PATTERN-BASED REQUIREMENTS MODEL USING SYSML FOR A HELICOPTER'S PILOT ASSISTANCE SYSTEM

M. Deshmukh, German Aerospace Center (DLR), Braunschweig, Germany
F-M. Adolf, German Aerospace Center (DLR), Braunschweig, Germany
M. Heisel, University of Duisburg-Essen, Duisburg, Germany

Abstract

Development of a pilot assistance system involves many disciplines. Requirements analysis and traceability becomes difficult while developing software for such multidisciplinary system. The modeling with SysML facilitates better design of software along with non-software components. The pattern based requirement analysis using Problem Frames (PFs) encourages knowledge reuse and provides better understanding of domain and requirements. Since PFs do not have standard notations and tool support, their use in industry is limited. In this paper, we introduce a systematic model and pattern based software development method which combines SysML and PFs in order to cope with the development complexity of a helicopter pilot assistance system. It also provides SysML based standard notations for PFs. The advantage of the proposed method is that all the artifacts modeled during the software development are linked together. A smooth and synchronized transition from requirements elicitation to software design, implementation, testing and maintenance is achieved.

1. INTRODUCTION

At the German Aerospace Center (DLR), the Institute of Flight Systems is currently working on the Assisted Low Level Flight and Landing on Unprepared Landing Sites (ALLFlight) project. In the ALLFlight project, a helicopter's pilot assistance system is developed, which allows the intuitive operation of a manned helicopter from start to landing on unprepared landing sites and an intermediate low level flight in the presence of obstacles in a degraded visual environment. Such a pilot assistance system provides advanced visual and tactile cues, intelligent control augmentation, reduces the pilot workload and increases the situational and mission awareness. Control software named Online ReConfiguration and Supervision (ORCS) is developed as a part of the ALLFlight project [8]. ORCS supervises the current environment situation by building its model based on the inputs from the different components of a helicopter. The input data includes current velocity, stick positions, way-points, sensor data etc. ORCS processes the input data and sends back the required reconfiguration parameters to the respective components. Figure 1 shows the role of ORCS (supervision and reconfiguration) with respect to other components of the helicopter using bidirectional arrows. All components of ALLFlight project are developed, implemented and tested in the Flying Helicopter Simulator (FHS) at DLR.

While developing software for a multidisciplinary project such as ORCS, gathering the requirements and establishing the relationships between them is a difficult task. Also there is no standard way for gathering the domain knowledge (e.g., external environment parameters, trajectories, waypoints etc.) and representing it in terms of software engineering. It is difficult to find the feasibility of requirements against the facts and assumptions of the participating domains. Maintaining the changing requirements and requirement traceability is challenging. There is a need to model the requirements and establish their links to further design components, implementation and testing.

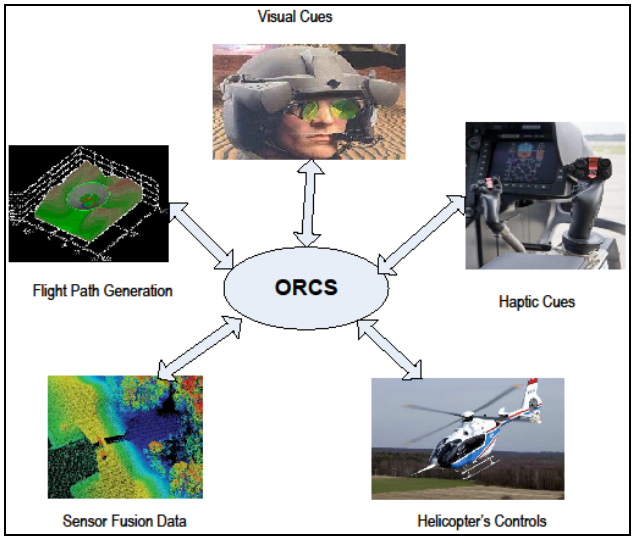

Figure 1 Role of ORCS in helicopter operations

Model-based software development is one of the approaches used for the development of defect-free, robust and reliable software. It improves software development processes and the quality of a developed product (in terms of defect count) [14]. So in order to overcome the previously stated problems, we describe a model-based software development approach using SysML [12] and Problem Frames [10] in this paper. The proposed approach follows the Development Process for Embedded System (DePES) defined by Heisel et al [9].

This approach is successfully applied to the development of ORCS.

In the remaining paper, first we describe the available software modeling techniques, their shortcomings and need of our proposed approach. Then as a first step of our approach, we provide standard notations to Problem Frames using SysML nomenclature and diagrams. Then we explain how the requirements analysis can be carried out by combining Problem Frames and SysML. Later we describe modeling of software architecture, interactions between software components using SysML diagrams and their linking to requirements analysis, implementation, and testing of software. The paper ends with the conclusion and future work. As a proof of concept, the proposed approach is applied to the development of ORCS.

2. MODELING APPROACHES

Unified Modeling Language (UML) [13] is widely used in industry for software modeling. However, UML is software design oriented and lacks in performing system engineering. Instead of UML, System Modeling Language (SysML) is more effective for system engineers to model multidisciplinary system like ORCS, which includes software and non-software components like hardware, processes, information, personal, and facilities etc. For the structural modeling of a system, SysML provides *package*, *internal block*, and *block definition* diagrams. For the behavioral modeling of a system, it provides *sequence*, *state chart*, *use case*, and *activity diagrams*. Newly added *requirements* diagram in SysML provides a way to specify requirements and relations between them and a *parametric* diagram shows parametric constraints between structural elements. However, SysML lacks in methodological support. Both UML and SysML do not cover capturing domain knowledge, problem descriptions, sub-problems decomposition and domain relationships. The requirements analysis phase is not modeled.

