Validation within Safety Critical Systems Engineering from a Computational Semiotics Point of View

G. Doeben-Henisch
Department of Computer Science
FH Frankfurt am Main
University of Applied Sciences
D-60318 Frankfurt am Main
Germany
Email: gerd@doeben-henisch.de
Senior Member, IEEE

M. F. Wagner
Department of Computer Science
FH Frankfurt am Main
University of Applied Sciences
D-60318 Frankfurt am Main
Germany
Email: mfwagner@ieee.org
Member, IEEE

***Abstract*— Software controlled Safety Critical Systems are increasing in importance in all areas of application. Combining efficient agile development processes with powerful but complex modeling and formal methods imposes grand challenges on software designing organizations. Especially the semiotic dimension which relates stakeholders and engineers in an engineering process is evoking such challenges. Focusing on these processes as basis for domain modeling and as point of reference for validation the authors outline a simulation based approach to overcome the semantic gap. But they know, that this still can only be a preparatory step for much more detailed semiotic analyses.**

I. Introduction

A. Safety and Mission Critical Systems

Today most safety (SCS) and mission critical systems (MCS) rely on software as their most important but also most complex component. Advances in the design and manufacturing of hardware, e.g. electronic miniaturization, have enabled the spread of digital safety critical systems from military, aerospace and nuclear energy applications to all other technical domains, i.e. medical diagnostics and therapy, automotive systems, railroad systems, facility management, etc.. Further technological advances will produce more software intensive *High-Integrity Systems*. Safety is an emergent system property ([1]) requiring more stringent development methods than in other areas. The development of SCS is in many domains regulated by international and national standards (for an overview compare [2]). The increasing demands pose many challenges to the organizations developing software for SCS. In addition there is a need for efficient development processes to survive in a global economy. Agile development paradigms are being used successfully in many application developments. The use of modeling in various phases of the diverse software engineering process models represents another approach to cope with software and systems requirements analysis and the complexities of design. An increasing number of computer scientists believe that the time is now ripe for the use of formal methods in software development and that it is now possible and feasible to guarantee high-quality and safe software products ([3]). Although this expectation seems to be true to some extend, it becomes evident, that the semiotic dimension which relates stakeholders and engineers in an engineering process, is revealing difficulties which cannot simply be solved. In this paper we want to present some of these difficulties accompanied with some suggestions, how we possible can cope with them.

B. Assumed Core Elements of the SCS-Engineering Process

To be able to talk more precisely about the difficulties rooted in the semiotic dimension of the SCS-Engineering Process we have to define the *Core Elements* of the SCS-Engineering Process (cf. [1], [10], [11], [12], [6]) with their relationships to several known safety standards adding system safety requirements to such engineering processes (cf. IEC 61508 [4], RTCA DO-178B [5], [2], [7])

For our discussion we are focussing on the following elementary phases of an SCS engineering process: (i) Requirements specification, (ii) modeling a possible solution, (iii) implementing the model, and (iv) evaluating the implemented system against the requirements specification. Between this terminology and that used in the MDA-Approach ([6]) – which will be reviewed later in this paper– we assume the following approximate coupling: (i) CIM as requirements specification, (ii) PIM, PM, and PSM as modeling. The phases implementation and evaluation are not official parts of the MDA - approach.

Some more assumptions are necessary.

As one can see in Fig.1 we assume as *Core Actors* the stakeholder and the engineer. These have to clarify in a com-

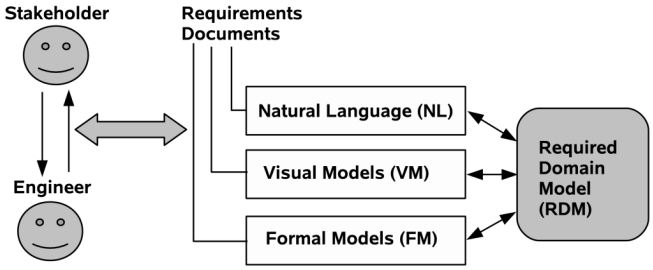

Fig. 1. Requirements Specification

munication process the requirements for the problem which has to be engineered. The communication process is supported by different kinds of documents. *Core Types of Documents* are those in *Natural Language (NL)*, those as *Visual Models (VM)*, and those as *Formal Models (FM)*. Whereas the relationship between NL-documents at one site and VM- and FM-documents otherwise can by semiotic reasons not easily be defined, we do assume a 1-to-1 mapping between VM- and FM-documents, which can be supported by computable algorithms.

