NEW RESULTS AND TRENDS IN FORMAL TECHNIQUES & TOOLS FOR THE DEVELOPMENT OF SOFTWARE FOR TRANSPORTATION SYSTEMS — A REVIEW

Dines Bjørner
Computer Science and Engineering, Informatics and Mathematical Modelling
Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark
E–Mail: db@imm.dtu.dk, URL: www.imm.dtu.dk/~db

Abstract: We characterise what is meant by a metod in the context of software development. Then what is meant by a formal technique. We refute the possibility of formal methods, but express the desirability of formal techniques. Some such techniques are briefly surveyed. We will outline what has been done recently and what is currently being done using formal techniques in the area predominantly of railway systems. Problems are outlined, as are avenues for future research. Being an invited survey, the paper features an extensive, albeit far from complete, literature reference list of almost 180 entries, taking up half the paper size! There are no examples of formal techniques being actually shown in this paper — but there should be at least three papers (Pěnička et al., 2003; Strupchanska et al., 2003; Haxthausen and Peleska, 2003b) in these proceedings which illustrate such techniques as are the subject of the current review.

Keywords: Formal Method, Domain Description, Requirements Prescription, Software Design, Provable Correctness, Safety Criticality, Real–time, Embedded Systems, Interlocking

1. INTRODUCTION

Transportation systems pose extraordinary challenge when it comes to their monitoring and control by combinations of classical automatic control systems and digital computers.

1.1 Infra–Structure

This is in particular the case for rail and air systems due to their hard real–time characteristics combined with the need for very high dependability. In these kinds of infrastructure systems we see a need to integrate many diverse management planning, and operational execution, monitoring and control facets.

1.2 Sub–System Interfaces

Thus the challenge is compounded by the possibility, when using computers, of combining many diverse "sub-systems", sub–systems that, in the days of only combinations of classical automatic control system and human monitoring and control, were quite “separate”: Where information from one sub–system was basically only handed on to another sub–system via human intervention. Such human intervention often entailed data vetting: Is the information to be passed–on relevant and valid?

With automated interfaces, even within purely digital computer, ie., software, controlled sub–systems, the problem of "switching domains" is staggering, but enticing.
1.3 Hybrid Systems
Perhaps, from a scientific and engineering point of view, the most obviously interesting area of study is that of hybrid systems: These are not just combinations of continuous and discrete systems, but are such in which there is not just one, but several controllers — to use a standard terminology in automatic control theory. The software controls the decisions when to exchange one controller for another. Such hybrid systems have been studied at UNU/IIST, the UN University’s Intl. Inst. for Software Technology, Macau, nr. Hong Kong: (Wang et al., 1994; Chen et al., 1994b; Chen et al., 1994a; Yu, 1994a; Hung and Wang, 1994; Widjaja et al., 1994; Wang and He, 1995; Zhou et al., 1995). An interesting concept in this connection is ‘Hybrid Automata’ (Henzinger, 1996). Hybrid automata combine discrete transition graphs with continuous dynamical systems. They are mathematical models for digital systems that interact with analog environments. Hybrid automata can be viewed as infinite-state transition systems, and this view gives insights into the structure of hybrid state spaces.

1.4 Structure of paper
The topic of this invited paper was suggested by the Programme Chair. In effect they chose the title! It therefore behooves me to explain some of the terms of the title such as they are understood in computing science and software engineering. We therefore first explain — to an audience usually associated with the field of automatic control — what is meant by a method, its principles and techniques; when such techniques can be based on mathematics, including notably mathematical logic.

2. ON TECHNIQUES & TOOLS

2.1 What is a Method?
By a (software development) method we shall understand a set of principles for selecting a number of techniques and tools for the study and solution of problems — in the form of the construction of an artifact (here: Software).

2.2 Impossibility of Formal Methods!
The method principles amount to criteria for selection and application: When and what to choose. Such principles can not be automated. Problems under investigation are usually too complex and “never the same”. So the choice has to be done by the developers. Hence cannot be formalised. Unfortunately the term ‘Formal Method’ has stuck.

2.3 Desirability of Formal Techniques & Tools
But many techniques can be formalised, and tools can be provided for the support of such formal techniques. These techniques apply to oftentimes very large scale engineering documents, formally specified, and far too large to be analysed by humans. It is therefore desirable to use such formal techniques and tools — since they may help ensure, amongst others, correctness of software with respect to likewise formally prescribed requirements.

