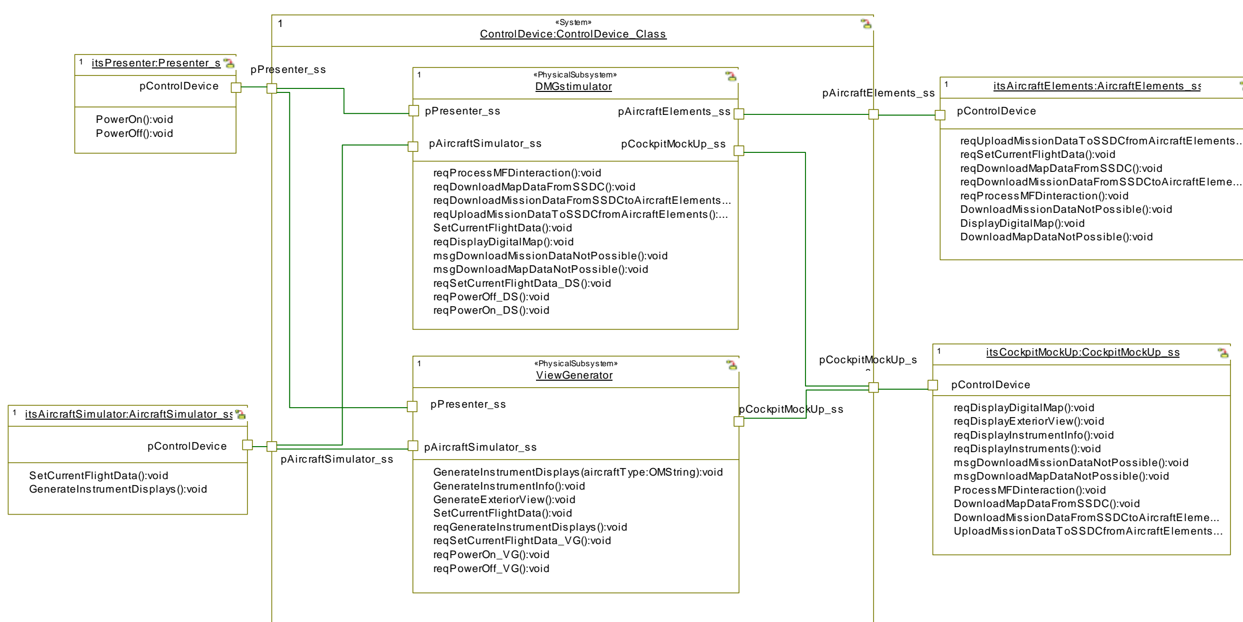

Figure 2-17 Example of a System Architecture in a Structure Diagram

Define Segment State-Based Behavior

The next step is to define the state-based behavior for each segment identified. The behavior is modeled on individual state charts per segment. As with the black box state charts, the state charts of the segments are modeled based on the use-case scenarios and activity diagrams: The segment state charts must implement the behavior that was specified in the use-case sequence diagrams. As a consequence, the sum of all segment state machines has to be consistent with the black box system state-behavior. Figure 2-18 shows an example of two segment-state charts that are consistent with the system state-machine specification.

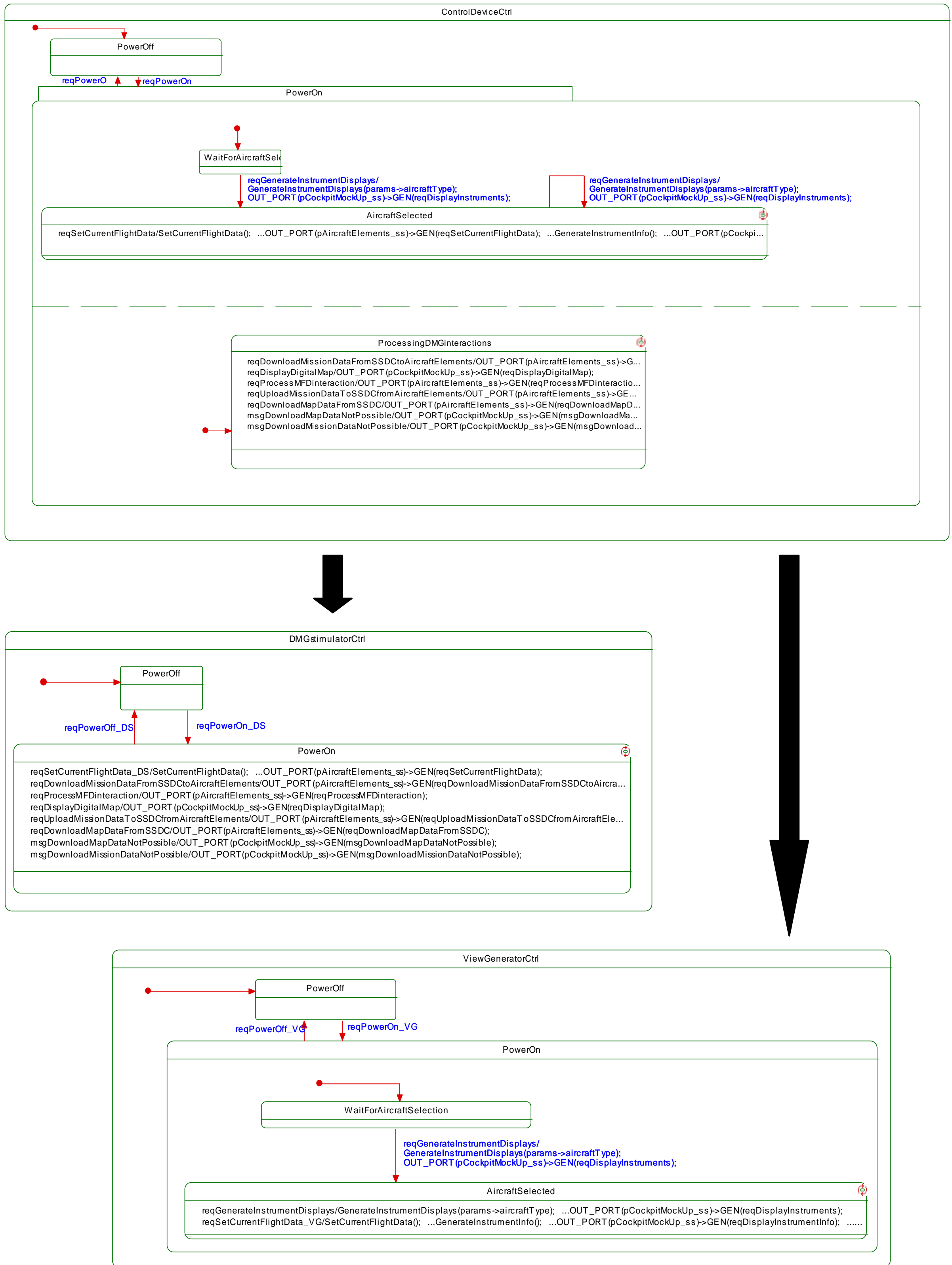

Figure 2-18 Example of a Segment State-based Behavior

Verify System Architectural Model through model execution

Finally, the system architecture has to be verified against the system black box specification. This is done by executing the model. During the model execution, the white box segments are stimulated and white box sequence diagrams are recorded. These are then compared to the black-box sequence diagrams. If the specified black box sequences match the white box sequence specifications, the verification was successful.

2.5 Systems Modeling Language (SysML)

This chapter will give a brief introduction to the Systems Modeling Language. Aspects of the language that are relevant for this thesis will be described in detail in the individual sections of chapter 3.The information and diagrams are based on the official website [OMG2007].

The Systems Modeling Language (SysML) is a standardized modeling language based on UML for the modeling of systems. The development was initiated in 2001 by the OMG (Object Management Group) together with the INCOSE (International Council on Systems Engineering). The goal was to create a common language that allows the analysis, design and evaluation of systems. In July 2006, version 1.0 was presented as "Final Adopted Specification", it is expected to be finalized in the mid of 2007.

As a traditional software modeling language, the UML has its strengths rather in the design of object-oriented software systems which stands in contrast to the primarily function-driven systems engineering approach. The language is very complex and many elements that the UML offers are not needed for systems design. On the other hand, the UML lacks in some departments that are important for complete description of systems. Last but not least, the nomenclature is tailored to that of the software engineering discipline. Systems engineers however don't think in "classes and objects".

To address this issue, the *Object Management Group* (OMG) has developed the *Systems Modeling Language* (SysML). The language is based on the UML but simplifies it in that only the elements that are suitable for the modeling of systems are included but on the other hand enhances it with new design elements to allow a more complete system description. In addition, the nomenclature is adapted to the one that is common in the systems engineering discipline.

Overview

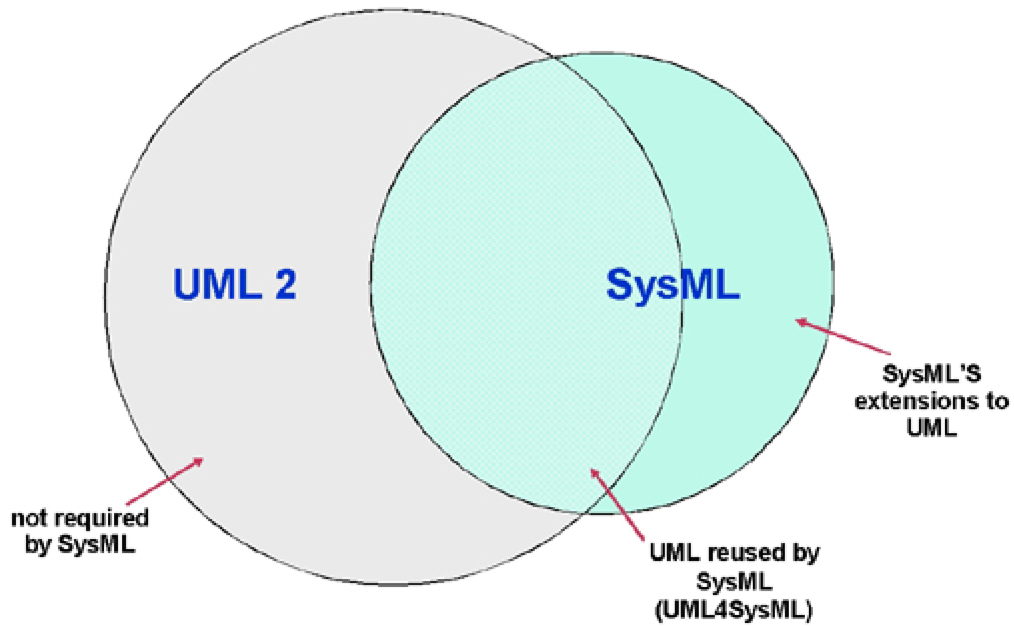

Figure 2-19 Relationship between UML and SysML

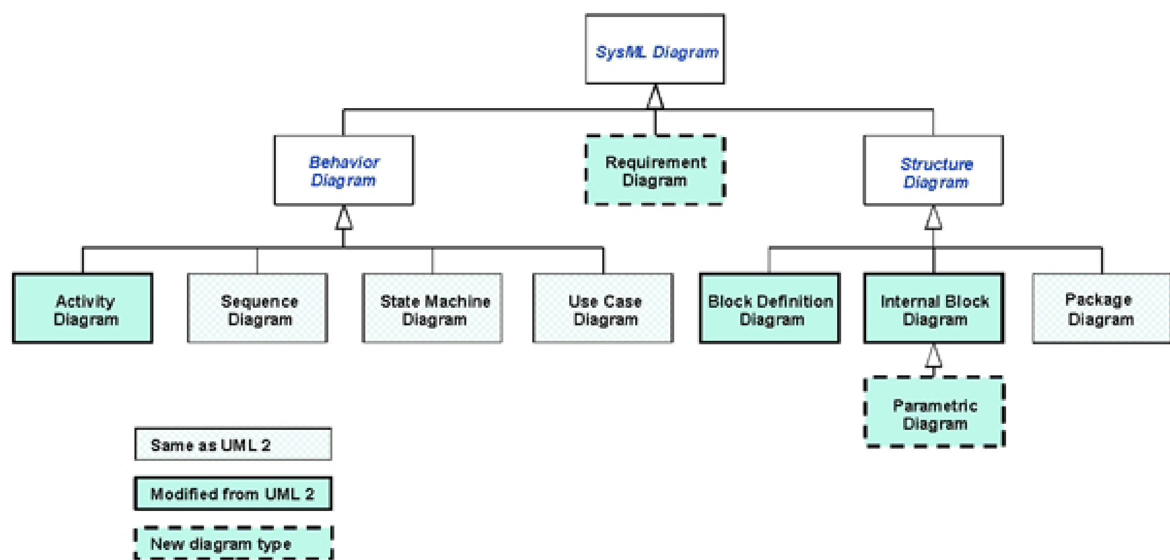

Figure 2-20 SysML Diagram Types

The SysML is a subset of UML 2.0 with extensions to satisfy the needs for the modeling of systems (Figure 2-19). Figure 2-20 shows the SysML diagram types. As extension to UML 2.0, SysML features two new diagram types, the *Requirement Diagram* and the *Parametric Diagram*.

As the name suggests, the requirement diagram enables the modeling of textual requirements and to show hierarchies and relations among them. A requirement can be related to a model element that satisfies or verifies a requirement.

Parametric Diagrams show constraints on system property values such as performance, reliability and mass properties and can be used to model parametric equations.

The basic element of SysML is a *block*. A block can basically represent any system element, hardware, software, personal, items or data. System structure can be shown on two diagrams, the *Block Definition Diagram* or the *Internal Block Diagram*. The block definition diagram describes the system block relations and hierarchies. The internal diagram shows the internal structure of a system by means of parts, ports and connectors. When compared to UML 2.0, a block can be seen as the SysML representation of a class, the block definition diagram corresponds to the UML class diagram and the internal block diagram to the UML composite structure diagram.

The UML 2.0 activity diagrams are enhanced in SysML in that it is now possible to show continuous flows, streaming activities and rates of flow. Control can be modeled as data, which allows activities not only to enable but also to disable other activities.

The other behavioral diagrams, the sequence diagram, the state machine diagram and the use case diagram, are taken from the UML without modifications.

3 Analysis and Concepts

3.1 Problem Definition and Approach

Problem Definition

While the OPES4 functional modeling process is basically capable to describe system functionality, behavior and structure, it also leaves some things to desire.

First of all, it uses a mixed set of SysML and UML model elements and is thus not fully compliant to the SysML language specification. In addition, it does not consider the enhancements for systems modeling that the SysML adds to the UML. Thus, a main goal of the thesis is to analyze the application of SysML and its enhancements to the current process. The used tool Telelogic Rhapsody is primarily a software engineering tool based on UML 2.0, the supports for SysML enhancements is fairly new and not evaluated by the department yet. Thus the SysML capabilities of Rhapsody have to be evaluated either.

Another problem the current process doesn't address is the transition from the systems engineering process to the subsequent development of segments, hardware and software in detail. The process describes how the system can be decomposed into hardware and software units and it describes *that* model handoffs have to be created for the hardware and software development teams. However, it is not described in detail *how* the handoffs should be generated to maintain a seamless development workflow.

The functional modeling process does only capture hardware and software units that provide "real functionality". The fact that software units do not run stand-alone but require basic software and computer hardware to execute on is not considered yet. However, this is an important aspect for the development of software: Units might span over different CPUs or have to communicate over various buses of a carrier board. As a consequence, the deployment of software to physical hardware with use of SysML has also to be analyzed in this thesis.

Approach

As a process document, the OPES4 functional modeling process does only define the different process steps without a detailed guideline how to apply the process to a real project. Thus, a modeling handbook is required that can be used as work reference and for self-studies. This modeling handbook will be the output of this thesis. The handbook

will apply the process in detail on a continuous example problem. The results of the process analysis in this thesis will be includes in the handbook.

The analysis of the current process and the evaluation of different concepts that address the stated problems will be performed on an example model. The example model will apply the modeling process to a fictitious example problem in a complete manner, reaching from the definition of system use cases over the creation of a systems architecture, the generation of a segment hand over model, the identification of software and hardware units, the deployment of them on a physical architecture and finally the generation of a software handover model. Possible first steps from the software unit handover model to software engineering activities will also be analyzed in a final step.

3.2 Example Problem

The analyses and concepts in the context of this thesis were performed based on a fictitious example problem. The kind of system however was chosen according to the main development field of the OPES4 to maintain the relation to the "real world".

The system in the example shall protect airplanes or helicopters from incoming infrared seeking missiles. An incoming missile shall be detected based on the UV radiation that the missile engine emits. The seeker head of the missile shall then be "jammed" by means of a high-intensity laser beam that is directed on the seeker head. The laser beam is modulated with a special jamming frequency that is chosen based on the analysis of the backscattered laser reflection from the seeker head. Such a system is often called DIRCM, which is short for **D**irected **I**nfra **R**ed **C**ounter **M**easure.

However, the DIRCM system is relatively complex and thus most concepts in this thesis are shown on more simple examples to support comprehensibility. In addition, the original allocation of HW/SW units to a physical architecture in the DIRCM example contains company confidential information because a real physical hardware was modeled. Thus, the concepts are shown on an alternative, fictitious architecture.

3.3 Modeling of System Structure

General Analysis and Concepts

The current process uses the Rhapsody representation of a SysML system block as basic system element. However, when analyzing the semantics of a Rhapsody block, it turns out that a block in Rhapsody is a proprietary concept that isn't compliant to the SysML. A Rhapsody block is technically and semantically an instance of a UML class. However, a block can be defined as *implicit*, which makes it possible to define class features

in a block, like operations and attributes. The definition of the class is thereby technically hidden. This is the concept that is used in the current modeling approach.