Furthermore we do assume that the intended meaning of all these different kinds of requirements documents has to be *equal*. Although it is difficult to define this required equality precisely, it is clear, that without such an equality the set of all those documents will be inconsistent.

Because the term 'meaning' is a technical term, highly depending in its meaning from the presupposed theoretical framework, we have to state that we are assuming here a semiotic framework. Because the reference to semiotics as such is still very fuzzy –take for instance the authors *de Saussure*, *Peirce*, *Moris*, and *Hjelmslev* (cf. [15])– we will use here a basic formalization of the sign concept of Peirce, which I have proposed in [16]. Within his phenomenological approach Peirce is using a sign concept with three interacting moments, the *object (O)*, which is signified, the *sign vehicle (S)* as representative of the object, and the *interpretant (I)*; the *I* represents the cognition produced in the mind of a human system on occasion of *O*. This concept is dynamic insofar as the *I* can be selective with regard to the *O* and the *I* can change. In my formalization of this concept (cf. [16], p.127) I have introduced the grounding operator $grounding : O \times S \times Q \rightarrow I$ with *Q* as a set of qualities with $I, S, O \subseteq 2^Q$. Thus, we can state that a *sign* is technically speaking a minimal structure like $SIGN(x, p)\ iff\ x = \langle I, S, O, grounding \rangle$, where *x* is the *sign* and *p* that *person p* to which this sign belongs. The 'meaning' is then an inherent property of this structure. We could say that the 'meaning' of a sign *x* of a person *p* in the context of a certain *O* and *I* is *extentionally* the *O* –as long as the *O* is a direct *empirical* object without being again a sign vehicle *S*– and *intentionally* the *I* as that moment, which represents the extensional meaning of the sign vehicle *S* in the person *p*. When the *O* is itself a sign vehicle *S'* as moment of another sign *x'*, the concept of meaning can become more and more nested. We could define $meaningInt(x, p)_{O,S} = I$. Then, using this formalization we could represent the intentional meanings of the stakeholder SH and the engineer ENG (cf. Fig.1) as $meaningIint(x, SH)_{O_{SH},S_{SH}} = I_{SH}$ and $meaningIint(x, ENG)_{O_{ENG},S_{ENG}} = I_{ENG}$ We call these intentional meanings the *Required Domain Model (RDM)* (cf. Fig. 2).

In the context of engineering we assume that the extensional meanings $O_{SH} = O_{ENG}$ are equal with regard to the processes which have to be modeled. In practice are the objects *O* not purely extensionally but mixed up with sign vehicles nesting the possible extensional meanings in fuzzy ways. The minimal condition for an equality of the *RDMs* as the intentional meanings I_{SH} or I_{ENG} is, that the representing sign vehicles S_{SH}, S_{ENG} as natural language texts, as visual models or as formalized structures (cf. Fig. 2) are congruent. But even if we have a complete congruence $S_{SH} = S_{ENG}$ we have no guarantee, that this implies a congruence between the intentional meanings $I_{SH} = I_{ENG}$ too. Because the internal coupling of the *I* to the *O* and *S* is only implicitly given, we can never be completely sure that $I_{SH} = I_{ENG}$ or that $RDM_{SH} = RDM_{ENG}$. This describes the *semantic gap* of requirements modeling in software engineering.