Problem Frames (PF) are the patterns invented by M. Jackson [10] that capture and define a commonly found class of simple sub-problems. They help developers in understanding the problems. M. Jackson [10, ch. 3] states that the high level problem description of two problems could be completely different. However, the sub-problems could be of the same type. For example, consider a library management system and ORCS. If we look at a very high level, given problems fall into two different categories. However, when we start decomposing these problems, we realize that they consist of sub-problems of similar type such as accepting input from a librarian is similar to accepting operational commands from a pilot or displaying book query result on a display is similar to displaying situation information on a display. Thus using PFs, we can effectively understand and analyze the problem and then re-use our knowledge in solving them. However, problem frames are less adapted in the industry because of the lack of standard notations and tools.

For capturing domain knowledge and relations, a context diagram could be used [10, p. 20]. It explores the relation between software and the real world in which it would be integrated. This gives a clear distinction and understanding about the problem to be solved and the participating domains. The context diagram includes connections (interfaces) and phenomena that are shared between machine and problem domains. The phenomena could be events, actions or operations that occur between the domains. As of now there is no standard template available for the context diagram. Therefore, context diagram need to be adapted with modeling language notations (UML or SysML) for its convenient use.

3. RELATED WORK

In order to overcome the above stated problems and to make use of good features of SysML and Problem Frames, Colombo et al. [4, 5, and 6] proposed to combine SysML and PFs. However, the suggested approach does not completely comply with Jackson's problem frames notations. The suggested notation does not take advantage of SysML diagrams such as a requirements diagram. The approach does not cover the complete software development process and it is still design oriented.

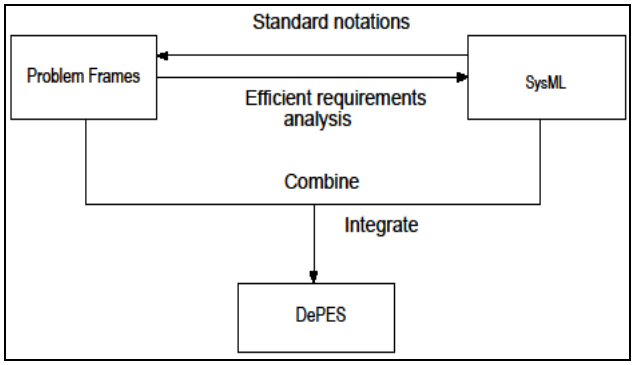

Figure 2 Overview of a proposed approach

In order to cover software development phases (requirement analysis, design, implementation and testing) along with the integration of SysML and PFs, DePES is used. DePES provides a systematic software development method based on the international software development standards. DePES consists of twelve steps which covers complete software development process. Problem frames and UML are already integrated in DePES methodology. In this paper, combination of SysML and PF is integrated with DePES (see Figure 2) which results into a synchronized model of software consisting of all the artifacts during the development of software. DePES covers the development of both hardware and software components. However in this paper, the scope is restricted to software development only.

4. STANDARD NOTATIONS TO PROBLEM FRAMES USING SYSML

Consider a transformation problem frame which deals with data manipulation sub-problems [10, p.99]. In general, the task of data transformation based on predefined processing rules is represented using a transformation frame shown in Figure 3.

In order to take advantage of PFs for efficient requirements analysis, we propose to use the Internal Block diagram and stereotypes of SysML to represent

frame diagrams. Figure 4 shows the transformation problem frame diagram using SysML.

The participating domains (Transformation Machine, Input and Output) shown in the original frame diagram (see Figure 3) are shown with the help of blocks in Figure 4.

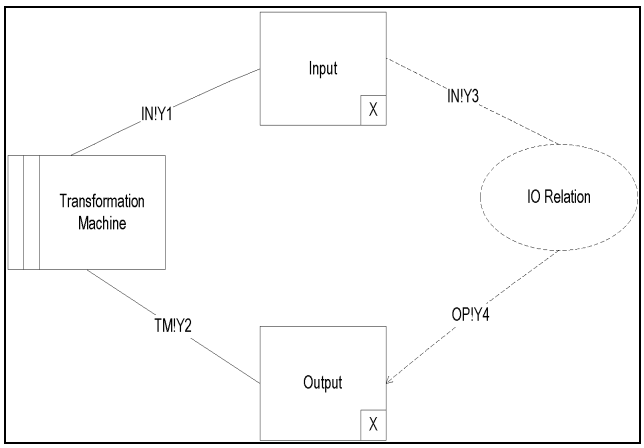

Figure 3 Transformation Problem Frame Using Jackson's notations

The type of each domain such as causal, lexical, and machine is modeled using stereotypes. The interfaces between the participating domains are modeled using binding connectors. The flow items of the binding connectors are used to represent the phenomena (IN!Y1 and TM!Y2). The requirement oval (IO Relation) is represented by the requirement element of the requirements diagram. The textual description of the requirement can be added into the description tab of the requirement. A unique identifier can also be assigned to the requirement for future reference.

Similarly, the generic templates of all problem frame diagrams could be represented with the help of internal block diagram of SysML and then could be instantiated as needed.

5. INTEGRATION OF PROBLEM FRAMES AND SYSML WITH DEPES

In this section, we apply all 12 steps of DePES to the development of ORCS based on the proposed approach. Since the development of hardware components is not considered in this paper, step 5 and 6 of DePES are skipped. In this paper, we demonstrate the approach with only one requirement of ORCS. However, the approach has been successfully applied to the whole set of requirements of ORCS [7].

5.1. Requirement Analysis

A clear understanding between the customer and the developer during the requirements analysis phase is a key to the successful implementation of the intended software. In DePES, requirements and domain modeling is carried out with the help of context diagram, problem frames and sequence diagrams as described in the following steps.

