Deutscher Luft- und Raumfahrtkongress 2010
DocumentID: 161170

BRIDGING THE GAP BETWEEN USERS AND DEVELOPERS WITH MODEL-BASED USAGE ANALYSIS

B. Langer,
Diehl Aerospace GmbH, An der Sandelmühle 13 – 60439 Frankfurt, Germany

Abstract
Increasing complexity is one of the major challenges for modern avionic systems. The number of subsystems is growing and at the same time, more stakeholders are involved in the development process. To enable efficient co-operation in this changing environment, experts with very different domain knowledge have to jointly create a common and unambiguous specification of a large and distributed system. Model-based Usage Analysis takes a step in this direction, allowing users and developers to record and synchronize their requirements and constraints in form of an abstract model of the system. In this paper we present the key features of Model-based Usage Analysis with examples taken from real projects. Subsequently, we will discuss the benefits focussing on the big goal to minimize cost and risk.

1. INTRODUCTION

In the aerospace industry, the process of developing safety critical systems has changed significantly in the last few years. Since back when developing an aircraft function simply meant for the supplier to refine textual customer requirements into an implementation, times have changed.

Whereas traditional embedded devices used to work mainly isolated from each other, the system of the future will embrace numerous subsystems, all of them massively relying on information exchange. New architectural concepts such as shared communication resources and safety partitioning, force everyone who has a part in the development process to collaborate not only very closely but also at a very high level of detail. This trend shows for example in the Integrated-Modular-Anvionics concept (IMA) that was introduced in the context of the Airbus A380 development. Typical roles involved in a IMA development process are 'platform suppliers', 'application suppliers' and 'system integrators' [15]. Each stakeholder has his own understanding of the complete system and has to explain his position to the other stakeholders.

Due to an even higher degree of specialisation in the different the development disciplines, such as system-engineering, software-engineering and hardware-engineering, the number of involved stakeholders and perspectives has grown even more [1]. This has also brought a large variety of domain specific vocabulary into the game. Since most of the specification work is still based on textual requirements, there is often a wide scope of interpretation for the stakeholders [2].

What makes things even worse is, that the perimeters and interfaces of subsystems are evolving throughout the whole development process. This means that the idea of a strict waterfall-oriented development process where the next development phase only starts after the previous one has been finished, is not realistic anymore.

Beyond the fact that all of the above has increased the complexity of the entire development process, it has also led to a larger gap between users and developers of a system. In other words: What the developer implements is not necessarily what the user really needs.

A structured approach and the implementation of model based methodologies might be the solution how to meet this challenge [1]. In this paper, we present an approach that outlines models to define requirements at an early stage. The approach is designed to facilitate a common understanding of the system's properties for all stakeholders involved in the development process.

This paper is structured as follows:

First, we will explain briefly how systems are generally developed in the aerospace domain (chapter 2).

In chapter 3, we will give a short introduction to model-based development and use this knowledge as a basis to explain the Model-based Usage Analysis in chapter 4.

Looking at two current technology projects, we will demonstrate the benefits of the presented methodology in chapter 5.

Finally we will discuss the results and give an outlook towards future fields of research (chapter 6).

2. DEVELOPING SAFETY CRITICAL SYSTEMS

The development of avionic systems is tied to strict processes and guidelines. Official authorities control whether the suppliers follow these guidelines, and demand evidence in form of documentation. If inconsistencies in the documentation are found, authorities can force the supplier to re-do parts of the development.

This is why it is important to understand the special demands of the avionics domain when introducing a new methodology.

2.1. Requirements Based Engineering

The whole system development process is described in the document SAE ARP-4754 [13].

In the first phase, the aircraft manufacturer develops requirements on aircraft level, identifies functions and allocates them to corresponding systems. For each system, a PTS (Purchaser Technical Specification) is distributed to potential suppliers. The PTS mainly consists of textual requirements on system-level.

Subsequently, system developers refine the System-Requirements and define a system-architecture. Each element of the architecture is differentiated by hardware and software parts, and the High-Level Requirements are developed for each of them.

As a next step, hardware and software developers refine the High-Level Requirements until they are detailed enough to transform them directly into source-code or a hardware design. Such requirements are called "Low-Level Requirements".

At the end, it needs to be ensured that every element is traceable to a requirement on aircraft-level [10]. Traceability allows to validate the entire design on completion, and guarantees that customer’s high level requirements are in line with the implementation details.

Sometimes, design decision make it necessary to introduce new requirements. These requirements are not linked to an upward requirement, therefore, they are called "Derived Requirements". Derived Requirements need to be analysed separately to evaluate their impact on the safety of the final system.

The different levels of requirements are illustrated in Figure 1.