The assumed content of the RDM is in this paper informally described as the *User* who is acting to solve some *Tasks*, then the *Environmental Conditions* which have to be taken into account, and the *User Interface (UI)*. We distinguish between two states of the *UI*: The user interface core (UIC) and the user interface shape (UIS). The *user Interface Core (UIC)* contains the logical substrate of all the assumed interactions between the user and the wanted system. This 'logical' concept of the UIC can be realized by different kinds of 'real' *User Interface Shapes (UIS)*. Furthermore one has to keep in mind that the relationship between a user, a system and the environmental conditions can vary through time.

The overall view of the assumed engineering process is shown in Fig. 3. The modeling activity in the second phase has to be in conformity with the UI. The requirements documents (NL,VMs,FMs) together with the UI and the Model are constituting the *Symbolic Space*. Only in this Symbolic Space is *Verification* possible! The mapping from the symbolic Model (PIM, PM, PSM) into a concrete running system is a transgression from the symbolic space into the physical world. This mapping is never 1-to-1, it can only be an approximation. The final system has then to be evaluated in the *Validation* procedure. Validation assumes the RDM as a valid point of reference and tries to compare the implicitly given RDM with the aid of the sign vehicles NL, VM, or FM to the *Implemented Domain Model (IDM)*. One gets the IDM out of the RDM by replacing the abstract concepts by real properties of a real running system. As long as the stakeholder did not see the real running system there can never be a complete clarity whether the RDM is really conforming with the running system in all its variations.

From this overall picture there arise many questions. Some of these we will discuss in this paper.

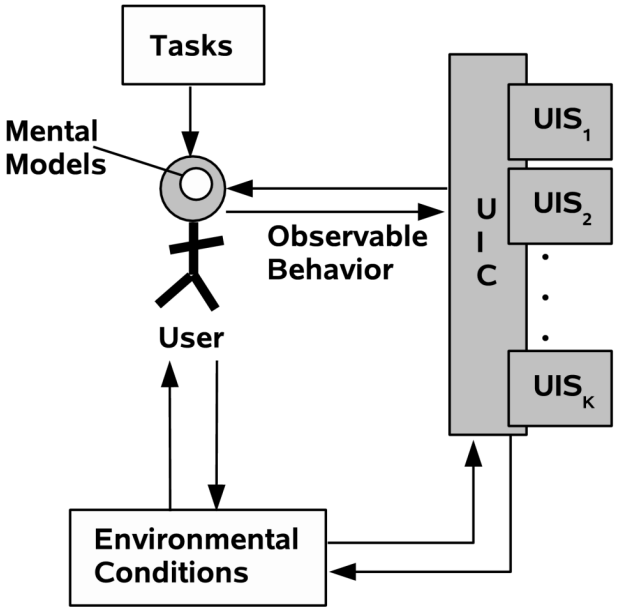

Fig. 2. Required Domain Model (RDM)