5.1.1. Step 1: Describe system in use

Generally, the information about the existing system, the participating domains and the interrelationships between

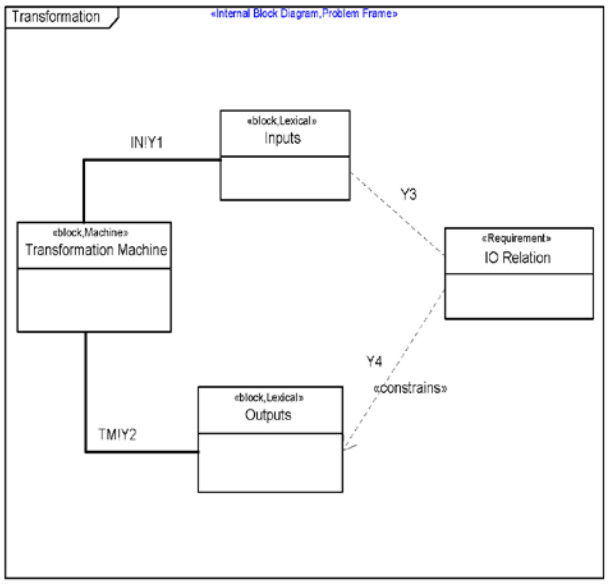

Figure 4 Transformation Problem Frame using SysML Notations

them are given by the customer. The customer could provide it in natural language description or using some diagrams. Since such information is provided by the customer, we cannot define a specific type of diagram in this step. However, for a synchronized software modeling we recommend user to use SysML block definition diagram for defining the context of a system.

5.1.2. Step 2: Describe system to be built

In this step, the context diagram provided in step 1 is refined with the addition of the solution domain (software to be built) and its relations with the problem domains. As a first step in the creation of a synchronized model, we draw a context diagram using a block definition diagram of SysML. The participating machine and other problem domains are drawn using a block. The type of each domain such as lexical or causal or biddable could be represented by predefined stereotypes. Stereotypes provide great flexibility while designing a model. The interfaces between these domains could be shown by associations. With the help of stereotypes, we can specify the type of associations. The interface name abstracts the phenomena shared between the domains. The domain knowledge provided by the domain expert can be added into the description field of a block. A constraint can be used to model conditions or limitations. If required, the decomposition of domain can be represented by adding internal block diagrams to the respected domain block.

In Figure 5, a high-level abstraction of ORCS is shown along with its connections to other domains. All the components of ALLFlight which are controlled by ORCS are implemented either on the Experimental Computer (EC) or Experimental Co-Computer or Sensor Co-Computer (SCC) of the FHS at DLR. The components (Domains) include trajectory planning, trajectory

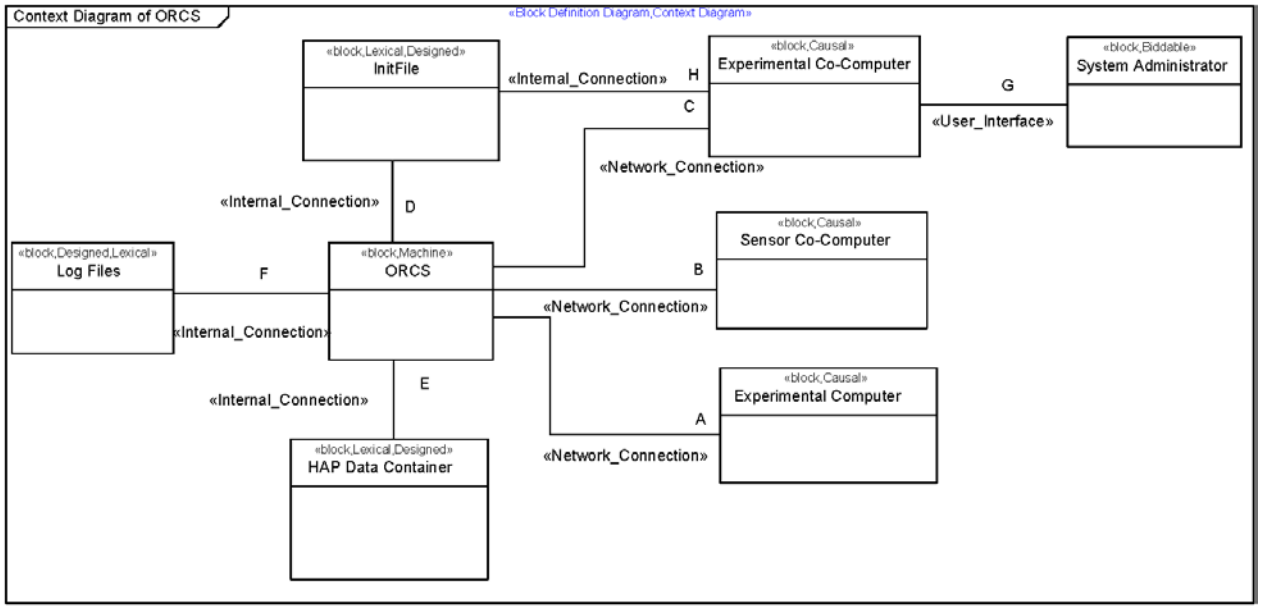

Figure 5 Context Diagram of ORCS using SysML Block Definition diagram

generation, helicopter control, haptic and visual cues and external environment.

Some of the interfaces between ORCS and participating domains are defined as follows which define exchange of actions between them. Similarly other interfaces are defined.

A = {sendOrcsEcUdpOutput, sendEcOrcsUdpInput}

E = {receiveEcOrcsUdpInput, receiveEccOrcsUdpInput, receiveSccOrcsUdpInput, parseInput, storeInput, filterInput, generateOutput, parseOutput, requestRREvent, checkInputData}

F = {writeLogFile}

After defining the problem domains and the interfaces between them, the facts and the assumptions about each problem domain are collected. Then the requirements of each problem domain are analyzed. Domain experts describe the desired characteristics in terms of input, data processing rules, and output requirements. For example, a trajectory planner provides waypoint calculation and navigation related requirements, a sensor expert provides requirements of processing of data captured by the different sensors from the external environment and a helicopter controller expert provides requirements related to handling of side stick and center stick position inputs etc. The requirements are consolidated from the domain knowledge gathered in Step 1 and the meetings with the corresponding experts. As an example, the requirements of ORCS with respect to problem domain *HubschrauberRegelung (*in English: Helicopter Control) (HR) are shown in Table 1.