The system structure in the current process is shown on a UML composite structure diagram exclusively. The diagram is used to show the internal structure of the system blocks, the ports, interfaces and connectors.

The actual SysML concept for structure modeling however is different and provides extended modeling capabilities. A SysML block is the *specification* of a system element and can be used as *part* to show the role of the system element in a specific context. The semantic is similar to the UML *Class* and *Role*.

System block specifications, relations between blocks and block hierarchies are shown on the SysML *Block Definition Diagram*. In the block definition diagram, the relations between blocks are realized as associations. The associations may be directed and have multiplicities. System block decomposition can be shown by using the composition association. Figure 3-1 shows an example block definition diagram. The block *Car* is decomposed into one *Engine* and four *Wheel*s, which is shown with the composition association. The engine block has an association of multiplicity 2 with the wheel block to show that only two of the four wheels on the car are driven by the engine.

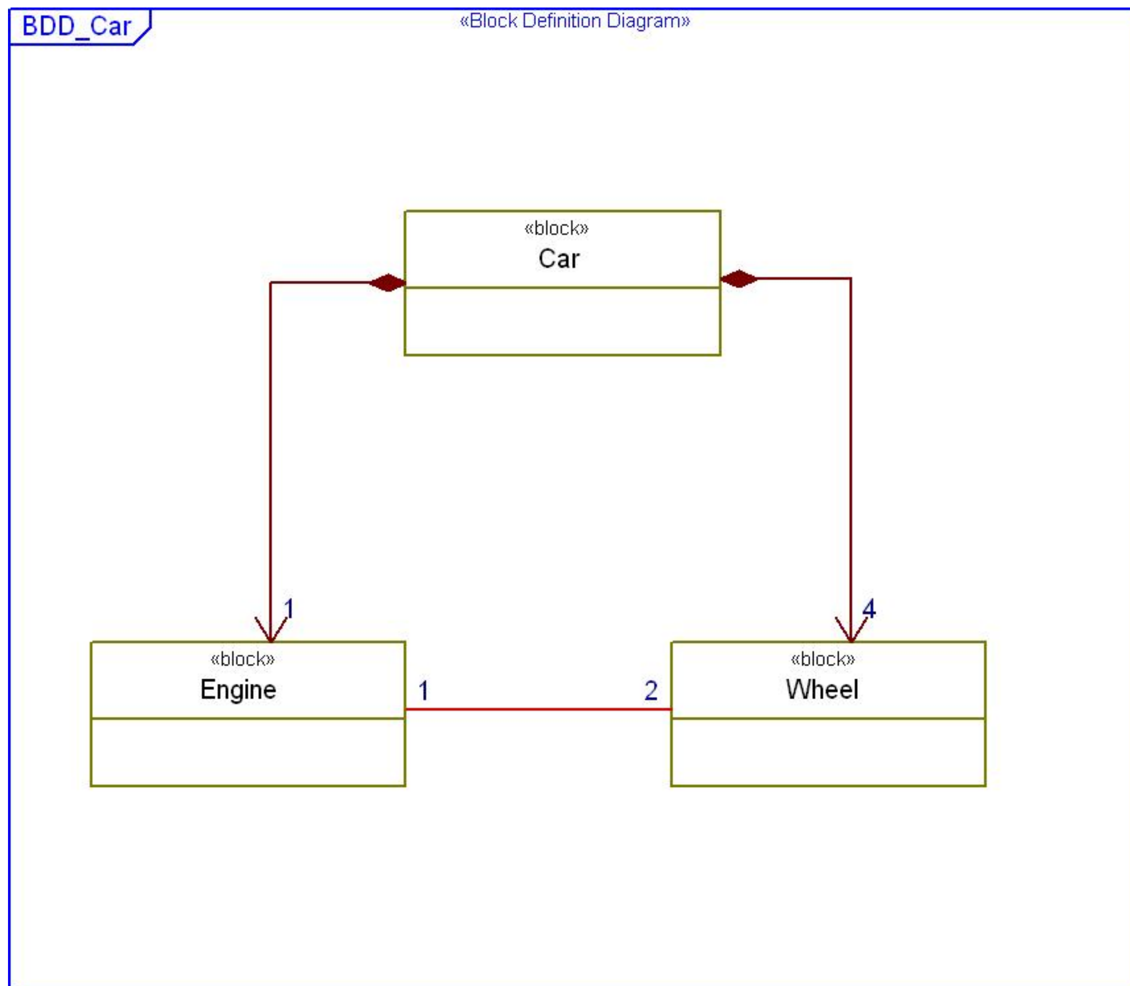

Figure 3-1 SysML Block Definition Diagram

The internal structure of system elements is shown in the SysML *Internal Block Diagram*. In this diagram, the internal structure is shown by means of parts, which are specified by blocks but show the usage of the blocks and the connections between them in a certain role. Here, part nesting, ports, interfaces and connectors between system parts are described. This concept is similar to the concept of the current process, which uses an UML composite structure diagram to show the internal system structure. The difference however is that the parts used in the internal block diagram are specified by a system block while the implicit blocks used in the composite structure diagram are both specification *and* usage in a specific role.

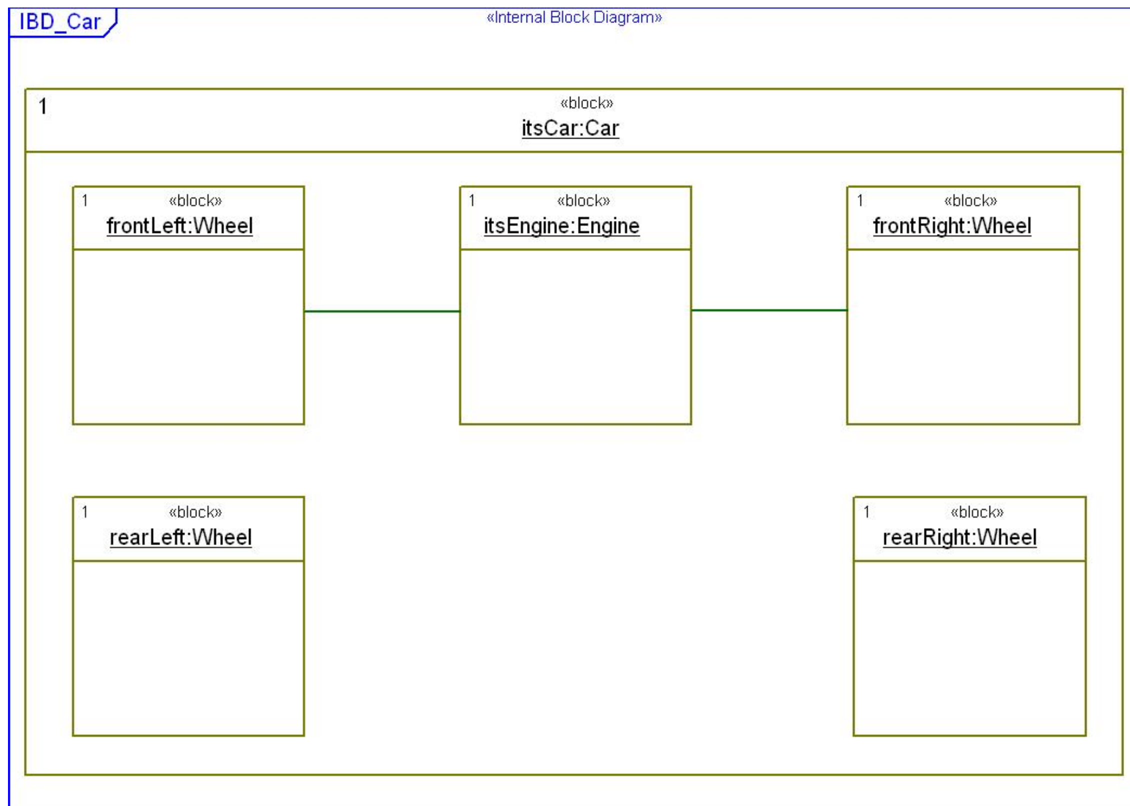

Figure 3-2 SysML Internal Block Diagram

Figure 3-2 shows the internal structure of the car example in an internal block diagram. Here, the block *Car* is shown in a specific role as part *itsCar*. The decomposition is shown by nesting; the blocks *Engine* and *Wheel* appear as parts within the border of the *itsCar* part. In the block definition diagram, the structure of a car was specified in that it has four wheels and an engine that drives two of the wheels. In the internal block diagram, the single wheels and the engine are shown in a specific context that shows that the two front wheels are driven by the engine.

The SysML concept has two important advantages over the current concept:

First of all, by separating specification and usage of blocks, the concept of abstraction is supported: Blocks with similar structure and behavior can be specified once and then be reused as parts within concrete contexts. In the example above, the block *Wheel* is specified once in the block definition diagram. This block could for example contain some properties, like the physical size or the pressure of the tire. The concrete parts then in the internal block definition diagram automatically inherit the specified properties but can have different values for them, like for example different tire pressure for front and rear wheels. The concept with the implicit blocks in the current process doesn't allow this. Blocks with a semantic similar structure or behavior would have to be duplicated.

As a second advantage, the separation of specifications and usage adds an additional high-level viewpoint on the system. With a block definition diagram, the basic system

elements, relations and hierarchies can be shown without revealing anything about the internal structure, ports and interfaces of the system. The current process lacks this feature. This becomes particularly evident as soon as the system structure gets more complex. Figure 3-3 shows the white box system structure of the DIRCM example problem on segment level. From the diagram, the top-level relations to the actors as well as the decomposition of the system into several segments become visible in a concise manner. An internal block diagram or composite structure diagram on its own is not able to provide this view (Figure 3-4). This becomes even more evident when different internal portions of the system are shown in different internal block or composite structure diagrams.

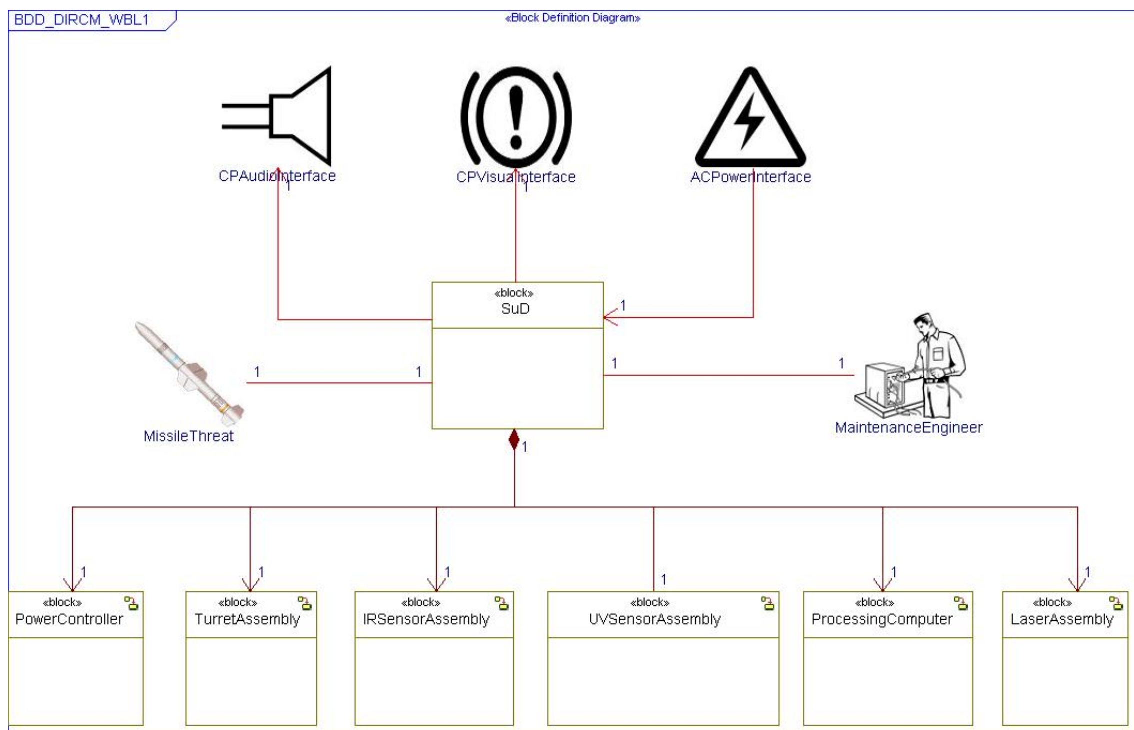

Figure 3-3 Block Definition Diagram showing the DIRCM White Box System Structure

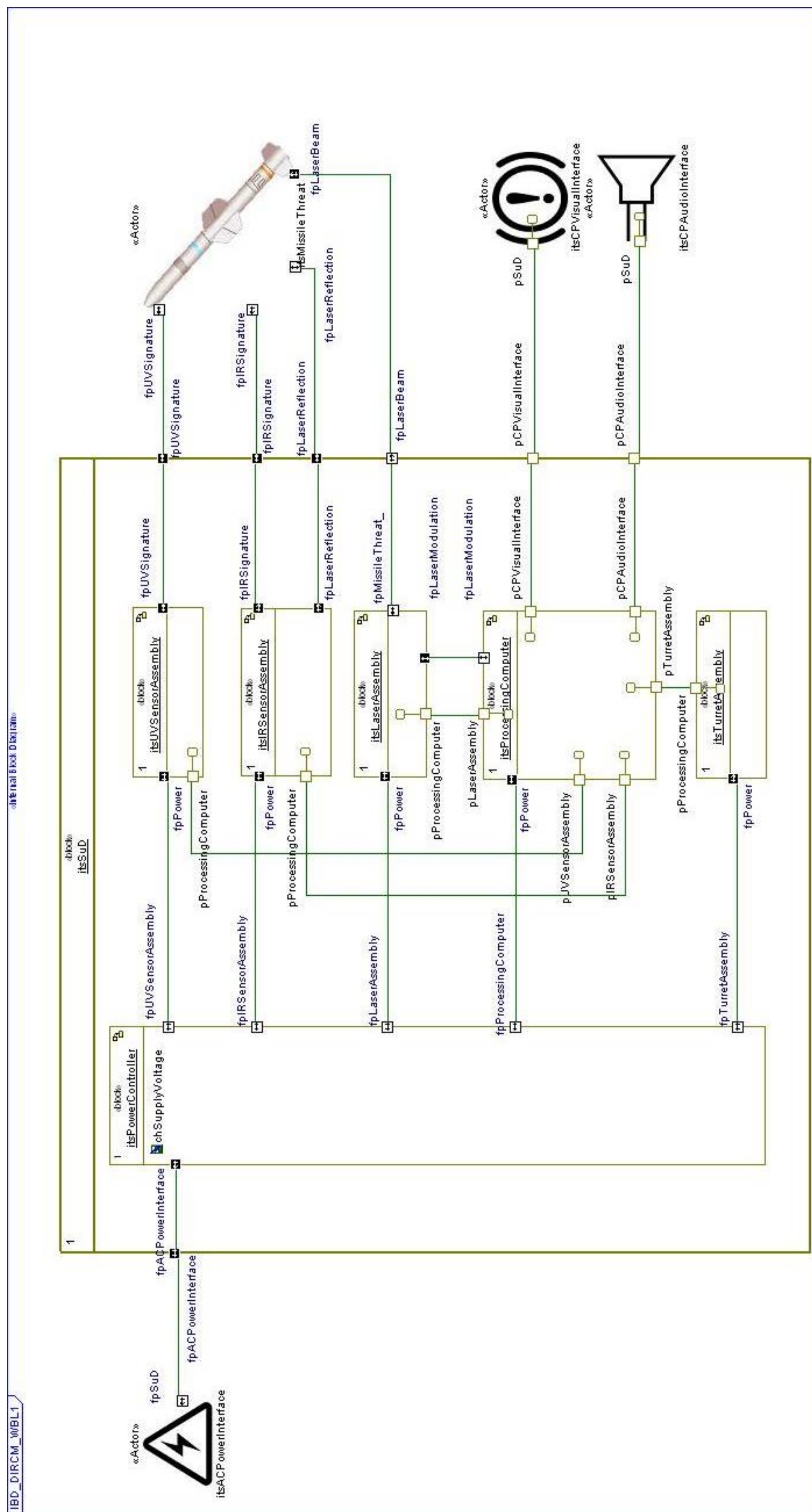

Figure 3-4 Internal Block Diagram showing the DIRCM White Box internal System Structure

Implementation of the Concepts in Rhapsody

With the introduction of Rhapsody 7.1 MR3, Telelogic provides a SysML profile together with Rhapsody that supports both block definition diagrams and internal block diagrams. When the SysML profile is activated within a Rhapsody project, the two diagram forms can be directly created from the browser. The concepts of system block as specification and part as usage of a block specification is also supported. This is done by the addition of the new model elements *System Block* and *Part*. The two elements are the semantically correct implementations of the SysML part and block specifications.

3.4 Modeling of Items and Data

The current modeling process does not define how items or data in the scope of a system can be modeled. However, items and data are an integral part of a functional system model and thus modeling concepts have to be created.

Modeling primitive Types

SysML introduces the concept of *ValueTypes* for the definition of primitive types. A ValueType defines values that have no identity and are not referenced by a system block. SysML defines the ValueType as a stereotype of the UML Data Type. This is done to *"establish a more neutral term for system values that may never be given a concrete data representation. For example, the SysML "Real" ValueType expresses the mathematical concept of a real number, but does not impose any restrictions on the precision or scale of a fixed or floating point representation that expresses this concept. More specific value types can define the concrete data representations that a digital computer can process, such as conventional Float, Integer, or String types."* [Sys2006]

SysML ValueTypes can be bound to units and dimensions. This addresses an important issue: In systems engineering models, it is necessary that the unit of a value is clearly defined to avoid serious design mistakes. A popular mistake is the failure of the *Mars Climate Orbiter* mission. The probe got lost at planet mars because of a unit fault in the navigation system. While the *NASA* was working with SI-units, the system was working with the imperial unit system [Wei2006].