2.4 Examples of Techniques & Tools
We take specification languages, and correctness (of software or hardware) verifiers and model checkers to be tools. By techniques we then mean the specific way in which the developer performs ‘calculations’ (i.e., development steps), including applying these tools. Verifiers are software packages that either assist the developer in conducting proofs of correctness or which perform such proofs more or less automatically. Model checkers are also software packages which symbolically — almost “exhaustively” — tests the software (hardware), while subject to usual computer interpretation, enters only desirable states. Certain kinds of compilers from domain specific language scripts are tools that transform specifications into semantically consistent executable systems.

2.5 Why Formal Techniques?
There are several, and in the mind of the current author, fully equivalent, good reasons for why one should apply formal calculi (i.e., techniques) in the pursuit of computing systems development: Usually one is mentioned as being the most important one: Correctness of software — with respect to requirements (i.e., that “the software is right”). To this we add that “it is the right software”, i.e., that it affine to users expectations. Finally: “it is fun, it is professionally satisfying” to use formal techniques.
2.6 Convincing the Skeptics

The subject of so-called Formal Methods, is, strangely, to the current author — who is one of the “pioneers” of the field (within software) — still somewhat “controversial”. So a number of popularising papers have been and are being offered: (Wood, 1990; Wing, 1990; Thomas, 1992; Bowen and Stavridou, 1992; Bowen and Stavridou, 1993; Bowen, 1993; Rushby, 1993; Bowen and Hinchey, 1994; Butler et al., 1995; Cleland and MacKenzie, 1995; Hinchey and Bowen, 1995; Kelly, 1995; Liu et al., 1995; Rushby, 1995; Kelly, 1996; Caldwell, 1996; Kelly, 1996).

These papers explain why developers might very well wish to use formal methods in the development of software. (Rushby, 1993; Kelly, 1995; Rushby, 1995; Kelly, 1996) explain very well NASAs position on the need for formal verification of safety critical on-board software.

3. SOME TECHNIQUES & TOOLS

In this section we overview, ever so briefly, some of the formal techniques that have shown effectiveness in solving problems — also in the domain of railway operations and management.

The possible distinction between a method, like ASM, B, RAISE or VDM, a specification language, like CSP, RSL, VDM-SL or Z, or a technique cum tool, like SPIN — for all of these see below — has here been deliberately blurred.

The next section will then comment on specific uses of these formal techniques.

We have ordered the presentation of the techniques chronologically: In the approximate order of their publication.

3.1 Petri Nets

Petri Nets are a two-dimensional, ie., a graphic, yet formal notation for expressing concurrent behaviours and true simultaneity. Leading book references are: (Jensen, 1997; Reisig, 1985; Reisig, 1992; Reisig, 1998). The nets are named after their “creator” Carl Adam Petri (Petri, 1962).

3.2 VDM-SL

VDM stands for the ‘Vienna (software) Development Method’. VDM-SL stands for the VDM Specification Language. It was researched and developed at the IBM Vienna (Austria) Laboratory in the early 1970s and can be said to have offered first comprehensive formal techniques for general software development. VDM-SL is now an ISO standard (Larsen et al., 1996).

VDM basically offers model oriented, ie., discrete mathematics means of specifying and reasoning about software. Major texts are (Bjørner and Jones, 1978; Bjørner and Jones, 1982; Fitzgerald and Larsen, 1997).

The author of this paper was one of the co-designers of VDM.

3.3 CSP

CSP stands for ‘Communicating Sequential Processes’. Put forward by Tony Hoare in 1978 (Hoare, 1978) CSP has become one of the leading means for specifying, succinctly and elegantly, the interaction between parallel processes. Leading books on CSP are: (Hoare, 1985; Roscoe, 1997; Schneider, 2000).

3.4 Z

Z derives from the Z in Zermelo, who, as a mathematician, together with Frankel, established the Zermelo-Frankel axiomatic basis for a set theory.

Proposed around 1980 by Jean-Raymond Abrial, Z has become one of the leading model oriented, ie., discrete mathematics means of specifying and reasoning about software. The Z literature is abundant, but we refer only to the delightful text book: (Woodcock and Davies, 1996).

3.5 Statecharts

Statechart, primarily put forward by David Harel in the mid 1980s, and supported by the Statemate tool set (Harel and Naamad, 1996), is a two dimensional graphics, ie., a pleasant visually oriented way of presenting concurrent behaviours by the (“similar”) behaviour of compositions of modularised sets of finite state machines. Statechart is a “feature” offered by UML.

A leading text book is: (Harel and Politi, 1998).