Similarly all requirements are gathered. In addition to the textual description of requirements, we visually model the ORCS requirements within IBM Rational Rhapsody® using requirements diagrams. A developer can add relationships between the requirements as per his/her understanding which could be then verified by the customer. Requirements diagrams provide different types of relations e.g., trace, derived, satisfy etc. in order to model relation between the requirements and other model elements.

Name	Requirements
Req.3.1_2.1	In order to create a helicopter's current situational model, helicopter's current control data (e.g., velocity, stick position etc.) need to be gathered.
Req.3.1_2.2	Based on the helicopter's current data model, a possible switching from current to next regime needs to be calculated.
Req.3.1_2.3	The calculated regime switching must be conveyed back to the helicopter's pilot.

Table 1 ORC S - HR Requirements

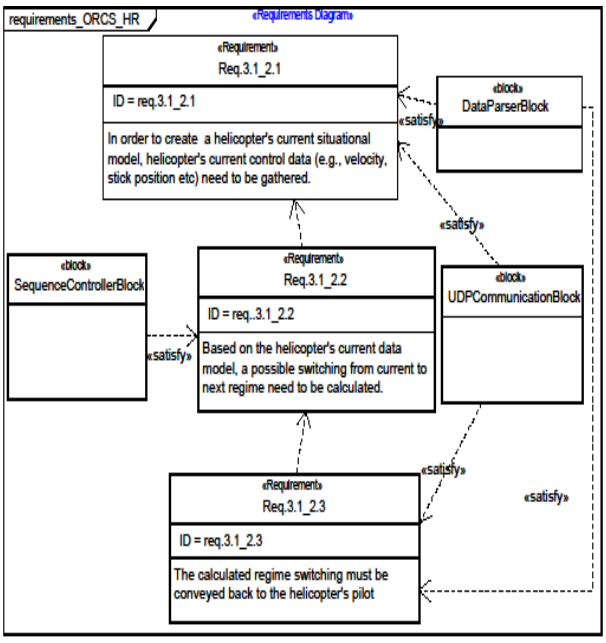

Figure 6 SysML Requirements Diagram of ORCS with respect to HR

Figure 6 represents the requirements diagram showing the requirements of ORCS with respect to HR. It visually represents the requirements stated in Table 1. It shows how the given 3 requirements are sequentially dependent on each other. Also it models their relationship with the elements which implement them. For example, *UDPCommunicationBlock* satisfies the requirements Req.3.1_2.1 and Req.3.1_2.3. This helps in tracing the requirements. Furthermore test cases could also be added to the diagram which would verify the requirements.

5.1.3. Step 3: Decompose problem

Once we know the requirements (problems) to be satisfied, next step is to perform proper structuring and decomposition of the given problem. For this, as suggested in DePES we use Problem Frames. PF models the relations between the software to be developed, external users (in case of ORCS, flight test engineer, evaluation pilot and safety pilot) and the real world. Mapping of our requirements to the problem frames provides us with common problem patterns and thus helps in utilizing previously tested knowledge. While analyzing ORCS requirements using PFs, we find mainly three common problem patterns namely *model building*, *transformation*, and *data-based control*. So instead of implementing each requirement separately, implementation based on the PF patterns takes the advantage of software reuse and in turn saves time and improves quality.

The second requirement (Req.3.1_2.2) states that the current and next regimes are generated based on the incoming data. The incoming data includes velocity and signals from the side stick and center stick. The regime recognition is the process of analyzing this incoming data and mapping them to a defined flight profile. The current and the next regimes are generated using the Regime Recognition state chart [1] based on the input data received from the HR and stored as output data. This is clearly a case of data transformation from one lexical domain to the other. Therefore requirement Req.3.1_2.2 can be modeled using the transformation problem frame. The instantiated transformation problem frame is depicted in Figure 7.

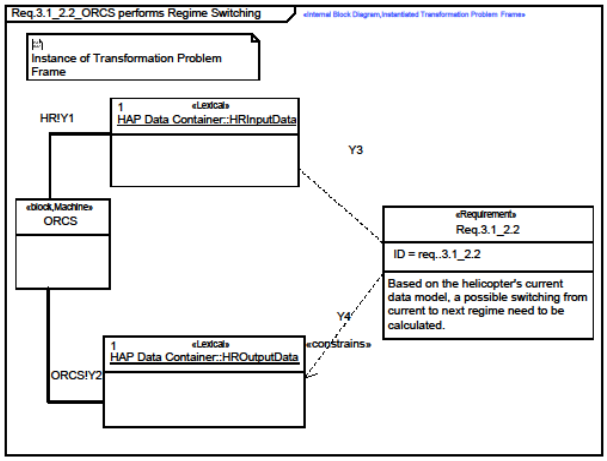

Figure 7 Instantiated Transformation Problem Frame for the requirement Req.3.1_2.2

We model requirements based on gathering of data from real world (e.g., Req.3.1_2.1) using model building problem frame and requirements based on controlling real world from generated output (e.g., Req.3.1_2.3) using data-based control problem frame. Thus with the help of problem frames, we classify the requirements, identify the pattern, and identify the role of each domain in each requirement. Such detailed problem analysis increases developer understanding. We use SysML based problem frame notation described in Section 4 which helps to use problem frames as a standard SysML modeling element and to further reference it with other modeling elements.

