

# Formal Specification Techniques as a Catalyst in Validation

Bernhard K. Aichernig, Andreas Gerstinger  
Institute for Software Technology (IST),  
Graz University of Technology,  
Inffeldgasse 16b, 8010 Graz, Austria  
(aichernig | agerstin)@ist.tu-graz.ac.at

Robert Aster  
Frequentis Nachrichten GesmbH  
Spittelbreitengasse 34, A-1120 Wien,  
Austria  
raster@frequentis.com

## 1 Introduction

The American Heritage Dictionary defines a catalyst as a substance, usually present in small amounts relative to the reactants, that modifies and especially increases the rate of a chemical reaction without being consumed in the process. This article reports on the experience gained in an industrial project that formal specification techniques form such a catalyst in the validation of complex systems. These formal development methods improve the validation process significantly by generating precise questions about the system’s intended functionality very early and by uncovering ambiguities and faults in textual requirement documents.

This project has been a cooperation between the IST and the company Frequentis<sup>1</sup>. The Vienna Development Method (VDM)<sup>2</sup> has been used for validating the functional requirements and the existing acceptance tests of a network node for voice communication in air traffic control. In addition to several detected requirement faults, the formal specification highlighted how additional test-cases could be derived systematically.

Figure 1 gives an overview of the processes which were carried out. The shaded processes and documents were performed or created by the formal methods engineer using IFAD’s commercial VDMTools<sup>3</sup>.

## 2 Requirements Validation

The system still under development is a large voice communication system for air traffic control in Austria. The requirements are contained in a regularly updated document of almost 3000 requirements formulated in natural language. The task of the formal methods engineer was to create an executable model of a part of

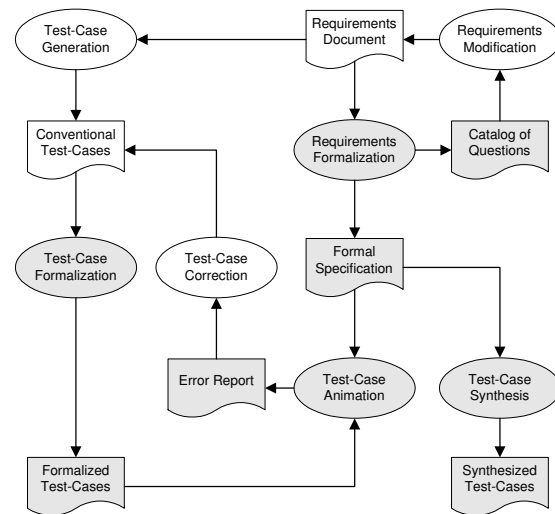


Figure 1. Processes with inputs and outputs

the system—specifically a network node—which plays a crucial role. There was also a first version of the architectural design document available which was helpful for gaining a general overview of the network node, however, the formal specification is solely based on the requirements. The formal methods engineer had no domain knowledge in air traffic control. Therefore, he was introduced to the domain and the system to be developed in a  $1\frac{1}{2}$  day training session held by a system engineer. It should be mentioned that the formal methods engineer was not directly involved in the main project—the formalization was done in parallel.

The first step was an intense study of the requirements and architectural design document. The formal methods engineer was not given a list of requirements to be formalized, he was rather instructed to formalize certain functionalities. Although the requirements document did contain some structure, one of the most

<sup>1</sup>Frequentis’ homepage: <http://www.frequentis.com>

<sup>2</sup>Information on VDM: <http://www.csr.newcastle.ac.uk/vdm/>

<sup>3</sup>IFAD’s homepage: <http://www.ifad.dk>

challenging tasks was the selection of the relevant requirements.

The next step was the actual formalization of the selected requirements. The formal model was constructed in two stages: first a model of the system's state, using types constrained by data-invariants, then the functional model containing the operations. This formalization process yielded the most valuable output. Formalizing requirements requires understanding them rigorously and in an unambiguous way. Since natural language requirements are doomed to be ambiguous, several questions have been raised during the formalization process. These questions have been incorporated into a "catalog of questions" which was regularly exchanged between the formal methods engineer and the requirements manager. During the whole project, 108 questions have been raised, with 33 of them resulting in changes in the requirements document. These questions uncovered inconsistencies, ambiguities, errors, omissions, redundancies and inaccuracies in the requirements document—which are unavoidable in textual requirements documents of this size. This led to an improvement in the quality of the requirements.

Studying the documents and selecting the relevant requirements took 5.5 weeks, the actual formalization process took another 6.5 weeks for 140 requirements. The size of the formal specification was 60 pages including comments.