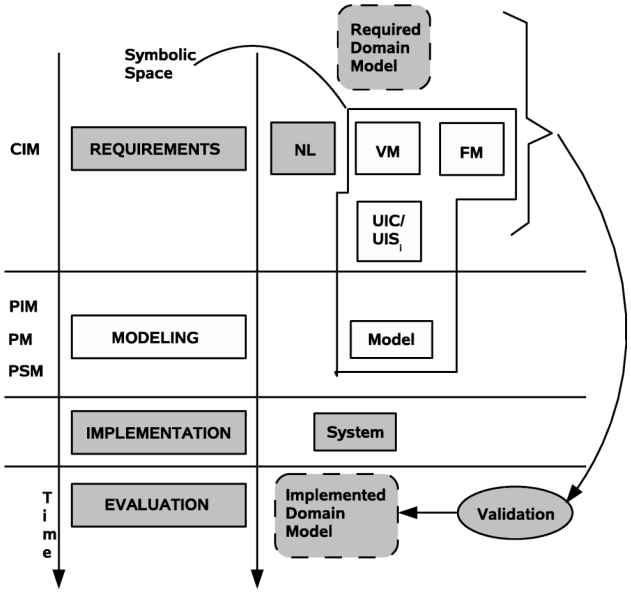

Fig. 3. SCS Core Elements

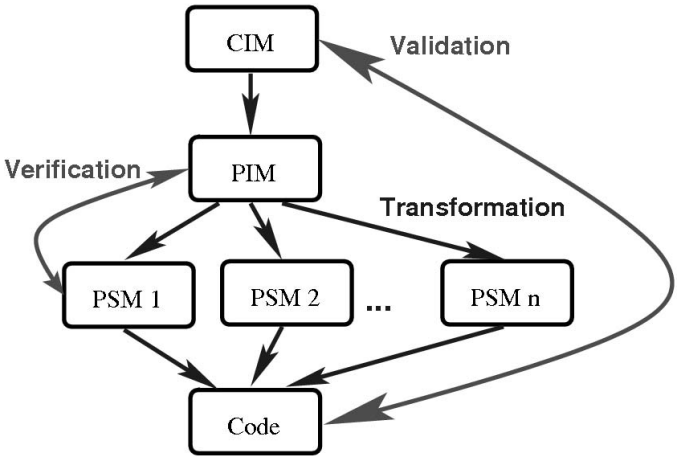

Fig. 4. Model Driven Architecture cf. [6]

C. Safety Certification

Development of SCS must be accompanied by a certification process demanding to fulfil the requirements of appropriate standards or guidelines (cf. [2]). This certification process must be incorporated into the system and software engineering process model. In modern approaches modeling on various levels gains importance. We will discuss as an example Model Driven Architecture (MDA) [6] and the difficulties for the validation of emergent system properties like safety.

II. Safety Standards

A. Overview

There are many safety standards, domain specific or generic, national and international standards (cf. [2]). The newer ones cover explicitly the development of the most complex system component, software. For our purposes the generic IEC 61508 [4] should serve as an example for the challenges to satisfy safety standard requirements in software development.

B. IEC 61508

IEC 61508 is a generic international standard that addresses the functional safety of systems, and primarily systems developed using electrical, electronic and computer technology. It is the base for the development of many SCS with a high software content. The framework for IEC 61508 is a safety life cycle model which is superimposed on a systems life cycle model (cf. [4]).

In addition to the normal complexities of software development system-level and domain specific safety requirements have to be traced through the whole software process. Therefore the software process model has to be enhanced in order to fulfill the traceability of the safety requirements (cf. [7] for a discussion of the merger of the IEC 61508 life cycle with the Rational Unified Process).

IEC 61508 places special emphasis on validation planning and validation as indicated in figure 2 of part 1 of IEC 61508 (cf. [4]).

III. Model Driven Architecture

The extensive use of modeling for the development of SCS especially in the context of the Model Driven Architecture ([6]) approach represents a new paradigm compared to standard methods in traditional process models.

Compared to other techniques MDA emphasizes the early phases in software engineering process models. The application and problem domains are being modeled using the

UML. MDA's vision is to generate code automatically from this visual model (VMD). MDA's approach is not altogether different from earlier approaches. The main difference is the goal to automate the transformations between CIM, PIM, PSM and Code. Figure 4 draws only a simplified picture of the possible transformations (model-to-model, model-to-code) of the MDA. For the development of SCS the transformations represent the crucial parts of the MDA. The incorporation, verification and validation of emergent and domain specific safety requirements (cf. 4) are subject to great research efforts ([8], [9]). For complex SCS with common safety goal the MDA is based on separate domains, each representing a specific perspective on the RDM, such as a specific body of theory related to the application or a technology. Safety requirements exist at every level of abstraction (CIM, PIM, PSM) Many safety requirements only emerge during the different analysis and design phases and can be considered as *derived* safety requirements. Therefore it may not be possible to state the safety requirements completely at the PIM level. For development projects based on MDA, it is necessary to consider the implications of the safety lifecycle as part of the process (cf. [8]).