3.6 HOL

From the HOL home page we quote: “The HOL System is an environment for interactive theorem proving in a higher-order logic. Its most outstanding feature is its high degree of programmability through the
meta-language ML. The system has a wide variety of uses from formalizing pure mathematics to verification of industrial hardware. Academic and industrial sites worldwide are using HOL. The system is available without charge.” Leading texts are: (Gordon and Melham, 1993)

3.7 Isabelle
From the Isabelle home page\(^3\) we quote: “Isabelle is a popular generic theorem proving environment developed at Cambridge University, England (Larry Paulson), and at the Technical University of Munich, Germany (Tobias Nipkow).” Isabelle is strongly related to HOL. There is a forthcoming book on Isabelle (Nipkow et al., 2002).

3.8 ccs
ccs stands for ‘Calculus of Communication Systems’. Put forward by Robin Milner (Milner, 1980; Milner, 1989) ccs provides a mostly theoretical framework for studying, investigating and experimenting with concurrent behaviours. ccs is reminiscent, but, in most respects, independent of CSP.

3.9 RAISE
RAISE stands for Rigorous Approach to Industrial Software Engineering. RAISE, with its Specification Language RSL, is a derivative of VDM, incorporating algebraic semantics (scheme, class, object) structuring constructs and CSP. Leading texts on RAISE are (George et al., 1992; George et al., 1995). RAISE can be claimed to be an object-oriented (i.e., OO) language.

The author of the present paper instigated RAISE in the mid 1980s. He is now launching a major text book of software engineering using RAISE: (Bjørner, 2004).

3.10 PVS
From the PVS home page\(^4\) we quote: “PVS is a verification system: that is, a specification language integrated with support tools and a theorem prover. It is intended to capture the state-of-the-art in mechanized formal methods and to be sufficiently rugged that it can be used for significant applications. PVS is a research prototype: it evolves and improves as we develop or apply new capabilities, and as the stress of real use exposes new requirements.” Leading people “behind” PVS are John Rushby and Natarajan Shankar. Seminal manuals are: (Shankar et al., 1993; Owre et al., 1999a; Shankar et al., 1999; Owre et al., 1999b).

3.11 B
B “derives” from the name of the group of set-theory oriented French mathematicians: Bourbaki. Put forward by the “father” of Z, Jean-Raymond Abrial, B offers utterly elegant means, within again a model-oriented simple, but reasonably abstract (imperatively oriented, as is Z) to specify and notably reason about — and, by means of strong tool support, to formally prove — properties of designs.

The leading text book is (Abrial, 1996).

3.12 ASM
ASM stands for Abstract State Machines. Put forward around 1985 by Yuri Gurevitch, ASM offers operational, some would say algorithmic, ways of specifying and reasoning about software. ASM provides so by means of a state transition system notation. States are algebras. Interpretation of ASM specifications leads to a notion of evolving algebras. A leading European “behind” ASM is Egon Börger. The literature on ASM is abundant. Leading books (incl. proceedings) are: (Börger, 1995; Gurevich et al., 2000; Gurevich et al., 2000; Börger and Stärk, 2003; Börger et al., 2003).

3.13 SPIN
SPIN is a reachability analysis tool designed for the general verification of distributed systems. First made available publicly in 1991, SPIN is widely used both for teaching and for industrial applications, and has inspired many other verification tools. In April 2002 the tool was awarded the prestigious System Software Award for 2001 by the ACM. The originator of SPIN is Gerard J. Holzmann. Leading texts are: (Holzmann, 1991; Grégoire et al., 1997; Holzmann, 2003).

\(^3\) www.cl.cam.ac.uk/Research/HVG/Isabelle/
\(^4\) http://pvs.csl.sri.com/
3.14 Duration Calculi (DC)

Of several notations for describing temporal (i.e., real–time) properties of systems, we single out the Duration Calculi. The main proposer of the DC was and is Zhou Chao Chen, but see (Zhou et al., 1991). DC offers a continuous time temporal logic for specification and reasoning. DC has a number of variants for dealing with probabilistic phenomena, for incorporating first order differential calculi in DC expressions, etc. Leading papers are: (Zhou et al., 1991; Hansen and Zhou, 1992; Zhou, 1993; Liu et al., 1993; Zhou et al., 1993; Zhou and Li, 1994) — with (Zhou and Hansen, 2003) being a monograph on the DC.

3.14.1 HyTech

HyTech (Henzinger et al., 1997a; Henzinger et al., 1997b) “is a symbolic model checker for linear hybrid automata, a subclass of hybrid automata (Henzinger, 1996) that can be analyzed automatically by computing with polyhedral state sets. A key feature of HyTech is its ability to perform parametric analysis, i.e. to determine the values of design parameters for which a linear hybrid automaton satisfies a temporal-logic requirement.” A leading person “behind” Hybrid Automata and HyTech is Tom Henzinger.