Units and dimensions in SysML are specified model elements. A SysML dimension is *"a kind of quantity that may be stated by means of defined units. For example, the dimension of length may be measured by units of meters, kilometers, or feet"* [Sys2006]. The SysML dimension is specified as a stereotype <<*dimension*>> of the ValueType element.

A dimension can be associated by a unit. A unit is *"a quantity in terms of which the magnitudes of other quantities that have the same dimension can be stated. A unit often relies on precise and reproducible ways to measure the unit. For example, a unit of length such as meter may be specified as a multiple of a particular wavelength of light"* [Sys2006]. The SysML unit is specified as a stereotype *<<unit>>* of the ValueType element. The association of a unit to its dimension can be done by setting the *dimension* property that a unit inherits from its <<unit>> stereotype.

The SysML specification provides the SI-unit system as a model library. In this library, the standard SI dimensions and units are implemented as SysML elements and the associations between the dimensions and units are realized appropriately. This library could be used as basis for concrete systems engineering models.

The unit and dimension can be associated with a ValueType by means of the *dimension* and *unit* property that a ValueType automatically inherits from its <<valueType>> stereotype. This unifies ValueTypes, units and dimensions into a "package" and thus describes the ValueType in a complete manner.

Figure 3-5 depicts the ValueType concept on an example. The SI dimension *Mass* and unit *Kilogram* are modeled with the unit and dimension SysML model elements. The unit Kilogram is associated with its dimension Mass via the dimension property. The concrete ValueType *SystemWeight* then is specified with its unit and dimension by the unit and dimension properties.

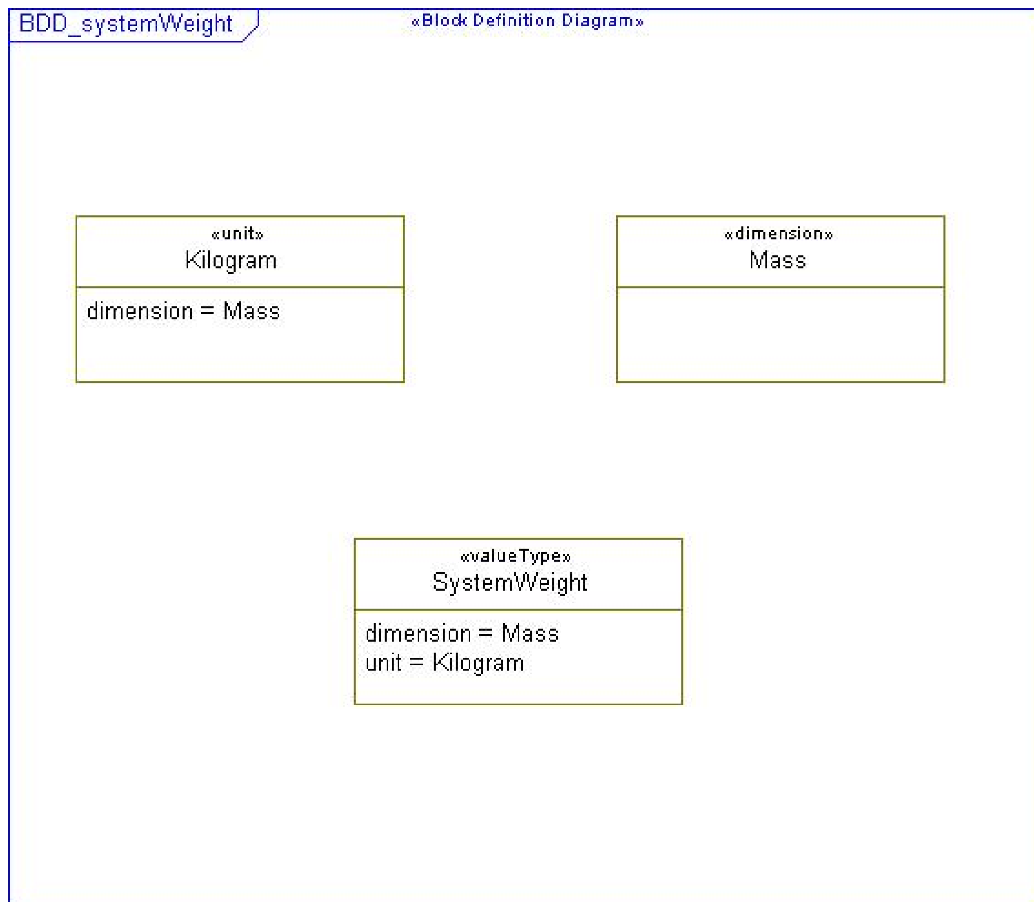

Figure 3-5 SysML ValueType Concept

Modeling Complex Items and Data

Often it is useful and necessary to model more complex data or item types that may themselves also contain primitive or complex types as properties. For example, the missile in the problem example shall be detected by means of the UV signature of the missile engine. The UV signature itself is specified by the intensity and wavelength, which are primitive numerical ValueTypes.

In SysML, complex data or items types can be modeled as system block. This technique broadens the block concept from the current modeling approach: A block can not only represent a logical or physical system entity, but also items or data. Figure 3-6 shows the missile UV signature as *UVSignature* system block. The UV intensity and wavelength are block properties, typed by the value types *Intensity* and *Wavelength*.

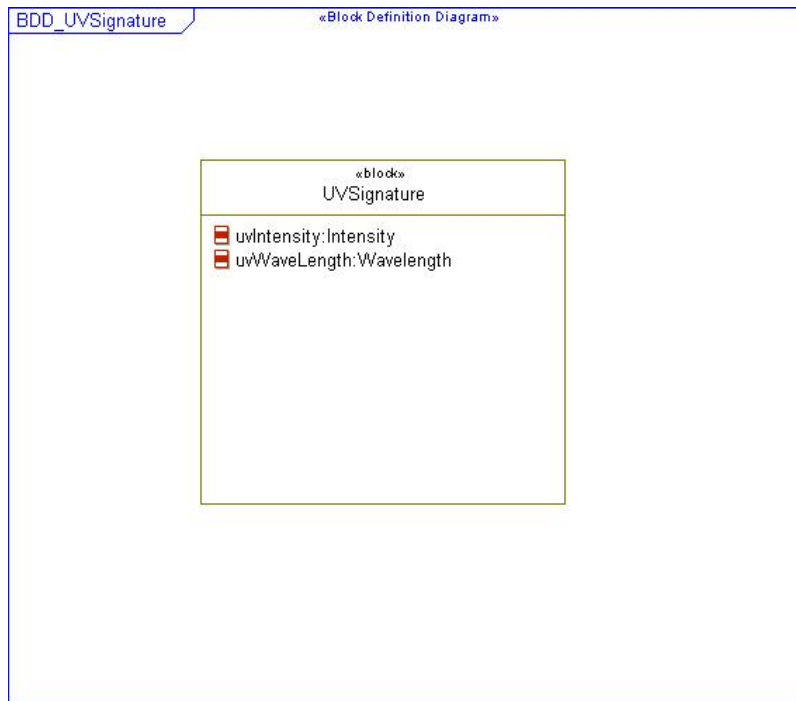

Figure 3-6 Modeling Complex Types as Block

Implementation in Rhapsody

As a UML software engineering tool, Rhapsody natively only supports UML Data Types. As a consequence, the concept of modeling dimensions and units and the possibility to associate them to types is not natively implemented. Thus the customization capabilities of the tool have to be analyzed and it has to be checked if the SysML concepts can be realized through them.

In Rhapsody, it is possible to define custom stereotypes. For a custom stereotype, the possible model elements to which the stereotype may be applied can be specified. In addition, a stereotype can be configured to appear as a new element in the browser when applied to a model element. By using this technique, the model elements ValueType, dimension and unit can be defined. To achieve this, the stereotypes <<valueType>>, <<unit>> and <<dimension>> have to be created and they have to be specified to be applicable to UML Data Types. By configuring the stereotypes to define a new element then, the ValueType, unit and dimension elements can be directly created from the project browser and also appear syntactically correct on diagrams.

The dimension property on units and the unit and dimension properties on ValueTypes can be implemented by so-called *tags* in Rhapsody. A tag is a property that can be applied to every model element within Rhapsody. By adding a tag to a stereotype, the element that the stereotype is applied to automatically inherits the tag from its stereotype. Thus, when adding the tag *dimension* to the <<unit>> stereotype and the tags *unit* and

dimension to the <<valueType>> stereotype, every newly created unit and ValueType in the model automatically inherits the tags. A problem is that the tags can not be set to reference to other elements within the model. That is, it is not possible to reference a dimension tag to a concrete dimension model element within the model. The information can only be entered textually. Thus, the consistency between the textual information of the tags and the actual model elements that the tag value should reference to is not automatically maintained in the model. However this problem could be solved by implementing custom tool macros that check the tags for consistency. Figure 3-7 shows the implementation of the aforementioned concept. The unit, dimension and ValueType are all stereotyped UML Data Types. The properties are realized as tags, which are shown in the body of the elements in the diagram.

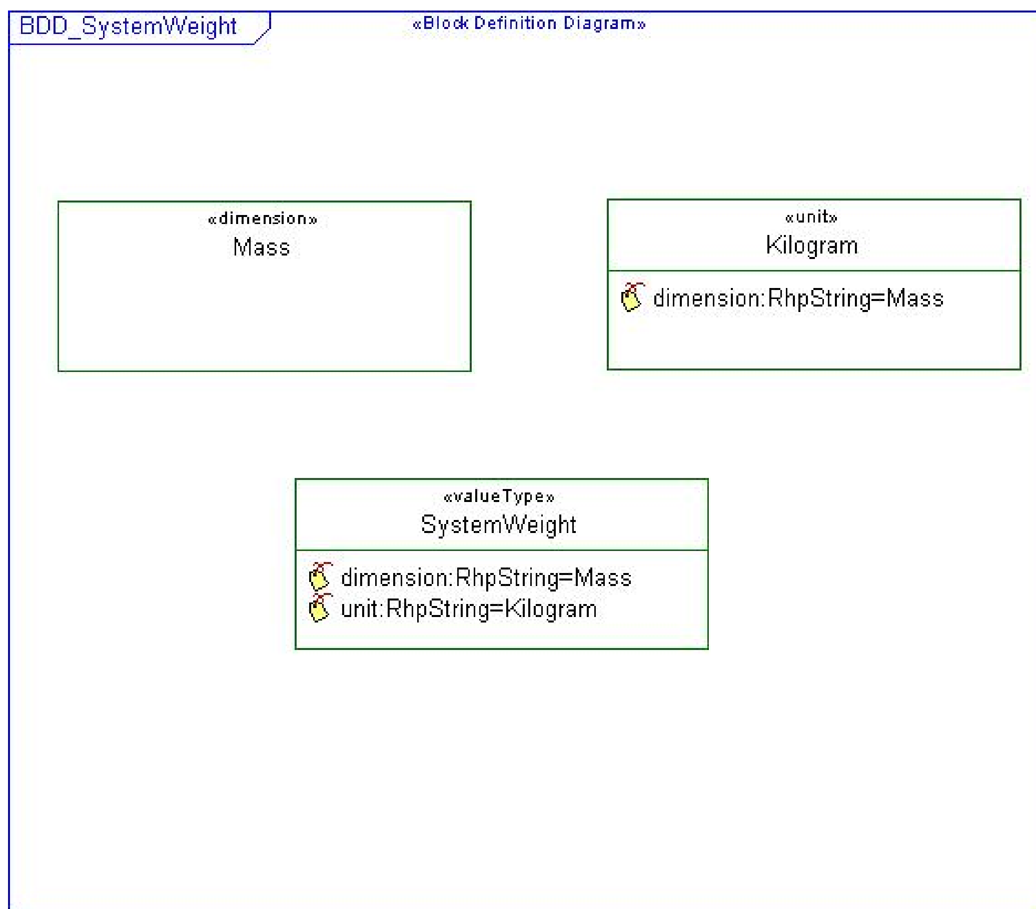

Figure 3-7 ValueTypes, Units and Dimensions in Rhapsody

The corresponding browser view of the example is depicted in Figure 3-8. The ValueType, unit and dimension SysML elements are realized as individual new model elements, using the stereotype feature to define a new model element from a stereotype.

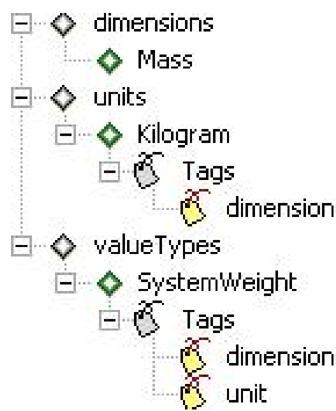

Figure 3-8 ValueTypes, Units and Dimensions in the Rhapsody Browser

Item or data system blocks can be modeled in Rhapsody in a straight-forward manner. With Rhapsody 7.0 MR3, new system blocks can be created simply in the project browser or in block definition diagrams and properties can be added and typed by ValueTypes. However, two problems occur when the item and data blocks are used in the simulation engine:

First of all, if an item or data type is used as part of an interface specification, for example as an argument of an event, and the item or data block is specified in a different model package than the interface, a compilation error arises. The reason is that Rhapsody does not automatically create references from the interface package to the item or data block. To solve this, the references have to be created manually in the model.

The second problem is a little more severe. In cases where item or data blocks are used in item or data flows (refer to chapter 3.5), another compilation errors arises. The simulation engine at some points checks if data has changed. This is done with the C++ negated comparator “!=”. However, blocks are generated as C++ classes by the simulation engine and the “!=” operator is not implemented by default. This results in a compilation error, because the “!=” operator is missing. To solve this problem, the “!=” operator has to be created manually for each block that is used in a data or item flow. This however might be performed automatically with a custom tool script.

3.5 Data and Item Flow

General Analysis and Concepts

In the current process, the system blocks communicate with each other by means of asynchronous *events*. The process defines three types of events:

- request events: A request event requests a certain service from the receiving block. A request event may contain data that is needed for the receiving block to perform the service.
- return events: A return event is the answer to a service request. A return event may contain data or just acknowledge the completion of a requested service
- simple events: A simple event may inform about some event that occurred in the system.

The events are distinguished by naming prefixes, which are "req" for a service request, "ret" for a return event and "ev" for a simple event. Events can contain data. The data that an event can contain are specified as attribute of the event model element.

Currently, data flow can only be modeled by using a service oriented approach: The sending block must send a request to set the data in the receiving block. The receiving block then has to implement the request. Figure 3-9 depicts this concept on a sequence diagram.

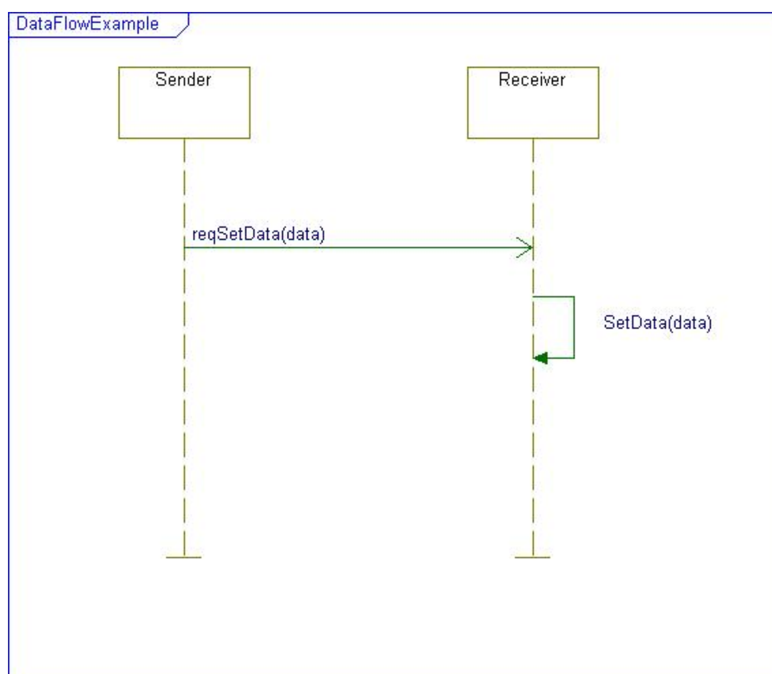

Figure 3-9 Service Oriented Approach to model Data Flow

The communication between the blocks is realized by means of SysML *Standard Ports*. Standard Ports are interaction points over which blocks can communicate. The communication over a port is specified by a set of required and provided interfaces. A required interface specifies the operations a block can call or the events a block can send to its environment over the port. A provided interface specifies the operations that can be called or the events that can be sent from the environment to the receiving block through the port.

Because of the asynchronous communication approach that the current process follows, the interfaces of the ports only specify event receptions.

Ports may be connected to other ports of internal parts of the block owning the port. For that reason, ports have a property to specify if it is behavioral or not: In case of a behavioral port, the incoming events and operation calls are relayed to the block owning the port. If the port is not behavioral, incoming events and operation calls are relayed to the internal parts of the block. Standard ports follow the single-cast semantic: If an event or operation call arrives at a non-behavioral port that is connected to more than one internal part, the message is only relayed to one internal part. *Which one* of the part receives the events or operation calls is not defined.

The concept of communication over ports and interfaces is depicted in Figure 3-10.