5.1.4. Step 4: Derive a machine specification for each sub-problem

For checking the implementation feasibility of the requirements stated in Step 2, we derive the machine specifications from them. The specifications state the responsibility of the machine in order to satisfy the corresponding requirements. This gives us idea whether the stated requirements are implementable or not. Similar to the modeling of requirements using a requirements diagram, we model the specifications using a requirements diagram. The specification diagram describing the specifications of ORCS is shown Figure 8.

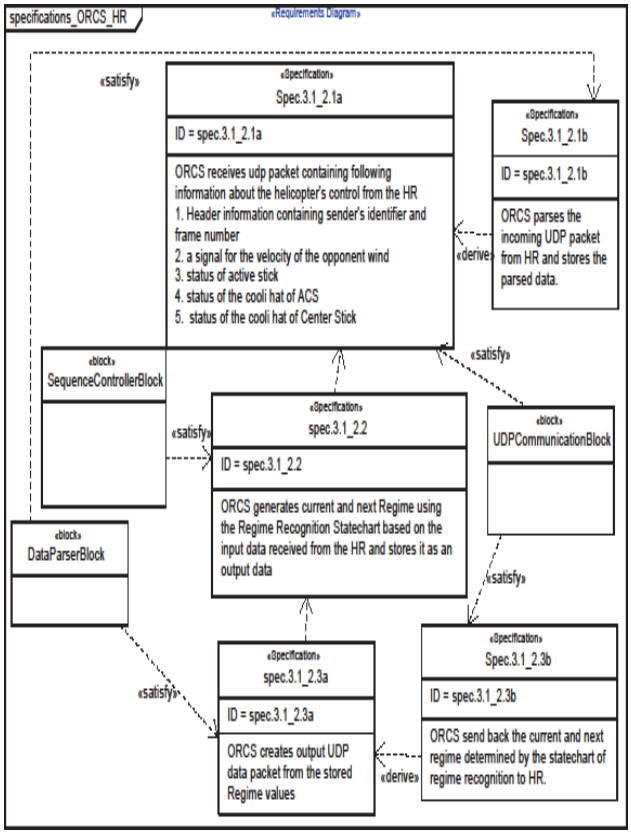

Figure 8 Specifications Diagram of ORCS with respect to HR using SysML Requirements diagram

Instead of describing the specifications in natural language, graphical visualization increases understandability. Specifications are represented using a Requirement with a *stereotype* ``Specification''.

Specifications have the same *ID* as that of the corresponding requirement (see Figure 6 and Figure 8). Only the prefix req is replaced with *spec*. The suffixes *a*, *b* etc. are appended in case multiple specifications are derived from one requirement. The use of same IDs increases traceability. Like a requirements diagram, software components satisfying the specifications could be added later. The relationships between the specifications are same as relationships between the requirements.

After stating the specifications, a sequence diagram is drawn at least one per requirement in order to represent the flow of actions and control between the domains. The correspondence between context diagram phenomena, problem diagram phenomena and messages in sequence diagram verifies whether the given requirement is implantable or not. The sequence diagram provides an outline for the testing in Step 12.

The sequence diagram for the requirement Req.3.1_2.2 is shown in Figure 9. It shows the sequence and direction of interactions taking place between the domains participating in requirement Req.3.1_2.2. The mapping between the messages in the sequence diagram and the phenomena defined in the previous section helps in maintaining the requirement traceability. This mapping is shown in Table 2 in Section 6. These messages give an insight into implementation details. Likewise sequence diagrams for other requirements are drawn and validated.

5.2. Design

In order to make software comprehensible, DePES propose to draw the explicit design of software architecture. Well-designed software can be implemented systematically and better maintained.

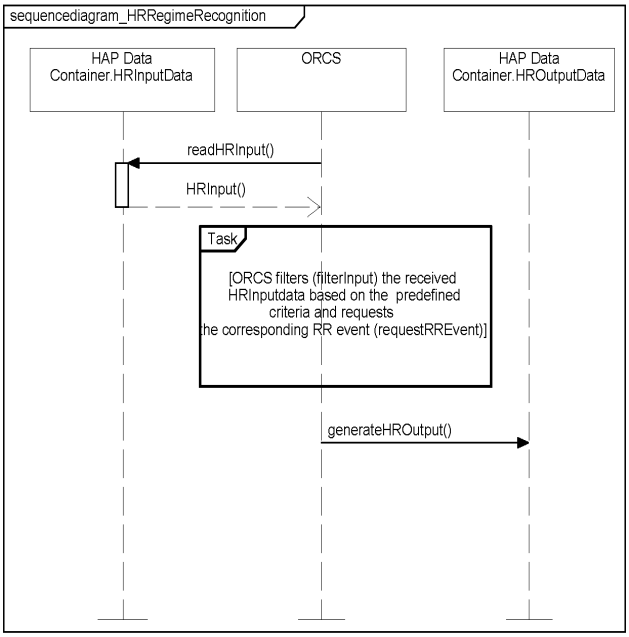

Figure 9 SysML Sequence diagram for the requirement Req.3.1_2.2

In this approach we use an internal block diagram of SysML for representing the software component architecture. The purpose and responsibilities of the architectural components along with the specification of interfaces are described in detail. The functional responsibilities of each component are based on the underlying common sub-problems. Finally the architectures of each sub-problem are merged into one global software architecture which provides a solution to the main problem. The merging of components is based on the principles stated by Choppy et al. [3]. As stated before, in this paper only development of software components is handled. Therefore the steps 5 and 6 of DePES are skipped as they describe the development of hardware components.

5.2.1. Step 7: Design an architecture for all programmable components of the global system architecture that will be implemented in software

ORCS contains only one programmable component. Now as proposed in DePES, first we decompose ORCS into sub-components for each sub-problem identified in Step 3. The decomposition is based on the object-oriented software architecture. Figure 10 shows the architecture of components that satisfy the requirement Req.3.1_2.2. The requirement Req.3.1_2.2 deals with the calculation of regime switching.

