TRAIN MAINTENANCE ROUTING[1]

Martin Pěnička[1], **Albena Strupchanska**[2], **Dines Bjørner**[3]
[1] *Czech Technical University, Faculty of Transportation Sciences, Department of Applied Mathematics*
Na Florenci 25, 110 00 Praha 1, Czech Republic
Phone: (+420) 224 890 712, Fax: (+420) 224 890 702, e–mail: penicka@fd.cvut.cz
[2] *Linguistic Modelling Department, Bulgarian Academy of Sciences*
25A Acad. G. Bonchev Str. Sofia 1113, Bulgaria
Phone: (+359 2) 9796607 , Fax: (+359 2) 707273, e–mail: albena@lml.bas.bg
[3] *Technical University of Denmark, Computer Science and Engineering,*
Informatics and Mathematical Modelling
Building 322, Richard Petersens Plads, DK–2800 Kgs.Lyngby, Denmark
Phone: (+45) 4525 3720, Fax: (+45) 4593 0074, e–mail: db@imm.dtu.dk

Abstract: We present an informal narrative and a formal model of the train maintenance routing problem. The problem to be tackled is as follows: There is a railway net. There are stations and lines in the net. Trains travel from station to station according to given schedule and rolling stock roster. Trains are composed of assemblies. There are several types of assemblies and several types of maintenances. Each assembly has to be maintained before a certain time interval between maintenance elapses, and at a limited numbers of maintenance stations. At stations pairs of assemblies may be exchanged.
Keywords: formal method, domain model, maintenance, schedule, routing

1. INTRODUCTION

Railway planners handle time–consuming tasks of railway operations. There are a number of tasks which can be solved by computers using operation research algorithms. These tasks are mainly being solved separately without any relations or integrations between them. We would like to focus not only at their integration, but we would like to find some common parts of these tasks already during the software development stages.

This is the main reason for presentin, in this paper, a formal methods approach to one of the railway optimisation problems — train maintenance routing. Formal methods approaches to other problems (crew rostering and optimal train length/train composition and decomposition) is presented in a separate paper (Strupchanska et al., 2003), respectively in a separate report (Karras and Bjørner, 2002).

In future, this approach should lead to much deeper, better and easier integration of railway applications in and among all tasks of the railway monitoring, control and planning processes.

1.1 Synopsis

Each railway company operating trains deals with the problem of maintenance of their rolling stock. By a circulation plan we shall understand a schedule of sequences of station visits. Each train is composed of carriages, which, according to a circulation plan, can be grouped into in–divisible parts — called assemblies. In other words, an assembly is one ore more carriages that have the same circulation plan. We do not discuss how to device circulation plans for assemblies in this paper. This is a task of rolling stock rostering and of optimal train length determination.

In this paper we define notions of rolling stock rosters and of maintenance events, and we show how maintenance

[1] The writing of this paper, as well as the papers (Strupchanska et al., 2003; Bjørner, 2003), also contained in these proceedings, and their presentation at Budapest, is sponsored by the EU IST Research Training Network AMORE: Algorithmic Models for Optimising Railways in Europe: http://www.inf.uni-konstanz.de/algo/amore/. Contract no. HPRN-CT-1999-00104, Proposal no. RTN1-1999-00446

events can be added into rolling stock rosters. By maintenance we do not mean only regular check of all systems (assemblies, etc.) in the depot, but we present maintenance in a more general sense. We understand maintenance as all activities, which must be done with rolling stock, regularly according to some rules, and which should be planned in advance except for the operation of rolling stock itself. That is the reason, why we also include outside and inside cleaning of carriages, refuelling diesel engines, refilling supplies into restaurant carriages, water and oil refilling, etc.

Each carriage, according to its type, has associated certain types of maintenance tasks. Each task has a defined frequency of necessary handling of this task upon the carriage. The frequency can be expressed by the elapsed number of kilometer or oeprating house since a previous maintyenance, ie., are intervals, for which maintenance need be done.

Basically there are two different ways on how to add maintenance to the rolling stock plans.

Rostering with Maintenance is the first possible way. Maintenance is planned already in the rolling stock roster planning process (maintenance is seen just as a one of the tasks in the roster). All maintenance actions for all rolling stock are planned in advance. This approach seems to be appropriate for long–distance trains and also for maintenance types that have to be done quite often (eg., inside cleaning, diesel fuelling, etc.).

Maintenance Routing is the second possible way. In this case, the rolling stock roster has several maintenance opportunities only. That means, that not all carriages have maintenance in their plans. In this approach it is necessary to have on–line statistics about actually elapsed kilometers and operating hours for all assemblies. Later, during train operation, maintenance checks are planned for those assemblies which are close to reach a given kilometer or time limit. It is done by modifying previous plans in such a way, that all assemblies are routed to maintenance stations. These modifications are called night and day exchanges or empty rides between stations. Maintenance routing better fits short-distance trains, typically trains "around" big cities, and also for those types of maintenance, where irregularities can be expected. For example, a broken engine must be routed to the maintenance station immediately, with no care about kilometer distance for which it was slated at the time of the breakdown event.

In this paper we deal only with the second approach — maintenance routing. This means that our input is a rolling stock roster with several maintenance opportunities (not assigned to concrete assemblies yet). For all assemblies in the network we can find their position in the network according to the schedule at a given time, number of kilometers and hours elapsed from the most recent maintenance checks. Certain events (like breakdowns) must be recorded and taken into account as well.

Output from the maintenance routing planning is a list of changes in rolling stock roster for the next few days. Once a day, changes and recorded events are applied to the current rolling stock roster plan and are used as input for the follwing day's maintenance routing planning.

