38 REQUIREMENTS VERIFICATION USING A MODEL BASED APPROACH

Dipl. Ing. (FH) Bernd Freyenberg-Richter
Dipl. Ing. Alexander Piechullek-Königer
Dr.-Ing. Thomas Sutor
T-Systems International GmbH
Ottobrunn, Germany

Abstract

Requirements verification can be established by the use of formal requirement description. From safety critical areas (e.g. flight control software) formal concepts emerged into other fields of application. Although formal languages deliver fail proof requirement verification due to their mathematical rigor, they suffer low acceptance. Especially huge learning curves for the non intuitive expression languages used for problem description render this approach difficult. In this paper the authors will compare the capabilities of formal requirement verification with a light weight model based approach. Our method offers easy handling in daily service. Because theoretic optimality properties as known from formal languages are hard to achieve and complicate usability, they will be considered in a limited way only.

Within the NH90 avionic project a method for effective and quick requirements verification was developed. Verification in this context not only investigates isolated requirements but moreover tries to cover the complex relationships between a set of requirements. As consequence the method is not restricted to bare requirements but also has to cover the functional breakdown or even system engineering aspects. In doing so a link to well used system engineering processes like DoD 2167 or the V-model can be established. As additional benefit testing staff is automatically familiar with this approach and does not need extra training.

As stated previously formal requirement description languages deliver fail proof verification results. No other method known to date has this capability. In comparison the limitations of our approach have to be considered. Originally the method was developed as rapid verification tool. In the meantime the set of covered requirements has been extended. The model also had to be refined which immediately unveiled maintainability issues. Another aspect is the complexity of the model which has to be kept within a reasonable size. These two topics will be explained in detail before the integration of the method into existing process and tool chains will be demonstrated. Finally possible solutions for further applications will be described briefly.

Requirements Verification Using a Model Based Approach

1. Importance of requirement engineering

The most important stage in the development of any real-time system is the generation of a consistent design that satisfies a mandatory specification of requirements. Almost always requirements are based on informal wishes expressed by one or several customers. Requirement engineering has to understand and specify the customer's needs through transformation from natural language to a suited domain. The latter is a critical activity since it is at the basis of a contractual relationship between customers and suppliers.

Four major kinds of requirement engineering activities can be distinguished [4]:

- **Elicitation:** collects information from customers
- **Modeling:** processes the collected information and maps the informal descriptions into concepts of a specification method
- **Analysis:** aims at the detection of problems in the requirements documentation
- **Validation:** controls the adequacy of requirement specifications

This paper mainly deals with analysis activities.

2. Formal requirement engineering methods

The specification language of choice must be expressive to support a straight forward mapping between the requirements collected from customers and the specification language concepts available. If the language is not expressive enough, extra artificial elements will have to be incorporated into the specification to capture requirements. These elements are not necessary from the customers view but are necessary to restrict the set of possible solutions at the implementation level. Verification and validation may become difficult, because reference to artificial elements introduced in the specification will have no meaning to customers. Any specification language should be a *formal* language built on *formal* semantics that give a precise meaning to all specifications written in the language. Because any syntax can be easily stated formally (using some form of backus-naur-form) most attention should be paid to the semantics. Semantics

based on imprecisely-worded prose render useless, because they will get subject of interpretation and misunderstanding. All semantic issues better shall be solved once for all and there will be no more critical phase in the system life cycle [6]. Useful products like counterexamples, failure scenarios, test cases, proof obligations, refinements, code fragments then can be generated automatically.

A formal specification serves as a single, reliable reference point for all requirement engineering activities. Especially mathematical/logical semantics will permit (automated) discovery of potential incompleteness or inconsistencies by

- Rules of deductive inference
- Theorem proving and model checking
- Abstract interpretation frameworks
- Decomposition

Unfortunately the tools supporting the formal analysis of requirements specifications are not ideal, because considerable knowledge and experience is still required to achieve a formal automated analysis [7].
A number of formal specification languages have been proposed for the purpose of describing software components. Older methods and design goals thereof can be found in [2]. Today the languages Albert II [1], Troll, VDM and Z gain the most support and attention.

2.1 An example for formal specification

Nevertheless both VDM and Z are not capable to deal with real-time issues, we will use a fragment of a specification written in Z to demonstrate automatic requirements verification in brief. The approach is similar for other formal specification languages. Z is popular with governments, academics and parts of industry, especially those developing critical systems where the reduction of errors and quality of software is extremely important.
The formal notation Z is based on set theory and predicate calculus and has been developed at the Oxford University Computing Laboratory since the late 1970's. Data is described by means of *predicate logic* which allows to describe the effect of each operation abstractly. In addition *schemas* which decompose the specification are used to describe both static and dynamic aspects. An international Z standardization effort was completed in 2002.

