Towards a Unified Toolset for Embedded Systems Development

A.G. Bakhmurov[3] V.I. Chervin[2] M.V. Chistolinov[3]
J.F. Groote[4] V.A. Kostenko[1,3] R.L. Smeliansky[1,3]
D.V. Tsarkov[3] Y.S. Usenko[4] K. Winter[5] V.A. Zakharov[3]

[1] *Redlab Ltd.*
Moscow, Russia

[2] *State Research Institute of Aircraft Systems (GOSNIIAS)*
Moscow, Russia

[3] *Moscow State University (MSU)*
Moscow, Russia

[4] *Centrum voor Wiskunde Informatica (CWI)*
Amsterdam, The Netherlands

[5] *German National Research Center for Information Technology (GMD FIRST)*
Berlin, Germany

Abstract

This paper contains a comparative analysis of three toolsets and associated techniques for development of embedded systems. The comparison is based on the experience acquired by applying the techniques to a common case study. The results of the analysis are being used to design a unified set of tools for embedded systems development.
Keywords: *Formal Methods, Embedded Systems, Tool Comparison.*

The work has been carried out under the EU INCO-COPERNICUS Project No.: 977020 'DR TESY' (http://www.first.gmd.de/~drtesy)

1 Introduction

One of the most important distinguishing features of the modern technologies for embedded system development is the extensive application of formal methods at the earlier stages of the design process. This is inspired by increasing complexity of such systems and, hence, by growing development cost. One should notice that the most expensive stage of embedded system designing is testing, debugging and improving the system to satisfy the target requirements. Design errors discovered on testing and field trial stages yield sharp cost growth since the design should be re-done from the

beginning. But, the development cost is a crucial factor now. The solution is to use the technology and flexible tools for virtual prototyping of embedded system on the early design stages. The main requirements the technologies of this kind have to satisfy are as follows

- to provide the highest design flexibility,
- to minimize the cost of changes and analysis of design variants,
- to enable its user to build and analyze system prototypes rapidly,
- to decompose the system into software and hardware parts correctly,
- to perform full cycle of simulation,
- to develop and validate algorithms of operation,
- to implement programs,
- to estimate performance characteristics, information streams and resource requirements.

To measure the cost reduction efficiency in application of a new technology to the design of embedded systems one has to estimate how effectively the tasks listed above could be solved by means of its tools and techniques.

Due to these requirements, stated by industry, we resume our experience made with the use of three different kind of formal methods, applied to the case study of industrial relevance, namely a navigation system of airplanes. The methods used are μSZ (cf. [3]), μCRL (cf. [4]), and the language of the DYANA tool environment (cf. [5]). The results of our comparison should guide our further investigations in finding the best suited combination of the three approaches. That is, we are looking for the most advantageous feature of each language and working for an integration of these to end up with one method to be applied when developing embedded safety critical systems. Since all approaches appears to be complementary in their features this approach appears to be promising.

The experience made so far is related to the formal specification of the case study (the navigation system, cf. [1] and [2]). Each of the contributing partners provides a formalization in his/her own notation. Along these documents we are able to compare the different approaches or languages according to the following measurements:

- applicability
- expressiveness
- readability and documentation facility
- abstraction and refinement means
- solutions and relevance for the problems in the case study
- solutions for validation and analysis tasks, i.e. safety, liveness, real-time, fault-tolerance issues

- formal semantics.

We add the last item since we found out that the formal semantics of the different approaches is elaborated to different extent. Nevertheless, it is a crucial point for the development of further verification and validation support.

2 Comparison along the Measurements

For an overview we will start this section with a tabular that summarizes the good and weak points of each notation for each measurement. This is followed by explanation remarks for each of the judgments.

The given notation in the table is used as follows:

+ good support
+/- less good support
-/+ less weak support
- weak support.

	μSZ	μCRL	DYANA
Applicability	-/+	+/-	+/-
Expressiveness	-/+	+	+/-
Documentation Facility	+	-	-/+
Abstraction/Refinement	+/-	-	+
Solutions/Relevance for Case Study	-/+	-/+	+
Validation/Verification Support	-	+	-/+
Formal Semantics	+/-	+	-/+

2.1 Explanations for the Judgments

2.1.1 μSZ

Applicability The formal language is well suited for modeling of embedded systems on a high level of abstraction. The major weakness in applicability is that no explicit notion of time is available. At least for analysis timing issues are not supported.