A additional problem is the lack of tools to help in the process [13].

IV. Validation and Verification Problems

A. The Definitions of Verification and Validation

The definition of verification as well as validation varies greatly depending on the software engineering process model in use (waterfall model, spiral model [10], V-Modell XT [11], Unified Process [12], etc.) (cf. [19], [18]). Following [18] we assume here as key definitions the following ones: *Verification* is the task of checking that a model matches a given specification. A model is correct if it fulfills all the properties given in the specification. *Validation* is the process of checking whether the system or model behaves as the stakeholder expects it to. This includes the complete required domain model. It is important to notice, that in the framework, which we are assuming here, verification and validation are completely *independent* from each other. While a verification process can give a 100% error-free result, can the validation process nevertheless deliver some faults, and vice versa. Thus an engineering process has to handle both processes as *individual processes* which have to be kept separately. In this paper we will mainly focus on the validation process, some of it's problems and possible solutions.

B. Problems with Validation

From the definition of validation above follows that the expectations of the stakeholder (= I_{SH} = RDM_{SH}) are the point of reference with which the behavior of a real system (= IDM) has to be compared. We distinguish at least two important phases in the validation process: (i) *Adequacy of models:* The mapping of the intentions of the stakeholder (also called *mental models* or *cognitive models*) into sign vehicles like NL, VM, and FM, and (ii) *Correctness of system behavior:* the comparison between a real running system (= IDM) and the textual, visual, or formal models representing the RDM.

In the ideal case we have the stakeholder using the running system some time and then he judges that everything 'behaves well'. But this is not the normal case. In most real cases we have requirements documents describing a user and system behavior which is far too complex for a direct inspection by a human. The same holds for the test of the real system.

The difference between the intentions of the stakeholder (= RDM) and the possible meanings of the different kinds of (textual, visual, formal) representations is often called *Semantic Gap*. Especially if the representational structure is given as a formal language which is additionally encoded with the aid of some formal semantics, then is the meaning of such an representational device for a stakeholder nearly not understandable; even the software experts which have to build software based on such representations have generally great difficulties to reconstruct such formally encoded meanings.

On the other side exists the need for formalization to be able to cope with complexities and to prove essential properties of a model or a system.

V. Known Solutions

A. SpecTRM

Leveson et al. [20] developed a set of tools for application to process-control systems with the goal to lower the semantic gap between domain experts on the system level and software engineers. SpecTRM-RL (Specification Tools and Requirements Methodology - Requirements Language) is part of a specification framework that includes both formal and informal specifications. The blackbox behavioral requirements specifications are described using a control loop type front end, which has an underlying state-machine model (Mealy automaton). The control loop serves as a visual model (VM), whereas the state machine is the formal model (FM) in our language.

B. Safety Patterns

An approach using *Safety Argument Patterns* is proposed by [8], [21] that combines the need for consistent traceable arguments with the need to recognize separate domains, each of which contributes to emergent safety behavior. The concept is based on the idea that the safety argument patterns fit together to make up the safety argument in its entirety. The conditions of one pattern could be derived from the output of the preceding pattern. By this approach the safety argument is constructed coherently across the different domains and combined in the overall process. It includes clearly defined interfaces to reduce the problem of inconsistencies. Another advantage is the possible reuse of common structures of safety case arguments [22]. The patterns are defined using Goal Structuring Notation (GSN) [21] as a VM.