3.15 \( \pi \)-Calculus

The \( \pi \)-Calculus, like ccs, both put forward by Robin Milner, is a reasearch vehile for studying systems whose behaviour can conveniently be understood in terms of a varying number of processes and channels. Processes can interface over dynamically varying channels. The \( \pi \)-Calculus is not intended as an industry–oriented ‘technology’. Main texts are: (Milner, 1999; Sangiorgio and Walker, 2001).

3.16 Remarks

Unlike the natural sciences — where the phenomena studied are manifest, and created by The Almighty God — in the computing sciences, as in mathematics, we deal with concepts conceived by humans. As a result we see, as illustrated above, a plethora of notational systems, diagrammatic and textual. Each reflecting a didactics, a mind–set specific to the time at which the specification language was first proposed — with many such concepts transcending into a long future.

For the natural sciences and its derived engineering branches (civil [building], mechanical, chemical and electrical & electronic engineering), the major notational system of analytic expressions and the major calculi of differentia and integral calculi pervades and have become “standards”. No–one would employ a “classical” engineer who was not thoroughly familiar with that mathematics and those calculi.

Alas, this is yet to happen for software engineering!

There are may other formal techniques and tools than those briefly surveyed above. Some will be mentioned in the next section.

4. RECENT WORK

In this section we shall go a little bit into actual applications of formal techniques in connection with railway systems. The present section offers but a mere glimpse. In no way does this section claim to be comprehensive. More, it is a reflection of what the current author has encountered and felt intrigued and/or inspired by.

4.1 FME Rail Workshops

Dr Peter Gorm Larsen, a former student of the current author, initiated a three year, I should say, rather successful, EU sponsored, collaboration (network): 1997–1999.

Proposed through the “offices” of the FME (Formal Methods Europe\(^5\)) association, FMERail brought practitioners in the railway industry together with researchers from that industry as well as from academia at five workshops: (Larsen, 1998; Woodcock, 1998; Fahlé, 1998; Montigel, 1999; Lecomte and Larsen, 1999).

Many of the references below derive from these workshops. We refer to www.ifad.dk/Projects/FMERail/fmerail.htm\(^6\).

4.2 A Survey

The survey is ordered by the alphabetic name either of the specific formal technique or tool being predominantly used in the referenced applications, or — in a few cases — by the application subject. Many papers listed under the name of some technique or tool could, as well, have been listed under

---

\(^5\) www.fmeurope.org/

\(^6\) The author hopes this web page stays alive for some more years.
some application area. This is particularly true for **Interlocking** — as very many papers indeed study that subject.

**ASM:** (Börger et al., 2000) A report on the use of ASMs at Siemens AG (from May 1998 to March 1999) to redesign and implement the railway process model component of FALKO, a railway timetable validation and construction program.

**B:** Using B (Guiho and Mejia, 1984; Dehbonei and Mejia, 1994a; Dehbonei and Mejia, 1994b; Dehbonei and Mejia, 1995) reports on what must be considered one of the most successful and spectacular uses of formal methods. The application is that of the software to automatically control high speed urban trains in Paris, France. More than 80,000 lemmas and theorems were proved, using the B tool set Atelier B, in order to gain confidence in the correctness of the specified design. “**Using the B Method to Design Safety–Critical Software Systems for Railway Systems**” provides an easy overview.

**Category theory:** In (Roanes-Lozano et al., 1998) the authors put forward a very interesting use of category theory to investigate, by means of some AI techniques, railway interlocking.

**ccs:** In (Morley, 1991; Morley, 3 4; Morley, 1996; Morley, 1997) Morley uses ccs to investigate the well-formedness of the signalling data, i.e., the information about rail nets relevant to the switching of rail points, and also studies the use of such data in actual interlocking.

**Galois Theory:** In (Ingleby and Mitchell, 1992a; Ingleby and Mitchell, 1992b; Ingleby, 1994; Ingleby and Mee, 1995) Michael Ingleby uses classical predicate logic, and, excitingly, also Galois Theory (Ingleby, 1995), to investigate rail net structures for the purposes of structuring proofs of correctness of interlocking schemes. Ingleby proposes to decompose nets into such components that together satisfy a Galois Connection criterion — in that way many proofs carry over as lemmas in Galois Connection structured theorems.

**Constraint Logic Programming:** Jimmy Lee and his colleagues apply constraint–based logic programming methods (Chiu et al., 2002; Chiu et al., 1996) to help the Hong Kong MTR (Metropolitan Transit Railway Corporation) schedule train traffic.