A Z specification document consists of interleaved passages of formal, mathematical text and informal prose explanation. The formal text consists of a sequence of paragraphs which gradually introduce the schemas, global variables and basic types of the specification, each paragraph building on the ones which come before it. Each paragraph may define one or more names for schemas, basic types, global variables or global constants. The scope of each global name extends from its definition to the end of the specification [5].

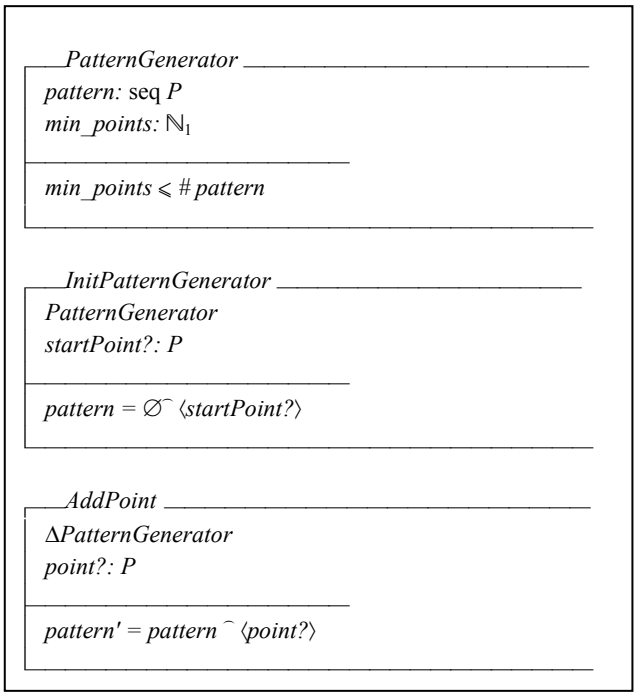

Figure 1: Fragment of a Search-And-Rescue (SAR) pattern expressed in Z.

The symbol [*P*] defines the set of patterns. There is neither a restriction of the pattern shape (ladder, expanding square, clover leaf) nor an assumption how to represent this type. Beginning with the next paragraph, the schema *PatternGenerator* specifies patterns as sequences. There is a predicate which demands a minimum number of points for the pattern. Initially the pattern only consists of an arbitrary start point as is expressed in schema *InitPatternGenerator.* Finally there exists a schema *AddPoint* which allows adding of points to the pattern. Additional schemas would refine the specification e.g. to only cover patterns of ladder form or would constrain the type point to geographical coordinates.

As a first example for requirements verification, expansion shall be applied.

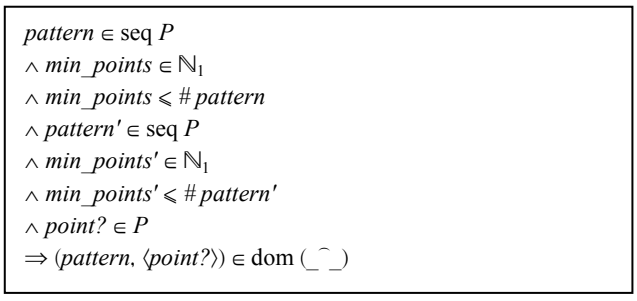

Figure 2: Inference through expansion.

All predicates required to add a point to the pattern obtained are stated in figure 2. Notably there are three deduced rules, which apply after a point was added to the pattern. Besides expansion, reduction and normalization also theorems can be proven in a explicit manner. Roughly spoken theorems provide formal spot checks.

$$start, A, B : P$$
$$\langle start, A, B \rangle \in \text{seq } P$$

$$Case1 \;\widehat{=}\; InitPatternGenerator[startPoint? := start]$$
$$Case2 \;\widehat{=}\; Case1$$
$$\fatsemi AddPoint[point? := A]$$
$$\fatsemi AddPoint[point? := B]$$

theorem evaluate
Case2

Figure 3: Sequential composition.

