FORMAL SOFTWARE TECHNIQUES FOR RAILWAY SYSTEMS

Dines Bjørner

Department of Information Technology
Technical University of Denmark
DK–2800 Lyngby, Denmark
E–Mail: db@it.dtu.dk

Abstract: We discuss a rôle for the use of formal techniques in understanding the **domain** of railways, in expressing **requirements** to, and in the **design** of trustworthy software for the backbone and program packages for railways. We exemplify such techniques as applied to the recording of our **domain** understanding and hints at their use in **requirements** and **design.** We motivate the use of formal techniques, and survey, ever so briefly, the state-of-affairs of such use.

1. A MOTIVATION

We see the rôle of formal techniques (of software development in the context of transportation systems) from two composite viewpoints: The sociological and socio-economic, and the computing science and software engineering viewpoints.

1.1 *Sociological Motivation*

The domain of railways was, for perhaps a hundred years, characterised by stable staff. Once hired into a railway company, the employee remained there for life. And by not so rapidly changing technology. Manually thrown switches were in use for more than a hundred years. Electro-mechanically operated switches for at least fifty years. Now staff "drifts" in and out of the domain, old loyalties have gone; new staff has to be continually retrained; and staff which is not kept vigilantly to the same operating routines, day after day, easily forget them. Rail and train technology changes more rapidly than the now much shorter employment-time of staff. And computers and communication have come to stay.

Do they, computers and communication, offer a relief from the problems of migrant staff and itinerant technologies ?

Yes, but !

The knowledge that loyal, life-time staff learned, through training, was predicated on that staff getting that knowledge re-inforced, again and again, year in and year out, by "drill-work". That knowledge was usually presented in master class: The experienced staff verbally taught the new staff.

Computing has to take heed and embody that knowledge. We need to codify that knowledge. To write it down. In precise ways. i In both humanly understandable form, and in forms that can be manipulated by machine.

The technology of former days was based on natural sciences: Switches were mechanical, then electro-mechanical, and are now seemingly electronic — or are they ? The technology of today — computing — is not based on natural sciences. It is based on mathematics and on the "man-madeness" of infrastructure components: transportation, etc.

It takes time for this view to prevail.

In this paper we wish to indicate a way in which the knowledge of for example railways can be "codified": Written down in a form that is guaranteed to last beyond the current fashions of object-oriented, UML-like models. Namely in good, reliable, old-fashioned mathematics. Not the math-

ematics, however, of analysis (differential equations, integrals, statistics, etc.), but a simpler, more readily accessible mathematics: that of discrete mathematics, categorical algebras and, notably, logic.

In this paper we wish, briefly, to illuminate the changes that are ingerent in computing. These are changes that will profoundly affect age old institutions like railways — and, for that matter, any transportation domain.

1.2 *Engineering Motivation*

1.2.1. *Some Delineations* Computer science is the study and knowledge of the phenomena that can exist inside computers: Data and processes. Computing science is the study and knowledge of how to construct those devices: Data and processes. Software engineering is, to us, the triptych of engineering understandings of the domain of applications — void of any reference to computing (*&c.*), of engineering requirements to software for computing applications in the support of operations of the domain, and of engineering the design of that software.

By a method we understand a number of principles, deployed [here] by engineers, to select and apply, a number of techniques and tools in order efficiently to analyse and efficiently to synthesize efficient [as here] software. No one method suffice for a full application. Some techniques of some methods can be formally based and for some of these formally based tools can be made available. The term 'formal method' should really only be understood as 'formal technque cum tool'. Some methods span phases, stages and steps of development (viz. VDM [2], [20], [11] and Raise [12,13]). Other methods cover just a single aspect of a single step (viz. STeP [22], [23], Petri Nets [19], [24]).

1.2.1.1. *Domain* models initially express the very basics of their domain: This paper will illustrate such an (i) *intrinsics* — void of any reference to how that domain may be instrumented: How it is done so through (ii) *supporting technologies,* (iii) *rules & regulations,* (iv) *human behaviour,* etc. Domain models go on to then capture (ii–iii–iv–...).

Another paper of these proceedings, [21], illustrate, from the perspectives of both domain and requirements, issues of, amongst orhers, modelling the *support technology* of the group interlocked control of rail switches.