**CSP:** In (Simpson, 1994; Simpson et al., 1997; Simpson, 1998; Woodcock and McEwan, 2002) CSP is used, together with the CSP oriented model–checking tool FDR, to engineer verified train protection and interlocking systems for the British railway infrastructure. To the present author this work is seminal.

**Duration Calculi (DC):** In (Zhou and Yu, 1994; Yu and Zhou, 1994; Yu, 1994b) DC has been used as a means to study scheduling and stability issues of train traffic. DC is slated to be far more used for these purposes than hitherto reported.

**Formal Testing:** In (Peleska and Siegel, 1996; Peleska, 1996; Peleska, 2002a; Peleska and Tsiolakis, 2002; Peleska, 2002b) formal theories are being established for the actual testing, including test case generation, of safety–critical designs. This work is done for various railway operators (and for the aerospace industry) in Europe.

**Interlocking:** The switching of rail points (switches, point machines, turn–outs) in groups, from station entry to platform or siding track, is clearly of major safety–related concern: It is also a typical real–time problem that can be computerised. To do so is studied and practiced intensely — as is evidenced by many of the above, and later on below, citations.

In (Culliver and Wai, 1990; Wong Wai, 1991b; Wong Wai, 1991a; Culliver and Wai, 1993) Wong uses graphic means to study interlocking. (Holzbacher et al., 1997) uses graph grammars. (Bernardeschi et al., 1996) studies state explosion problems. (Petersen, 1997; Borálv, 1997) apply...

---


8 [http://www.atelierb.societe.com/other_papers/english/using_B/using_B.htm](http://www.atelierb.societe.com/other_papers/english/using_B/using_B.htm) (French by Pierre Desforges; translated into English by André Danne.)

9 [www.hkcrystal.com/hiking/mtrkcrmap.htm](http://www.hkcrystal.com/hiking/mtrkcrmap.htm)

Stålmarck’s (model checking) approach, using Prover\textsuperscript{11} technology to prove properties of interlocking schemes. (Eriksson, 1997b; Eriksson, 1997a) pursue very similar ideas. (Jackson, 1998) applies process algebraic notions and uses the CMU model checking tool SMV\textsuperscript{12} to engineer British Rail interlocking schemes.

**Petri Net:** It cannot surprise anyone, given the graphical nature and purpose of Petri Nets that it has found widespread use in modelling railway system issues. A large variety of applications can be reported: From studies of railway topologies ((Montigel, 1992; Montigel, 1994)), and interlocking ((Basten et al., 1994; Basten et al., 1995; Billington and Janczura, 1996)) incl. deadlock avoidance, railway stations ((van der Aalst and Odijk, 1995)), via studies of simulation of railway control systems ((zu Hörste, 1999)), and models of train movement ((Decknatel, 1999)), to test case generation for interlocking ((Casaza et al., 1999)), liveness test ((Giua and Seatzu, 2002)), and even an education project ((Berthelot and Petrucci, 2001)).

**PVS:** In (Skakkebæk, 1994) DC is used, amongst others, to study safety–criticality of railway–road level crossings. In the study models of a DC proof system has been built using PVS.

**RAISE:** Since the current author, besides being one of the originators of VDM also instigated the development of RAISE, it can come as no wonder that we shall also survey the use of RAISE in connection with railways.

In (YuLin et al., 1994) a Chinese MSc student investigates issues of railway station management. In (Björner et al., 1999a; George, 1996) issues of train traffic (global, resp. distributed ‘Running Map’–based) scheduling are studied.

In these proceedings (FORMS2003) (Pěnička et al., 2003; Strupchanska et al., 2003), a part result of the EU IST Research and Training Network AMORE on Algorithmic Methods for Optimising Railways in Europe, two problems are analysed: What it means to re–schedule train carriages for regular maintenance (service), respectively allocation of railway staff to trains (staff rostering).

In (Björner, 2000; Björner et al., 2002) the current author suggest a foundation for how to model railway net topologies, respectively the principles and techniques for such domain modelling — irrespective of requirements and actual software design, but as precursors for those development phases.

In (Björner, 2003) the present author continues the line of (Björner, 2000) and attempts a study of the dynamics of railway nets.

I now come to a series of papers which I believe will be trend–setting:

(Haxthausen and Peleska, 2000; Lindegaard et al., 2000; Haxthausen and Gjaldbæk, 2003; Peleska et al., 2000; Haxthausen and Peleska, 2002; Haxthausen and Peleska, 2003b; Haxthausen and Peleska, 2003a) (the last three are treated below, in paragraph Domain Specific Languages).