Expressiveness μSZ comprises facilities for describing the overall architecture, the behavior of the components and their data structure and logical properties; the object-oriented approach allows the decomposition into several components (called process classes) which encapsulates certain behavior and data structure; it is sophisticated with inheritance and multiple instantiation of an object or process class; for communication between the class a broadcast mechanism is applicable. Implementation issues related to the design of the software to be intended are not sufficiently expressible.

Documentation Facility The notation yields a very good documentation facility by means of the graphical notation of Statecharts (which are part of μSZ); moreover, the object oriented approach allows to structure the specification; the overall method supports the notion of literate specification, i.e. the formal specification is interleaved with textual explanation.

Abstraction/Refinement μSZ supports the formalization on a high level only, i.e. implementation issues are kept abstract. However, there is an abstraction mechanism given through the notion of Statecharts: a state may be refined through another statechart. The main drawback in this abstraction mechanism is given by the fact that interdependencies might remain hidden between different layers of abstraction; it is very difficult to track those inter-level dependencies.

Solutions/Relevance for Case Study For the case study and their requirements at hand the notation yields a good documentation facility and supports the description on a high level of abstraction. Real time estimation or real time requirements can not be investigated within this approach.

Validation/Verification Support Although the given tools support test case generation the approach fails for the case study at hand. The navigation system is mainly concerned on the control flow that is to be specified by means of Statecharts. Since the test case generation operates on the data part (given in terms of Z) it is not applicable here. Other approaches for verification are not supported. For the validation we might rely on the documentation facility that yields a good survey over the whole architecture.

Formal Semantics The formal semantics for μSZ comprises the semantics of the sub-languages, namely Z and Statecharts. Although the formal semantics of Z is well defined and precisely specified by means of ZF set theory this does not hold for the formal semantics of Statecharts. For this graphical notation the formal semantics is determined by the (simulation) behavior of the tool that is used. This is a major drawback of this approach.

2.1.2 μCRL

Applicability The language μCRL was defined to describe interacting processes that rely on data [9]. The major design objectives for μCRL were that

- μCRL had to be so expressive that 'real life systems', generally consisting of a set of interacting programs, could be described
- μCRL had to be so simple and clear that it was suitable to form a basis for mathematical analysis
- the definition of μCRL had to be sufficiently precise to allow for the independent construction of computer tools for μCRL, capable in assisting in the actual development of systems.

Expressiveness According to this design goals of μCRL and the experience in specifying the case study, the language is generally applicable for specification of such kind of systems. Some parts of the case study were not possible to specify in the current version of μCRL. The extensive computations with real number arithmetics are hard to specify using algebraic specification techniques that are used in μCRL for data

definitions. Despite the research in this direction [11], it is considered to be a better approach to separate the computation part from the control and communication parts of the system during the early design phases.

The timing aspects of the case study were not specified in μCRL, as this is still an active research topic and the tool support for the timed version of μCRL is limited. However, the case study has helped to indicate the extra requirement for timing support in μCRL which deal with execution time and busy/idle time registration.

Other problems were due to the presence of priority and interrupt mechanisms in the case study, which are hard or not possible to encode in μCRL.

Documentation Facility The current documentation facilities of the μCRL toolset [10] are limited to the pretty printer to LaTeX. The μCRL language allows to write very structured, clear and extremely precise specifications, however the explanatory text must be provided to explain the system decomposition as well as data, actions and processes used.

Abstraction/Refinement The μCRL toolset currently does not provide any support for the stepwise specification development. The abstraction on the level of actions can be achieved by renaming some of them to the internal, non-observable action. As a refinement mechanism, action refinement techniques [8] can be applied by hand.

Solutions/Relevance for Case Study Generally, the presented case study is a one bulk unit that is hard to split into independent parts, and therefore hard to specify formally. The algorithms presented in the study are mainly sequential computational algorithms that are hard to model in μCRL. The communication protocols in the case study were either very compound and trivial, or just absent. Because of this, most of the errors that can be found in the system are under-specification or bookkeeping errors, while it is hard to imagine existence of conceptual errors, those that the verification in μCRL is primarily targeted on.

Validation/Verification Support Currently, the μCRL toolset supports the analysis of specifications which have finite models. It is possible to automatically generate finite labeled transition systems for such specifications. After that, existing model checking tools can be applied for the analysis of the transition systems. Some optimization tools allowing to reduce the size of generated transition system are available in the toolset.

The active research and development is going on now towards automatic symbolic verification and validation of μCRL specifications without generation of the entire transition system. At the moment symbolic techniques can only be applied by hand, which limits the size of the systems to which such techniques can be applied.