This phase could have been completed in considerably shorter time if it would have been performed by a domain expert. We believe however, that in this phase, the lack of domain knowledge is not a disadvantage—in contrast—it might be advantageous because of the type of unbiased questions which arise. This phase resembles a validation of the requirements.

### 3 Example

The following example demonstrates the kind of issues being raised during the formalization process. A relatively simple example is chosen to keep the explanations at a minimum. However, a basic knowledge about the system is required:

The primary goal of the new system is the complete interconnection of air traffic control centers via a fully digital network. The digital network lines will be used for voice (telephone and radio) and data transmissions. There are several interlinked air traffic control centers, each of them consisting of a network node and a voice communication system (VCS) (Figure 2). The VCS is a switch with a large number of interfaces, which can be used to connect several types of voice communication to each other (Figure 3). The most important

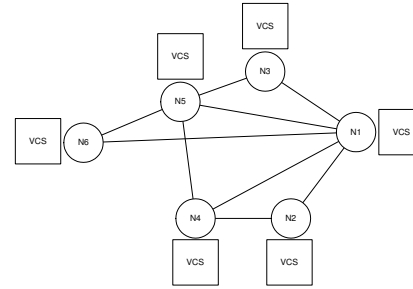


Figure 2. A network for air traffic control

interface from our viewpoint is the controller working position (CWP). This is an integrated work-place for the air traffic controller with a touch panel, headsets, handsets and loudspeakers. A VCS can, depending on its configuration, support a large number of controller positions.

One of the advantages of the new system is that lines between the network nodes are allocated dynamically. Therefore, additional lines can be allocated to compensate a broken line or to provide additional bandwidth in times of high network traffic. This dynamic allocation is also valuable from a cost effectiveness point of view. Another important concept are "roles". Roles are simply functionalities: for instance the functionality to control a certain sector of air traffic. These roles - or functionalities - are not bound to specific CWPs, but can be distributed within the network, to optimize the work-load on the controllers. Therefore, it is possible to control air space, independent of the physical location of the controller.

The example is about the key layout on the touch panels of the CWPs. If a CWP handles the functionality of a specific role, we say the CWP has the role assigned. Roles are grouped into function groups. A function group contains roles which have similar func-

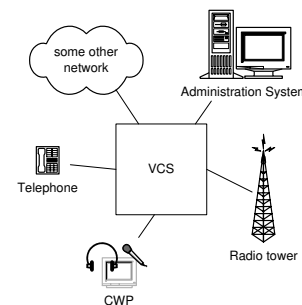


Figure 3. Voice Communication System (VCS)

ID	Text
#0025	A function group contains all roles which have about the same function and therefore have the same key layout. There are up to 9 function groups 1..9. Each role belongs to exactly one function group. The function group 0 contains the key layout for CWP's which own roles from more than one function group.
#0042	If a CWP has roles from more than one function group assigned, a default layout shall be displayed.
#0062	If all assigned roles belong to the same function group, the key layout corresponding to the function group shall be displayed.
#1047	The key layout of the Test-CWP must always be the layout of the function group 0.

**Figure 4. Requirements for the touch panel layout on CWP's**

tionality. A VCS may have a special CWP with additional functionality which is called Test-CWP. The original requirements for displaying the correct layout of a touch panel are summarized in Figure 4.

If these requirements in plain text are read quickly, they seem to be quite reasonable. However, if they are read more thoroughly they give rise to questions. If they are read with the process of formalization in mind, they give rise to even more questions. And once the actual formalization takes place, even more issues may be raised. This was the process by which the whole project was conducted: some issues were detected just by reading and having the formalization in mind, but some issues require the actual formalization to be raised. Since the goal of the formalization was an executable specification—a high level prototype—the focus was also on implementation. Issues were raised which might otherwise have been detected much later in the project, namely in the implementation phase.

The most obvious inconsistency is the following. Requirement #0025 is inconsistent in itself: First, it is stated that there are 9 function groups which are numbered from 1 to 9. The same requirement, however, mentions function group 0! This is an easy-to-detect inconsistency, which probably does not require formalization to be uncovered.

However, if we read on, we see that requirement #0042 talks about a certain default layout. If we don't

have the formalization in mind, we might just think about this as a certain layout. However, if we view the "default layout" in connection with the other requirements, the question is raised if the "layout of the function group 0" and the "default layout" are equivalent. This is in fact true, so there is an inconsistent terminology in the requirements.

An unambiguous specification can be given using VDM type definitions:

$$\text{Functiongroup} = \mathbb{N}$$

$$\text{inv } f \triangleq f \in \{1, \dots, 9\};$$

$$\text{Layout} = \text{Functiongroup} \mid \text{DEFAULT};$$

A data-invariant and a union type model the fact that there are 9 function groups numbered 1 to 9, but ten layouts: one for each function group plus an additional default layout.