1.2.1.2. *Requirements* usually reflect a triple of concerns: To secure (a) support of domain operations (the *domain requirements*), (b) the *interface* between humans (and/or other interfacing quantities) and the hardware/software to be built (the *interface requirements*), and (c) performance, dependability, maintainability, platofrm and other "ilities" issues (the *machine requirements*).

We shall not cover this facet in the current paper.

1.2.1.3. *Software design* finally deals with what is normally considered the domain of software engineering — which in this paper is thus argued to encompass also the more novel domain (and the, since some time, well-established requirements) engineering aspects. *Software architecture* design implements the domain and parts or all of the interface requirements, and *program organisation* implements machine requirements. Further step of software design focuses on specific platform issues and it is here that modularisation, the use of standard, today typically object oriented packages (CORBA, etc.) enters the design process.

Throughout the development process "reams and reams" of documents are being constructed: Informal briefs, scope & span synopses, rough sketches, analyses, narrative and terminologies, as well as formal documents: formalisations of the narratives and verification of properties. Validation is the act of securing, with stake-holders of the domain that domain and requirements models to a highest, if not fullest, extent, satisfy human, hence informal, expectations. And throughout the development process, one or another construction method is being applied.

1.2.2. *Promises of Better Engineering* Domain engineering, as a separate development phase, addresses, typically the *"separations of concern"* problem of engineering. Formal techniques, with their use of and facilities for abstraction, have made it possible to capture the domains of indeed very large scale applications.

This modern software engineering, now being applied, systematically and rigorously, to major components, viz. transportation systems, health care systems, financial service indusry, etc., of societal infrastructures, enable us to embark upon and tackle the development of very large scale software systems whose thousands of packages cover, in the end, highly interwove, highly interrelated, communicating and synchronising domain phenomena.

So far only software can do that: The "one step" applications of automatic control, or of operations analysis technques, cover only well defined "package" parts where computing is able to link "it all"

together. Automatic control is, classically based on classical mathematical models, say in the form of differential equations, being first used to model what can be modelled by such mathematical tools. Similar for operations analysis. The technques and tools of automatic control and operations analysis are indeed very profound. Computing science has only begun to sratch the surface of its profoundness. But computing can do what the others can't: Link the various "packages" together. In doing so, computing does not, as of yet, guarantee that there is a formal, mathematical understanding of what that "linking" means. The problem is, however, being under intense study. In general it could be labelled the problem of codesign, but currently that label is used in a much more narrow sense: The co-design of hardware & software to solve an isolated, "small" problem.

2. A RAILWAY MODEL

The purpose of this section is to illustrate — by a top-of-the-iceberg example — that we can indeed capture all, of what need be captured, in precise informal as well as formal terms. We will explain, in simple words, and in simple mathematics — where the formula fits, "hand-in-glove" the simple, informal text, what is meant (in a technology independent abstraction) by a the statics of a railway net, by rail lines and railway stations, by the rail units they consists of, the connectors that "glue" units together, etc.; and by the dynamics of the states of signals and swtiches, by open and closed routes, by train movement along routes (even off the net!), by time tables, schedules and traffic in general.

The model given is claimed to describe the domain, independent of any computing application. As shown, elsewhere, in other papers, we can then base requirements on domain models, like the one shown next, and from these requiremtns, in a trustworthy manner, develop dependable software.

After this section we shall expand the view and postulate further applications.

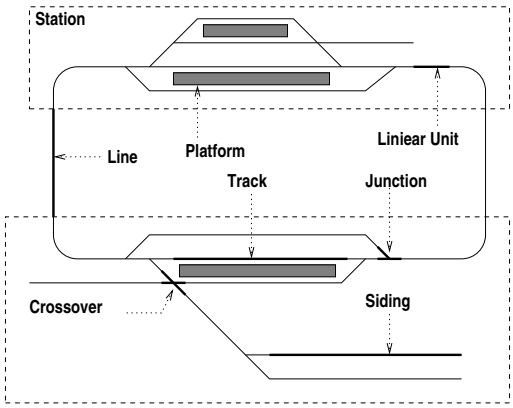
We divide the presentation into two parts: a hierarchical ("topdown") first part followed by a configurational ("bottom-up") second part.

2.1 *Hierarchical Presentation*

We focus on the railway net perspective of railway systems. We designate such components of the rail net which can be physically demonstrated, but we abstract from a number of physical attributes — they can always be simply "added" later on — and focus on the transport logics.

Pls. inspect figure 1.

