FROM RAILWAY RESOURCE PLANNING TO TRAIN OPERATION

a brief survey of complementary formalisations

Martin Penicka*
Department of Applied Mathematics, Faculty of Transportation Sciences, Czech Technical University, Na Florenci 25, CZ-11000 Prague
penicka@fd.cvut.cz

Dines Bjørner*
Department of Computer Science, School of Computing, National University of Singapore, 3 Science Drive 2, Singapore 117543
E–Mail: bjorner@comp.nus.edu.sg, URL: http://www.comp.nus.edu.sg/cs/

Abstract From seasonal planning via day-to-day train operation to real-time monitoring and control of trains, software applications are becoming increasingly integrated. Timetabling implies train traffic. Train staff rosters and train car maintenance are initially derived from timetables and influences future timetables.

In this extended abstract we shall sketch a formal model of Railway Nets, Timetables, Rosters, Maintenance, Station Interlocking, Line Direction Agreement and Automatic Line Signaling. The last three formal models are based on four integrated formal techniques (RAISE, Petri Nets, Live Sequence Charts and State Charts). The formal sketches are all "backed-up" by either a publication or a research report.

Keywords: Railways, Planning, Timetabling, Rostering, Control, Interlocking, Signalling, RAISE, Petri Nets, Live Sequence Charts, State Charts, Technique Integration

1. Railway System

A *railway system* (Ω) can be modelled as a function from *time* (`T`) to states of a *railway net* (`N`), to states of all *rolling stock* (`RS`), to its *timetable* (`TT`), to states of all *passengers* (`P`) and *freight* (`F`).

*This paper was written while the authors were at Inst. of Informatics & Modelling, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark. The first author gratefully acknowledges support from the EU 5FP Network of Excellence Programme: CoLogNET: Computational Logic Network

```
type
    T, N, RS, TT, P, F
    Ω' = T → (N × RS × TT × (P × F))
    Ω = {|rw:Ω' • wf_Ω(rw)|}

value
    wf_Ω: Ω' → Bool
    wf_Ω(rw) ≡ well-formedness of railway states
```

The well-formedness describes constraints that must be met by any railway system. They are laws of nature, e.g. trains move monotically, net changes states accordingly, etc.

2. Allocation & Scheduling of Resources

From Passenger Statistics to Railway Nets and Timetables

Passenger statistics (`STA`) express predicted number of passengers between pairs of *geographical centers* (`C`) (urban area were potential passengers live) in *time intervals* (`TxT`). From a *cartographical map* (`MAP`) one can observe the geographical centers and their positions.

```
type
    MAP, C, POS
    STA' = (C×C) -m-> ((T×T) -m-> Nat)
    STA = {|sta:STA' • wf_sta(sta)|}

value
    wf_sta: STA → Bool
    wf_mapsta: MAP × STA → Bool
    obs_Cs: MAP → C-set
    obs_Pos: MAP × C → POS
```

A railway net (`N`) is composed from stations and lines. *Lines* (`T`) connect stations. A *railway station* (`S`) is a place where trains stop to allow passengers to enter and get off a train as reflected in timetable.

A *timetable* (`TT`) expresses for all *planned trains* (`Tn`) their *journeys* (`J`). A train *journey* (`J`) is a list of line visits (departure time from a station, line, arrival time time to a station, train *capacity* (`K`), and possible *periodicity* (`PRD`) (e.g. 24hour, or 20 minutes) and *restrictions* (`RST`) on the days for which it applies (e.g. Mon-Fri, only or summer season only)).

Now one can define a function `genNTT` which from a given geographical map and passenger statistics, and according to a given set of *predicates* (`P`), generates all possible pairs of *nets and timetables* (`NTT`), such that these satisfy the map and the passenger statistics.

```
type
    N, L, S
    Tn, PRD, RST

    TT' = Tn -m-> J
    TT = {|tt:TT' • wf_TT(tt)|}
    NTT' = N × TT
    NTT = {|ntt:NTT' • wf_NTT(ntt)|}

    J = (T×S×L×S×T×K×PRD×RST)*
    P: N × TT → Bool

value
    p: P
    wf_TT: TT' → Bool
    wf_NTT: NTT' → Bool

    genNTT: (MAP × STA) → P → NTT-infset
    genNTT(map,sta)(p) as intts
        post ∀ ntt: NTT • ntt ∈ intts ⇒
            satisfy(n,map) ∧ satisfy(tt,sta) ∧ p(n,tt)
```