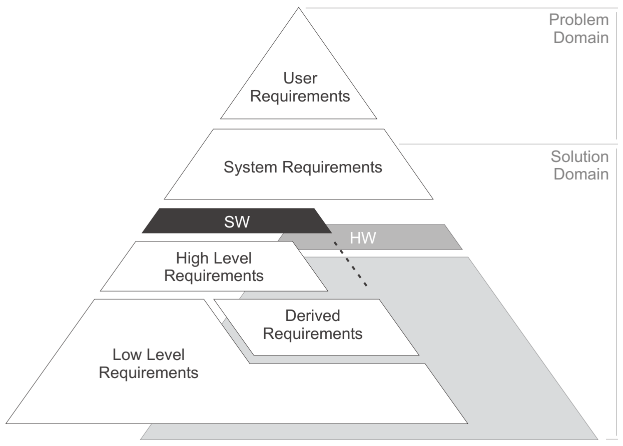

Figure 1 - requirements hierarchy

There are different kinds of requirements. In [6] we present a taxonomy for requirements. Non-functional requirements are mainly concerned with quality, safety and environmental topics.

Functional requirements specify precisely what the system shall do.

2.2. Validation and Verification

Validation of requirements is the process of ensuring that the specified requirements are sufficiently correct and complete so that the product will meet applicable airworthiness requirements.

Thus, in Figure 2, the arrows labelled “validation” are drawn backwards to preceding phases. In each phase, all requirements must be validated against the requirements of the previous phases before a project can proceed.

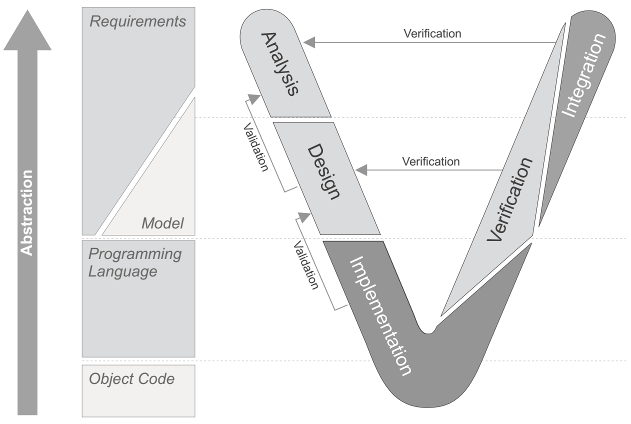

Figure 2 – schematic representation of the development process

Whereas validation considers only requirements, verification connects requirements and implementation. It is the process of assessing the correctness of the implementation according to given requirements that have been defined during the analysis phase.

Normally, the system is verified through a set of tests that has to be derived solely from the requirements without taking the implementation into account.

2.3. The gap between problem and solution

It is obvious that the mandatory development process for an avionics system is completely based on requirements and their verification.

Verifying requirements is the most expensive and time-consuming task in system-development. This is why aircraft manufacturers nowadays tend to handover the full responsibility for the costly requirements engineering to the supplier.

Slowly this leads to an increasing gap between the problem-domain and the solution-domain. Therefore users often find it difficult to articulate their requirements [3]. This is one of the reasons why only about 10% of the requirements in a current PTS are functional requirements [5]. Whilst non-functional requirements are strongly standardized by official guidelines defined by authorities and do not leave much room for interpretation, functional requirements are much less formalized.

Users often specify requirements in natural language [1] or use their domain specific vocabulary. This often leads to misinterpretations when performing a validation from a lower-level requirement. If this happens it cannot be guaranteed that low level tests correctly verify the user requirements. As a consequence, the whole verification process must be called into question.

To avoid this, it is important that the functions of a system are specified up to an unambiguous level jointly with the users and the developers. Model-based development is one of the approaches to address this challenge.

3. MODEL BASED DEVELOPMENT

In [3] Nuseibeh and Easterbrook give an overview of requirements engineering and state that a significant proportion is about developing domain descriptions. This is often done with documents that contain simple diagrams and a textual description. In [1] they state “Because of the lack of proper syntax and semantics, other project members often misinterpreted the diagrams”.

Models introduce syntax and semantics for an unambiguous description of a system, and therefore help avoid misinterpretations. A model is a structured and formal description of a certain aspect of a system and usually highlights properties of interest from a given viewpoint. In Model Driven Engineering (MDE), the model of the system to be developed controls the entire development process.

This approach is finding more and more acceptance, even among the responsible authorities like the EASA or the FAA. The next version of DO-178B [10] will contain a model-based supplement that regulates the usage of models during the development of a safety critical software system.

Models are usually expressed in a modelling language and use diagrams to visualize certain aspects. No matter what modelling technique is used, they have certain aspects in common:

- They describe the properties and structure of model entities.
- They detail the relationships between model entities.
- They allow a precise, formal description of the system's behaviour.
- They guarantee consistency at all levels.

Most often, models are used in the later phases of the system development process to formulate the detailed design. Domain specific modelling languages on the other hand could be used to formally describe the environment of the system. Therefore they are useful in earlier development phases.