(Haxthausen and Peleska, 2000) concerns the formal development and verification of a distributed railway control system using RAISE. The idea is to start with a domain model of static and dynamic aspects of railway networks, then the safety requirements are defined in terms of that and finally the control system is stepwise developed and verified to satisfy the safety requirements. The RAISE model and verification is generic wrt. the network topology.

(Lindegaard et al., 2000; Haxthausen and Gjaldbæk, 2003) concerns the use of RAISE for the formal modelling and verification of interlocking systems for stations and lines, respectively, at the Danish Railways. These papers build on the same methodological ideas as (Haxthausen and Peleska, 2000), but the control systems are quite different.

**Domain Specific Languages:** (Peleska et al., 2000; Haxthausen and Peleska, 2002; Haxthausen and Peleska, 2003b; Haxthausen and Peleska, 2003a) \textsuperscript{“}concerns a development and verification method for railway/tramway control systems based on domain–specific descriptions. The work described in these papers extend previous methodological ideas by providing a domain–specific specification language for railway/tramway control systems. The idea is that for each control system to be developed, application–specific parameters are specified in a domain-specific language, and from this specification a control system is automatically generated and verified to be
safe. The control components automatically generated from the domain–specific specifications are specifications of the rules of a state transition system that is made executable by a generic interpreter technique. Hence, we generate “executable models”. (Haxthausen and Peleska, 2002) (extends (Peleska et al., 2000) with more info) focuses on the domain–specific language, (Haxthausen and Peleska, 2003b) on the automatic generation of control systems from domain-specific descriptions and (Haxthausen and Peleska, 2003a) on the verification and testing issues” — ends the quote.

### 5. FUTURE RESEARCH

Future research is sometimes based on current problems. Some such problems can perhaps best be undertaken in the context of ‘integrating’ two or more formal approaches. Some such research may be undertaken in the form of a “Grand ("Man–on–the–Moon") Challenge”. The next three subsections deal with the previous three sentences!

#### 5.1 Technical/Scientific Problems

**Problem 1:** The foremost pressing current technical/scientific problem seems to be that most realistic software developments need combine two or more techniques (languages etc.). Where, for example Petri Nets or Statecharts have proven very useful for expressing concurrency and transitions, there is no easy “other” formalism in which to express the “contents” of the transition actions. Not one whose semantics and proof rules “fit” the graphics of either of the two techniques just mentioned.

Where, for example RAISE, with RSL, has been used successfully to express action “contents”, concurrency and synchronised communication ("rendez–vous") proposals have now been made to supplement this with language constructs for expressing temporal properties: Timing and durations (a la DC): (Xia and George, 1999; Haxthausen and Xia, 2000).

The above, the foremost pressing current technical/scientific problem, is general, not specific to the railway domain.

**Problem 2:** The secondmost pressing current technical/scientific problem is, in my, undoubtedly prejudiced mind, that the very many otherwise sound approaches to the formal treatment of railway problems do not build on a common understanding of what a railway systems is.

One easily runs the risk that one, say a train scheduling algorithm’s developer’s conception of a railway net, differs substantially from that of a signalling engineer’s conception — with the possible result that automatically generated reschedulings

---

are at odds with “corresponding” interlock schemes, and can be so fatally!

In the engineering based on the natural sciences there are such common domain understandings. These have been and are being provided by their “back–up” science: Physics (mechanics (as from Kepler and Newton), electricity), chemistry, etc., and have been hundreds of years under way.

In Section 5.3 we put forward a proposal for a joint, world–wide “Grand Challenge” project that aims at providing a theory of [railway]\textsuperscript{14} transportation!

5.2 Unifying Theories of Programming

Similar such “integration” of two or more “formal methods” have been and are increasingly being proposed. It seems that a general semantics framework for combining notations have been put forward by Tony Hoare and he JiFeng in (Hoare and Feng, 1997).

Applications of the Hoare/He concept of ‘Unifying Theories of Programming’ are appearing, for example, in (Butterfield and Woodcock, 2002; Woodcock, 2002; Cavalcanti et al., 2002; Sampaio et al., 2002; Woodcock and Hughes, 2002).

5.3 A Project Proposal

We propose that researchers in university computing as well as transport engineering departments together with scientists and engineers at railway infrastructure providers and at train operators go together around the following possibly 6–10 year joint R&D project: One that establishes a set of commensurate, finely interface “tuned”, formally, as well as precisely, but informally, described models of a conceptual railway system, i.e., “the railway system”. By a conceptual railway system we mean one that designates a class of actual railway systems, that is one to which each of the actual railway systems around the world relate in precisely described ways.