Similarly the architectures for all requirements of ORCS are drawn. All architecture diagrams of the requirements of ORCS are not included in this paper. Because of the common patterns in the sub-problems, the same components are re-used. Finally, we merge the sub-problem architectures into a global architecture as shown in Figure 11.

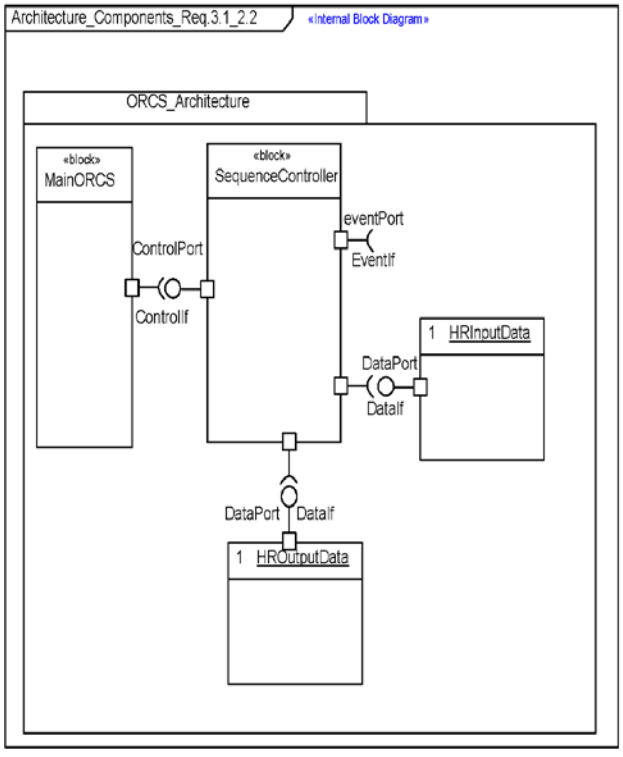

Figure 10 Components Architecture for the requirement Req.3.1_2.2 using SysML Internal Block diagram

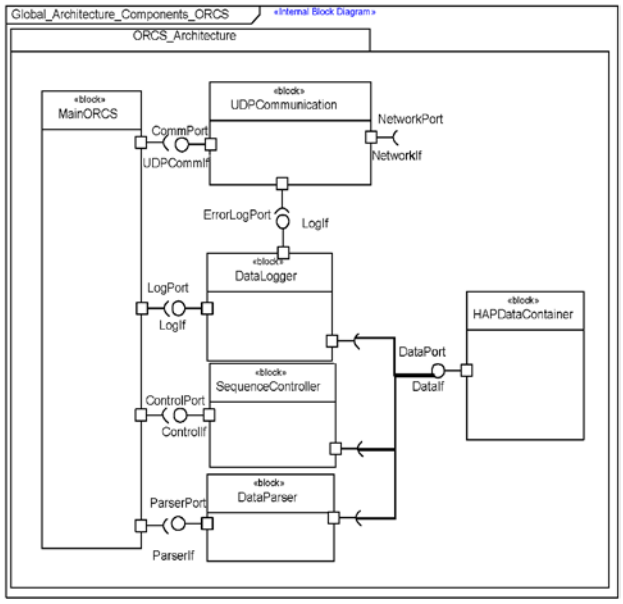

Figure 11 Global Architecture of ORCS after the merging of its all sub-components using SysML Internal Block diagram

5.2.2. Step 8: Specify the behavior of all components of all software architectures using sequence diagrams

In this step, the behavior of each component from the global software architecture is modeled using a sequence diagram. A separate sequence diagram is drawn for each sub-problem. The signals from the global software architecture and the specifications from Step 4 are re-used in this step. The sequence diagrams model the flow of data, actions and control between the sub-components of the global software architecture which is required in order to implement the corresponding sub-problem.

Figure 12 shows the interactions between the components with respect to the requirement Req.3.1_2.2.

The interface behavior of the components obtained from the sequence diagram forms the basis for the test specifications. Messages used in the sequence diagrams must be consistent with the interface signals defined in the previous steps.

5.2.3. Step 9: Specify the software components of all software architectures as state machines

In this step, we describe the behavior of each component in terms of internal transitions using state machine diagrams. We define pre-conditions which are needed in order to execute the transitions and post-conditions that are the output of the transition.

From the architecture diagram in Step 7, we know that the component *Sequence Controller* is the decision making component of ORCS (for Req.3.1_2.2). The regime calculation which is the core logic of ORCS is based on the helicopter's control data (velocity, stick positions etc.) received from the component HR. The behavior of the component *Sequence* Controller with respect to regime calculation is shown in Figure 13 with the help of a state machine diagram.

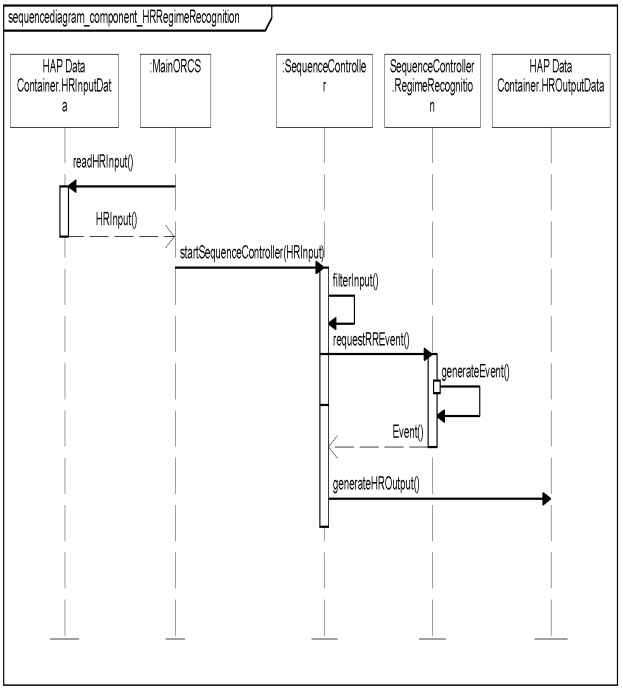

Figure 12 SysML Sequence diagram showing behavior of the components of the requirement Req.3.1_2.2