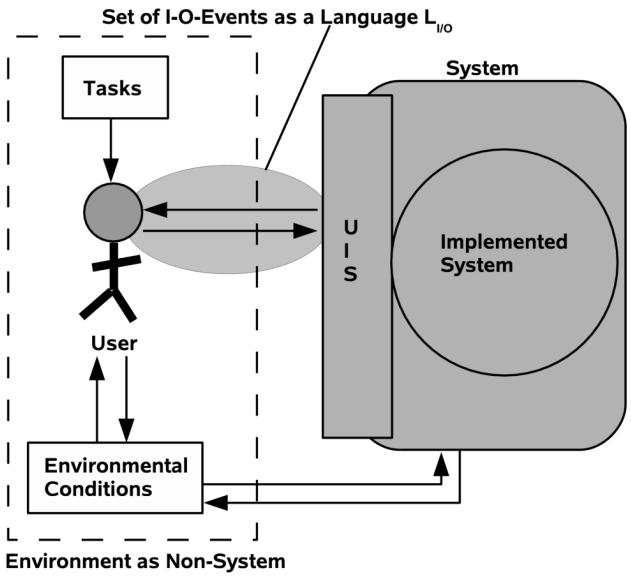

Fig. 5. Reformulating the IDM

C. Formal Methods for Safety

Bitsch [23] proposes the use of *Safety Patterns* to transfer domain expert knowledge and to enable them to check especially safety requirements for the SCS under consideration. Bitsch's approach uses a set of patterns for complete requirements. The patterns are explained in natural language, but expressed in formal logic. Different formal notations enable the use of model checkers for verification and validation of the requirements.

VI. Validation Revisited

Accepting that there is a real need for formal methods to cope with the complexity of the requirements and to be able to prove essential properties, one is left with the task to solve the problem how to bridge the semantic gap. Because the source of the semantic gap is the stakeholder with his typical way of understanding one has to find a solution which is in conformity with the way how the user understands.

The best known approach today to bridge the semantic gap is to extend the natural language description of the problem with *visual models (VM)* which can represent a RDM close to the way how a human experiences the behavior of objects in his world. The problem of the semantic gap would then be converted into the adequacy problem of the computed visual models. This we call the strategy of *adequate simulations*. The interesting question is, whether one could use adequate visual models for the correctness test too?

One possible strategy to do this is shown in Fig. 5.

One structures the whole domain in the implemented system S with the attached UIS at one side and all the other components (tasks, user, environmental conditions) as the Non-System, understood as the environment E of the system. What is left then are the possible interactions between E and S. One task is to translate all these interactions into the expressions of an input output language L_{I-0} which has to be described by a formal grammar G_{I-O}. If such a translation is not possible one has a first strong indicator that the whole set of interactions is not computable. If the grammar G_{I-O} can be constructed one has to model the environment E as an automaton A_E which can cope with the language L_{I-0}. A minimal condition is that the automaton is a turing machine. But because turing machines can correlate with complexity classes which are in practice not really feasible one has to try to find 'simpler' automata which can handle the task in an acceptable time (this implies that the corresponding grammar G_{I-O} itself has also a lower position in the Chomsky-Hierarchy for formal languages). A first *domain simulator (DSIM$_{I-O}$)* would then be a structure like $< L_{I-0}, A_E >$, where the automaton is the operator working on the language as set of expressions.

To prove the correctness of the IDM one has to compare the behavior of the IDM with the RDM. This can be done by applying the concept of the domain simulator DSIM also to the RDM. This implies that one has to construct the requirements models for the RDM accordingly. One approach would be to use for the visual models so called *state charts* (as described in the SysML-Standard, cf.[24]). One state in such a state chart would then represent a complete *situation* containing the user U, the actual task T, and the actual environmental conditions E. An action by the system S would be represented as an input event I to the state which will cause the state to change, usually paired with the generation of an output action O. Such an output action will again be an event which causes different system responses as new input events I. To represent this formally one needs a richer language L_{REQ} than the language L_{I-0}. This *domain simulator (DSIM$_{REQ}$)* would then be a structure like this: $< L_{REQ}, A_{REQ} >$.

In principle it is no problem to use DSIM$_{REQ}$ instead of the DSIM$_{I-O}$ for the correctness test too. On the contrary such an approach would improve the transparency of the correctness test because all assumptions about the user and his environment have to be made explicit. The same domain simulator could then be used for the adequacy and the correctness test within validation.