From Nets & Timetables to Operational Planning

Let us now define a usually heuristic operational planning process, `Planning`, which from a given geographical map and passenger statistics according to a given set of *predicates* (`P`), generates one possible pair of *nets and timetables* (`NTT`).

```
Planning(map, sta)(p) ≡
    let ntts = genNTT(map,sta)(p) in
    if card ntts = 1 then ntts
    else
        if ∃ p′: P • 1 ≤ card genNTT(map,sta)(p′) < card ntts in
            then let p′: P • 1 ≤ card genNTT(map,sta)(p′) < card ntts in
                Planning(map,sta)(p′) end
            else let ntt: NTT • ntt ∈ ntts in ntt end
        end
    end
    end
```

From Timetables and Rolling Stock to Vehicle Scheduling

Given a net, a timetable and an available *rolling stock* (`RS`) one is interested in computing optimal *working plans* (`VWP`) for *vehicles* (`V`) of rolling stock such that these plans honour the timetable. A set of *predicates* (`P`) on rolling stock, nets and timetables has to be satisfied (one can operate electric powered engine only on suitable lines, etc.). We model the set of predicate as one "grand" predicate.

```
type
    V, RS = V-set
    VWP = (V ⇸ (Tn × J)*) × RS

    P: N × TT × RS × RS → Bool
value
    p: P

genVWP: (N×TT×RS)→P→VWP-set
genVWP(n,tt,rs)(p) as vwps
    post ∀ (vwp,rs′):VWP • (vwp,rs′) ∈ vwps ⇒
        dom vwp ⊆ rs ∧ rs′=rs\dom vwp ∧
        honour(n,tt,vwp) ∧
        p(n,tt,rs,rs′)
```

From Timetables and Human Resources to Rostering

Given a net and a timetable one can determine the number of *human resources* (`HR`) of each type (drivers, conductors, etc.) needed to honour the timetable. We refer to [5]. Let us just show, how one can generate a set of *working plans* (`HWP`) for *railway employees* (`H`). A set of *predicates* (`P`) on human resources, nets and timetables has to be satisfied. We model the set of predicate as one "grand" predicate.

```
type
    H, HR = H-set
    HWP = (H ⇸ (Tn × J)*) × HR

    P: N × TT × VWP × HR × HR → Bool

value
    p: P
```

genHWP: $(N\times TT\times VWP\times HR)\to P\to HWP$**-set**
genHWP$(n,tt,vwp,hr)(p)$ **as** hwps
  post $\forall$ (hwp,hr′):HWP • (hwp,hr′) $\in$ hwps $\Rightarrow$
  dom hwp $\subseteq$ hr $\wedge$ hr′=hr**dom** hwp $\wedge$
  honour(n,tt,vwp,hwp,hr) $\wedge$
  p(n,tt,vwp,hr,hr′)

From Timetables to Vehicle Maintenance

The earlier vehicle working plans did not specify that any vehicle has to undergo preventive maintenance. We now define a set of functions, which modify vehicle working plans to reflect timely maintenance. By a maintenance we understand all regular activities which must be done with rolling stock and according to some *rules* (R). Each vehicle, according to its type, has associated with it certain types of maintenance tasks to be performed with a frequency which can be expressed by elapsed number of kilometer or operating house since a previous maintenance.

Given a *railway net* (N), *vehicle working plans* (VWP), a *timetable* (TT) and a *planning period* (TxT) the job is to generate all the possible *sets of changes* (CS), necessary and sufficient to secure maintenance. Given these sets, one is selected and used for update of existing vehicle working plans.

```
type
   CS = (DayChange | NightChange | EmptyRide)-set
value
   genCS: N×TT×VWP×(T×T) ~→ CS-set

   MPlanning: N×TT×VWP×(T×T)→(VWP×CS)
   MPlanning(n,tt,vwp,(tb,te)) ≡
      let css = genCS(n,tt,vwp,(tb,te)) in
         Select(vwp,css) end

   Select: VWP×CS-set → (VWP×CS)
   Select(vwp,css) as (vwp′,cs)
      post cs ∈ css ∧ equivalent(vwp,vwp′)
```

More details about this task can be found in [4].

3. Monitoring & Control

Railway Net States. *Lines* (L) and *stations* (S) are composed from “smallest” rail parts called *units* (U) (linear, points, cross-overs, switchable cross-overs). A unit define a number of connectors (linear:2, point:3, cross-overs:4). A subset of pairs of distinct connectors of a unit define paths though the unit. By a *unit state* (Σ) we understand any such subset. A unit may change state. The state space of a unit is called Ψ. A unit is said to be closed, if it is in *path state* ($\sigma_{\{\}}$) of no paths.

A *route* (R) is a sequence of units. A route is open, if all units are in non-closed states and if the units state paths connect [1].