During the formal specification of the actual layout calculation another issue was raised. So far we know:

- if a CWP has roles from only one function group, the layout of the function group shall be displayed.
- if a CWP has roles from more than one function group, the default layout shall be displayed.
- the Test-CWP shall always display the default layout.

Below, the formal definition is shown:

$$\text{CalcLayout} : \text{APLID} \times \text{VCSID} \xrightarrow{o} \text{Layout}$$

```

CalcLayout (cwpid, vcsid)  $\triangleq$ 
  (if is-TestCWP (vcss, cwpid)
   then return DEFAULT
   else let assigned-roles =
         get-roles-of-CWP (rv, cwpid),
         functiongroups =
         {get-fg (roles, vcsid, rid) |
          rid  $\in$  assigned-roles} in
   cases functiongroups:
     {}  $\rightarrow$  return DEFAULT,
     {f}  $\rightarrow$  return f,
     others  $\rightarrow$  return DEFAULT
  end)

```

The if-clause is used to display the correct layout on the Test-CWP. Then, the set of assigned function groups is constructed from the set of assigned roles with the help of a set comprehension expression. In the following cases statement, a layout is returned depending on the set of function groups. A set can have one, more, or no elements. The case for one or more

function groups (i.e. the second and third branch of the cases statement) is covered in the requirements. However, what if a CWP has no roles (and therefore no function groups) assigned? This case (i.e. the first branch of the cases statement) is not covered in the requirements!

This is not necessarily an error in the requirements, since this case may have been deliberately omitted, and left to the discretion of the engineers responsible for the following phases. In this case, it could have been left open in the formal specification as well. However, it was not meant to be left open. There should be a requirement saying:

*If a CWP has no roles assigned, a default layout shall be displayed.*

## 4 Test-Case Validation

For parts of the functionality covered by the formal specification, conventionally designed test-cases already existed. These test-cases were formulated in a very clear—we call it semi-formal—way. Each test case is broken down to atomic actions and the results of each action are specified in great detail.

The task of the formal methods engineer was to formalize these test-cases and animate them on the formal specification using the interpreter tool. There were 65 test-cases consisting of approximately 200 single steps of which 60% were formalized. Due to the detailed specification of the test cases, their formalization was a straightforward task.

By comparing the expected results specified in the original test-cases with the results returned by the formal specification—which acted as a prototype—a test-case validation has taken place. 16 errors within the original test-cases have been detected and could therefore be corrected. Furthermore, the animation of test cases also led to a validation of the formal model. Seven minor errors were found in the formal specification and consequently corrected. Another output of this stage was the calculation of the test coverage of the formalized test cases with the help of the test coverage metrics built-in in IFAD's VDMTools. It was realized that this coverage metrics, which is basically expression coverage, is rather weak and that complete coverage was quickly attained. This is also due to the fact that the conventional test-cases were designed in a very thorough manner: for each requirement one or more test-cases have been specified. The time effort for the formalization and animation of the test-cases was 2 weeks.

## 5 Test-Case Synthesis

The last stage of the project was the synthesis of test cases with the help of the information contained in the formal specification. Several methodologies have been used to design additional test-cases. The remarkable aspect is that essentially white-box techniques on the specification have been used to design black-box tests for the final system which led to a well defined process to elicit test-cases. The time effort for this stage was 5 weeks.

However, the output of this stage was not as substantial as the previous outputs. We believe that for the creation of test-cases, the formal model should more closely resemble the final system. This can be achieved if the formal model is created by one of the domain experts.

## 6 View of the Requirements Engineer

In the mentioned project, the application of VDM took place at a relatively late stage. At this time, each member of the project team already possessed a high level of insider-knowledge. Considering this fact, it was even more astonishing, that the VDM-engineer quickly became familiar with the complex subject. Subsequently, the application of VDM quickly led to qualified questions. Consequently, several inconsistencies in the requirements were uncovered and corrected prior to the implementation phase. Apart from this, it was realized that several requirements were well known to the members of the project team without being actually written down.

## 7 Concluding Remarks

Summing up, the formalization of 140 requirements uncovered 108 open issues leading to 33 changes in the requirements document, and 16 errors in the acceptance test-cases have been detected. This was done in 14 man weeks. Considering the fact that requirements engineering is one of the most difficult parts in a project this effort seems to be acceptable. Especially, in a safety-critical domain like air traffic control, where voice communication must not fail. Especially, the customer Austro-Control appreciated the increased rate of early questions for clarification. This emphasizes the important catalyst role the formalization plays in validation. However, the experience indicates that this method could be much more effective if it would be more tightly integrated into the conventional development process.