3.1. Domain Specific Models

Requirements engineering is concerned with interpreting stakeholder terminology, concepts, viewpoints and goals. Each stakeholder must be able to express his thoughts and ideas in a language that is close to the one that he uses on a daily basis. This is why numerous domain specific techniques and tools have evolved in the past.

In [8] Rumbaugh, Jacobson and Booch present the Unified Modeling Language (UML). In [1] they found out, that “UML was the most commonly used notation for architectural modelling”. The 'System Modeling Language' (SysML) and the 'Architecture Analysis and Design Language' (AADL) have also become popular during the last years.

In addition to the well-known modelling languages mentioned above, there are numerous languages originating from academic groups (e.g.: Intermediate Language for Model Verification (FIACRE)) or adoptions of pre-existing processes (e.g.: Structured Analysis Method (SAM)). These languages can sometimes satisfy the needs of a specific problem better than the main-stream languages.

Since not all stakeholders are willing to use the same modelling language, it is important to find ways to translate a model from one language to an equivalent model in another language.

3.2. Model Transformation

The fact that a modelling language delivers formal syntax and semantics allows us to automatically transform models between different representations. Parts of a AADL model that has been used to perform architectural analysis could be translated into a UML model that serves as a basis for the software design.

Translation rules have to be specified to detail how a construct in one language is transformed into a construct in another language. Model transformation engines use these rules to translate a model piece by piece into another representation. The nature of the translation rules determines if the transformation is partial or complete. A complete translation yields an equivalent representation while a partial translation only considers certain aspects of the original model.

One of the most common model transformations used in current projects is the generation of documents from a model. This approach has the advantage that changes in the model are automatically reflected in all referring documents.

Model transformation enables all stakeholders to work with their own modelling languages and tools. Given a set of appropriate transformation rules, the modelling process is completely independent from the selected languages and tools.

3.3. Model Analysis

Formal methods can be used to verify parts of a model. If the requirements are formalized to a sufficient degree, this process could be achieved automatically. In [5] we give an overview of what kind of requirements can be formalized, and how this can be accomplished. We will give some examples of effectively occurring requirements, and will outline formal methods to verify the respective properties.

The Object Constraint Language (OCL) is a possibility to formally define a restriction or a rule that must be followed by the system that is currently modelled. In [9] they demonstrate how to use formal methods in UML/OCL models. With this approach it would be possible to formally verify whether a system fulfils certain requirements.

Since it takes much effort to create such formal description, sometimes the preferred method of model verification is a simulation. A simulation can help detect errors in the specification and find solutions together with the customer.

3.4. Modelling Tools

There are numerous tools available that can be used to create and analyse models. One of the most common commercial tools in the aerospace world is Rhapsody by IBM. Modelling tools usually provide means to create and manipulate models. The Eclipse Modelling Framework (EMF) is the first attempt to provide an abstract infrastructure that enables model transformation and model analysis on an open-source basis.

In the context of this work, we have selected the open-source tool TOPCASED that is based on EMF for the creation of our model parts.

TOPCASED is the acronym for "Toolkit in OPen source for Critical Applications & SystEms Development". The Toolkit is developed and maintained by a consortium that originated from the AeroSpace Valley, one out of the six “world-class” competitiveness clusters selected by the French Government in July 2005 [11]. The consortium, is led by Airbus France.

The tool contains generic services for model serialization and manipulation, and it comes with the integrated Atlas Transformation Language (ATL).

An integrated Configuration Management Interface, a sophisticated subcontracting-mode and the possibility to share models across different teams allows a seamless collaboration with many stakeholders.

4. MODEL-BASED USAGE ANALYSIS

As described above, the model-based approach is the basis for an efficient and unambiguous exchange of information between different stakeholders. We propose to use this technique as early as possible in the development process to avoid misinterpretations in later phases. For this purpose, we would like to introduce a possible model structure and the corresponding semantics to formally express the knowledge and needs of all stakeholders in a single model.

We have chosen a use-case driven methodology and open-source tools to perform the model-based usage analysis. The analysis is performed following the steps below.

- First the domain-model is specified.
- Then the usage-scenarios are defined.
- At the same time, the architecture is refined.
- When the model is detailed enough, analysis tools can be used to assess the specification with respect to correctness or other attributes.

The different models that are created during the usage-domain analysis are structured hierarchically. Each step includes the results of the previous step, and adds new information to it.

This model structure allows us to distinguish clearly between user-domain and solution domain. In Figure 3 there is an overview of the three main models that are used to describe the results of the analysis. The domain-model contains pre-existing entities coming from the domain knowledge of the stakeholders.

The architecture-model describes parts of the developed system, and is therefore more relevant for the developers than for the users.

The usage-model bridges the gap between the domain-model and the architecture-model. Since it contains knowledge of both, users and developers, it can be considered as the missing link between both worlds.