Fig. 1. A "Model" Railway Net !



Our natural, professional railway language description proceeds as follows: [1]

(1) A railway net consists of lines and two or more stations.
(2) A railway net consists of units.
(3) A line is a linear sequence of one or more linear units.
(4) The units of a line must be units of a net.
(5) A station is a set of units.
(6) The units of a station must be units of a net.
(7) No two distinct lines and/or stations share units.
(8) A station consists of one or more tracks.
(9) A track is a linear sequence of one or more linear units.
(10) No two distinct tracks share units.
(11) The units of a track must be units of the station (of that track, and hence the net)
(12) A unit is either a linear unit, or a switch point, or a simple crossover, or a switch-able crossover, etc. [2]
(13) A unit has one or more connectors.
(14) For every connector there are at most two units which have that connector in common.
(15) Every line of a net is connected to exactly two, distinct stations.

A corresponding, representationally abstract formal specification — please read it carefully, line-by-line, is:

type
Net, Line, Station, Track, Uni, Connector
value
1. obs_Lines: Net → Line-**set**
1. obs_Stations: Net → Station-**set**
2. obs_Unis: Net → Uni-**set**
3. obs_Unis: Line → Uni-**set**

[1] We enumerate the sentences for reference.
[2] A linear unit has two distinct connectors, a switch point has three distinct connectors, crossovers have four distinct connectors.

5. obs_Unis: Station → Uni-**set**
8. obs_Tracks: Station → Track-**set**
12. is_Linear: Uni → **Bool**
12. is_Switch: Uni → **Bool**
12. is_Simple_Crossover: Uni → **Bool**
12. is_Switchable_Crossover: Uni → **Bool**
13. obs_Connectors: Uni → Connector-**set**

axiom
forall n:Net, l,l′:Line, s,s′:Station,
t,t′:Track, u:Uni, c:Connector •
1. **card** obs_Stations(n) ≥ 2,

3. l ∈ obs_Lines(n) ⇒
∀ u:Uni • u ∈ obs_Unis(l) ⇒
is_Linear(u) ∧

7. l′ ∈ obs_Lines(n) ∧ l≠l′
⇒ obs_Unis(l) ∩ obs_Unis(l′) = {},

7. l ∈ obs_Lines(n) ∧ s ∈ obs_Stations(n)
⇒ obs_Unis(l) ∩ obs_Unis(s) = {},

7. s′ ∈ obs_Stations(n) ∧ s≠s′
⇒ obs_Unis(s) ∩ obs_Unis(s′) = {},

8. **card** obs_Tracks(s) ≥ 1,

9. s ∈ obs_Stations(n) ∧ t ∈ obs_Tracks(s)
⇒ ∀ u:Uni • u ∈ obs_Unis(t)
⇒ is_Linear(u),

10. t′ ∈ obs_Tracks(s′) ∧ t≠t′
⇒ obs_Unis(t) ∩ obs_Unis(t′) = {},

14. **card**{u | u:Uni • c ∈ obs_Connectors(u)}
≤ 2,

15. ∃ s,s′:Station,l:Line • s≠s′ ⇒
let sus = obs_Unis(s),
sus′ = obs_Unis(s′),
lus = obs_Unis(l) **in**
let u:U • u ∈ sus,
u′:U • u′ ∈ sus′,
u″,u‴:U • {u″.u‴} ⊆ lus **in**
let scs = obs_Connector(u),
scs′ = obs_Connector(u′),
lcs = obs_Connector(u″),
lcs′ = obs_Connector(u‴) **in**
∃! c,c′ • c ≠ c′ ∧ scs ∩ lcs = {c}
∧ scs′ ∩ lcs′ = {c′}
end end end

2.2 *Configurational Presentation*

We wish, now, to build up to the dynamics of the net: modelling, but abstractly, signals and their states and the state of switches as well as the intended use-state of straight rails:

(1) A path, p:P, is a pair of connectors, (c,c′), of some unit. A path of a unit designate that a train may move across the unit in the direction from c to c′. We say that the unit is open in the direction of the path.
(2) A state, $\sigma : \Sigma$, of a unit is the set of all open paths of that unit (at the time observed). The state may be empty: the unit is closed.
(3) A unit may, over its operational life, attain any of a (possibly small) number of different states ω, Ω.
(4) A route is a sequence of pairs of units and paths —
(5) such that the path of a unit/path pair is a possible path of some state of the unit, and such that "neighbouring" connectors are identical.
(6) An open route is a route such that all its paths are open.
(7) A train is modelled as a route.
(8) Train movement is modelled as a discrete function (map) from time to open routes such that for any two adjacent times the two corresponding open routes differ by at most one of the following: a unit path pair has been deleted from (one or another end) of the open routes, or (similarly) added, or both, or no changes — a total of seven possibilities.