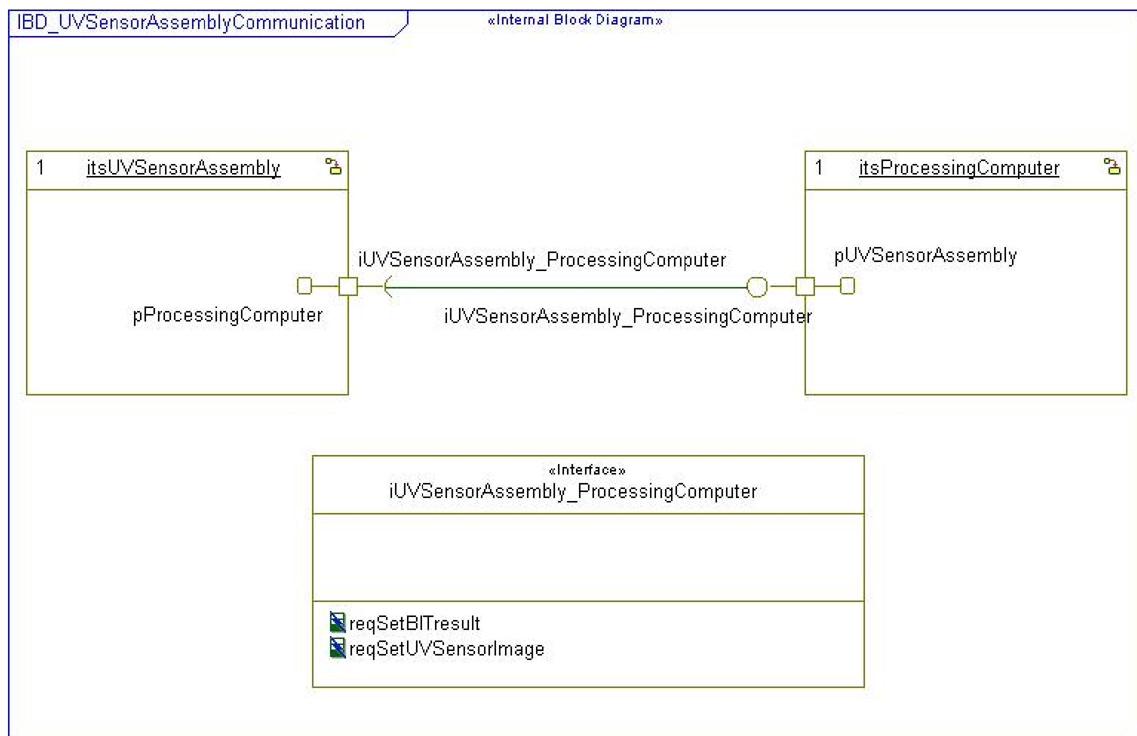

Figure 3-10 Communication between Blocks with Ports and Interfaces

The current approach seems adequate to capture the communication with discrete data or items. However, data or item flows in systems can also be continuous, like for exam-

ple the flow of energy or material, where the interval between the items approximates zero. As already described in chapter 3.4, in the DIRCM example the missile is detected by means of the UV signature emission of the missile engine. That means that a continuous flow of radiation energy is flowing from the missile to the system under design. In the current service oriented approach a *reqSetUVSignature* event would have to be sent from the missile to the DIRCM system. The interval between the individual messages would have to approximate zero to maintain the continuous nature of the energy flow. Following the approach of the current process, the data flow would have to be modeled with UML ports. Figure 3-11 depicts this approach.

However, this concept has disadvantages:

- The association of continuous data or item flow with discrete events is logically misleading
- Continuous flows can not be distinguished from non-continuous flows

Another problem is the single-cast semantic of the standard ports. With standard ports, broadcast communication like bus systems can not be modeled.

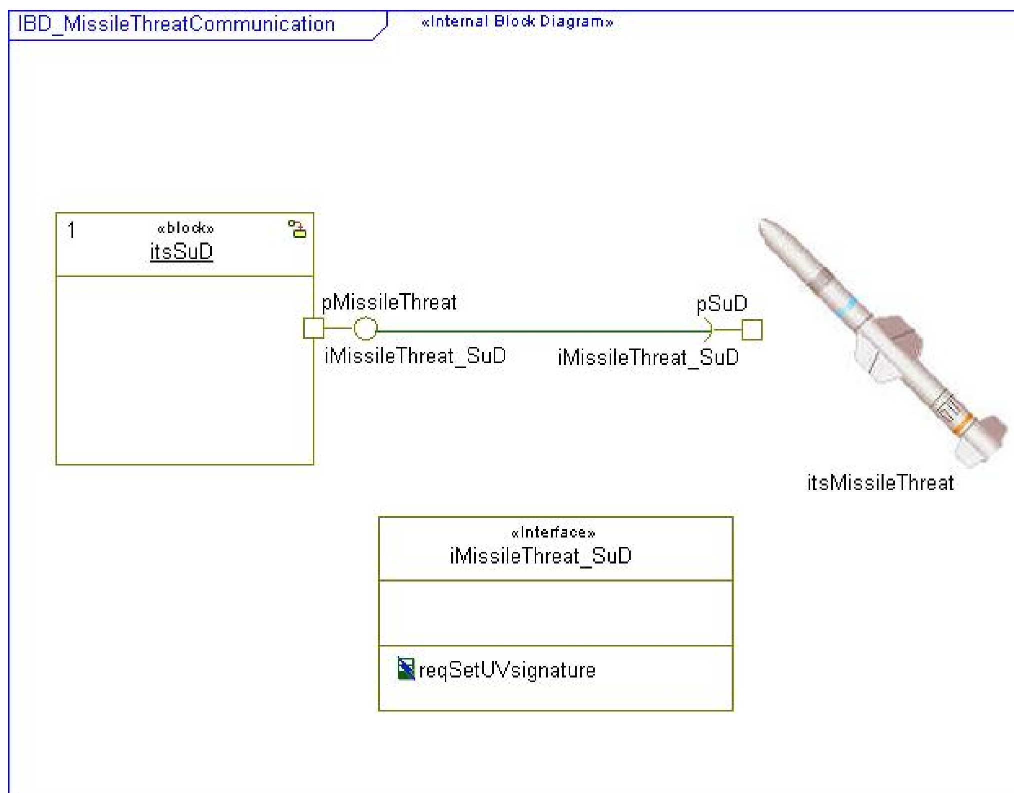

Figure 3-11 Continuous Item Flow with UML Ports

While the asynchronous approach with UML ports is an adequate method for specifying service oriented peer-to-peer interactions, which are common in software component architectures, it is insufficient when continuous flows of energy or material, like torque or fluid, want to be modeled. Also broadcast bus systems like a CAN bus can not be described correctly. The SysML standard port concept isn't designed to model such flows, which doesn't surprise: The SysML standard port is a direct adoption of the UML Port and broadcast communication and continuous flows are normally not needed for the design of software.

The SysML however provides a second approach that addresses the need to model continuous or broadcast item and data flows. This can be done with the SysML *Flow Ports*. Flow ports specify interaction point over which data or items can flow in or out of a block. The communication over flow ports follows the broadcast semantic, that is, data or items that arrive at a port is relayed to *all* connectors. There are two types of flow ports:

- atomic flow port: An atomic flow port specifies one single item or data type to flow through it. The direction of the data type is specified by the direction of the port, which can either be in, out or bidirectional. The item is specified by typing the port with the flow item.
- non-atomic flow port: A non-atomic flow port specifies a set of items that can flow in or out of the flow port. The flow items are specified in a *flow specification.* The direction of the individual items is specified per item in the flow specification.

A non-atomic flow port can be conjugated. If a flow port is conjugated, the direction of the flow items specified in the flow specification is reversed. This is done to specify the correct flow direction of a sender and receiver flow port.

Blocks owning a flow port have to implement the items that are specified either in the flow specification in case of non-atomic flow ports or directly on the flow port in case of non-atomic flow ports. The implemented items must match the specified items in terms of both name and types. The implementation is the physical link between the specification and the concrete values and items that flow. That means that an item or data that arrives at a flow port is relayed to the implementation and vice versa.

Figure 3-12 applies the SysML flow port concept to the UV signature flow between the missile and the system under design in the DIRCM example. The UVSignature item is specified in the flow specification *ifUVSignature*. The black color of the flow port on the system under design marks the port as conjugated. This way, it can be seen from the

diagram that the UV signature flow direction is from the missile to the system under design.

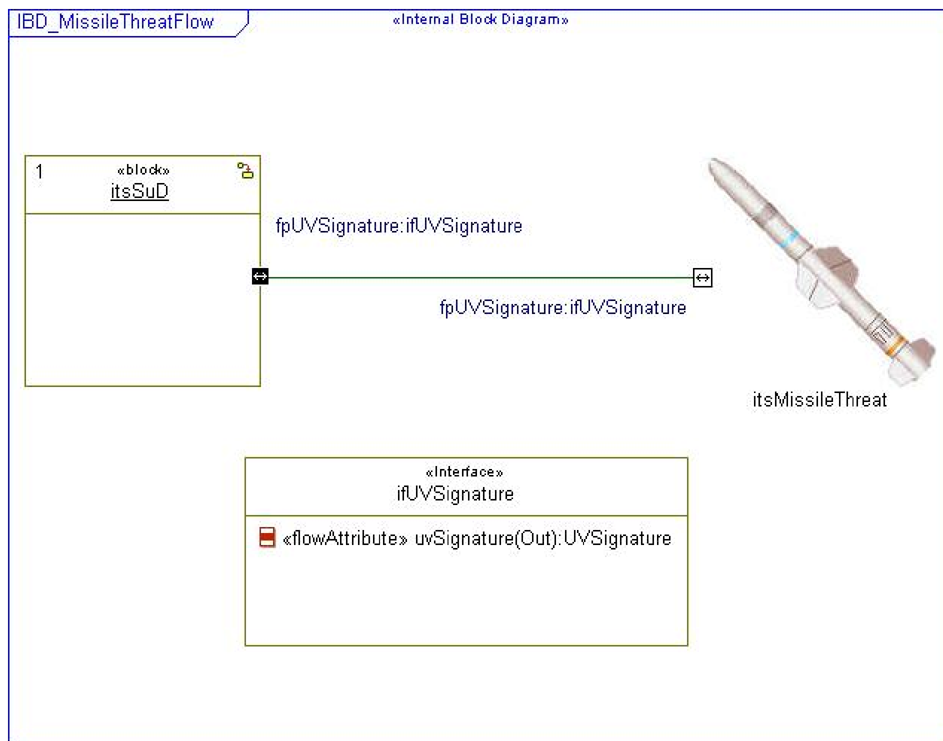

Figure 3-12 Continuous Item Flow with SysML Flow Ports

The example makes use of non-atomic flow ports although an atomic flow port could also be used because only one flow item is specified. However, the usage of a flow specification has the advantage that the flow is specified in a clearly defined interface. With non-atomic flow ports, the flow items are only specified directly on the ports.

Implementation in Rhapsody

Rhapsody supports SysML flow ports in the SysML profile that is provided in the version 7.0 MR3. Both atomic and non-atomic flow ports are supported. Flow specifications can be created by adding one or more properties with the stereotype *<<flowAttribute>>* to a common interface. The flow attributes can be typed by both ValueTypes and data or item blocks. The direction of a flow attribute can be set in a *direction* tag that is inherited for each newly created flow attribute. All three directions, in, out and bidirectional can be set. Port conjugation is also supported. The Rhapsody implementation is depicted in Figure 3-12.

Flow port simulation is supported by means of an automatically generated *set* operation for each flow attribute that is implemented in the block owning the flow port. When the *set* operation is called within in the simulation, the flow attribute is automatically updated with the set value in the receiving block that is connected to the flow port. The

call of the *set* operation can be traced in animation sequence diagrams and appears as synchronous message. Figure 3-13 depicts the flow port communication concept on a sequence diagram. In the example, the missile updates its emitting UV signature. The value is then automatically updated in the *SuD* block by means of a synchronous operation call.

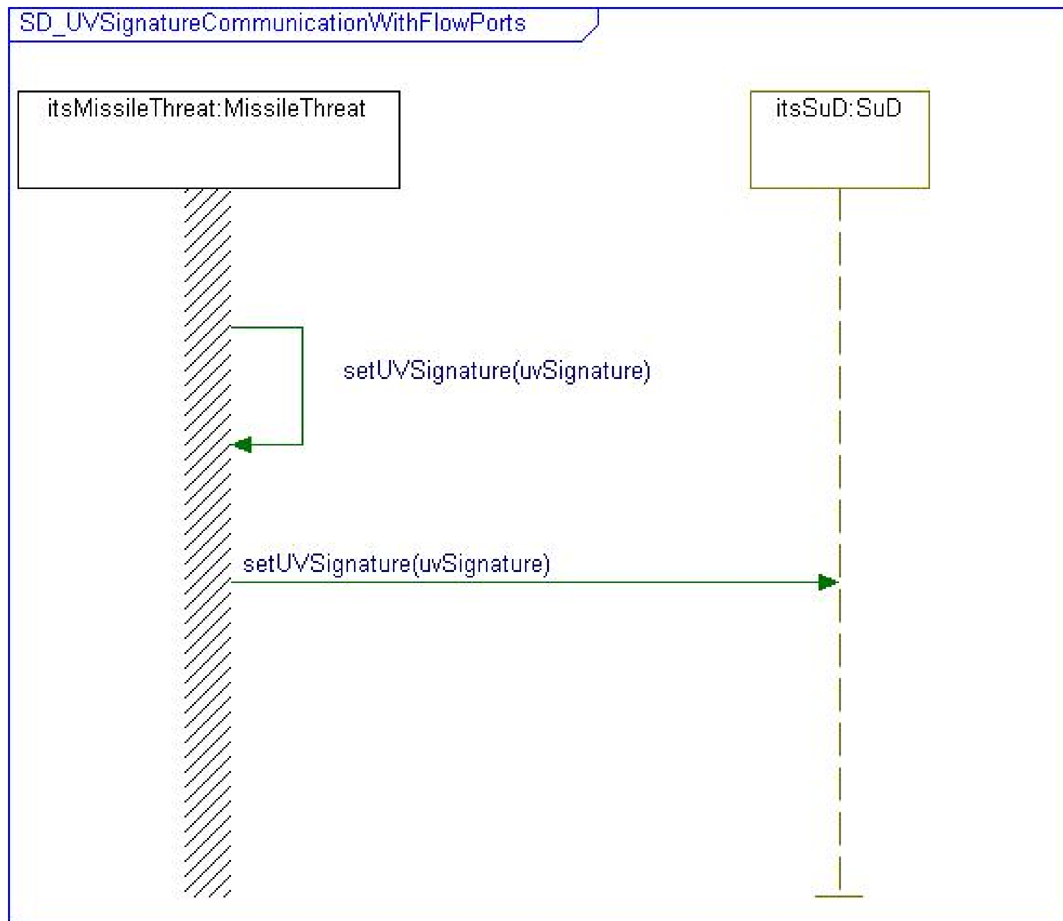

Figure 3-13 Flow Port Communication in Sequence Diagrams

Support for flow ports in state charts is also provided: Whenever an attribute of a block is changed by an incoming flow attribute value, an event is generated in the receiving block. The state machine can then react to the event. This concept can be shown on a simple example: Figure 3-14 shows a voltage flow from a power supply to a computer mainboard. The flow is realized by means of flow ports between the *PowerSupply* and the *Mainboard* blocks. The mainboard reacts to a change of voltage in its state chart, which is shown in Figure 3-15. Every time the voltage changes on the power supply and thus also on the mainboard, the *chVoltage* event is thrown. The mainboard reacts to the event in that it changes from the *Off* state to the *On* state when the voltage is 12 Volts and from the *On* state to the *Off* state when the voltage is below 12 Volts.