By *traffic* (TF) we mean a function from time to net and *train states* ($T\Pi$). A train state contains information about train position on the net, its actual and planned velocity and acceleration, etc. Our *primed definition of traffic* (TF’) defines values that do not respect laws-of-nature (there are no “ghost trains”, trains do not“jump” all over the net, etc.). Also desired *railways properties* (RR) are not respected in TF’ (train movements only on open routes, only one train on a open route, obeying interlocking rules and regulations, etc.). See [3].

```
type
    N, TΠ, Tn, POS, RR
    U, C, P=C×C
    Σ = P-set
    Ψ = Σ-set

    TF′ = T → (N × TP)
    TF = {|tf:TR′ • wf_TF(tf)|}
    TP = Tn -m-> TΠ

value
    obs_Cs: U → C-set
    obs_Σ: U → Σ
    obs_Ψ: U → Ψ

    wf_TF: TF′ → Bool
    wf_TF(tf) ≡ obeyLaws(tf)

    obs_POS: TΠ → POS
    regs: TF × N × RR → Bool
```

From Timetable to Traffic

The syntactic quantity a *timetable* (`TT`) denotes the semantic quantity a set of possible *traffics* (`TF`) which satisfy the timetable. See [2].

```
type
    TR, TT, N

value
    denote: TT × N → TF-infset
    denote(tt,n) as itfs
        ∀ tf:FT • tf ∈ itfs ⇒
            regulations(tf,n) ∧ obeysTT(tf,tt,n)
```

From Traffic to Station Interlocking

The above model of dynamics of units did not show how units change states. This is what we will now consider. We will use Petri Nets to model conditions for state changes of a unit.

Interlocking has to do with setting up proper routes from station approached signal to a track (platform) in the station and from these to the lines. We shall focus on one way of constructing models for proper interlocking control scheme using Place Transmission Petri Nets. Petri Net for stations can be built up from four subparts: Petri Net for units, for switches (ie., point or switchable crossover), for signals (Fig. 1), and finally Petri Net for routes. The Petri Net of a route (Fig. 2) is then a composition of all its unit, switch and signal Petri Nets — where the composition is specified by an interlocking table. Since we do not show a specific station nor its interlocking table we refrain from showing the full Petri Net [3]. The table expresses for each interesting route the state requirements for switches (points and switchable crossovers) and the requirements for signal states.

So, on one hand we have our RSL model for nets and trains states; on the other hand we have shown Petri Nets that control state spaces. In other words we have brought two different specification techniques. Hence from RLS specification of a station one can build a interlocking table and then the Petri Net.

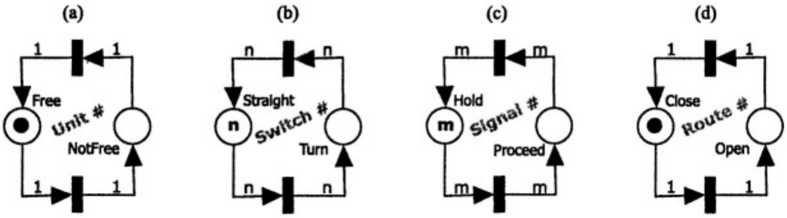

Figure 1. Petri Net for unit, switch, signal and route

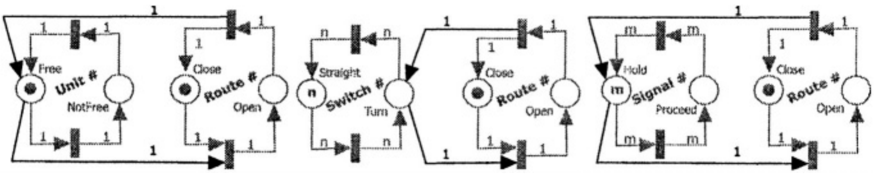

Figure 2. Adding arcs for unit|switch|signal and route

From Traffic to Line Direction Agreement

The above RSL model of traffic did not show how certain rules could be obeyed. In this section we with to show one of the important safety properties of a railway line: that two trains are not allowed to move in opposite directions on a line. One way of ensuring this is by a so called Line Direction Agreement System (LDAS).

The externally visible behavior of the LDAS can be illustrated using Live Sequence Charts. The three entities are: Station A (SA), the Line Direction Agreement System (LDAS), and Station B (SB). The charts in Figure 3 illustrate only the partial communication as seen from Station A. The mutual exclusive control of the LDAS can illustrated by a Statechart. For more details we refer to [3]. Transitions of the Statechart corresponds to messages of the LDAS.