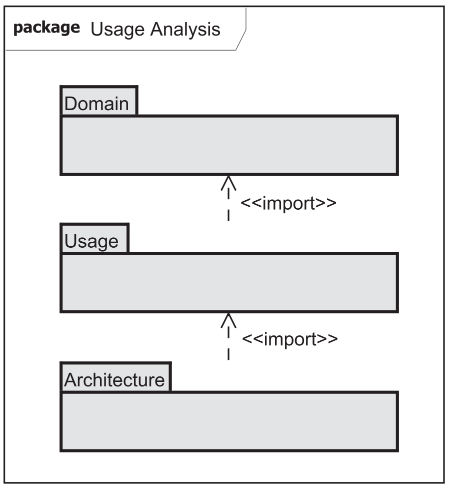

Figure 3 - different models of the usage analysis

The different models are explained in more detail in the next three chapters.

4.1. The domain model

In [3] a roadmap is given on how to improve the specification of systems in the future. One of the major suggestions was to formally model and analyse properties of a system's environment.

As suggested in [2] our domain model represents a description of the environment in which the envisioned system will operate. It is comparable to a glossary where the basic building blocks and their relationships are defined. It is important that every stakeholder can find representations of his main objects in the domain model.

We use UML class diagrams and simple associations between classes for the domain model. All stakeholders are given the task to add a textual description to their own building blocks in order to help the others with the interpretation of each model element.

In Figure 4 we have created a simple domain model to explain the main concepts.

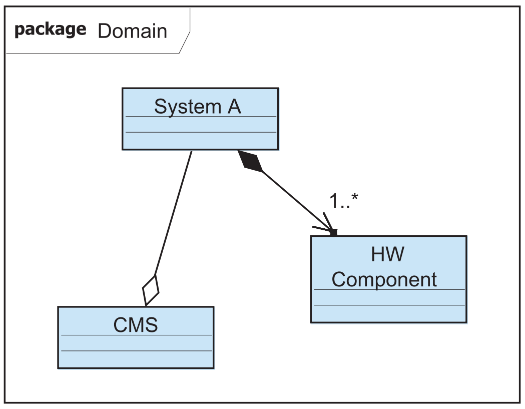

Figure 4 - a simple domain model

The example system (*System A*) consists of several hardware components, as shows the filled diamond at the bottom of the association arrow connecting *System A* and *HW Component*. Such a relationship between two entities is called a composition in UML.

Furthermore there is a *Centralized Maintenance System* (*CMS*) that is connected to the system. The *CMS* is not part of *System A*, but it is associated with it. A hollow diamond indicates that the connected entities are only loosely coupled. Such an association is called an 'aggregation' in UML. In natural language this can be translated with "A knows B".

One simple diagram (three boxes and two arrows) is already sufficient to demonstrate some very important aspects of the system in an intuitive way.

4.2. Usage Scenarios

In the next step we define how the system will be used and how the entities of our domain model interact with each other.

The Use Case Approach from Ivar Jacobson [12] has not been widely adopted in the avionics domain. Nevertheless in [1] they found out that "another qualitative technique employed was scenario-based analysis”.

Instead of starting with the system's architecture, we begin with the question of how the system will be used in operations. In a first step, we declare all the stakeholders of the system as actors in a use-case diagram as shown in Figure 5.

In our example there are two different user classes. The maintenance engineer, for instance, will use the system differently in comparison to the normal user. Normal users can be either a pilot or a crew member.

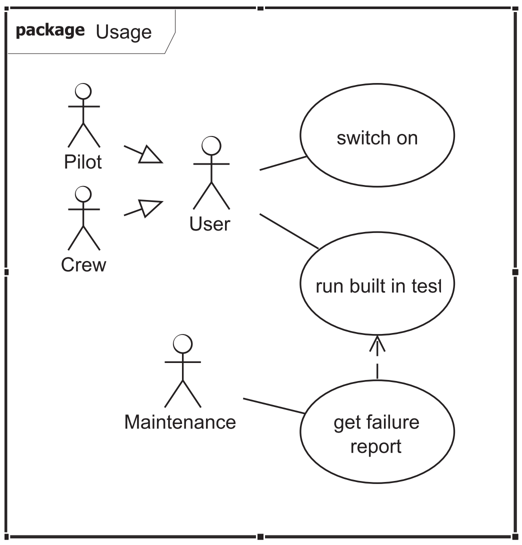

Figure 5 - Use Case Diagram for our example

To this end, a requirements engineer can elicit information about the tasks that users currently perform, and also on those that they might want to perform [4].

The use-cases associated with the system can be detailed together with the corresponding stakeholders in one or several workshops (depending on the complexity of the use-case and the skills of the stakeholders). We use UML interaction-diagrams to give practitioners the possibility to precisely document the use-cases.

Lifelines in interaction diagrams represent objects from the domain-model that play a role in the corresponding use-case. The Lifelines are illustrated as vertical, dotted lines; the rectangle on top shows the name of the domain object.

Messages between lifelines indicate how the entities interact. A message can be either data or a command. With this simple diagram we have captured the control-flow and the data-flow of each use-case as requested by [10].