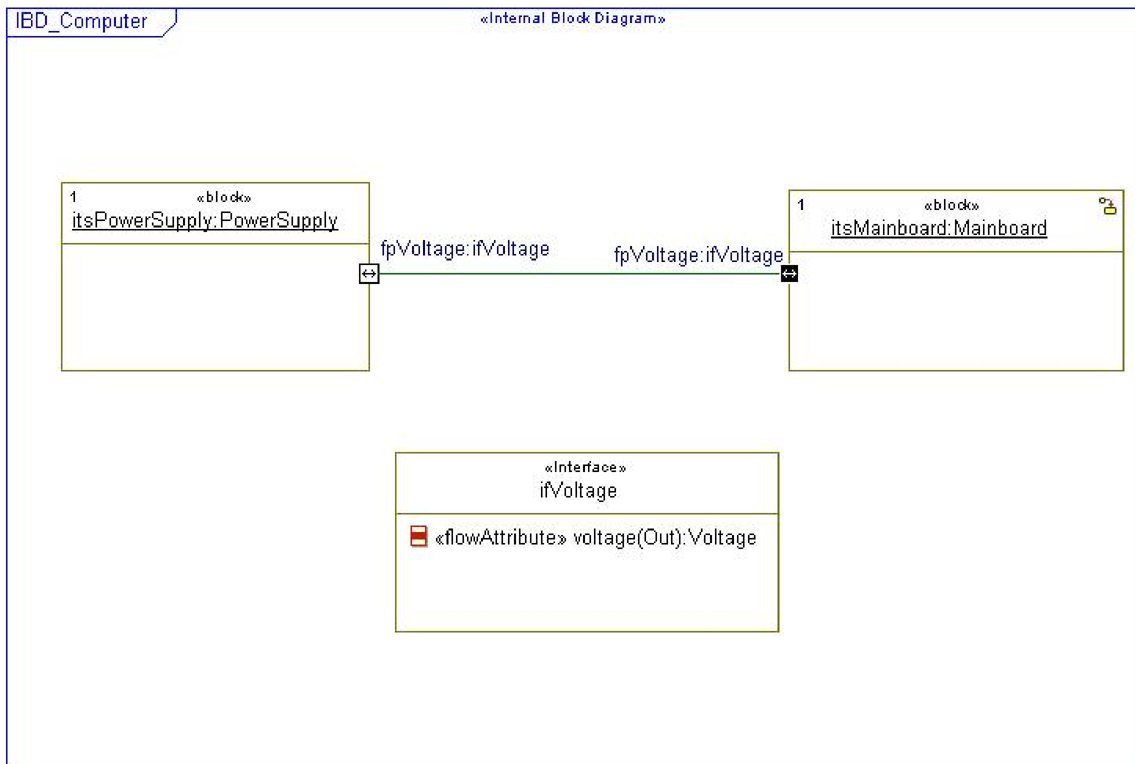

Figure 3-14 Voltage Flow over a Flow Port

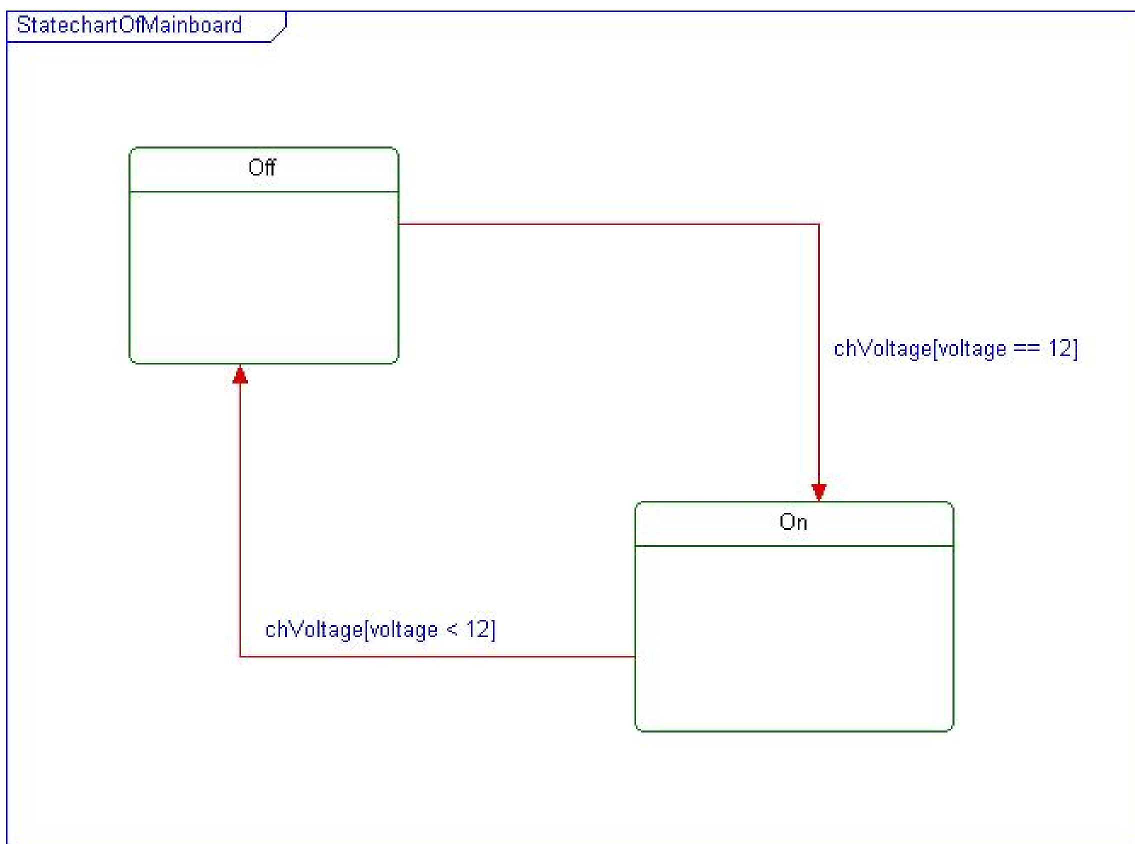

Figure 3-15 Flow Port Support in State Charts

However, the simulation lacks support for one of the most important features of flow ports, the broadcast messages. The values of ValueTypes and item or data blocks are not relayed when more then one part is connected to a port. Thus, bus systems for example cannot be simulated in Rhapsody. This significantly minimizes the value of using flow ports from the simulation perspective.

An additional problem is that only ValueTypes and item or data blocks can be specified as flow items. It is not possible specify events.

Another thing to consider is that the decomposition of flow specifications is not possible. When more than one flow item is specified in a flow specification at a higher abstraction level of a system, the individual flow items can not be relayed to individual parts when decomposing the system. The consequence is that a separate flow specification and with it a separate flow port have to be created for each flow item already at the top-level tier of the system. This is necessary to enable a maximum degree of freedom to assign the individual flow items to parts later in the design process.

3.6 Segment Handover

In case that the individual segments will be developed by different design teams, the current process describes that model handovers have to be created. The segments themselves usually consist of both hardware and software and are not decomposed in concrete hardware and software units yet. While the process describes that handover models have to be created, it doesn't describe in detail how they can be created. Thus concepts have to be elaborated to create a model handoff that both support the collaboration between the different teams and the traceability to the high-level context.

When beginning thinking about a sensible way to create a model handoff technically, first of all it has to be thought of what model artifacts are essential for the subsequent development teams. These are:

- *the system block representing the segment*: This block contains both the operational contracts as well as the state based behavior and ports to the environment.
- the segment *use cases*, *scenarios* and *activity diagrams*
- the *interfaces* to the neighbor systems and actors
- the *context* in that the segment fits by means of a block definition diagram and internal block diagram

Segment Block

The segment block can in principle be handed over “as is”. The attributes, ports, interfaces and state based behavior of the block build a self-contained entity and thus can remain unchanged.

Context

For the development of segments, the functionality of the neighbor segments is not relevant in the first instance. The model handover should represent the segment as black box. Thus, the external systems should not appear as system blocks in the handover model but instead as actors. However, the actors should reassemble the original ports of the system blocks, which also should be linked in a concrete context to the segment block.

Use Cases

In the system-level model, the use cases are detailed with sequence and activity diagrams per system level use case. For the segment development, the system level use cases might however be too abstract to be handed over directly. For example, in the DIRCM system, the main system-level use case is *DetectAndDefeatMissile* (Figure 3-16). The system components are mounted on a rotary turret (the system block *TurretAssembly* in Figure 3-3), whose primary function is to move the turret to a commanded position. Handing over the abstract system-level use case *DetectAndDefeatMissile* to the turret development team would be improper.

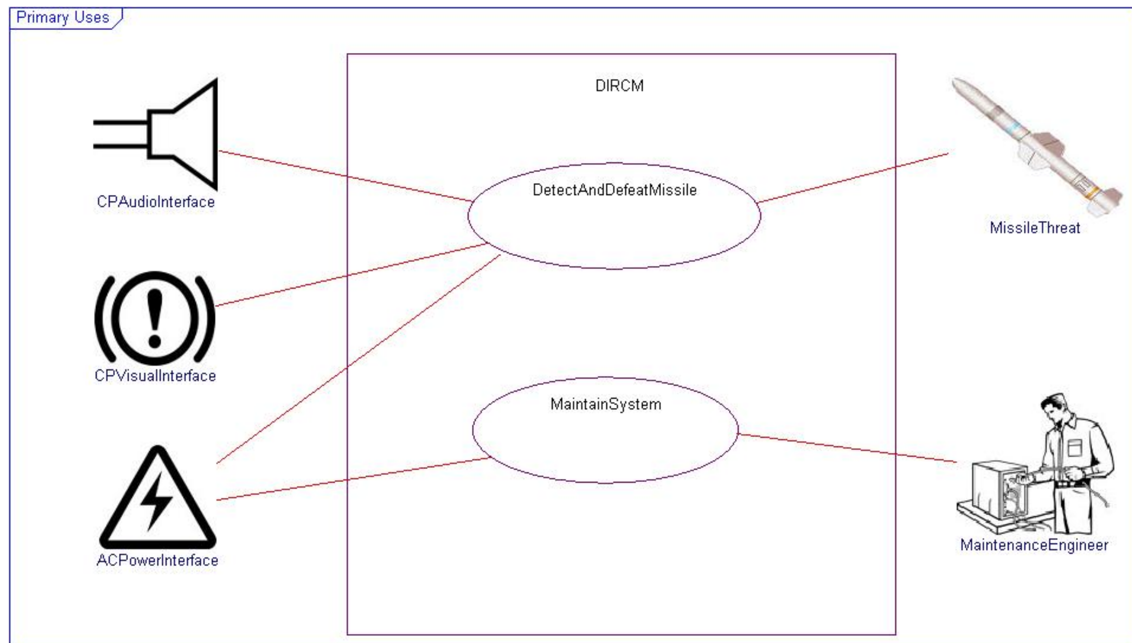

Figure 3-16 Use Cases of the DIRCM System

A way to address this issue is to decompose the system-level use cases into segment-level use-cases. Segment use cases can be identified by the operational contracts of the segment blocks. The individual segment-level use cases can then be allocated to the appropriate segments. This narrows the use-case context for the individual segments. However, by linking the segment-use case to the system-level use case in the model, a "bottom-up" view is provided to see how the segment-level use case fits into the context of the system-level use case.

The allocation should be performed in the system-level model on a separate use-case diagram as an additional view on the use-case model. A way is to show the use-case allocation on a separate use-case diagram per system-level use case. This helps for a concise use-case model structure. Figure 3-17 shows the decomposition of the *Detec-tAndDefeatMissile* use case in the DIRCM example. The decomposition is performed by means of include dependencies from the system-level use case to the segment-level use cases. The segment-level use-cases are allocated by setting the segment-level use cases into new system-boundaries that represent the segments. To mark the abstraction-layer of use cases, the stereotypes <<*SystemLevel*>> and <<*SubsystemLevel*>> are used.

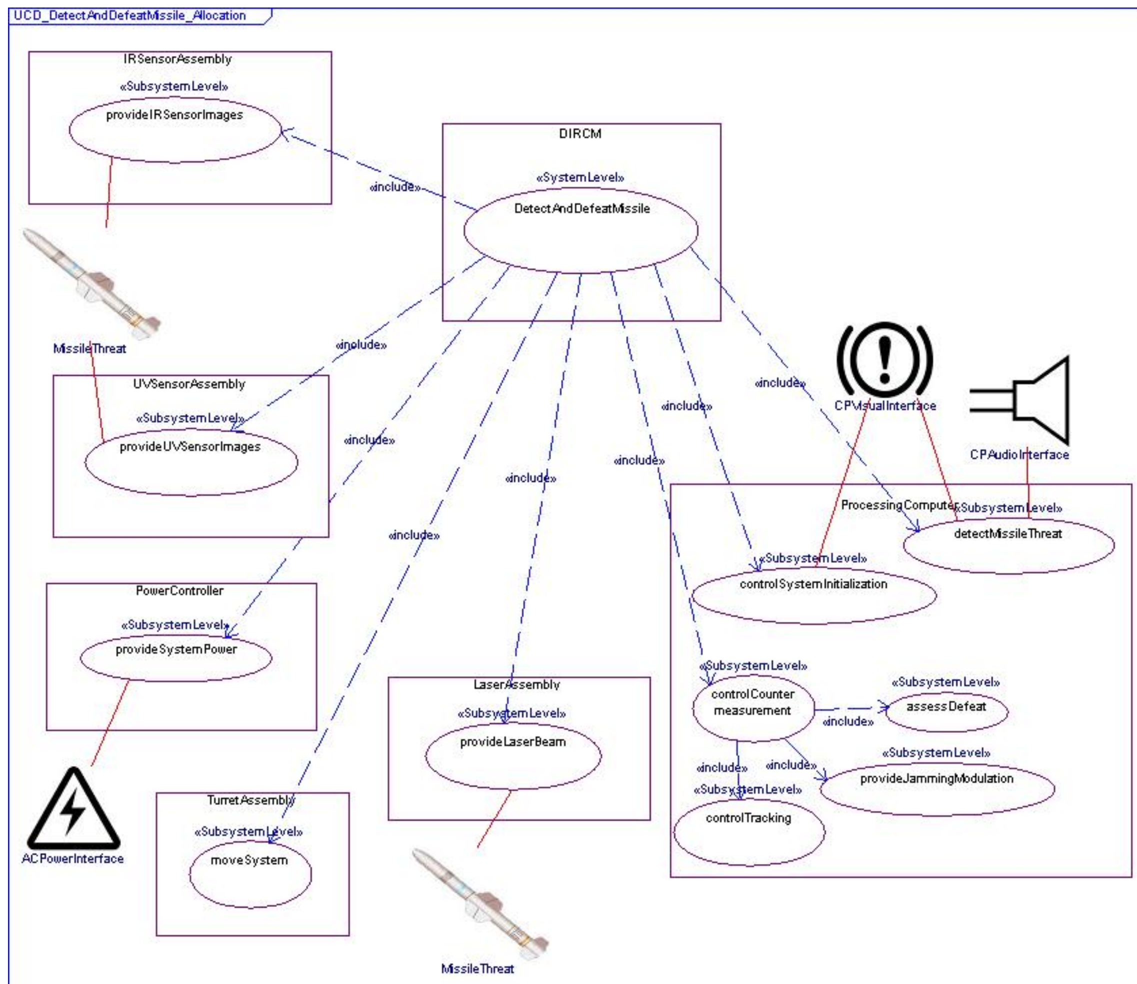

Figure 3-17 Use Case Decomposition and Allocation

When the use cases are decomposed and allocated to segments in the system model, the decomposed use cases can be handed over to the segment development team. New scenarios and activity diagrams then have to be created for the segment use cases. These scenarios and activity diagrams can be derived from the system-level use case.

In the scenarios, the newly created actors should then be used to represent the neighbor systems. These are then automatically drawn with hatched lifelines.

The activity diagrams can then be modeled as black box diagrams: Control and item flows in the system level use case diagrams then have to appear as event receptions, send signal actions and activity parameters.

Interfaces

The interfaces between the segments and the actors are a vital part of the specification. These shouldn't be modified by the segment teams and have to stay consistent on system level. Therefore the interfaces should be handed over as referenced read-only package.

"Bottom-Up" View

For the segment development teams, it is often necessary to see how their individual segments fit into the overall context of the next-higher system abstraction tier. Narrowing the focus of segment development to a black-box only perspective may result in misinterpretations on the interfaces and the functional roles in the overall context. A way to address this issue is to import the complete higher-level system model into the individual segment models. The segment teams can then browse to the next-level system-model context if required. By importing the higher-level model as read-only reference, care is taken that the higher-level system model can not be modified by the segment-level team.

Model Verification

When the segment model has been created, it should be verified against the higher-level system specification by means of model execution. This guarantees that the segment model has been created correctly.

3.7 Modeling the Physical Architecture

Modeling Basic Software and Carrier Hardware

The modeling approach covers mainly functional aspects of systems. That means that only hardware and software is modeled that provides the essential system functionality. Examples are functional hardware like FPGAs or application and interface software units. However, especially software units cannot run stand-alone but require a carrier computer hardware that consists of CPUs, RAM, bus systems etc. In addition, also basic software like an operating system and a board support package is required. Another thing to consider is the communication between the functional units. While communication in functional models is only specified from the logical perspective, in the end the communication is realized by means of communication over physical hardware interfaces. The concept can be seen as layered system structure, as depicted in Figure 3-18.

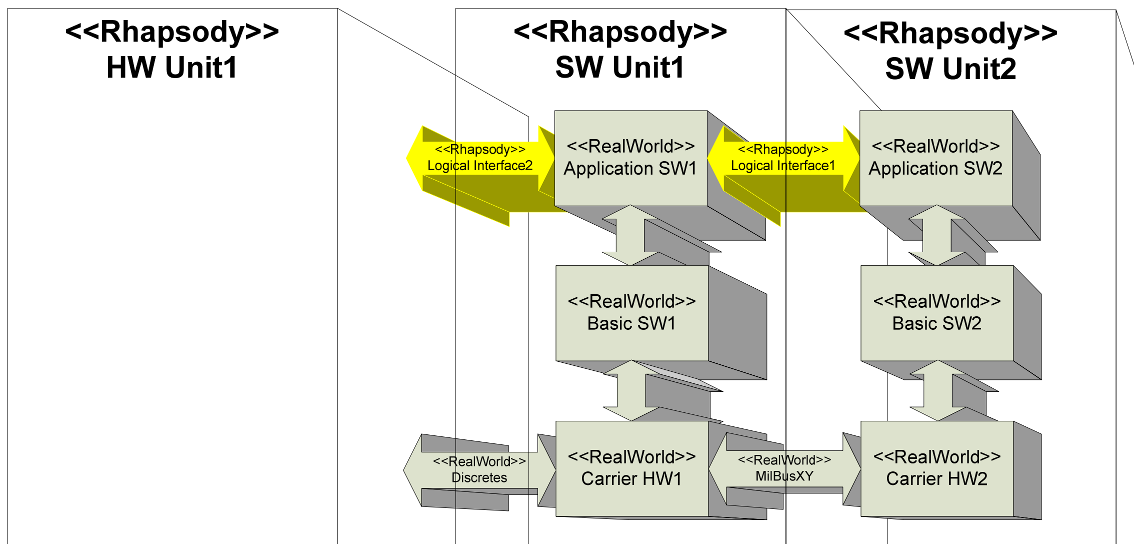

Figure 3-18 Functional vs. Physical World

Application software, basic software and carrier hardware are developed by different specialized teams. Past experiences of the OPES4 have shown that there were often difficulties in the communication between the development teams especially regarding startup sequences or the distribution of functionality. Another problem is the allocation of timing requirements. For example constraints regarding maximum startup times or communication delays are only specified for the overall functional system block in the system models. However, these have to be broken down in timing constraints for the functional software, the basic software and the carrier hardware.