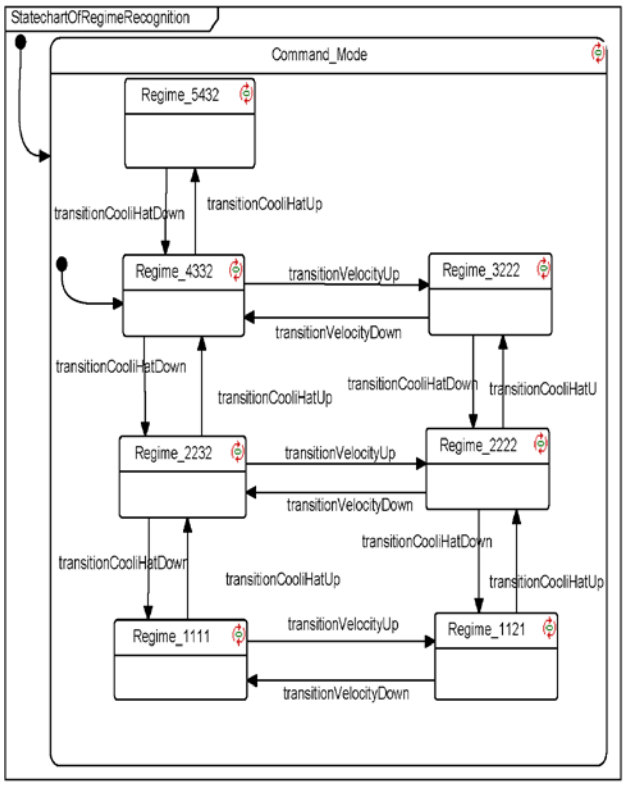

Figure 13 SysML State machine diagram describing the behavior of the component Sequence Controller of ORCS

5.3. Implementation

5.3.1. Step 10: Implement software components and test environment

Classes and objects are the basic building blocks of object-oriented programming. In order to synchronize software implementation (coding) with its design, we draw class diagrams based the component, sequence and state machine diagrams of design phase and then derive the actual code (automatically or manually) using the classes and their relations shown in the class diagrams.

Figure 14 shows the class diagram of ORCS. All the classes can be easily mapped to corresponding components shown in Figure 14.

The methods correspond to the messages in sequence diagrams and the processing logic described by the state machine diagrams. As already mentioned, the design phase diagrams are linked with the requirements. Therefore this linking between design phase diagrams and class diagram facilitates the linking between the requirements and the implemented code. Such linking helps in achieving requirements traceability and verification.

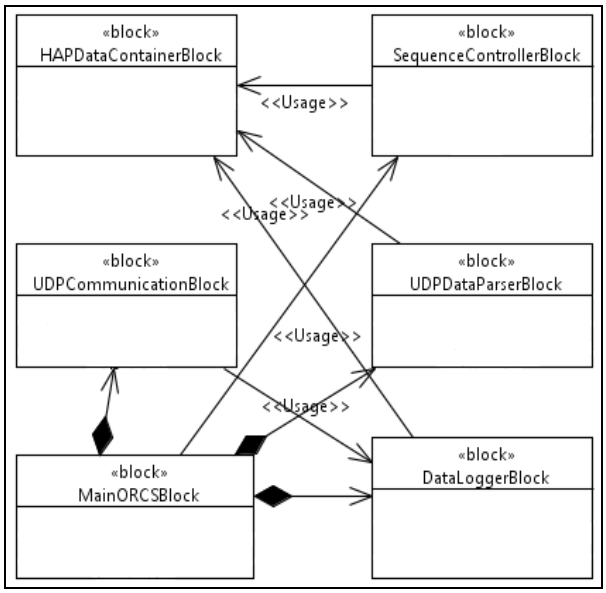

Figure 14 Class Structure of ORCS using SysML Block definition diagram

5.4. Testing

In general a systematic testing reveals the defects created during the development and ensures the quality of a product. The last 2 steps of DePES, Steps 11 and 12 describe the details of the testing phase. The focus of this paper is not on testing techniques. Therefore we are not going into the details of ORCS testing. However, we suggest some testing techniques which could be adopted in future for improving the model-based software development process.

5.4.1. Step 11: Integrate and test software components

The unit testing of ORCS is carried out using manual test cases based on the sequence diagrams created in previous steps. For testing, DePES does not suggest any model-based process. So in order to model the complete software development process, we suggest some improvements to testing. Model-based testing [2] could be adopted within DePES in future. The addition of test cases to requirements diagram would increase traceability. In DePES, sequence and state machine diagrams are already used for the representation of system behavior. We state that these diagrams would form the basis for the test cases. Therefore the test plan proposed by Schwarzer [14] based on state charts could also be integrated within DePES for model-based testing.

5.4.2. Step 12: Integrate and test hardware and software

In this final step, both the software and hardware components are integrated. The machine is actually integrated in the context along with the other existing components. The actual results of the system after integrating the developed machine into it are tested against the expected change in the existing system described by the requirements. This *acceptance testing* is performed by the customer/ end user. The test cases are based on the sequence diagrams from Steps 4 and 8. In case of ORCS, the software is tested in FHS at DLR along with the other components of the ALLFlight project.