The μCRL toolset also includes a simulator, that can be used to find errors manually on earlier stages of specification.

Formal Semantics The formal semantics of μCRL specifications consists of static semantics, algebraic semantics for data types, and structured operational semantics for processes. All of this semantic definitions can be found in [9].

2.1.3 DYANA

Applicability DYANA is a programming environment intended for the design of embedded real time system (ERTS) architecture [7]. DYANA includes the subsystem for ERTS formal description and a number of subsystems for simulation, performance analysis and logical analysis of ERTS. The subsystem used for formal description of ERTS is based on the MM (Model with Messages) language [6].

Expressiveness By using DYANA model description language MM one can specify

- the hardware of ERTS, its structure and communications;
- timed characteristics of hardware components;
- the overall behavior of ERTS.

The model description language MM language supplies the hierarchy of data and message types, message exchange mechanism, modularity, natural number arithmetic, timing aspects of computations. MM is compatible with programming language C which is integrated in DYANA. By using MM language one can make a well-structured precise specification of the ERTS hardware, and describe communication, control and computation parts of ERTS software.

Documentation Facility DYANA supplies no documentation support; no graphical notation, e.g. flow-charts or state diagrams, is available for the moment. The description of ERTS model in MM language is presented as a set of text files augmented with some explanatory text as comments.

Abstraction/Refinement DYANA supports the top-down stepwise development of ERTS description down to the implementation level. MM provides both abstraction and refinement techniques by means of modularity of ERTS description, the hierarchy of message types, by choosing an appropriate level of details in describing the hardware and software components of ERTS.

Solutions/Relevance for Case Study

- DYANA provides
    1. adequate tools for the formal description of ERTS;
    2. pre-design the software that is to be constructed;
    3. simulation, visualization and performance analysis of ERTS behavior;
    4. the definition and verification of the logical requirements for software and hardware;
    5. design of the software in detail.

- DYANA partly supports:
  The coding and testing of basic program modules which are indivisible parts of a program that allows separate verification (e.g. ASM procedures).
- DYANA does not support:
  1. the integration and testing of fully implemented program components that are sets of program modules selected and united by functional or object criterion (e.g. ADA package).
  2. the integration and testing of program volumes that are sets of program components implemented for a given system board with CPUs, RAM, ROM etc.
  3. timed verification of ERTS behavior.

Validation/Verification Support DYANA supports a performance analysis in terms of time estimation by code execution. Moreover, the environment is sophisticated for logical analysis of the model behavior (this holds for program or models with more then 10^8 states (see [15])). But for the moment no real time behavior can be analyzed.

Formal Semantics The DYANA system is based on the formal model of distributed computer system developed in [12, 13]. The complete description of formal semantics of MM language is under the development.

3 Consequences for Future Work

In our comparison in the last section we can find that the formalisms have a complementary distribution of advantages and disadvantages. We may summarize the following advantages in the entire setting of all three formalisms we have at hand:

- We have a good documentation facility (given by means of μSZ).
- We have good skills for high expressive power (by means of μCRL).
- The most elaborated verification and validation support is given by μCRL.
- μCRL has a precisely defined formal semantics that allows independent construction of tools for the language.
- DYANA has a good support for abstraction and refinement that is necessary to develop a suitable design from a given formal specification.
- DYANA is well suited for the case study at hand and yields the best solutions for the problems arising within.

Combining these advantages coming with each formalism we get the following picture given with figure 1 (see next page) that sketches the intended way of integration.

Next steps for further development is already added to the picture. We refer to *test case generation from dynamics*, *logical analysis for DYANA*, and *timing analysis for μCRL*.

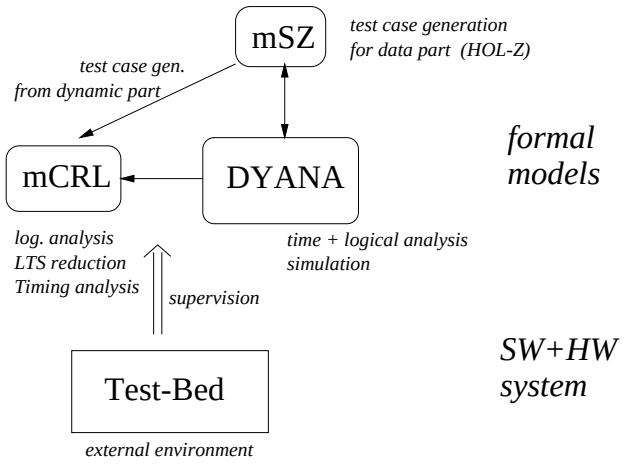

Figure 1: The given formalisms and their (current and future) facilities

Test Case Generation The given test case generation on the basis of the data part of a μSZ specification should be extended by a concept for test case generation derived from the dynamic part that specifies the overall control flow. Both facilities complement each other, and the entire mechanism will yield solutions for our problem at hand. Test case generation based on control flow is of particular interest for systems with hard real time requirements like our case study at hand.