A possible approach to address these issues is to bring the basic software units as well as the carrier hardware into the SysML model. The break-down of timing constraints is shown in an example. Figure 3-19 shows the structure of an external segment and a software unit that are connected by means of logical ports. The basic software and carrier hardware of the software unit are not yet considered. Figure 3-20 depicts a communication scenario between these two blocks. The external segment sends a request to perform a computation. The software unit then reacts to the request in that it performs the computation and replies with a return message. The time between the input of the request and the return message is specified to be maximal 120 milliseconds by means of a timing constraint model element.

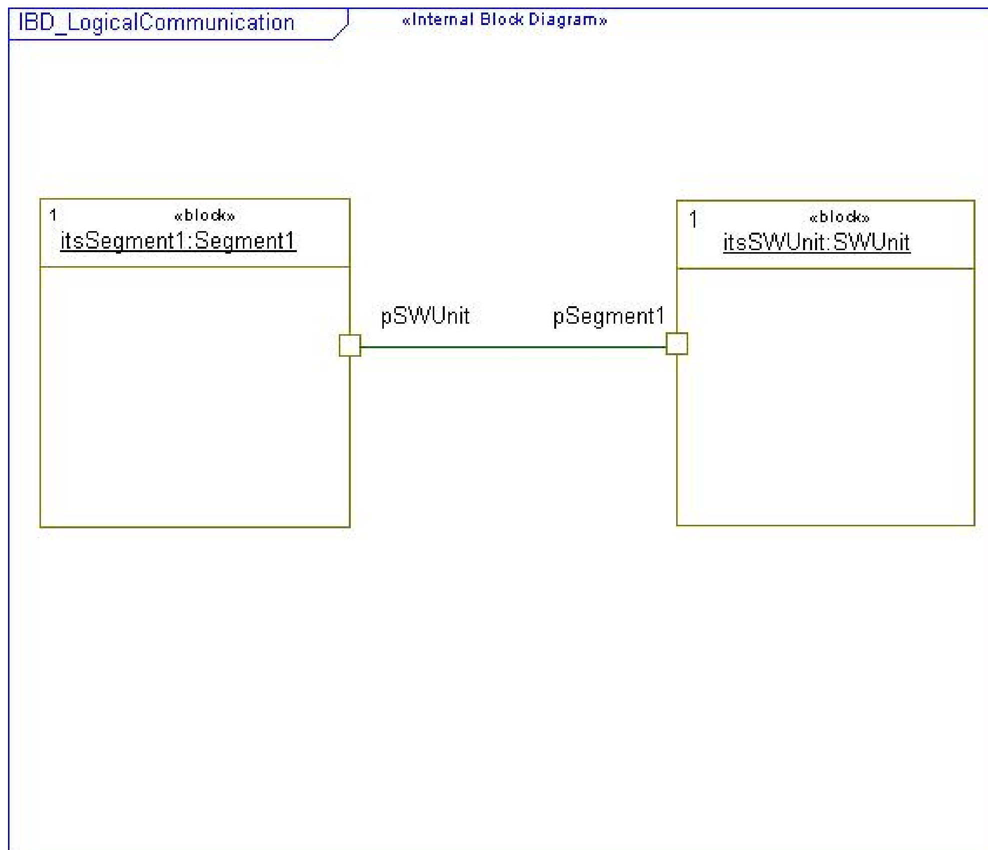

Figure 3-19 Internal Block Diagram showing Logical Communication Structure

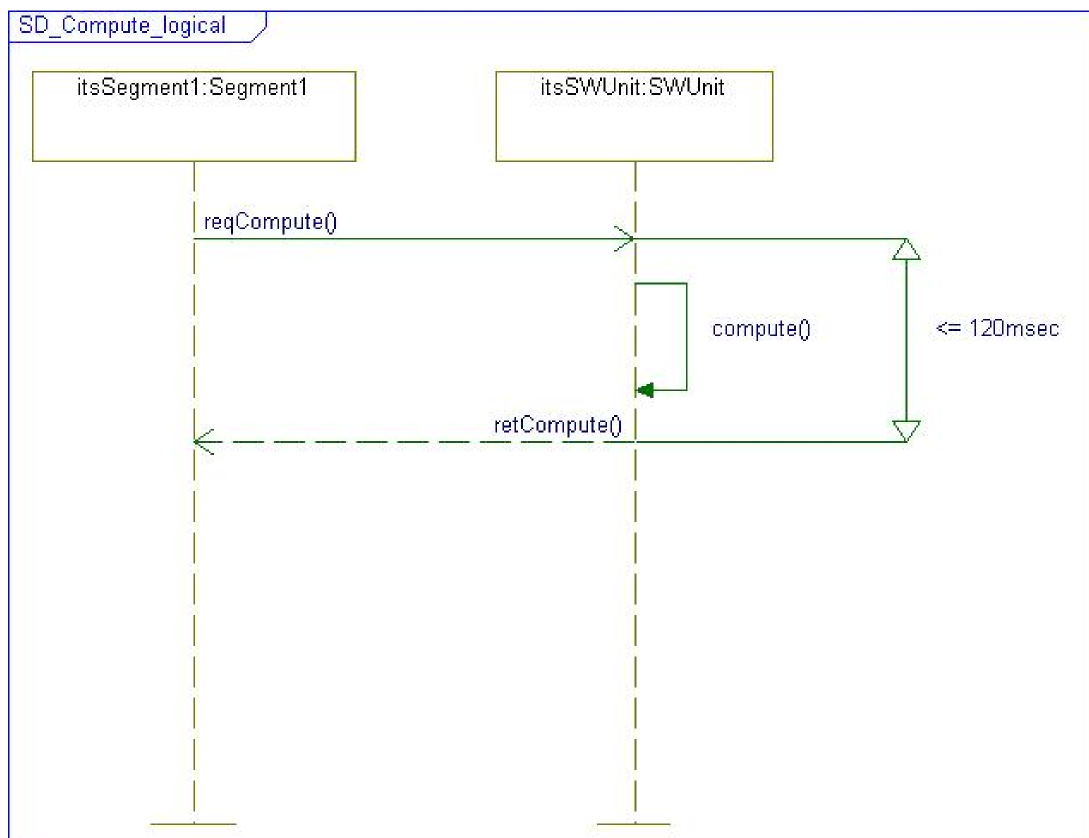

Figure 3-20 Logical Communication in a Sequence Diagram

Figure 3-21 shows the same example but considers both the basic software and carrier hardware. These are modeled as blocks and are linked together with the functional software unit in a layered structure. The external segment now communicates directly with the carrier hardware.

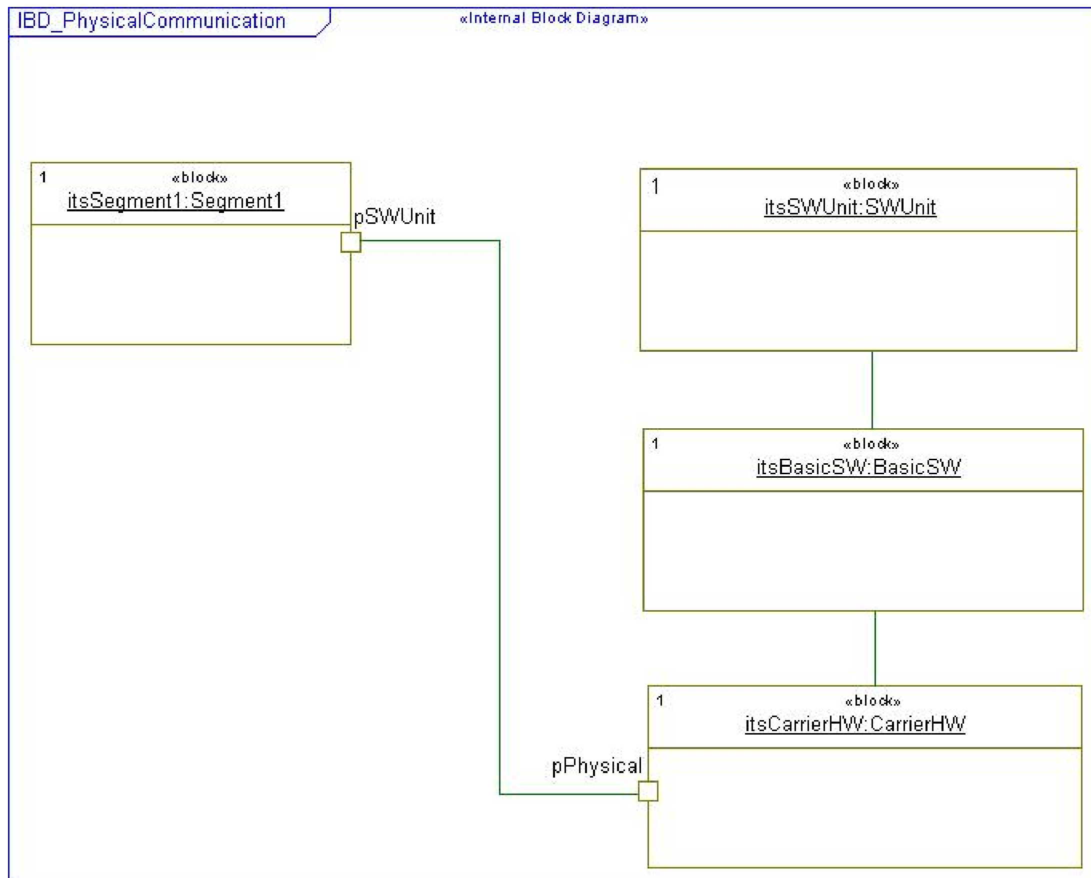

Figure 3-21 Internal Block Diagram showing Physical Communication Structure

The communication is depicted in Figure 3-22. The external segment now sends the request to compute to the carrier hardware, which relays the request to the basic software. The basic software again relays the request to the software unit which performs the computation and sends the reply back in the same communication hierarchy. By adding the carrier hardware and the basic software into the communication, it is possible to decompose the system-level timing constraints into individual timing constraints.

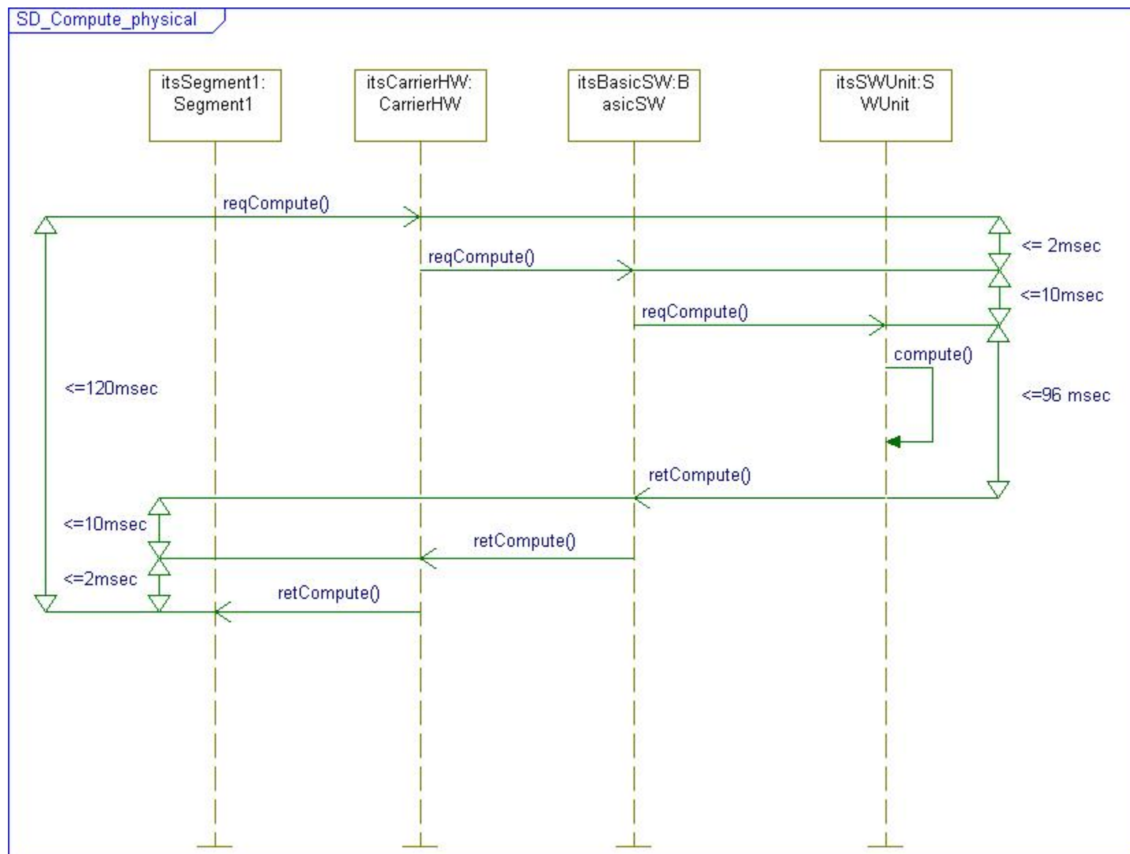

Figure 3-22 Physical Communication in a Sequence Diagram

The above example only considers timing aspects, which are modeled by introducing relay messages. These timing aspects are an important information for the specification of the carrier hardware, the basic software and the functional software units and are thus in the scope of systems engineering. The specific functionality that is implemented in the carrier hardware and the basic software to enable the hierarchical communication is not in the scope of systems engineering but in the scope of the specialized development teams. However, the models can be refined in collaboration with the specialized development teams. The refinement can be in principle made as far as behavior in form of state charts, concrete programming language functions and even access to hardware control registers can be modeled.

Deploying HW/SW Units on Physical Architecture

Especially if more than one functional software and hardware unit will be deployed on a single physical architecture and thus share resources like CPUs, RAM and buses, it makes sense to integrate the deployment information into a SysML deployment model. Integrating the deployment into a model has two advantages:

- The overall resource-sharing can be shown visually and thus can support the performance and resource analysis on system level.
- The developers of the individual hardware and software units can see how their functional specifications map to the physical constraints and environment.

The deployment can be performed in multiple views in the model, with each showing a different deployment aspect such as CPU, RAM or bus deployment.

Figure 3-23 shows the decomposition of the processing computer segment of the DIRCM example system. The processing computer is decomposed into the software units *ImageProcessing*, *SequenceControl* and *BasicSoftware* and the hardware unit *JammingControl*. The stereotypes <<*SWUnit*>> and <<*HWUnit*>> have been introduced to distinguish the types of units.

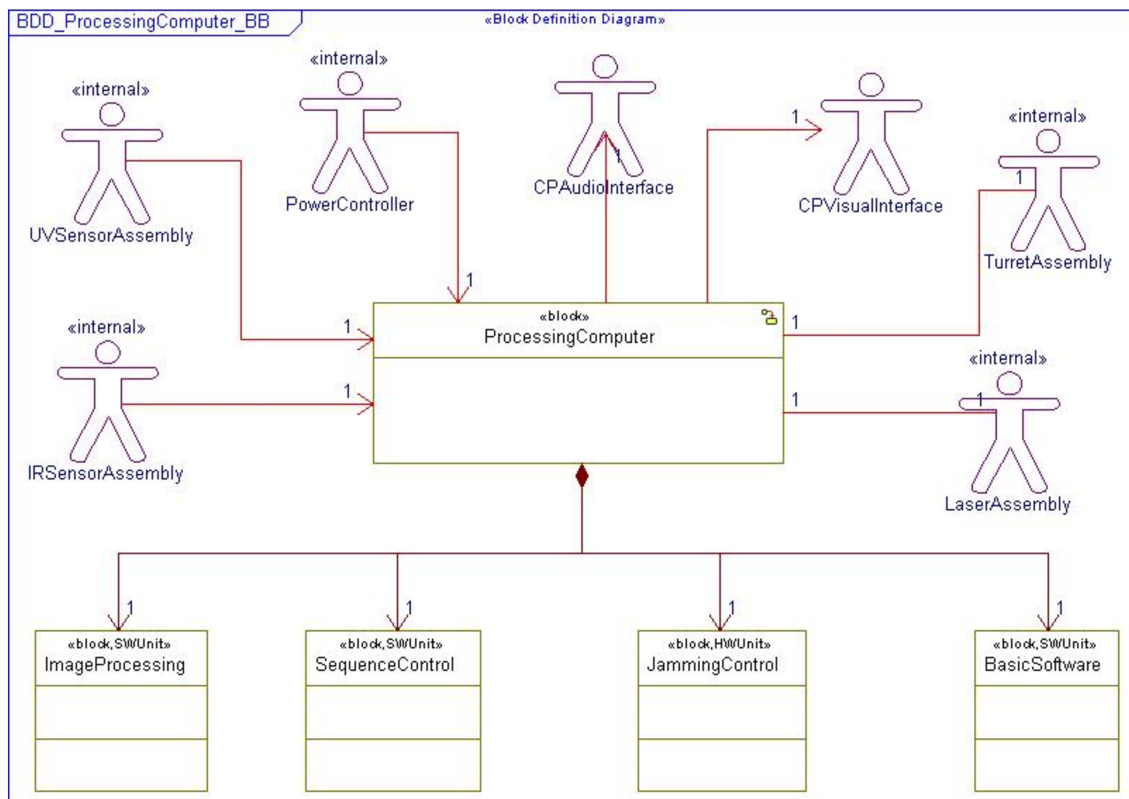

Figure 3-23 Decomposition of the ProcessingComputer Block in the DIRCM Example

The units shall all be deployed on a carrier hardware architecture. This architecture is modeled by means of an internal block diagram. This creation of a block definition diagram is omitted because for the architecture, only the concrete internal hardware structure is of interest. Figure 3-24 depicts the architecture for the processing computer carrier hardware. The computation units in the architecture are two CPUs and one FPGA. The two CPUs have access to individual RAM components and are connected via a standard PCI bus. The FPGA is connected to CPU2 via an I²C bus. In addition, CPU2 has access to an Ethernet interface and the FPGA is connected to discrete I/O lines.