type
1 P = C × C
2 Σ = P-**set**
3 Ω = Σ-**set**
4 R′ = (Uni × P)*
5 R ={| r:R′ • wf_R(r) |}
7 Trn = R
8 Mov = T $\xrightarrow[m]{}$ Trn
value
2 obs_Σ: Uni → Σ
3 obs_Ω: Uni → Ω
5 wf_R: R′ → **Bool**
wf_R(r) ≡
∀ i:**Nat** • i ∈ **inds** r **let** (u,(c,c′)) = r(i) **in**
(c,c′) ∈ ⋃ obs_Ω(u) ∧
i+1 ∈ **inds** r ⇒
let (_,(c″,_)) = r(i+1) **in** c′ = c″
end end
6 open_R: R → **Bool**
open_R(r) ≡
∀ (u,p):U×P • (u,p) ∈ **elems** r
∧ p ∈ obs_Σ(u)
8 wf_Mov: Mov → **Bool**
wf_Mov(m) ≡ **card dom** m ≥ 2 ∧
∀ t,t′:T • t,t′ ∈ **dom** m ∧ t < t′
∧ ∼∃ t″:T • t″ ∈ **dom** m ∧ t < t″ < t′ ⇒
let (r,r′) = (m(t),m(t′)) **in**
clauses (i) − (vii) **end**

We leave it as an exercise to define clauses (i)–(vii) above.

2.3 *Discussion*

Models, informal and formal, such as shown above, can be developed for "the whole" of railway systems — indeed for any (other) transportation infrastructure component: Air traffic (with airports and airlines), shipping (for example emphasizing container logistics), and metropolitan transport (the interplay between taxis, buses, metro, regional trains, etc.).

For railways we have, in this manner, developed, from the basis shown above, sub-models that include the following aspects:

- **Time tables:** T stands (or could, eg. stand for modulo, say a week) times, Sn for station names and Tn for train names. TT stands for time tables: Every applicable train mame associates to a journey of station visits — with their arrival and departure times:
  type
  TT = Tn $\overrightarrow{m}$ (T × Sn × T)*
- **Traffic:** Given that a train can be associated with not only its name but also its open route position on the rails, TF stands for traffic: A continuous function from time, now absolute, with no modulo necessity, to usually stable nets and certainly monotonically changing train (TR) positions:
  type
  TF = T → (**value**
  obsTn: TR → Tn
  obsRs: N → R**-infset**
  axiom
  ∀ tf:TF, t:T •
  t ∈ **dom** tf ∧ **let** (n,tps) = tf(t) **in**
  ∀ tr:TR • tr ∈ **dom** tps ⇒
  tps(tr) ∈ obsRs(n) **end**
- **Train Plans:** The train engine man is given, during briefing, before the train is despatched, a train plan. It gives a discrete, ie. not a continuous, description of where the train is expected to be at certain times, at which sequence, R of units. For all trains we get:
  type
  TP = T $\overrightarrow{m}$ (N × (Tn $\overrightarrow{m}$ R))
  Train plans are discretisations of traffics — and, as is traffic, is modulo the net.
- **Scheduling:** Given a net and a time table we can speak, as the meaning of the net and the time table, of all the traffics that satisfies the net and the time table:
  value
  Meaning: N × TT → TF**-infset**
  We can in particular devise a special planning function, scheduling, which decides upon a suiable set of
  scheduling: N × TT → TP
  Given a traffic and a trains plan we can now speak of monitoring whether trains in the traffic are on schdule:
  value
  ontime: N × TP × TF → **Bool**
  Etcetera.
- **Shunting and Marshalling:** We can define what is meant by shunting and by marshalling yards, marshalling plans, and marshalling.
- **Passenger Service:** And we can define what is meant by passenger services such as travel plan inquiry, ticket and seat reservation, ticketing, etc.
- **Logistics:** And, with nets, rolling stock control, passenger statistics and old time tables, we can speak of logistics planning: Constructing new time tables, etc.
- **Freight Services:** And we can define what is means by freight trains, by the logistics of planning and executing freight service: Reservation, loading, tracing and unloading freight, etc.
- **Rolling Stock Monitoring & Control:** And so on: We can model the rolling stock, its participation in, or waiting, at sidings, to participate in passenger and/or freight train traffic, the gathering of such rolling stock into operable trains, etc.
- **Net Development:** You noticed that the net, N, was a parameter in traffic: To model that the net regularly undergoes maintenance and development. Old rails taken out of service, new lines and stations being put into service.