Assuming a smooth transformation is given from one formalism into the other, we can exploit the most suitable foundation of a formal semantics for our concerns. Since μCRL provides a good and very well suitable formal semantics we will start our investigations on the basis of this formalism. Also, we can learn from other approaches that have already investigated test case generation on the basis of process algebras (cf. for instance [14]). We found that to be a good starting point.

Logical Analysis The logical analysis coming along within the DYANA tool is limited currently through the extend of system's state space. This is a well known problem for the overall model checking approach, called the *state space explosion* problem. To overcome this restriction the LTS reduction supported by the μCRL formalism should be facilitated within the DYANA tool. After transforming a given system specification into a linear form, a simple reduction algorithm is applicable. Thus systems' state space that have been too large will become smaller such that model checking can be applied efficiently. How to adapt this appealing facility for the DYANA modeling language will be the topic of our next investigations.

Timing Analysis The μCRL tool support is very well established. We found a wide range of well founded algorithms for analysis. Going to extend these range we are claiming that timing analysis is needed. The timing issue is newly added to the formal-

ism. Thus the tool support is not yet very well elaborated. Ongoing research for this issues will be guided by the given real time problems that arise within our case study at hand.

References

[1] *Informal Description of the Airborne Navigation System*, http://www.first.gmd.de/~drtesy/publications/IDNS/IDNS.doc, 1999

[2] *Report on Informal Specification*, http://www.first.gmd.de/~drtesy/publications/RIS.doc, 1999

[3] *Modeling the Navigation System in μSZ*, http://www.first.gmd.de/~drtesy/publications/FormSpec/NS-GMD.ps.gz, 1999

[4] *Formal Specification of the Navigation System in μCRL*, http://www.first.gmd.de/~drtesy/publications/FormSpec/NS-CWI.ps, 1999

[5] *The formal specification of the Airborne Navigation System by means of DYANA tools*, http://www.first.gmd.de/~drtesy/publications/FormSpec/NS-MSU.tgz, 1999

[6] *MM language for DYANA tool*, http://www.first.gmd.de/~drtesy/publications, 1999

[7] A.G. Bahmurov, A.P. Kapitonova, R.L. Smeliansky, *DYANA: An Environment for Embedded System Design and Analysis*, Proceedings of TACAS'99, Amsterdam, March 22-26, p.390–404.

[8] R.J. van Glabbeek and U. Goltz. Refinement of Actions and Equivalence Notions for Concurrent Systems. Technical report, Institut für Informatik, Universität Hildesheim, 1998.

[9] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes 1994*, pages 26–62. Workshop in Computing Series, Springer-Verlag, 1995.

[10] J.F. Groote and B. Lisser. Tutorial and reference guide for the μCRL toolset version 1.0. Technical report, CWI, 1999. To appear, available from URL http://www.cwi.nl/~mcrl/mutool.html

[11] P.H. Rodenburg and D.J. Hoekzema. Specification of the fast Fourier transform algorithm as a term rewriting system. Logic Group Preprint Series No. 27, Department of Philosophy, Utrecht University, December 1987.

[12] R.L. Smeliansky, *Distributed computer system operation model*, Moscow University Computational Mathematics and Cybernetics, 1990, N 3, p. 4–16.

[13] R.L. Smeliansky, *On program behaviour invariants*, Moscow University Computational Mathematics and Cybernetics, 1990, N 4, p. 54–58.

[14] J. Tretmans. Testing Concurrent Systems: A Formal Approach. CONCUR'99 – 10^{th} Int. Conference on Concurrency Theory, 1999, J.C.M. Baeten and S. Mauw, pages 46–65, Lecture Notes in Computer Science 1664, Springer-Verlag, 1999.

[15] D.V. Tsarkov and V.A. Zakharov, Efficient algorithms for the model checking in CTL and their application to the verification of parallel programs, *Programming and Software Engeneering*, 1998, v. 4, pp. 3-18.