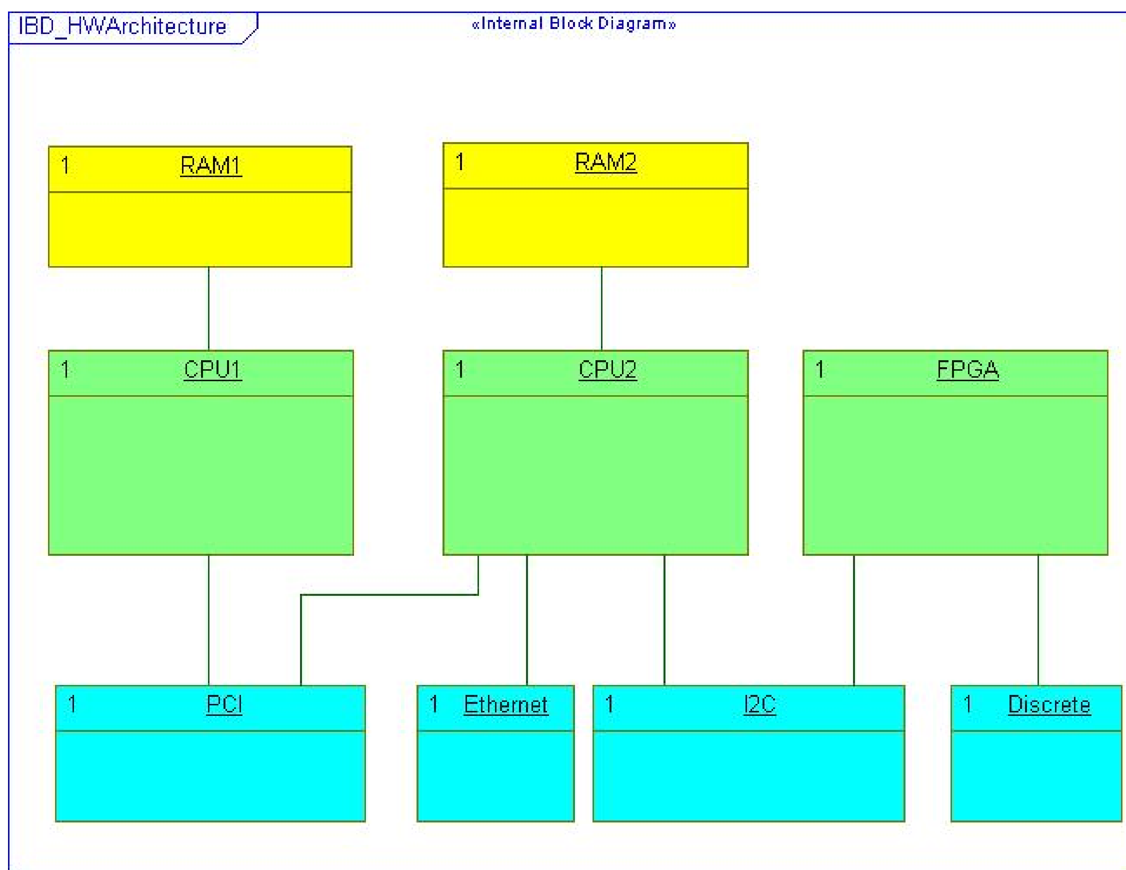

Figure 3-24 Carrier Hardware Architecture

Additional information about the individual hardware components can also be modeled, such as CPU frequency, RAM sizes, address spaces, bus speeds etc. The information can either be modeled as tags (refer to chapter 3.4) of the hardware parts or as a textual annotation. However, tags should be preferred because they can be directly accessed through the model browser and also be used by custom tool scripts that, for example, support document generation or perform analysis tasks. An example is shown in Figure 3-25. For the CPU, the information about manufacturer, model and frequency is modeled by means of tags.

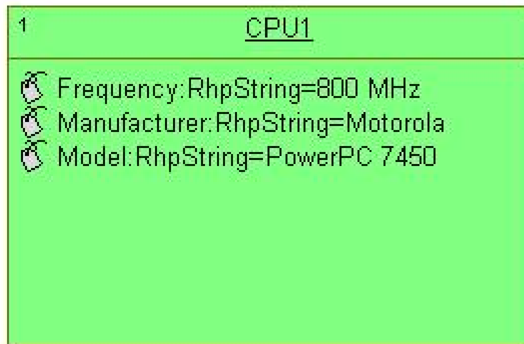

Figure 3-25 CPU Block with Information Tags

When the carrier hardware architecture has been brought into the SysML model, the different deployment views can be shown on individual internal block diagrams. The deployment is based on the basic hardware architecture.

Figure 3-26 shows the deployment of the hardware and software units in the DIRCM example to the processors of the carrier hardware. The deployment is realized by means of the SysML allocation relationship. The allocation relationship allows allocating basically every model element to another, such as behavior to structure, software to hardware, logical to physical entities etc. The original SysML stereotype of an allocation relationship is <<*allocate*>>. However, the allocation in Rhapsody is natively realized with the <<*allocation*>> stereotype.

In the example, the ImageProcessing unit is allocated to the CPU1, the SequenceControl to CPU2. The BasicSoftware is allocated to both CPU1 and CPU2. The hardware unit JammingControl is allocated to the FPGA.

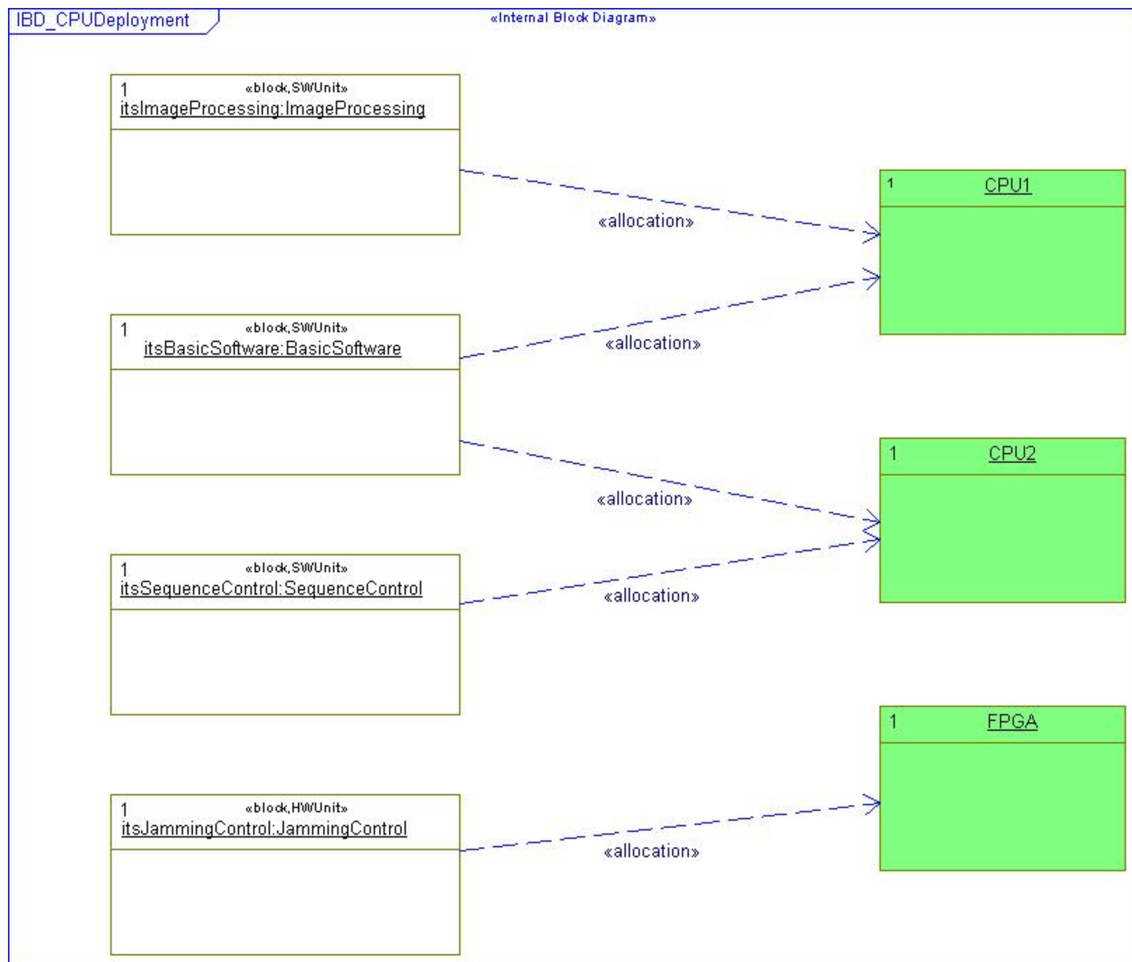

Figure 3-26 CPU Deployment

The allocation relationship can also be extended with comments or tags that contain more information. For the CPU allocation, additional information could be for example the maximum CPU time or scheduling information.

Another deployment view can be the memory deployment. In the memory deployment view, the allocation of software units to physical memory can be shown. The memory deployment for the software units in the DIRCM example is depicted in Figure 3-27. The ImageProcessing unit is allocated to RAM1, the SequenceControl to RAM2 and the BasicSoftware to both RAM1 and RAM2. As in the CPU deployment, the allocation relationship can be extended with additional information like address spaces or memory partitions by adding comments or tags to the relation.

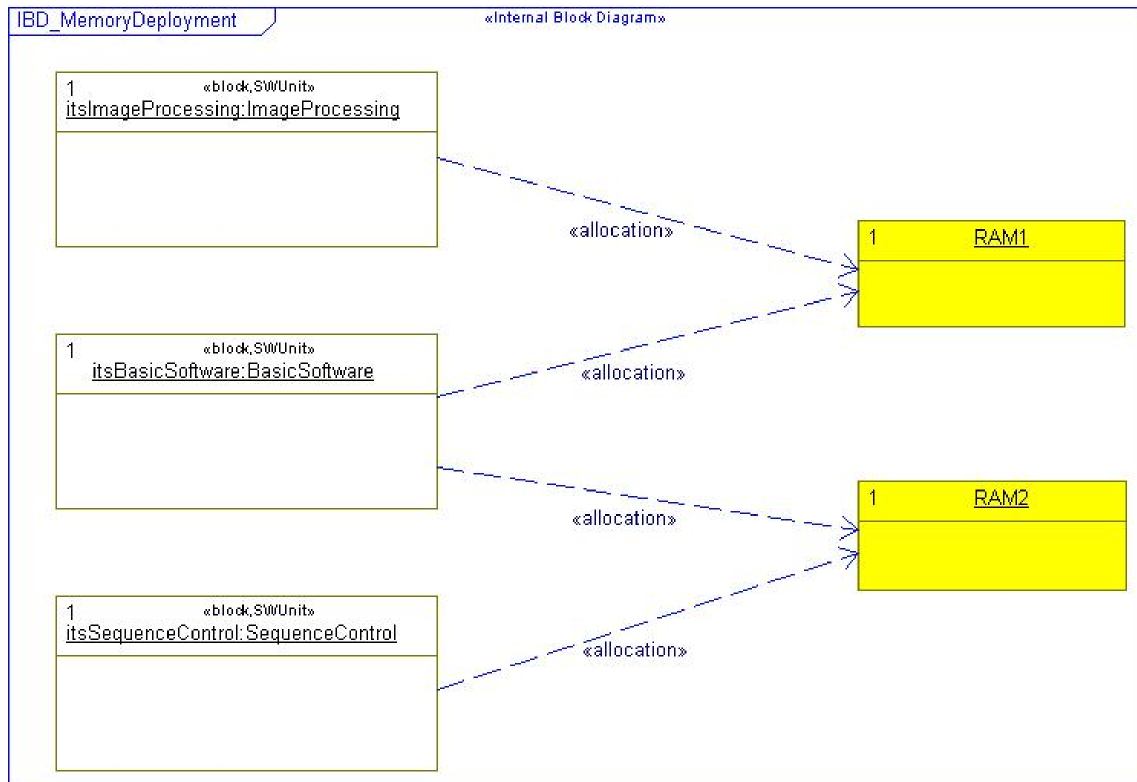

Figure 3-27 Memory Deployment

An additional possible deployment view is the interface deployment. In the interface deployment, the allocation of logical communication to physical interfaces can be shown. Figure 3-28 shows an excerpt of the logical communication between the SequenceControl, ImageProcessing and JammingControl units of the DIRCM Processing segment.

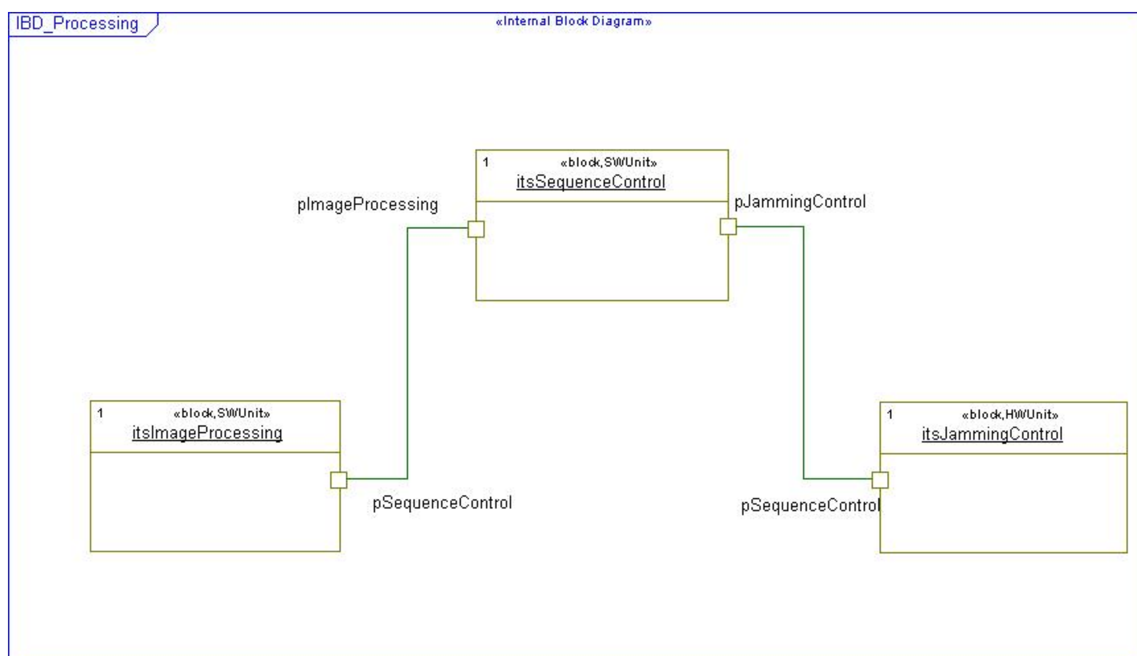

Figure 3-28 Logical Communication in the Processing Segment (excerpt)

By using the allocation relation, the logical communication interfaces can be mapped to the physical interfaces of the carrier hardware architecture. Figure 3-29 depicts this concept. The logical communication between the ImageProcessing and the SequenceControl blocks is mapped to the PCI bus by means of an allocation of the ports. The communication between the SequenceControl and the JammingControl blocks is allocated to the I²C bus. A semantically better way would be to directly allocate the link to the bus blocks. However, Rhapsody does not allow the allocation of links, thus the ports are allocated to the buses.

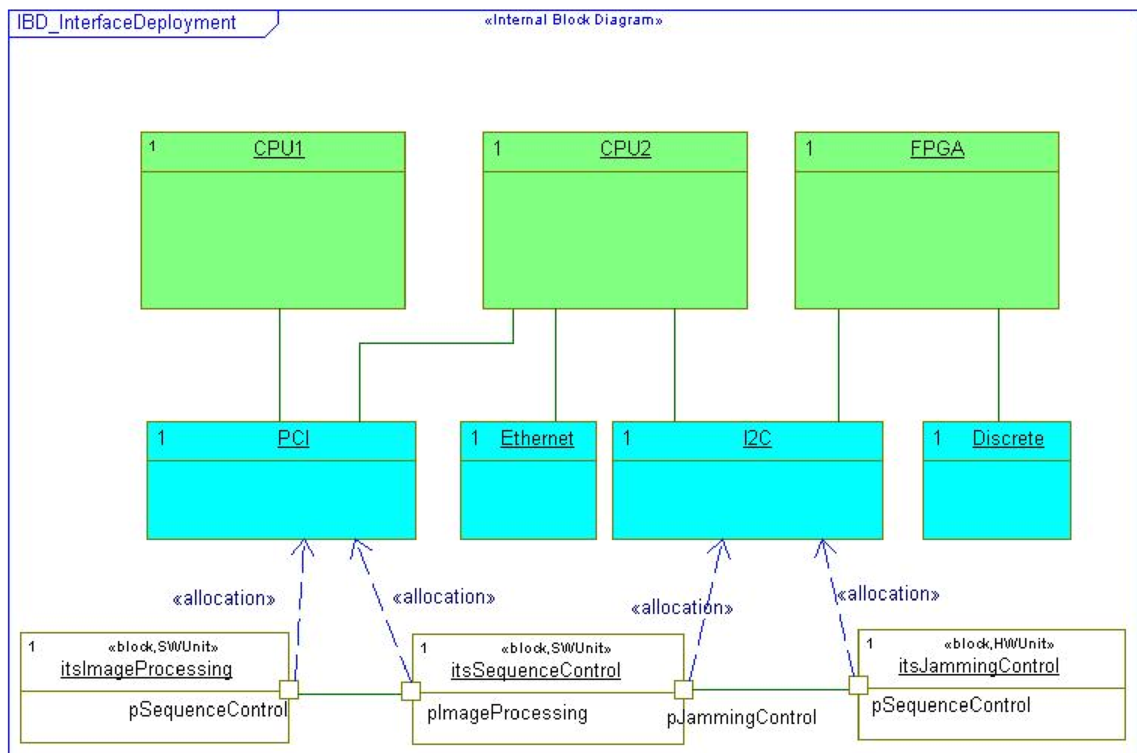

Figure 3-29 Interface Deployment

As with the two preceding deployment views, the interface allocations can be extended with additional information like for example bus addresses or the maximum allowed bus load.