An example for a detailed scenario specification is given in Figure 6.

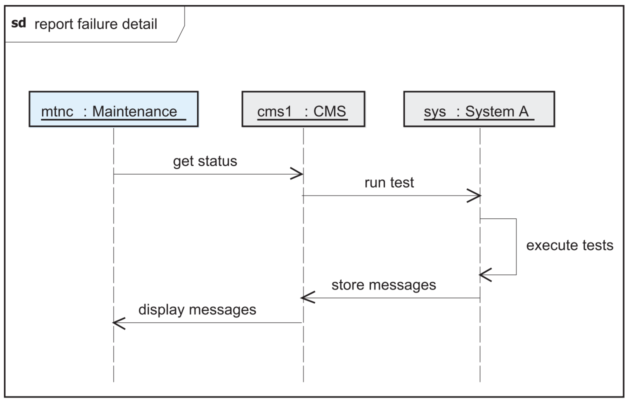

Figure 6 - example usage scenario

The interaction-diagram is a very efficient tool to help stakeholders think in a structured pattern, and the resulting specification is unambiguous. Furthermore the rule stating that a lifeline has to be linked to an object of the domain-model effects consistency throughout the different parts of the model.

As an additional benefit, we know that authorities already accept this kind of representation as a valid part of a specification.

4.3. Architecture Model

Whilst the domain model exclusively contains elements from the problem domain, the architecture model already describes parts of the solution. The architecture must be able to realise all usage scenarios defined in the previous step.

Again we use UML Class diagrams to define the architecture-model. The architecture-model should always contain the elements of the domain-model and put them in relation to new architectural elements. These new elements represent the parts of our system that have to be developed.

Architecture-models can be hierarchically decomposed. For each level in the hierarchy new usage scenarios have to be defined until the architecture is detailed enough to function as a basis for the detailed design.

Figure 7 shows the architecture of our example after the definition of the first usage scenario.

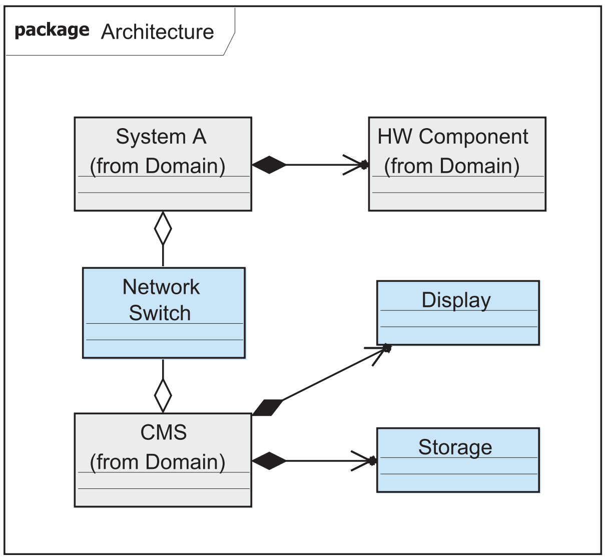

Figure 7 - example architecture

We have added a *Network Switch*, since we now know that the *CMS* has to exchange data with *System A* to realize the usage scenario from Figure 6. Furthermore, we have added a display to the *CMS*, because the maintenance engineer should be able to read the failure messages somehow. Additional memory ('Storage') must be available inside the *CMS* to store messages generated by *System A*.

In this manner the system architecture is bound to evolve step by step. The architecture model can be used as a pre-stage for the later design; it is the basis for detailed analyses as explained in the next chapter.

4.4. Model Exploitation

The usage analysis model generated thus far has some properties which enable a structured and automated analysis. It contains knowledge on dependencies between model elements. The interaction-diagrams implicitly define how the different elements of the system interact with each other.

A very basic possibility is to use the model as a database, and extract information with predefined queries. We used the Object Constraint Language (OCL) to develop queries on a model that has enabled us to calculate complexity metrics and also to extract interfaces for each model entity automatically.

In our case studies we used the model to generate large parts of the documentation automatically. The built-in document-generator of TOPCASED is able to process document templates and extract diagrams along with text from the model.

As a next step, we experimented with analysis tools to detect subtle errors in the architecture. Using the model transformation engine of TOPCASED we translated parts of the model into AADL and conducted latency and performance analyses. We even used a similar approach as described in [7] to perform an automated safety analysis.

When it comes down to verification, nothing is more convincing than a mathematical proof. In [4] they demonstrate that a sequence diagram can be formalized using template semantics. This high degree of formalisation will allow the usage of deductive methods for the verification of certain properties of the model.

5. CASE STUDIES

Currently, there is no quantitative study of the effectiveness available. Early stage case studies as a prove of concept has been done.

In this chapter we present the models generated during the usage analysis of two current technology projects. First we will explain the context of each project and give some examples for each model part of the usage analysis.

At the end we will explain how the models have been exploited in each project.