Results of sequences of operations can be investigated by defining a test case schema. Figure 3 shows, that the outputs of *Case1* are input to *Case2*. Thereby symbolic evaluation for theorems can be carried out. Figure 4 demonstrates the result obtained for the theorem *evaluate* in figure 3.

$$min\_points \in \mathbb{Z}$$
$$\wedge\ min\_points' \in \mathbb{Z}$$
$$\wedge\ pattern \in \mathbb{P}\,(\mathbb{Z} \times P)$$
$$\wedge\ pattern' \in \mathbb{P}\,(\mathbb{Z} \times P)$$
$$\wedge\ (\exists min\_points\_\_0 : \mathbb{Z}$$
$$\bullet\ (pattern = \{\} \frown \langle start \rangle$$
$$\wedge\ pattern' = \{\} \frown \langle start \rangle \frown (\langle A \rangle \frown \langle B \rangle)$$
$$\wedge\ \{\} \frown \langle start \rangle \in \text{seq } P$$
$$\wedge\ min\_points \geqslant 1$$
$$\wedge\ min\_points \leqslant \#(\{\} \frown \langle start \rangle)$$
$$\wedge\ min\_points\_\_0 \geqslant 1$$
$$\wedge\ min\_points\_\_0 \leqslant 1 + \#(\{\} \frown \langle start \rangle)$$
$$\wedge\ min\_points' \geqslant 1$$
$$\wedge\ min\_points' \leqslant 2 + \#(\{\} \frown \langle start \rangle)))$$

Figure 4: Conjectures after proofing.

Both the expected sequence $\langle start\ A\ B \rangle$ and additional properties are obtained by symbolic evaluation. A great part thereof deals with number spaces related to state or result variables. Besides the relation between minimum number of points and pattern elements unveiled conjectures which by sure would have been missed by manual analysis concentrating on the correct sequence.
Remarkably the used proofing tool was able to do symbolic evaluation, but was unable to simplify the pattern sequence.

2.2. Drawbacks of formal requirements engineering

As already mentioned most formal specification languages do not completely deal with the specification of real-time systems. This deficiency was recognized and influenced the development of Albert II. But the expressiveness of Albert II is so rich, that most system engineers will either have difficulties to understand the formal notation or will fail to extract the right subset of the notation that is proper for problem description. Apart from this formal methods have inherited well-known disadvantages from water-fall models, because they try to specify the complete system at any level in its entirety. And most important there is no doubt that structured or formal notation will not capture the requirements that the customer has failed to mention (irrespective that no known method will handle this situation).

Another important reason for not using a formal approach is the already presence of requirements. Nearly always a huge amount of knowledge is accumulated implicitly with the requirements and cannot be transferred or transformed easily. Besides decisions of the past which influenced choice of method or representation might even be of contractual interest.

3. Self developed model based approach

The NH90 avionics project suffers a similar situation. Since some time a set of requirements is engineered and maintained. Knowledge of staff is closely tied to the constituted form of the specification. The set of requirements describes complex functionalities of different categories at various levels of detail. Initially this specification was written in natural language. Unfortunately the attempt to migrate to a dedicated toolset failed. Because no additional funding could be raised and time was getting short, members of the formal qualification test team within the NH90 project at Eurocopter Germany decided to develop a light weight model based approach. A formal approach would have not been possible due to the stated constraints and therefore a lightweight model based verification was developed.

As an example for the model based approach again a Search-And-Rescue pattern is chosen. To prevent violation of secrecy with respect to the content of the requirements specification document (SRS) used, the information within this presentation is modified and obliterated.

3.1. Initial situation

The behavior of the Search-And-Rescue pattern is specified by a set of requirements within the SRS document. Mostly all of the functionality is expressed as pseudo-code (see figure 5). The correctness and