3.8 Transition to Hardware/Software

A main goal of the application of modeling techniques to the specification of systems is the improvement of communication between the different engineering disciplines. The ideal case would be that the subsequent hardware and software development teams can perform their unit-level analysis tasks either directly on or at least very close to the SysML unit specification models. To enable this, the handover models have to be well specified and configured.

The concept for a unit-level handover model in general should cover all aspects that were already elaborated for the segment handover (chapter 3.6). The handover should contain the unit functional and behavioral specification, the architectural context, the interfaces and the refined unit-level use cases. To provide the "bottom up" view, the higher-level segment-level model should also be referenced, especially when the physical deployment that was elaborated in chapter 3.7 is considered.

However, at present it is unlikely that the hardware development teams continue their development tasks on the SysML model. For hardware engineering other description languages such as VHDL and constitutive tools are common. The SysML model is likely to act as a pure unit specification only.

The case is different for software development. In software engineering the language UML has become the de-facto modeling standard. Because of the close relation of SysML to UML, many design elements can be reused. In case that the subsequent software development team also use Rhapsody for the development of software, the development activities for software development can continue based on the SysML unit model in a seamless manner. Rhapsody bases the SysML language elements on native UML elements, so SysML and UML can be both used within one model.

The concept of a possible transition from system specification to software analysis will be shown on an example. Figure 3-30 shows a functional sequence between the SequenceControl block and the ImageProcessing block on system level. The SequenceControl block, here shown as external actor, sends a request to search the target signature of an incoming missile in an image. The ImageProcessing block reacts in that it performs the search function and then replies to the SequenceControl block with a return message.

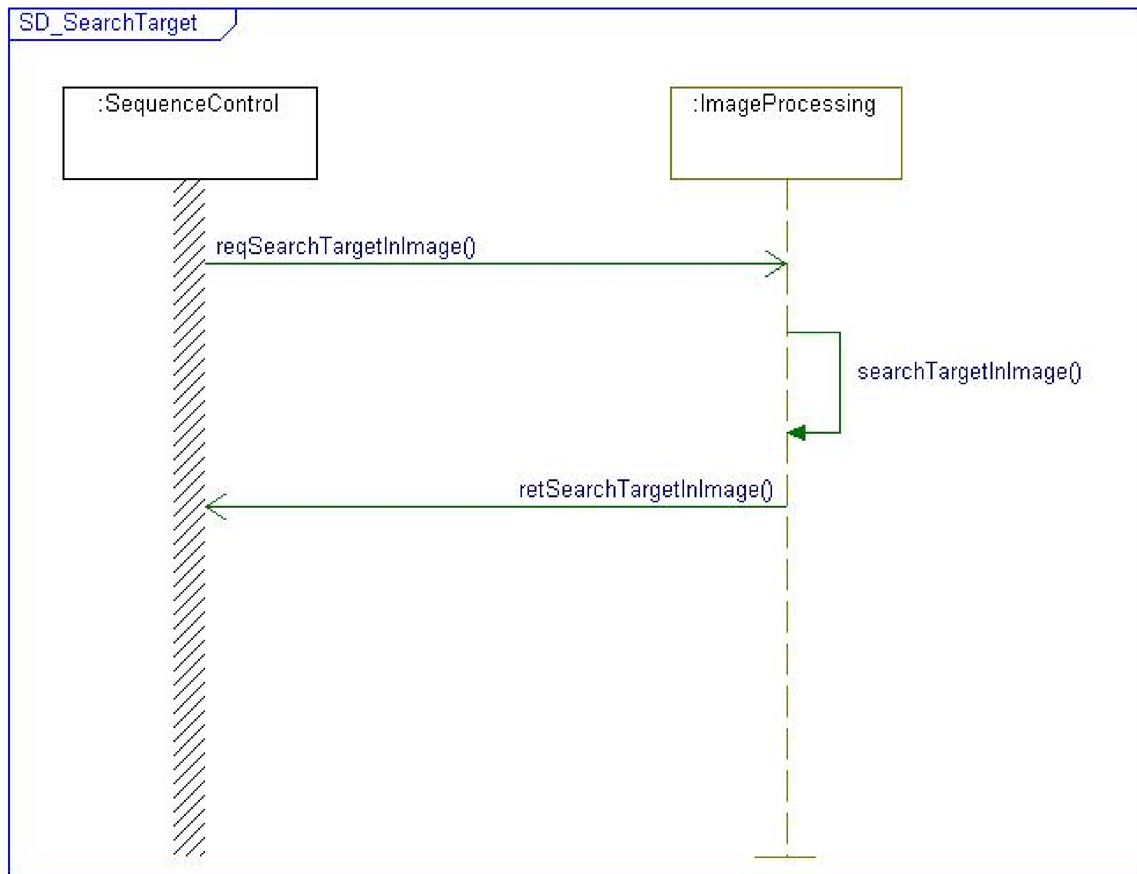

Figure 3-30 Functional Sequence on System Level

The system block can be then decomposed in the software model into a class architecture. The decomposition of the ImageProcessing block into classes is shown in a UML class diagram in Figure 3-31. The system block is decomposed into the classes *IOControl*, *ControlLogic*, *FilterAlgorithms* and *SearchAlgorithms*.

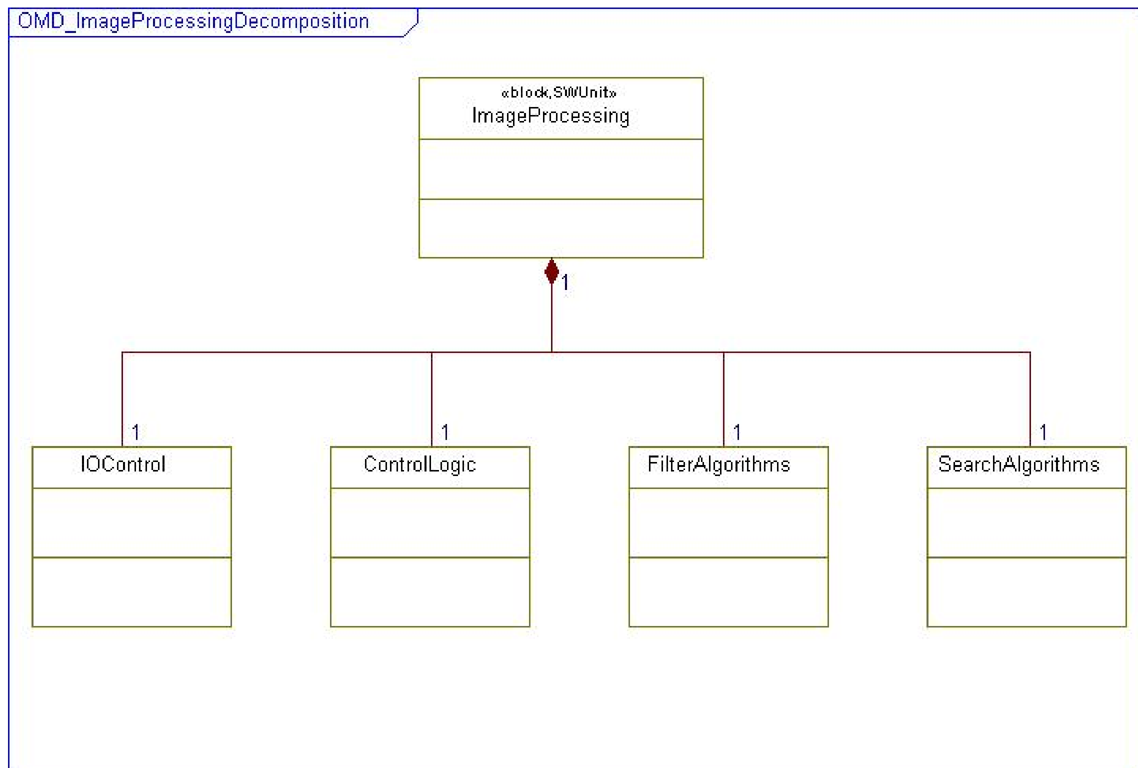

Figure 3-31 Decomposition of a System Block into Classes

The functional decomposition can then be elaborated on a sequence diagram that is based on the system level black box sequence specification. The decomposition of the system level sequence is depicted in Figure 3-32. The request from the SequenceControl block is now received by the IOControl class, which then informs the ControlLogic class about the reception by means of an asynchronous event. The ControlLogic class then fetches the image from the IOControl and sends it to the FilterAlgorithms. After the image has been filtered, it is sent to the SearchAlgorithms, which search the target position within the image and return the target position. The target position is then sent via the IOControl back to the SequenceControl block.

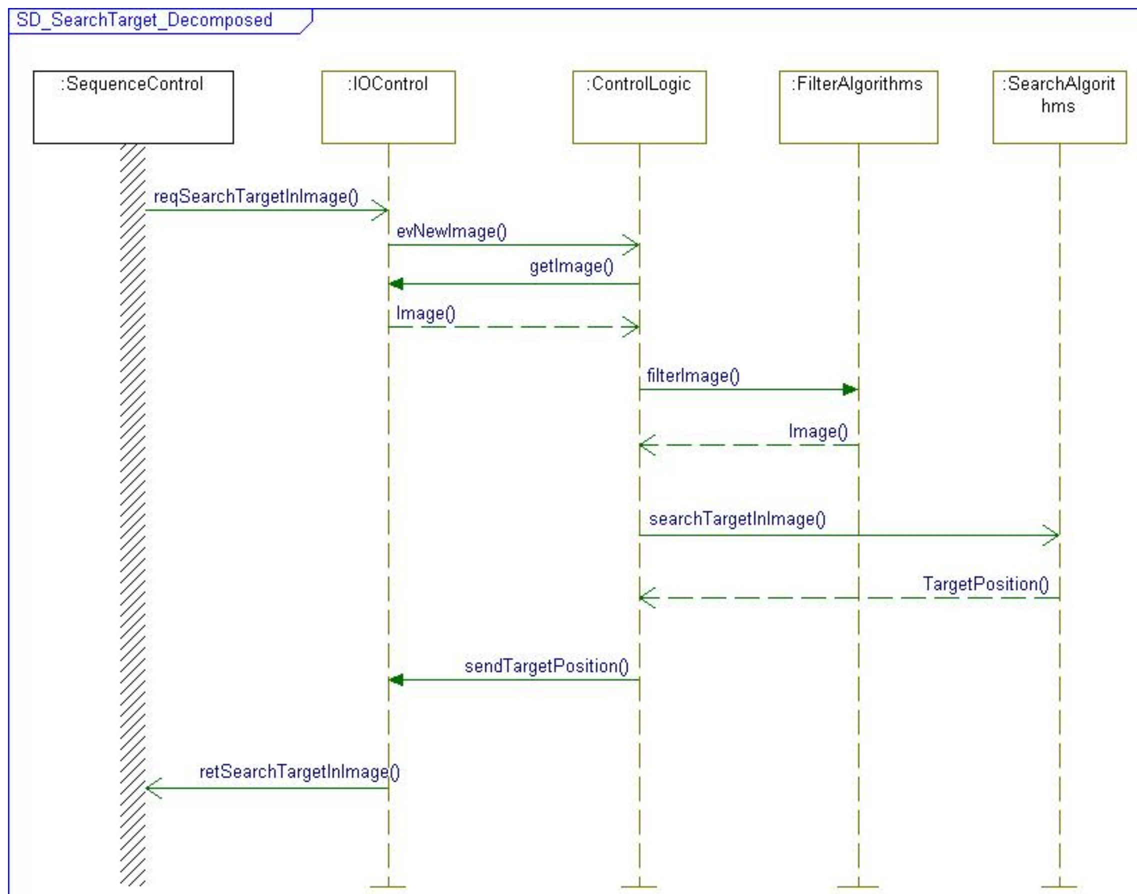

Figure 3-32 Decomposed Functional Sequence in the Software Model

The decomposition can then in addition also be continued by means of decomposing the system-level state chart and activity diagrams. The software models can be verified against the system specification at any time by means of model execution. The example shows that by reusing the system specification within the software analysis and design, a seamless transition from systems to software engineering can be reached.

4 Results

The concepts that were elaborated in the context of this thesis will on the one hand be reflected in the current modeling process document but will also result in a separate modeling handbook. The work on these documents has already begun during the thesis but, because of the limited time available, could not be finished. However, the concepts and foundation material for finishing the documents are for the most part set.

Modeling Process

First of all, the modeling process will be adapted to be compliant to and use the SysML language specification only. This is mainly done by using the provided Rhapsody SysML profile and adding an additional profile that specifies the missing stereotypes that are required to be SysML compliant. The structure modeling concept will be adapted to the one described in chapter 3.3. Also the concepts for modeling items and data (chapter 3.4) will be directly adopted. Because of the problems with the simulation engine when modeling flow ports – no support for events and broadcast communication - the flow port concept that was elaborated in chapter 3.5 will only be adopted partially. In the process, the use of flow ports will be only recommended to model continuous item or data flows. The modeling process will also be enhanced with the concepts for the segment handover model that were elaborated in chapter 3.6.The concepts for the modeling of physical architecture and basic software as well as the deployment (chapter 3.7) will be taken as basis for a process enhancement. In addition, the concepts for the seamless transition from systems engineering to hardware and especially software engineering will be reflected in the modeling process.

Modeling Handbook

Based on the DIRCM example model that was the basis for the concepts presented in this thesis, a modeling handbook will be created. The modeling handbook will guide the systems engineer through the modeling process and demonstrates the various process phases and the practical application of SysML on the DIRCM example model.

5 Summary and Conclusion

The concepts that were elaborated in the context of this thesis provide options to enhance the current process and solve its main problems and shortcomings.

One of the goals was to analyze the possible adaptation of the process to the SysML specification. The adaptation could be realized by means of implementation of the SysML profile and by adding the missing SysML stereotypes. By converting the current process to be fully compliant to the SysML specification, an improvement in terms of interchangeability of the designs is achieved. This is especially important when the designs are not exchanged by means of Rhapsody model repositories but instead in terms of documents, for example as part of an interface specification that is provided to other departments or even companies.

Another goal was to analyze the SysML enhancements to UML and the possible application of them to the current modeling approach. A severe shortcoming of the current process – the lack of capabilities to model items and data – could be addressed with the implementation of the SysML concept to model primitive and complex items and data as ValueTypes and blocks. By using this concept, items and data can be modeled in a concise manner. As a further enhancement is the introduction of SysML flow ports to the current process. With flow ports, the capability to model continuous item and data flows is added – something that is not possible with the current modeling process.

The last goal was to develop concepts to enable a seamless transition from the systems to the subsequent hardware and software engineering disciplines. This could be accomplished by describing an approach to generate model handoffs that provide a complete and self-contained description of segments, hardware and software units but in the same time also allow to trace back into the overall system context. In addition, the concepts to model physical architectures and basic software as well as the possibility to model different views of physical deployment improves the quality of system specifications and the communication and collaboration between the different engineering teams.

However, besides the work done in the context of this thesis, there still are more open topics that need to be addressed in future.

With the extensions of the process with the concepts described in this thesis, the systems engineering tool wizards that are provided together with Rhapsody cannot be used to its full extends anymore. Thus new customized tool wizards have to be implemented that support the process workflow and maintain consistency within the systems engineering models.

Another topic is the integration of Simulink models into a Rhapsody SysML model. According to Telelogic, a seamless integration of Simulink models is possible. However, this has to be evaluated and, if applicable, guidelines for the integration have to be created.

In the context of this thesis, only the SysML extensions to model data and items as well as flow ports are considered. The SysML however also provides capability to model parametric constraints and requirements. The applicability of these extensions to the current approach should be analyzed in future.

Last but not least, concepts to generate textual requirements from SysML models should be elaborated in future. Textual requirements are often required in development process models and thus it has to be analyzed how the requirements can be extracted from a SysML model in a concise and effective manner.

References

[Wei2006] Tim Weilkiens. *Systems Engineering mit SysML/UML*. dpunkt.verlag, 2006.

[VM2007] iABG. *Das V-Modell. http://www.v-modell.iabg.de, 2007*

[VMG2004] EADS. *VM-GBV*. EADS Intranet Geschäftssystem, 2004

[Gra1992] Robert Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice-Hall, 1992

[OP12007] EADS DE OPES4 Department, Internal Process Description Material, 2007

[OP22007] EADS DE OPES4, *Functional Systems Engineering Process*, 2007

[OMG2007] OMG. *Systems Modeling Language. http://www.omgsysml.org, 2007*

[Sys2006] OMG SysML Specification, Final Adopted Specification, ptc/06-05-04. *http://www.omgsysml.org, 2006*

[Mat2007] Mathworks website, *http://www.mathworks.com, 2007*

[Tel2007] Telelogic Website, *http://www.telelogic.com, 2007*

Versicherung über die Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, ____________________________

(Dennis Fadljevic)