Domain models of the above, and of requirements to actual or postulated software for the support of railway operations, whether of strategic, tactical or operational nature, have been developed and can be inspected on the Web:

(1) `http://www.it.dtu.dk/~db/i3/raildomain.ps`
(2) `http://www.it.dtu.dk/~db/i3/railreqs.ps`

3. OTHER FORMAL WORK

There is, by now, a very rich literature on applying formal technques to problems of railways.

An EU sponsored network, FME Rail carried out by FME: Formal Methods Europe under the leadership of Dr. Peter Gorm Larsen, `http://www.-ifad.dk/Projects/fmerail`, held five workshops on the topic across Europe in the period 1998–1999: In The Netherlands, England, Austria, Sweden and France gathering a total of more than 200 railway IT people. A very extensive, partially annotated bibliography and a light assessment of

international research and actual commercial use of formal technques can be inspected on:

- `http://www.it.dtu.dk/~db/fmerail/fmerail.ps`

So far the main emphasis — in the use of formal techniques — has been on a topic not listed explicitly in section 2 (including its subsection 2.3): Namely the safetfy critical aspects of controlling the switches of railway stations, in modern technology known as *interlocking*.

4. CHALLENGES AND CLAIMS

4.1 *An Infrastructure of Software*

The domain of railways is a micro-cosm: It represents a huge variety of possibilities for applications of software and all these applications "link together": Are tightly related. Results of computations by means of some software packages has import as input data to other packages — yet little, if any effort, is made to vet that data: To insure that an undelrying model, not just of information (ie. data), but of their semantics, ie. of the actions to be taken on data and the interaction sequences, ie. the processes in which these data are shard with other processes — that such underlying models indeed do exist and "bind" the information and its use, the data and its processing. As it is today there is hardly any guarantee that statistical data gathered from one applicationm say 'traffic and passenger monitoring', is indeed correctly used by 'time tabling' software.

The realm of automatic control, the subject of this conference, interacts strongly with many facets of statistics gathering, time table planning, train scheduling, train despatch, resource allocation and scheduling, etc. — in a cycle back to the more-or-less automatic control of train movemenents.

The field of **computing science** provides design tools, techniques, principles and methods that allow classical **automatic control** engineering designs to "link up" to more modern **operations analysis** (graph-theoretic and combinatorial resource optimisation). The computability based state, event and process modelling techniques of computing science provides means lacking in control theory and theories of optimisation.

The understanding of this truth takes time to propagate.

4.2 *A Socio–Economic Outlook*

This paper was invited with basically the given title. It therefore behooves us — based on many years of both scientific and engineering, theoretical as well as wuite some substantial practical experience — to make some statements as to the rôle of formal techniques in software development, and in particular in the development of software for the support of operations in the application domain of transportation and traffic.

For sociological reasons it seems to take some time before formal techniques enter the domain of software for general software for common railway cum transportation support.

In the area of software for safety critical applications there seems to be a strong trend towards using the formal techniques of tis and the companion paper [21] (See also the references of [21]).

In the area of general software there is hardly any "movement".

The problem, there, seems to be that most managers of IT departments of the transportation industry: providers as well as users, are not software professionals. In other words: They really are neither computing scientifically cum software technologically mature nor engineeringly responsible. The current lot of programmers possess the enviable postion of being considered indispensable — managed by non-professionals — where, as a matter of cool fact, they ought be laid off: The programmers and their management !

A long period therefore confronts us, in which the industry ever so slowly turns around and becomes mature and responsible.

Meanwhile a few have to "fight it out".

It will be interesting to see whether the inevitable transition to trustworthy software conception, from domains via requirements to software deisgn, will primarily be due to the candidates from universities wanting to have it no other way, or due to insurance companies demanding calimed "proofs of correctness" ?