the consistency of the requirements was unknown and had to be proven. For this purpose a huge number of test definitions and test procedures had to be developed in short time.

```
-------------------------------------------------------------------
----              creation of the L A D D E R pattern            ----
-------------------------------------------------------------------
--
-- inputs required for Ladder:
--
-- TYPE      = Ladder
-- WIDTH     = usually the small side of the complete search area
-- SPACE     = distance between the legs 1,3,5,7...
-- DIRECTION = angle towards north of the long side of the area
-- SIDE      = left/right turn onto the second leg; default: RIGHT--
-------------------------------------------------------------------
--
-------------------------------------------------------------------
-- calculation of some constant values;
--    ..LAT, ..LON etc in deg+tenth of degrees
--    999 means: value is not defined / not to be used
--
[LADD_WIDTH/=999 and LADD_SPACE/=999]/
LADD_D:=LADD_WIDTH LADD_SPACE;; --length of transverse legs

-------------------------------------------------------------------
-- first corner position of the right turn Ladder pattern:
--
[LADD_SIDE= R ]/
LADD_LAT_CO(1):=LADD_CSP_LAT+LADD_DLAT1;
LADD_LON_CO(1):=LADD_CSP_LON+LADD_DLON1;;
-------------------------------------------------------------------
-- first corner position of the left turn Ladder pattern:
--
[LADD_SIDE= L ]/
LADD_LAT_CO(1):=LADD_CSP_LAT LADD_DLAT1;
LADD_LON_CO(1):=LADD_CSP_LON+LADD_DLON1;;
-------------------------------------------------------------------
-- second corner position of right and left turn Ladder pattern:
--
/LADD_LAT_CO(2):=LADD_C_1_LAT+LADD_DLAT2;
LADD_LON_CO(2):=LADD_C_1_LON+LADD_DLON2;;
-------------------------------------------------------------------
-- remaining corner positions of right and left turn Ladder patterns:
--
[PATX_TYPE= LADDER  and LADD_WIDTH/=999 and
LADD_SPACE/=999 and LADD_DIR/=999]/

…
```

Figure 5: Example for pseudo-code (obliterated).

3.2. Using a spreadsheet tool

Because the available pseudo-code can not be executed within the tool chain, a transformation of the specification becomes necessary. Consequently all available information was extracted from the SRS document and implemented in a spreadsheet. Only small modifications of the terms (see figure 6) were applied. To verify the correctness of the defined algorithms (e.g. parameters used for the generation of the pattern) an x/y-diagram was applied for direct visualization. In doing so, the implementation of special test algorithms which in turn would have to be proven to work correctly, can be avoided. Due to the fact that the displayed information of the diagram can only be seen as a rough guide to verify the validity of the parameters, algorithms to control the output should be added to the final solution of the verification model.

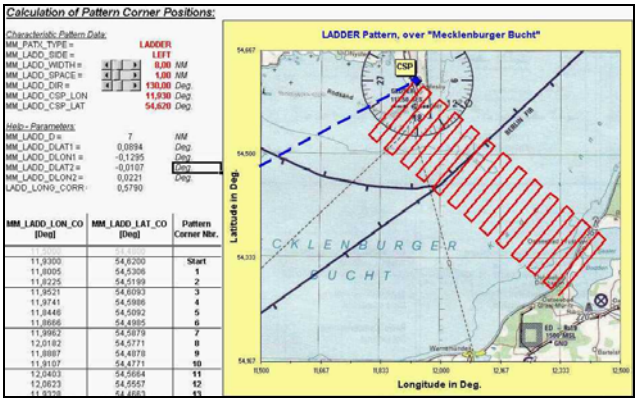

Figure 6: Snapshot of the spreadsheet solution.

3.3 Lessons learned from the spreadsheet solution

Using the spreadsheet approach as described in the previous paragraph, it became possible to verify the correctness of the specified algorithms and to improve them considerably in a straightforward manner. Several specification errors and inconsistencies were discovered at an early stage of the project. Because our systems engineering model is based on the waterfall-model, detecting these class of errors during later development would have a great impact on the time and cost schedule of the project. Therefore simply animation of the specification helped saving of cost and time. One major advantage is the fact that for this coarse solution only very low effort and standard applications are required. Approximately one man-month had to be spent to reach the status shown in figure 6.

The development of this verification approach unveiled both advantages and disadvantages to be considered.

Advantages:

- Complex algorithms can be easily verified with low effort.
- Discrepancies and errors can be found and solved at early stages and accelerates specification change management.
- Graphical visualization can be done with suitable spreadsheet diagram - types tailored to the application under verification.

Disadvantages:

- The implemented algorithms remain difficult to read and trace (now because of the spreadsheet pseudo-code).
- The structure of the methodology is static and can hardly be modified or expanded.
- The simulation of graphical input devices (e.g. Man-Machine-Interface, MMI) is not possible with this solution.

3.4. Upgrade of the spreadsheet solution in favor of a common programming language

In order to solve the first two disadvantages mentioned (readability and structure of method), it has been decided to use a common programming language. Preferably the language itself should be compatible to the spreadsheet application in order to allow comparison to our baseline approach.