6. REQUIREMENTS TRACEABILITY

Requirement traceability helps in improving the software development process by managing evolving requirements, in testing for tracing back from failed tests, and for system documentation and maintenance. It links each requirement to artifacts which are involved in its representation and implementation. DePES does not explicitly specify how to maintain the requirements traceability during software development. As an improvement to DePES, we propose to maintain the requirements traceability matrix throughout the system development. After each phase, the relationships between the different artifacts and the requirements is added or refined in the matrix as shown in Table 2

Also as stated earlier, the SysML *requirements diagram* can also be used for requirement traceability. The developed artifacts can be added in the requirements diagram and can be linked to the requirements using the dependency *satisfy* (see Figure 6).

7. CONCLUSION AND FUTURE WORK

This paper has proposed an effective domain as well as application engineering process using Problem Frames, SysML and DePES. It has provided SysML based standard notations to Problem Frames and enriched requirements modeling in SysML using Problem Frames. The classification of requirements based on problem frames helped in finding patterns in the requirements. The handling of requirements based on such patterns facilitates knowledge reuse. The knowledge reuse reduced the efforts and increases the quality because of already verified implementations. Because of the standard notations using SysML diagrams, problem frames can be easily understood in terms of a modeling language and thus helped in adapting them for the modeling of requirements. The modeling of participating domains, interrelations between them and their interfaces with ORCS provided better understanding for the software developer about the environment in which software is going to work. The proposed approach satisfied the standard *"V & V"* model of software development. As a result of this approach, a single synchronized model containing all artifacts from requirement analysis to implementation is obtained. The overall gap between requirements, design, implementation and testing is reduced. The proposed approach is successfully applied to the development of ORCS and validated in ground simulator testing in FHS at DLR.

In order to improve the process further, we suggest integrating model-based testing within the proposed approach. The use of formal specification languages such as OCL and Z will also improve the requirements verification. Synchronization between the models of different phases would be improved by using automated requirement traceability mechanism.

PHENOMENA IN THE CONTEXT DIAGRAM	PHENOMENA IN THE FRAME DIAGRAM	IMPLEMENTED OPERATIONS
SENDECORC SUDPINPUT	SENDHRINPUT	
RECEIVE ECORCSUDP INPUT	RECEIVEHR INPUT	RECEIVER()
PARSEINPUT	PARSEFLIGHT VELOCITY	PARSEANDSTORE CHARTOFLOAT()
	PARSEACTIVE STICKSTATUS	PARSEANDSTORE CHARTOCHAR()
	PARSESIDE STICKPOSITION	PARSEANDSTORE CHARTOCHAR()
	PARSECENTER STICKPOSITION	PARSEANDSTORE CHARTOCHAR()
	PARSECOS MODE	PARSEANDSTORE CHARTOCHAR()
STOREINPUT	STOREFLIGHT VELOCITY	PARSEANDSTORE CHARTOCHAR()
	STOREACTIVE STICKSTATUS	PARSEANDSTORE CHARTOCHAR()
	STORESIDE STICKPOSITION	PARSEANDSTORE CHARTOCHAR()
	STORECENTER STICKPOSITION	PARSEANDSTORE CHARTOCHAR()
	STORECOS MODE	PARSEANDSTORE CHARTOCHAR()

Table 2 Requirements Traceability Table for the requirement Req.3.1_2.1

References

1. M. Abildgaard, Entwurf einer Regime Recognition im Projekt Allflight, Technical Report (111-2009/13), Institute of Flight Systems, German Aerospace Center (DLR), 2009.
2. P. Baker, Z. Dai, J. Grabowski, O. Haugen, I. Schieferdecker, and C. Williams, Model-Based Testing, Springer Berlin Heidelberg pp. 7-13, 2005.
3. C. Choppy, D. Hatebur, and M. Heisel, Architectural Patterns for Problem Frames, IEEE Proceedings – Software, Special Issue on Relating Software Requirements and Architectures volume 152 pp. 198-298, 2005.
4. P. Colombo, V. Bianco, L. Lavazza, and A. Coen-Porisini, A Methodological Framework for SysML: a Problem Frames based Approach, 14th Asia-Pacific Software Engineering Conference, 2007.
5. P. Colombo, V. Bianco, and L. Lavazza, the Integration of SysML and Problem Frames, International Workshop on Advances and. Applications of Problem Frames 08, 2008.
6. P. Colombo, F. Khendek, and L. Lavazza, Requirements Analysis and Modeling with Problem Frames and SysML: A Case Study, European Conference on Modeling Foundations and Applications, LNCS 6138 pp. 74-89, 2010.
7. M. Deshmukh, Model-based Software Development for the Helicopter's Pilot Assistance System using Problem Frames and SysML, Master Thesis, University of Duisburg-Essen, March 2011.
8. M. Hamers, and R. Lantzsch, Assisted Low Level Flight and Landing on Unprepared Landing Sites, Technical Report 111-2008/10, Institute of Flight Systems, German Aerospace Center (DLR), 2008.
9. M. Heisel, and D. Hatebur, Development Process for Embedded System, Proc. Workshop on Model-Based Development of Embedded Systems, 2005.
10. IBM, System Engineering Tutorial for IBM Rational Rhapsody®, 2009.
11. M. Jackson, Problem Frames: Analyzing and Structuring Software Development Problems, Addison-Wesley, 2001.
12. Object Management Group, SysML: OMG System Modeling Language, http://www.omg.org/spec/SysML/1.2/, June 2010.
13. OMG Unified Modeling Language: Superstructure, UML Revision Task Force, http://www.omg.org/spec/UML/2.1.2/, November 2007.
14. B. Schaetz, M. Broy, F. Huber, J. Philipps, W. Prenninger, A. Pretschner, and B. Rumpe, Model-Based Software and Systems Development - A White Paper., http://www4.in.tum.de/ schaetz/papers/ModelBased.pdf, 2004.
15. R. Schwarzer, Testen modellgetrieben entwickelter Software auf Basis des Rhapsody-OX-Frameworks am Beispiel der ARTIS-Plattform, Special Interest Group - Model Driven Software Engineering, June 2007.