The fact is simple enough: There are enough universities around Europe — and Europe seems to be a leader in the research and use of formal techniques — which produce a reasonably wuantity of very highly qualified candidates. There are enough problems that ought be "formally" tackled. But procurers either do not demand the "proven" quality, or the providers, the suppliers are not capable, or both !

5. CONCLUSION

5.1 *Formal Techniques and Tools*

In this paper, as well as in its "companion" paper, [21], we have used the Raise Specification Lan-

guage, RSL [12], and thus implies use of the Raise software development method, [13]. As shown in [5] there are many complementary as well as "competing" approaches: B [1], VDM-SL [2], [20], [11], Z [18], [26], and others. It seems, to this author, that B is currently being very successfully applied in "real-life", commercial projects. We refer to [5] for references.

5.2 *A Reinforcement*

In a rather personal style — but that is fortunately what invited papers from long experienced people are all about, ie. to be expected — I have related some 25 years of hard-won experience: For the fascinating field of transportation, a highly structured and displined field, the combination of automatic control with operations analysis as linked together by computing science, that is: The well-managed collaboration of control engineers and operations 'researchers' with software engineers, brings exciting promises.

The former disciplines: Automatic cotrol and operations analysis, has, since long, adhered to strong, formal foundations, linking, in the case of automatic control, the natural sciences with mathematical models, and, in the case of operaitons analysis, the human decision processes with computing. The new corner in the triangle: That of computing science, is now no longer a trivial means to "hack code: theformerm but is offering radically more fundamental, broad and deep, all-encompassing models of "all" of railways.

That is the challenge: For owners, operators and procurers of the transportation domains, to understand and profesionally exploit this.

5.3 *Acknowledgements*

The author is grateful to Prof. Schnieder, Univ. of Braunschweig, for promoting this paper. Over the years the author has had the great joy to work with Messrs. Søren Prehn, Chris W. George and others on the specific (railway) as well as general (methodology) subjects covered and exemplified by this paper. My thanks goes to them in no small measure. The origins of the model of this paper seems to derive from work done by Prehn and George. Other models, [3], are more concrete — and would not have given us the degree of freedom and the desired generality. Finally I wish to acknowledge, with great pleasure, the leadership of Dr. Peter Gorm Larsen of the EU sponsored FME Rail project (`http://www.ifad.dk/-Projects/fmerail.htm`).

6. BIBLIOGRAPHICAL NOTES

We refer, rather unscientifically, only to "own" publications and reports — that is: work done by the author and his colleagues during the last 10 years.

In Section 3 we have referred to [9], [10] and to a Web document [5]. The latter list literally hundreds of mostly technical and cientific papers and rports — the "sum total" of which substantiate the claims of the current paper.

In addition we can refer to a number of publications and reports which also substantiate the claims made in this paper: [25], [8], [4], [14], [6], [16], [7]. [3] hints at the issues but, we now reckon, represents an altogether too concrete modelling approach. Hence [4] and its followers.

More recently, and perhaps more appropriate than this rather cursory paper we can refer to [17], [21].

7. REFERENCES

[1] Jean-Raymond Abrial. *The B Book: Assigning Programs to Meanings*. Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, England, 1996.

[2] D. Bjørner and C.B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

[3] Dines Bjørner. Trustworthy Computing Systems: The ProCoS Experience. In *14'th ICSE: Intl. Conf. on Software Eng., Melbourne, Australia*, pages 15–34. ACM Press, May 11–15 1992.

[4] Dines Bjørner. Prospects for a Viable Software Industry: Enterprise Models, Design Calculi, and Reusable Modules. Technical Report 12, UNU/IIST, P.O.Box 3058, Macau, 7 November 1993. Appendix: *"On a railway domain model"* by Søren Prehn and Dong Yulin, Published in *Proceedings from first ACM Japan Chapter Conference*, March 7–9, 1994: World Scientific Publ., Singapore, 1994.

[5] Dines Bjørner. The FME Rail Annotated Bibliography. URL: `http://www.it.dtu.dk/-~db/fmerail/fmerail.html`, Novembr 1999. A main deliverable of an EU project FME Rail.

[6] Dines Bjørner, C.W. George, and S. Prehn. Domain Analysis — a Prerequisite for Requirements Capture. Technical Report 37, UNU/IIST, P.O.Box 3058, Macau, February 1995. .