Such a conceptual model would include precisely harmonised descriptions of such “sub–systems”, cf. Section 1.2, as for example: (1) statics and dynamics of railway net topologies (Bjørner, 2000; Bjørner, 2003), (2) time table creation based on passenger statistics, (3) scheduling and rescheduling of trains (George, 1996; Bjørner et al., 1999a), (4) train maintenance (Pěněčka et al., 2003), (5) crew rostering (Strupchanska et al., 2003), (6) train composition and decomposition (along train routes, and according to seats reserved and to statistics) (Karras and Bjørner, 2002), (7) passenger and freight seat, resp. space inquiries, reservation (and for freight also tracing), sales etc., (8) railway net development (downsizing and upgrading, net maintenance, etc.), and much more.

We emphasize that we are searching for a model of the railway domain as it is, not as it should be. That is: The “advertised” domain models can then serve as a basis for — hence “normalised” — requirements to computing (monitoring, control & communication) systems. From the requirements one can then develop software. And, provided the software is correct wrt. the “normalised” requirements, we shall conjecture that it is significantly simpler to make sure that otherwise independently developed software is easy to fit safely and securely together!

We cite (Bjørner et al., 1999b; Bjørner et al., 1999c) as a pair of technical reports that suggests domain, respectively requirements models.

We refer to www.imm.dtu.dk/~db/train/train.html, and www.imm.dtu.dk/~db/train/train.ps for HTML and postscript documents which, although a few years old, outline details of such a project — called TRAIN, for: The RAilway INfrastructure project.

Care to join?

6. CONCLUSION

So what can we conclude? We formulate the conclusion in the form, first, of questions or conjectures, and, subsequent to all questions, to part answers, respectively comments: (0) What are the trends — in summary? (1) There is a need for using formal techniques, not only for safety–critical, real–time and embedded software–based systems, but for any related software — also for railways. But who is taking care of the need? (2) There are a plethora of formal techniques (cum methods, languages and tools) available — and that poses, perhaps, a problem: Which ones to choose? (3) Among this multitude: Which one are “The Winners!”? (4) Are these formal techniques being taught sufficiently at universities? (5) And are these formal techniques being accepted, adopted and adapted by industry?

---

\textsuperscript{14} the “Grand Challenge” project easily “carries over” to other transportation modes.
6.1 The Trends

**General:** The trend is towards increasing replacement of classical control equipment with such which is predominantly controlled by software. Such software need, it is increasingly being mandated — by the transportation system regulatory bodies — to be shown correct by means of formal techniques. The trend is furthermore to compose such software components into larger systems wherein functionalities spreading across the entire railway planning, development, operation and maintenance spectrum “deliver” data to one another. The complexity of such, in the past, rather mundane, i.e., “down-to-earth” applications, thereby increases “exponentially” — furthermore calling for the use of highly professionalised, accredited and certified software houses, support software and software developers.

Along general lines a trend 'Transformation Systems' (Peleska et al., 2000). They transform specifications into semantically consistent executable systems. If transformation has been proven, model checking or theorem proving is no longer or only needed for validation of the generated software.

**International Standardisation:** The European CENELEC norm requires that applications involving safety with safety-critical integrity levels be implemented using formal techniques. This applies also to railways.

**Specific & Personal:** The current author — being who he is — cannot refrain, given the opportunity, to point out the trends that he himself, with many colleagues around the world, are pursuing. Using such approaches as are designated by B, CSP, VDM, RAISE, and Domain Specific Languages, in order to achieve trustworthy, first abstract, subsequently more concrete models of domains, then requirements, and finally software designs. We refer to the paragraphs above on B ((Guiho and Mejia, 1984; Dehbonei and Mejia, 1994a; Dehbonei and Mejia, 1994b; Dehbonei and Mejia, 1995)), CSP ((Simpson et al., 1997; Simpson, 1998; Woodcock and McEwan, 2002)), VDM ((Hansen, 1994b; Hansen, 1994c; Hansen, 1994a; Hansen, 1996; Hansen, 1998)), RAISE ((Bjørner, 2000; Bjørner et al., 2002; Pěněk et al., 2003; Strupchanska et al., 2003; Bjørner, 2003; Haxthausen and Peleska, 2000; Lindegaard et al., 2000; Haxthausen and Gjaldbæk, 2003)), Domain Specific Languages ((Peleska et al., 2000; Haxthausen and Peleska, 2002; Haxthausen and Peleska, 2003b; Haxthausen and Peleska, 2003a)) —and the text accompanying these references. **Formal Testing** along the lines of (Peleska and Siegel, 1996; Peleska, 1996; Peleska, 2002a; Peleska and Tsiolakis, 2002; Peleska, 2002b) goes hand-in-hand with the above.