5.1. The integrated Airport – IPORT

In the framework of the project IPORT [14] we are currently developing an optimised taxiing process in collaboration with both an airport operating company and the responsible air traffic control authority. The goal is to achieve a higher level of automation compared to current procedures. Additional displays in the cockpit, an enhanced lighting system on the surface and a new front-end for the controller on the ground are designed to provide both more safety and efficiency for surface operations on airports.

The system considered in IPORT includes a large amount of subsystems and involves ground and airborne equipment. Due to the large scale of the system, many stakeholders from different domains are involved.

When the project was launched there were no textual requirements available. Even the understanding of the development process was inhomogeneous among the stakeholders. Different guidelines from different authorities had to be consolidated.

The usage-domain analysis has been used in IPORT to support the discussions of different operational concepts and different levels of automation.

5.1.1. Domain Model

The domain model of IPORT describes all entities that are involved in the surface operations of an airport. In Figure 8 parts of the domain-model of IPORT are depicted.

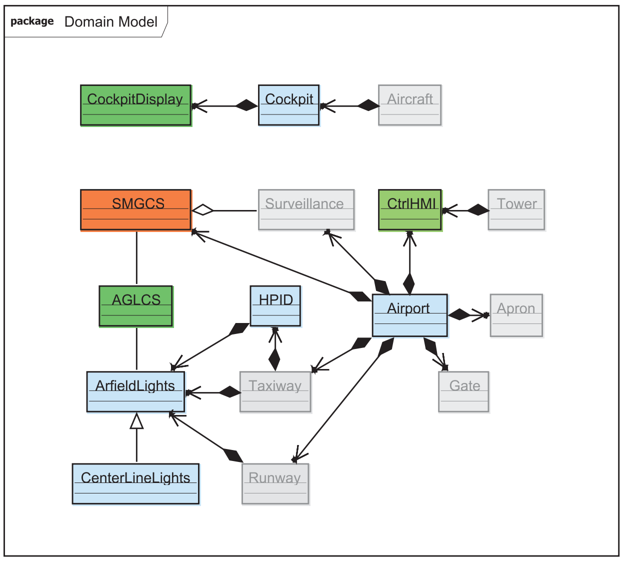

Figure 8 – the IPORT domain-model (extract)

In IPORT the stakeholders originate from very different domains. This is why it is important to describe the entities of the domain very precisely. For basic terms such as 'airport' and 'runway' we used the definitions from ICAO. The technical infrastructure was described by the corresponding domain experts.

5.1.2. Usage Scenarios and Architecture

In IPORT we managed to involve real users in the usage-analysis. In 12 use-cases we described surface operations that shall be performed with the support of new technical equipment.

In Figure 9 the detailed specification of the usage-scenario "runway incursion" is shown. A runway incursion occurs when an aircraft enters the runway without the clearance of a controller. In the worst case this can lead to a collision with another aircraft that is using the same runway for take-off or landing.

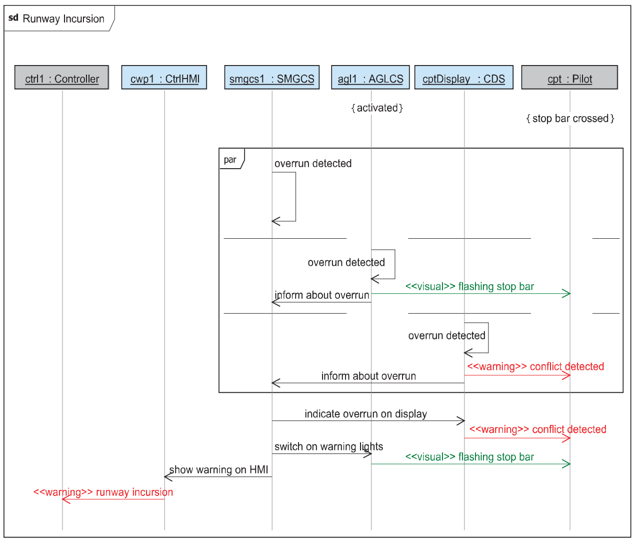

Figure 9 - IPORT usage-scenario – runway incursion

Tower-controllers, apron-controllers and pilots have actively participated in the specification process of the scenarios.

We used UML-stereotypes to distinguish between different kinds of interactions, like voice communication and visual indications. This is crucial for possible safety analysis. With this additional information, we were able to assess the impact of the new procedures on the usage of the voice communication channel.

5.1.3. Model Exploitation

In addition to an early feasibility analysis through simulation, the model is used for the verification of the system in the later phases of the development.

The interaction-diagrams were used to automatically generate test-cases for the verification of the final system. With OCL queries we extracted the necessary interfaces of all subsystems from the model.

Furthermore major parts of the operational concept documentation was automatically generated from the model.

5.2. The intelligent Cockpit – INTECO

The project INTECO is about developing a synthetic vision display for rescue helicopters. Bad weather conditions in conjunction with challenging mission profiles often lead to a very high workload for the helicopter crew.