Furthermore one can map the state chart representing a complete situation into a directed cyclic graph (DCG) where every task T_i corresponds to one directed cyclic graph DCG$_{T.i}$ whose different paths correspond to the different runs of the automaton depending from the input I from the system S. We call such a graph a *task graph (TG)*. Whereas *repetitions (= cycles)* seem to be realistic, it seems to be unrealistic to assume tasks which are for a user not finitely solvable. Therefore we assume that the task graph for a certain system has only a finite depth. This does not exclude that there can be runs which are *conditionally cycling*. This represents the case where a user has to repeat some actions until he reaches a certain state which results in the satisfaction of a certain condition. Thus a task graph is a directed cyclic graph (DCG) whose processing is bounded to finite runs.

From the point of view of SCS engineering one is during

validation not only interested in the overall correctness of the behavior of the system but also especially in the property of *safety* (cf. [25]). In this context does the property safety mean, that 'bad things' do not happen during all the runs. This means the task graph does not show a certain bad property in all its branches.

For practical reasons one can think of DSIM_{REQ} also as a collection of simulators like DSIM_{REQ-T}, DSIM_{REQ-U}, and DSIM_{REQ-E}, or one is using only a domain simulator for the user DSIM_{REQ-U} and interfaces this simulator with T, E and the real system S. Another consideration is to build also a *system simulator (SSIM)* for supporting development, testing as well as –later on– for customer training.

VII. CONCLUSION

These considerations show how one can formalize and automatize the validation process. Ideally one should provide here also a review of all the known software tools supporting the idea of a domain simulator as described above. But this task has to be delayed for a future paper. The authors want only to mention that besides the well know Petri-Net Approach (see e.g. [17]) they are engaged in an open-source project called PlanetEarthSimulator (PES-Simulator) (cf. [26], [27]), which is targeting the idea of a domain simulator which is based on a visual modeling language representing horizontal and vertical networks of input-output systems which internally for the underlying automaton are represented as directed cyclic graphs.

REFERENCES

[1] Leveson, Nancy G.: *Safeware - System Safety and Computers.* Addison Wesley, Boston, MA, 1995

[2] Herrmann, Debra S.: *Software Safety and Reliability.* IEEE Computer Society Press, Los Alamitos, CA, U.S.A., 1999

[3] Woodcock, Jim: *First Steps in the Verified Software Grand Challenge.* IEEE Computer, IEEE Computer Society, October 2006, p. 57

[4] IEC 61508: *Functional safety of electrical/electronic/programmable electronic safety-related systems.* International Electrotechnical Commission, Geneva, Switzerland, `http://www.iec.ch`

[5] DO-178B: *Software Considerations in Airborne Systems and Equipment Certification.* Radio Technical Commission for Aeronautics (RTCA) Standard DO-178B/ED-12B, Dez. 1999.

[6] *MDA Guide Version 1.0.1* Document Number: omg/2003-06-01 Date: 12th June 2003 `http://www.omg.org/docs`

[7] Frederiksen, Rune: *Use of the Rational Unified Process for development of safety-related computer systems.* Thesis, Hogskolen i Ostfold Avdeling for infomatikk og automatisering, 2002

[8] Audsley, N.; Conmy, P. M.; Crook-Dawkins, S. K.; Hawkins, R.: *Safety Challenges for Model Driven Development.* In Metamodelling for MDA; First International Workshop; York, UK, November 2003

[9] Ehrlich, Alwina: *Model Driven Architecture fuer die Entwicklung sicherheitskritischer Systeme.* Diploma Thesis, FH - Frankfurt am Main - University of Applied Sciences, 2007

[10] Boehm, B.: *A Spiral Model for Software Development and Enhancement.* Computer, vol. 21, no.5, May 1988

[11] *Das V-Modell XT. Version 1.2.0* Bundesrepublik Deutschland, `http://ftp.uni-kl.de/pub/v-modell-xt/Release-1.2/Dokumentation/html/`, 2007.