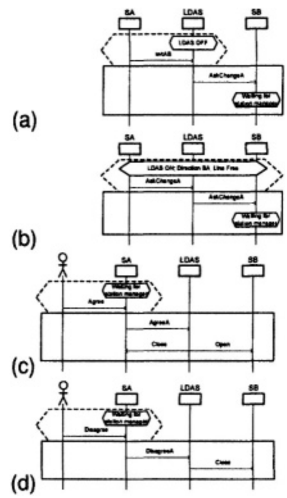

Figure 3. (a) Initial LDAS, (b) Request Direction Reversal, (c) Request Approval, (d) Request Rejection

From Traffic to Automatic Line Signalling

Again the above RSL model of traffic did not show how trains on a line can be separated. Lines connect exactly two stations and can be divided into several segments (see Fig. 4). Each segment can be either in '*Free*' state (when no train is detected in the segment) or in *'Occupied'* state. For each segment there are two signals (one in each direction of travel). With each signal we associate four possible states *('Hold', 'NextHold', 'Proceed'* and *'Off')*.

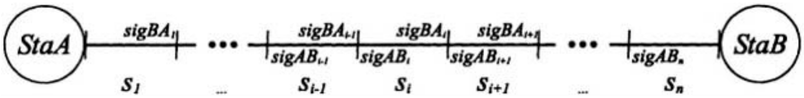

Figure 4. Signal positions on a line with n segments

```
type
    TR, N, L, S, SEG, SIG
    SegSt == Free | Occupied
    SigSt == Hold | NextHold | Proceed | Off

value
    obs_Ss: L → S-set
    obs_Segs: L → SEG*
    obs_Sigs: SEG → SIG × SIG
    obs_SegState: TR × SEG → SegSt
    obs_SigState: TR × SIG → SigSt
```

Statecharts can be used to specify requirements for pair of signals along a line including pairs including station interlocking.

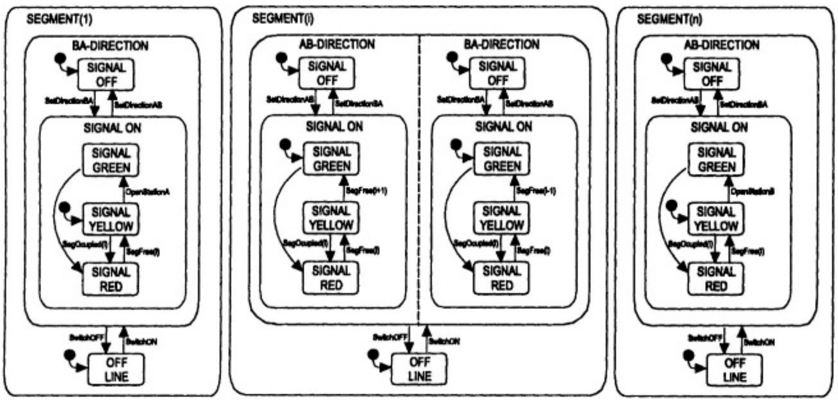

Figure 5. Sub-Statechart for Automatic signalling in lines

The Statechart for a line with three and more segments is composed from the three subparts shown on Fig. 5.

4. Conclusion

We have covered a part of the railway domain by rough sketching the synopses of several examples concerning scheduling & allocation and monitoring & control aspects of railways. Our focus has been the underlying formal models. Several different formal methods, f.e. RAISE, Petri Net, StateChart, Live Sequence Chart have been used.

References

[1] Dines Bjorner. Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In *CTS2003:10th IFAC Symposium on Control in Transportation Systems,* Oxford, UK, August 4-6 2003. Elsevier Science Ltd. Symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki.

[2] Dines Bjfirner, Jakob Braad, and Karin S. Mogensen (Eds.). Models of Railway Systems: Domain. In Thierry Lecomte and Peter Gorm Larsen, editors, *FME Rail Workshop* # 5, volume # 5 of *FME Rail Workshop; Toulouse, France.* FME: Formal Methods Europe, Steria, France, September 22–24 1999. (60 pages) Dept. of IT, Techn. Univ. of Denmark.

[3] Dines Bjorner, Chris George, Anne E. Haxthausen, Christian Krog Madsen, Steffen Holmslykke, and Martin Penicka. "UML"–ising Formal Techniques. In *INT 2004: Third International Workshop on Integration ofSpecification Techniquesfor Applications in Engineering.* Institut fur Softwaretechnik und Theoretische Informatik, Sekr. FR 6-1, Techn.Univ. of Berlin, Franklinstrasse 28/29, D–10587 Berlin, Germany, 28 March 2004, ETAPS, Barcelona, Spain. To be published in INT–2004 Proceedings, Springer–Verlag.

[4] Martin Penicka, Albena Kirilova Strupchanska, and Dines Bjorner. Train Maintenance Routing. In *FORMS'2003: Symposium on Formal Methodsfor Railway Operation and Control Systems.* L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.

[5] Albena Kirilova Strupchanska, Martin Penicka, and Dines Bjorner. Railway Staff Rostering. In *FORMS2003: Symposium on Formal Methodsfor Railway Operation and Control Systems.* L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.