A three-dimensional terrain representation shall increase the situational awareness of the pilot in low-visibility conditions and support the crew during difficult manoeuvres.

To facilitate the management of the high-resolution terrain data, a highly reliably and efficient database management system has to be developed. Furthermore a powerful graphics generation device has to be integrated into the system that is able to generate complex three-dimensional graphics for cockpit-displays.

Stakeholders in this project are the helicopter manufacturer, the function supplier responsible for the display applications and the platform supplier who delivers the hardware components and basic software services.

5.2.1. Domain-Model and Use-Cases

The main entities in the domain model of INTECO are the *database server* and the *graphics generator* (see Figure 10). The *database server* manages geographical information like terrain data and obstacle data which is necessary for creating a detailed depiction of the current mission area. The *graphics generator* is responsible for transforming data received from the mission system and the database server into a picture on the *cockpit displays*.

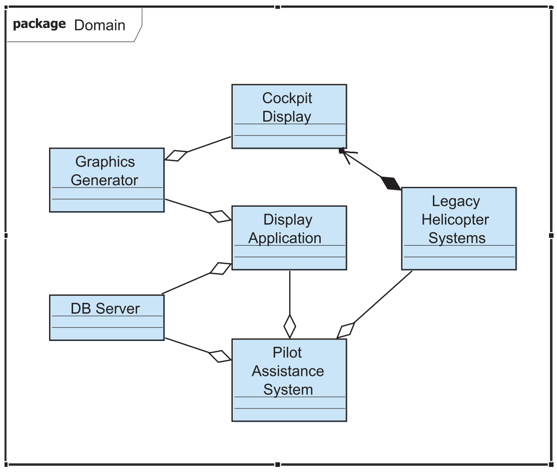

Figure 10 - INTECO domain model

Together with the helicopter manufacturer, high level requirements have been developed and at the same time the domain-model has been created. TOPCASED-mechanisms have been used to link the textual requirements to elements of the model.

Since the system to be developed in the frame of INTECO has the character of a platform, operational aspects do not play a key role in the model. Under certain circumstances the user of the platform has to embed the domain-model into his operational model to get a complete specification. This will be possible through the transformation mechanisms described in chapter 3.2.

Each element of the domain model is associated with use-cases that describe the expected functionality on a high level. In Figure 11 all use-cases associated with the *database server* are listed. The use-cases have been delivered from the helicopter manufacturer as part of the user-requirements. They serve as a basis for the verification of the system.

Furthermore, in INTECO we had access to the prototype of the system developed earlier. Thus we were able to validate the use-cases on a running platform. This is one of the reasons why the descriptions of the use-cases were already very well developed at an early project stage.

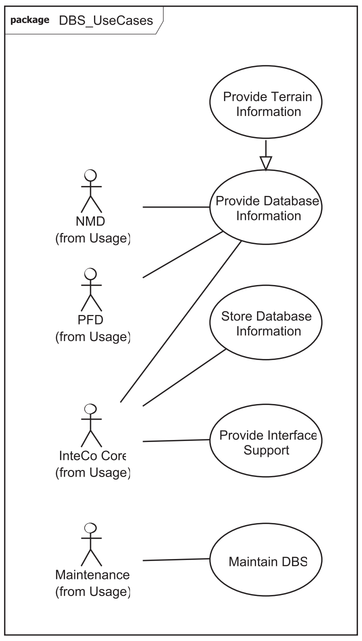

Figure 11 - INTECO DB Server Use-Cases

5.2.2. Usage Scenarios and Architecture

In several modelling workshops the use-cases have been refined into usage-scenarios. In collaboration with software engineers the use-cases were realized using real function calls of pre-existing software libraries.

In Figure 12 an example scenario is illustrated. The scenarios have been reviewed and commented on by all partners of the project. All comments have been directly incorporated into the model and validated through simulation.

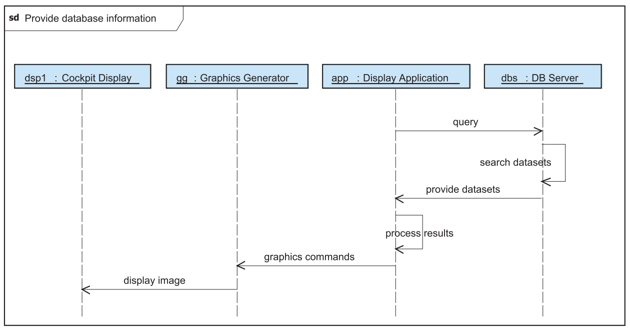

Figure 12 - INTECO scenario "provide database information"

It is remarkable that the INTECO scenarios do not include as many stakeholders as the scenarios in the IPORT project. This is due to the different natures of both projects.

While in IPORT we concentrated on the definition of an operational concept for the system, in INTECO we specified the technical protocol used to communicate with the *database server* and the *graphics generator*.