[7] Dines Bjørner, C.W. George, and S. Prehn. *Scheduling and Rescheduling of Trains*, page 24 pages. FACIT. Sringer–Verlag, London, England, 1999.

[8] Dines Bjørner, Dong Yu Lin, and S. Prehn. Domain Analyses: A Case Study of Station

Management. Research Report 23, UNU/IIST, P.O.Box 3058, Macau, 9 November 1994. Presented at the *1994 Kunming International CASE Symposium*: KICS'94, Yunnan Province, P.R.of China, 16–20 November 1994. .

[9] Dines Bjørner et al. Formal Models of Railway Systems: Domains. Technical report, Dept. of IT, Technical University of Denmark, Bldg. 344, DK–2800 Lyngby, Denmark, September 23 1999. Presented at the FME Rail Workshop on Formal Methods in Railway Systems, FM'99 World Congress on Formal Methods, Toulouse, France. Avaliable on CD ROM.

[10] Dines Bjørner et al. Formal Models of Railway Systems: Requirements. Technical report, Dept. of IT, Technical University of Denmark, Bldg. 344, DK–2800 Lyngby, Denmark, September 23 1999. Presented at the FME Rail Workshop on Formal Methods in Railway Systems, FM'99 World Congress on Formal Methods, Toulouse, France. Avaliable on CD ROM.

[11] John Fitzgerald and Peter Gorm Larsen. *Developing Software using VDM–SL*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 1RU, England, 1997.

[12] The RAISE Language Group. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[13] The RAISE Method Group. *The RAISE Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

[14] Kirsten Mark Hansen. Validation of a railway interlocking model. In M. Bertran M. Naftalin, T. Denvir, editor, *FME'94: Industrial Benefit of Formal Methods*, pages 582–601. Springer-Verlag, October 1994.

[15] Kirsten Mark Hansen. Modeling Railway Interlocking Systems. In *FME Rail Workshop # 2*, volume # 2. FME: Formal Methods Europe, Formal Systems (Europe) Ltd, Keble Court, 26 Temple Street, Oxford OX4 1JS, UK, Telephone: +44 1865 728460, Telefax: +44 1865 201114, October 1998. Scanrail Consult (DK).

[16] K.M. Hansen. *Linking Safety Analysis to Safety Requirements*. PhD thesis, Department of Computer Science, Technical University of Denmark, Building 344, DK-2800 Lyngby, Denmark, August 1996.

[17] Anne E. Haxthausen and Jan Peleska. Formal Development and Verification of a Distributed Railway Control System. In Wing and Woodcock, editors, *FM'99: World Congress on Formal Methods in the development of computing systems*, volume 1708–1709 of *Lecture Notes in Computer Science*, pages 1546–1563, Heidelberg, Germany, September 29 1999. FME, Springer-Verlag.

[18] Ian J. Hayes, editor. *Specification Case Studies*. International Series in Computer Science. Prentice Hall, Hemel Hempstead, Hertfordshire HP2 4RG, UK, 1987.

[19] Kurt Jensen. *Coloured Petri Nets*, volume 1–2–3 of *EATCS Monographs in Theoretical Computer Science*. Springer–Verlag, Heidelberg, 1985.

[20] C.B. Jones. *Systematic Software Development using VDM*. Prentice Hall International, second edition, 1990.

[21] Morten Peter Lindegård, Peter Viuf, and Anne Elisabeth Haxthausen. Modelling Railway Interlocking Systems. In *Control in Transportation Systems 2000*, The Boulevard, Langford Ave., Kidlington, Oxford, OX5 1GB, UK, June 2000. IFAC (Intl. Fed. for Automatic Control), Elsevier Science Ltd., Pergamon.

[22] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive Systems: Specifications*. Addison Wesley, 1991.

[23] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive Systems: Safety*. Addison Wesley, 1995.

[24] Wolfgang Reisig. *Theory and Practice of Petri Nets*. Springer–Verlag, Berlin Heidelberg, 1998.

[25] Jens U. Skakkebæk, Anders P. Ravn, Hans Rischel, and Zhou ChaoChen. Specification of Embedded, Real-time Systems. Technical report, Dept. of Computer Science, Techn.Univ. of Denmark, EuroMicro Workshop on Formal Methods for Real-time Systems, 1992 December 1991.

[26] J. Michael Spivey. *Understanding Z: A Specification Language and its Formal Semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, UK, January 1988.

Reference [21] refers to a paper in these proceedings !