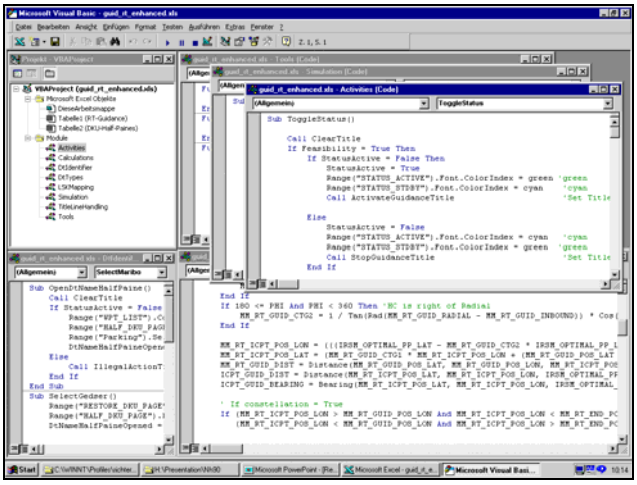

Figure 7: Snapshot of the common programming language modules.

The algorithms to be verified have now been taken nearly unchanged from the specified pseudo-code (directly from the SRS, not from the previous approach) and were translated into programming language modules. Also the backwards link to the spreadsheet solution was dropped completely because is rendered useless.

Only advantages of the approach are summarized, because the Man-Machine-Interface still has to be considered in the next paragraph.

Advantages:

- The algorithms to be verified and modeled can be taken nearly unchanged from the original source.
- By using modules it becomes possible to structure the model.
- The model can be maintained and expanded in a straightforward manner.
- Libraries and re-usable components become feasible.
- Cyclic references can now be implemented.

The use of modules and thus the structuring of specified and required algorithms turned out to be the greatest improvement with regards to methodology. Furthermore the tool became now capable of running dynamic simulations.

3.5 Development of an MMI simulation

Besides the verification of the specified algorithms of the Search-And-Rescue pattern, also the facility to test the behavior of the Man-Machine-Interface became evident. With help of both an MMI-simulation and requirements verification especially the interaction of functionality in conjunction with the MMI can be shown in a comprehensive way. It should be noted that the conditions and settings for the output on the MMI are not specified within the SRS but within another document. Due to this fact we again stress the importance of an integrated verification tool. Our experience shows, that the verification of requirements originating from arbitrary and different sources can become cumbersome and error prone if only isolated parts of the specification are considered. More elaborated models, which preferable should be run capable, promise simulation and verification of the functional chain and its entire properties from an integration viewpoint.

Figure 8: Snapshot of an approach with integrated Man-Machine-Interface.

In order to minimize the amount of human labor and time exposure a rapid prototyping solution emulating an input and output unit was added to the spreadsheet solution. By doing though, MMI specific aspects could be verified by visual inspection.
The example presented for the first implementation consists of an operational function, that is located on the target computer of the NH90. Mainly the “Display-And-Keyboard Unit (DKU)” has been modeled as MMI, using simple format features and control buttons (see figure 8).

Advantages:

- Model based requirements verification with simultaneous graphical simulation for rapid prototyping can be performed by using spreadsheets with low effort and low time budget.
- Manual analysis missed a lot of inconsistencies which were clearly apparent from our approach.
- Due to the dynamic simulation across different requirement sources even more inconsistencies were found.

- Algorithms verification can be performed sufficiently.

Unfortunately the possibilities for graphical representation are limited to the capabilities of the spreadsheet used and the modularity of graphical elements does not convince.

3.6 Final solution

Again the spreadsheet solution was somehow limited. Therefore a redesign of method for future extension has been carried out. As advantage customization and flexibility for new application areas are obtained.
Especially for the modeling of graphical interfaces utmost flexibility is required. Ease of handling, adherence to well-known interface standards and customer preferences are essential for successful deployment.

Figure 9: Snapshot of the final solution.

Further useful features like simulation control panels or state diagrams have also been implemented and have proven useful (see figure 9).

4. Conclusions

In the preceding paragraphs two different strategies - the formal requirement engineering method and the model based approach – have been introduced. Both cover the analysis phase of the requirement engineering process with different methodology. Analysis in this case is focused on the detection of problems or errors during the phase of specification of requirements.
Formal requirement methods enforce system engineers to create consistent specifications. As consequence a formal notation is precise and unambiguous. Thus the formal notation always provides the definitive description in the case of any misunderstanding. Using a formal notation increases the understanding of the operation of a system, especially at early stages. Thereby it becomes possible to reason about a system by stating and proving theorems about it. This provides a check that the system will behave as expected.
On the other hand formal techniques require a significant amount of training effort and practical experience to be applied successfully. E.g. nevertheless some specification languages are designed to be executable (although very inefficiently) there seems to be a general lack of convenient tool support and integration. Still insight and invention of staff is necessary, because there is no fool-proof methodology or magic formula that will automatically ensure a good or even feasible specification. Even if a system is proofed correct, there are still many assumptions which may be invalid. The specification must be ‘obviously right’. There is no way that this can be formally verified to be what is expected [3].