5.2.3. Model Exploitation

In INTECO we used the described method in the first place to assess the feasibility of the user-requirements. Working with the created model, we used simulation data and automated consistency checks with OCL-constraints to refine and validate the specification together with our customer.

Since the model was directly linked to the textual requirements, it was possible to analyse the impact of changes in the specification and identify problematic requirements.

Additionally, we developed a methodology to derive test cases from the behavioural description (i.e. the interaction diagrams) of the system; and we used UML activity diagrams for the low-level specification of the software functions. Since both definitions were available in a single model, we were able to partially generate the source code from the activity diagrams and run the generated test cases on the compiled software.

Combined with a model-based safety analysis that was based on information derived from the usage scenarios, the model greatly supported all necessary verification activities.

6. RESULTS

In all the above projects, stakeholders from different domains were involved. The model-based usage analysis has been performed to derive the specification and architecture for the final system.

Since there is no representative data available for the costs of an average requirements engineering process we were not able to calculate the real savings of the method presented.

All stakeholders agreed that the transparency that could be achieved by our approach and also the illustrative diagrams were helpful to create a common understanding of the system.

During the specification work of both projects new dependencies have been discovered that have not been thought of at the beginning of the project. Especially the possibility to integrate changes directly into the model helped to explain the consequences of each change to all stakeholders.

The creation of these models can be a time-consuming task. Then again, compared to a pure textual approach to define a specification, time and effort for explanations and clarifications could be reduced in our case-studies.

7. CONCLUSION

The high complexity of modern systems and the necessity for a full traceability, as explained in chapter 2.1, makes it necessary for the supplier to develop a large amount of requirements on his own.

We have presented a structured approach to refine and formally develop requirements for a complex system. Model-based methods have been used to formalize as much of the available domain knowledge as possible. The method helps propagate this knowledge throughout the different disciplines and domains of all stakeholders.

Furthermore, the created models can be re-used in later phases of the development (e.g.: as design models or verification models) and therefore the investment is always useful.

The tools used in the aforementioned technology projects are mature and the method has proven to work for real-life projects. Our method enables suppliers to offer an unambiguous and mature description of a system in a very early stage. Therefore it reduces the risk of expensive changes in the later stages of development.

Our goal for the near future is to improve the assessment options of the models by creating more analysis tools and formal constraints. We hope to increase the usability of existing modelling tools by customizing them for the usage-domain analysis approach and integrating them into the established toolset.

REFERENCES

[1] B. Graaf, M. Lormans, and H. Toetenel, "Embedded Software Engineering: The State of the Practice", IEEE Software, 2003, p. 61-69

[2] Bashar Nuseibeh and Steve Easterbrook, “Requirements engineering: a roadmap,” in Proceedings of the Conference on The Future of Software Engineering (Limerick, Ireland: ACM, 2000), p. 35-46

[3] P. Johnson, Human-Computer Interaction: psychology, task analysis and software engineering, McGraw-Hill, 1992

[4] H. Shen, A. Virani and J. Niu, “Formalize UML 2 Sequence Diagrams” in Proceedings of the 2008 11th IEEE High Assurance Systems Engineering Symposium, p. 437-440

[5] B. Langer and M Tautschnig, “Navigating the Requirements Jungle”, Leveraging Applications of Formal Methods, Verification and Validation, Communications in Computer and Information Science, Volume 17. Springer Berlin Heidelberg, 2009, p. 354ff

[6] N. Pontisso und D. Chemouil, “TOPCASED Combining Formal Methods with Model-Driven Engineering,” Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on, 2006, p. 359-360.

[7] J. Liu, J. Dehlinger, R. Lutz, “Safety analysis of software product lines using state-based modeling”, The Journal of Systems and Software 80, 2007, p. 1879-1892

[8] J. Rumbaugh,, I. Jacobson, and G. Booch, Unified Modeling Language Reference Manual, the (2nd Edition). Pearson Higher Education, 2004.

[9] A. Occello, A. Dery-Pinna, M. Riveill, “Validation and Verification of an UML/OCL Model with USE and B: Case Study and Lessons Learnt ”, IEEE International Conference on Software Testing Verification and Validation Workshop, 2008

[10] RTCA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992

[11] N. Pontisso und D. Chemouil, “TOPCASED Combining Formal Methods with Model-Driven Engineering,” Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on, 2006, p. 359-360.

[12] I. Jacobson, M. Christerson, P. Jonsson, und G. Overgaard, “Object-Oriented Software Engineering: A Use Case Driven Approach.", Harlow, Essex, England: Addison Wesley Longman, 1992

[13] SAE ARP4754 – Certification Considerations For Highly-Integrated Or Complex Aircraft Systems, Warrendale, PA, 1996

[14] iPort-Project Description : http://www.dlr.de/pt-lf/Portaldata/50/Resources/dokumente/VUE_2009_iPort.pdf (seen on 03.05.2010)

[15] RTCA DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations, 2005