[12] Jacobson, I., Booch, G., Rumbaugh, J.: *The Unified Software Development Process.* Addison-Wesley, 1999

[13] Houberdon, Jean-Louis; Babau, Jean-Philippe: *MDA for embedded systems dedicated to process control.* In Workshop on Model Driven Architecture in the Specification, Implementation and Validation of Object-Oriented Embedded Systems (SIVOEES), UML2003, San-Francisco, CA, U.S.A., October 2003

[14] Lutz, Robyn R.: *Software Engineering for Safety. A Roadmap.* in *The Future of Software Engineering*, Anthony Finkelstein (Ed.), ACM Press 2000

[15] Noeth, W., *Handbuch der Semiotik*, 2nd ed., renewed and extended, Stuttgart - Weimar: J.B.Metzler Publisher

[16] Doeben-Henisch, G. *Reconstructing Human Intelligence within Computational Sciences: An Introductory Essay*, In: Loula, A.; Gudwin, R.; Queiroz, J.; *Artificial Cognition Systems*, Eds., Hershey - London: Idea Group Publishing, 2006, pp.106-139

[17] Girault, Claude; Valk, Rüdiger (Eds.): *Petri Nets for Systems Engineers. A Guide to Modeling, Verification, and Applications*. Springer, Berlin - Heidelberg - New York, 2003.

[18] Moldt, D.; Kordon, F.: *Systems Engineering and Validation* In: Girault, Claude; Valk, Rüdiger (Eds.): *Petri Nets for Systems Engineers. A Guide to Modeling, Verification, and Applications*. Springer, Berlin - Heidelberg - New York, 2003, pp.405-415.

[19] Haddad, S.: *Introduction: Issues in Verification*, In: Girault, Claude; Valk, Rüdiger (Eds.): *Petri Nets for Systems Engineers. A Guide to Modeling, Verification, and Applications*. Springer, Berlin - Heidelberg - New York, 2003, pp.183-200.

[20] Leveson, Nancy G.; Heimdahl, Mats P.E.; Reese, Jon Damon:*Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future.* In Software Engineering - ESEC/FSE'99: 7th European Software Engineering Conference, Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Toulouse, France, September 1999. Proceedings

[21] Kelly, T. P.: *Arguing Safety - A Systematic Approach to Managing Safety Cases.* Dphil Thesis, University of York, UK, 1999.

[22] Kelly, T. P.; McDermid, J. A.: *Safety Case Construction and Reuse using Patterns.* In Proceedings of 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP'97), September 1997, Springer

[23] Bitsch, F.: *Safety Patterns - The Key to Formal Specification of Safety Requirements.* In: Proceedings of the 20th International Conference on Computer Safety, Reliability and Security, Springer Verlag Berlin Heidelberg, Lecture Notes In Computer Science; Vol. 2187, 2001, p. 176

[24] OMG-SysML, *OMG SysML Specification v. 1.0 (Final Adopted Specification) [May 2006].* http://www.omg.org/cgi-bin/doc?ptc/06-05-04 (Last visited March 21, 2007)

[25] Duttheillet, C.; Vernier-Mounier, I.; Illie, J.-M.; Poitrenaud, D.: *State-Space-Based Methods and Model Checking.* In: Girault, Claude; Valk, Rüdiger (Eds.): *Petri Nets for Systems Engineers. A Guide to Modeling, Verification, and Applications*. Springer, Berlin - Heidelberg - New York, 2003, pp.201-275.

[26] Doeben-Henisch, G.: *The Planet Earth Simulator Project - A Case Study in Computational Semiotics*, In: Proceedings IEEE AFRICON2004 Conference, 2004, pp.417-422.

[27] Doeben-Henisch, G.: *Reducing Negative Complexity by a Semiotic System.* In: Gudwin, R., & Queiroz, J., (eds). Semiotics and Intelligent Systems Development. Hershey et al: Idea Group Publishing, 2006, pp.330-342.