6.2 Caring for the Need

Thus the need for using formal techniques will increase significantly. Those software houses which can demonstrate professionalism in this area will simply replace those which cannot. Already now we see the emergence of a number of European software consultancy & design houses specialising in providing formal techniques–based software to the transportation industry.

6.3 The Multitude

It is too early to give definitive and unique advice on which formal techniques to deploy. Surely, for highly concurrent systems Petri Nets have shown great use. But CSP and RAISE, to mention two examples, can also serve well here — they lack the appealing graphics of Petri Nets, however, so an integration, a “unification”, might be desirable. For such concurrent systems which are furthermore highly reactive, using Statechart in a software design stage seems most reasonable. For major parts of actual domain, requirements and software development, any of the B, VDM, RAISE or Z approaches will do. HOL, Isabelle, PVS, and SPIN can, and should be used in conjunction with several of the above techniques and tools — it being noted that there are today reasonably powerful theorem proving assistants or automation provided for B, CSP, RAISE and Z.

6.4 “Winners ?”

Time will tell! We are simply far too short into the era of professionally sound, industrially scalable formal techniques and tools. Man will not stop thinking up profoundly new didactic bases for software development.

6.5 University Education & Industry Take–up

An increasing number of graduate students are now being offered courses at most European universities in which one or more of
the techniques and tools covered in this paper play a substantial rôle. There are, however, in my mind, a triplet dichotomy: (i) Students of software engineering very often “know better” than the lecturer/scientists, and, claiming that industry is not using formal techniques, avoid these courses. (ii) The computer science department staff increasingly turn to research in exactly the area of formal techniques. And (iii) increasingly we see the emergence of now dozens of small software houses, all over Europe, industries whose main livelihood depends on their using formal techniques and tools. So the students seem to become “losers”. They look at the large software houses—which will, eventually die out because of fossilisation: They missed the boat on professional, ie., responsible software engineering.

6.6 Closing Remarks

We have provided a brief survey with a long list of supporting references. The references of Section 3 were predominantly to leading books on their subject, while the references of Section 4 were to papers illustrating the application of formal techniques and tools to railway problems.

7. ACKNOWLEDGEMENTS

The author acknowledges the immense benefits he has had from technical/scientific interactions with Søren Prehn and Chris George over the last more than 25, resp. more than 18 years.

8. BIBLIOGRAPHY

The bibliography is extensive — typical of a survey paper. But it is approximately half of the number of entries culled, some years ago, during the FMERail project mentioned in Section 4.1.

We refer to www.imm.dtu.dk/~db/fmerail/fmerail and www.imm.dtu.dk/~db/fmerail.ps for a 340 item literature list

References


15 www.imm.dtu.dk/~db/documents/2ifacpaper.ps


on Abstract State Machines, Monte Verita, Switzerland, Local Proceedings, number 87 in TIK-Report. Swiss Federal Institute of Technology (ETH) Zurich.


YuLin, D., Bjørner, D., and Prehn, S. (1994). Domain Analysis: A Case Study of Railway Station Management. Technical Report db/03/01, UNU/IIST, the UN University’s International Institute for Software Technology, P.O.Box 3058, Macau; E-mail: library@iist.unu.edu. Presented at KICS’94: The Kunming (Yunnan, PRC) Intl. CASE Symposium, Nov. 1994.


Desirability of Formal Techniques & Tools

But many techniques can be formalised, and tools can be provided for the support of such formal techniques. These techniques apply to oftentimes very large scale engineering documents, formally specified, and far too large to be analysed by humans. In April 2002 the tool was awarded the prestigious System Software Award for 2001 by the ACM. The originator of SPIN is Gerard J. Holzmann. Leading texts are: (Holzmann, 1991; Grégoire et al., 1997; Holzmann, 2003). For software developers, trends tend to indicate where the industry is headed. New technologies, development methods and customer preferences can be key indicators regarding where you should focus your development skills. Staying on top of new technology trends is a great way to future-proof your software development skills, ensuring your abilities don’t age as poorly as a Korn CD. From retail to healthcare, the future of software development “lies within algorithms, big data and sophisticated AI tools to draw new and fascinating conclusions. Read also: Ciklum’s AI Experts Release A Scientific Study On Brain Activity. Trend 2: Cross-Platform Development Tools. A software review is “A process or meeting during which a software product is examined by a project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval”. In this context, the term "software product" means "any technical document or partial document, produced as a deliverable of a software development activity", and may include documents such as contracts, project plans and budgets, requirements documents, specifications, designs, source code