In contrast to the formal requirement methods the light weight model based approach seems to be a suitable way to cope with the analysis phase. Experience from use of the spreadsheet application showed, that it is possible to both verify algorithms and simulate man-machine-interfaces. This advantage gains importance, as customers are less interested in methodology discussions than rapid deployed results. The lessons learned that have been captured during the development of the model based approach can be summarized as the following statements:

- Errors can be detected in an early stage of the process.
- The handling is easy.
- Run capable models can be achieved very quickly.
- Rapid prototyping is possible.
- Exchange of data between different applications of the same supplier is possible.

5. Future prospects

As mentioned above both methods, the formal specification language and the model base approach, proved both useful during the verification process. The methodologies are mutually exclusive and can therefore be seen as complementary. Furthermore both methods could be used in a single project. For example the cost of proving a system correct may be justified in safety-critical systems where lives are at risk. Many systems are less critical, but could still benefit from formalization earlier on in the design process. Further development will have to consider a list of decision criterions in order to support engineers in future projects to find the best method.
Development of the model based approach is at the moment targeted towards tool integration. Especially the Functional Analysis Database, which holds

elements to support the requirements engineering process, shall be expanded.
Roughly spoken all requirements of the system and accompanying additional information shall be connected in order to support testing and formal qualification. Scattered information sources originating from different and separated documents have to be provided as single source. Users of the database will be able to access requirement specifications as original source (or any other inherited instantiation). Immediate verification of specified algorithms will also be possible from this entry point. Feedback can be written back in the form of comments. The creation and definition of single test steps intended to carry out requirement testing can also be done directly within the database. On that basis a complete integrated specification system especially suited for testing needs will be established. Nevertheless the described topics are rather requirement management tasks. The complexity of the NH90 avionics project demands an approach considering not only requirement engineering tasks. The work has already begun, but is still ongoing.

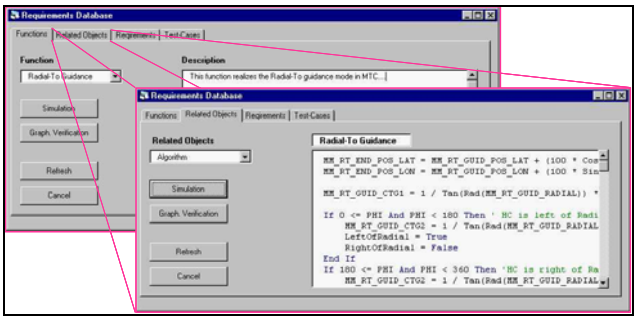

Figure 10: Snapshot of the Functional Analysis Database.

Acknowledgements

We'd like to thank our colleagues within the NH90 avionics project for many helpful discussions and suggestions. Also we'd like to thank Eurocopter Germany for allowance and support.

Literature

[1] Bois, P.D. (1995): *Semantic definition of the Albert II language.* Technical Report RP-95-007, Computer Science Dept., University of Namur, Namur (Belgium)
[2] Burns, A. and Wellings, A. (2001): *Real-Time Systems and Programming Languages (third edition).* Addison Wesley
[3] Davis, J. and Woodcock, J. (1996): *Using Z: Specification, Refinement and Proof.* Prentice Hall
[4] Dubois, E., Bois, P.D., and Zeippeny J-M. (1995): *A Formal Requirements Engineering Method for Real-Time, Concurrent, and Distributed Systems_.* Technical Report RP-95-004, Computer Science Dept., University of Namur, Namur (Belgium)
[5] Spivey, M. (1989): *The Z Notation: A Reference Manual.* Prentice Hall
[6] Xiashan, L., Zhiming, L. and Jifeng, H. (2001): *Formal and Use-Case Driven Requirement Analysis in UML.* Technical Report 23, UNU/IIST
[7] Zeippen, JM., Dubois, E., Bois, PD. (1998): *Supporting the Analyst when Reasoning on Requirements Specifications for Real-Time and Distributed Systems.* Technical Report RP-98-016, Computer Science Dept., University of Namur, Namur (Belgium)
[8] Information Technology - Z Formal Specification Notation - Syntax, Type System and Semantics, ISO/IEC 13568:2002, ISO