1.2 The Major Functions

Given a railway net, N, a traffic schedule TS, and a planning period (from day-time, DT, to day-time, DT) the job is to formally characterize and generate all the possible sets of changes, CS, necessary and sufficient to secure finely maintenance. What we understand by terms net, traffic schedule and sets of changes can be found further on in the paper.

type
    N, TS, DT, CS
value
    gen_Changes: N $\times$ TS $\times$ (DT $\times$ DT) $\overset{\sim}{\to}$ CS-**set**

Given these possible change sets, one is selected and applied to the traffic schedule to generate a new traffic schedule for a given period.

value
    ApplyChanges: TS $\times$ CS $\times$ (DT $\times$ DT) $\to$ TS

1.3 Requirements and Software Design

We emphasis that we formally characterize schedules, assembly plans and maintenance changes — such as they are "out there", in realty, not necessarily as we wish them to be. On the basis of such formal domain specificatons we can then express software requirements, ie., such as we wish schedules, assembly plans and maintenance changes to be.

The actual software design relies on the identification of suitable operation research techniques (ie., algorithms), that can provide reasonably optimal solutions and at reasonable computing times.

It is not the aim of this paper to show such operation research algorithms. Instead we refer to (Kroon and Fischetti, 2000; Kroon, 2001; Maróti, 2001).

1.4 Paper Structure

The paper is divided into two main parts. In the first part (Section 2) we give a full description of those railway domain terms that are relevant to the problem at hand. We start with a description of railway nets, lines and stations. Then it is explained what we mean by trains and assemblies. Further we explain the concept of traffic schedules and describe some functions on such schedules. The last part of Section 2 presents a detailed description of maintenance.

The second main part is Section 3: Planning. In it we explain the necessary and sufficient changes to rolling stock rosters, introducing the concepts of day and night changes and of empty rides. We explaon their generation, as well as the application of these changes to the new traffic schedule.

All these descriptions are presented in natural English language as well as in the Raise Specification Language (RSL) (George et al., 1992).

2. FORMAL MODEL

In this section we introduce the actual domain phenomena and concepts of railway nets, trains, schedules, rolling stock rosters and maintenance, and we build a domain model in a 'formal methods' approach — step–by–step.

First we define the concept of railway nets. We describe railway nets as a set of lines and a set of stations and all properties, which belongs to these concepts.

2.1 Nets, Lines, Stations

In the first part basic concepts of railway net, lines and stations are described. We present it in natural English description as well as in RSL.

Narrative:

Each net (`N`) is composed from two main parts: stations (`S`) and lines (`L`). Stations and lines can be observed from the net. Axioms (ie., constraints) are:

- There are at least two stations and one line in a net (α_1).
- Each line connects exactly two distinct stations (α_2).
- Each station is connected at least to one line (α_3).
- Each line has no zero length (α_4).

Formal Model:

```
scheme NETWORK_S =
    class
        type N, L, S, KM

        value
            zero_km : KM,
            > : KM × KM → Bool,
            obs_S : N → S-set,
            obs_L : N → L-set,
            obs_S : L → S-set,
            obs_Length : L → KM

        axiom
            (α1) ∀ n : N •
                        card obs_S(n) ≥ 2 ∧
                        card obs_L(n) ≥ 1,
            (α2) ∀ n : N, ℓ: L •
                        ℓ∈ obs_L(n) ⇒
                            card obs_S(ℓ) = 2,
            (α3) ∀ n : N, s : S • s ∈ obs_S(n) ⇒
                        ∃ ℓ: L ⇒ s ∈ obs_S(ℓ)
            (α4) ∀ n : N, ℓ:Lin •
                        ℓ∈ obs_L(n) ⇒
                            obs_Length(ℓ) > zero_km
    end
```

2.2 Time & Date

Narrative:

In this part, basic functions about date (`D`), time (`T`) and time intervals (`TI`) are presented. Since it is not the main subject of this paper, no detailed description is given.

Formal Model:

```
scheme TIME_S =
    class
        type T, TI

        value
            + : T × TI → T,
            + : TI × T → T,
            + : TI × TI → TI,
            − : T × TI → T,
            − : TI × TI → TI,
            > : T × T → Bool,
            > : TI × TI → Bool,
            < : T × T → Bool,
            < : TI × TI → Bool
    end

scheme DATE_S =
    class
        type
            D,
            Day,
            Month,
            Year,
            WD == mo | tu | we | th | fr | sa | su
        value
            obs_Day : D → Day,
            obs_Month : D → Month,
            obs_Year : D → Year,
            obs_WeekDay : D → WD
    end

scheme TIME_DATE_S =
    extend TIME_S, DATE_S with
    class
        type DT = D × T

        value
            + : DT × TI → DT,
            + : TI × DT → DT,
            − : DT × TI → DT,
            − : TI × TI → TI,
            > : DT × DT → Bool,
            < : DT × DT → Bool,
    end
```

2.3 Trains and Assemblies

Narrative:

There are trains (TR) travelling in the network from station to station. Each train has a train number (TRNo) and a train name (TRNa). Each train is composed of an ordered list of assemblies (A). In a real world assembly can be composed of cars, but in this task, the assembly is always the smallest part of the train and can never be divided into pieces.

Each assembly has its unique identification number (AID). There is a kilometer distance, which each assembly has run in total at certain time. There are several different types of assemblies (AT) operated by railway companies. These type could be passenger or cargo car, diesel or electric engine, doubledecker or sprinter unit, etc.

Since each train is a ordered list of assemblies, we can easily find out the position (POS) of an assembly in a train. In our case, we just distinguish, if an assembly is first, last or properly internal. If the train is composed of one assembly, then it is called solo.

There are some axioms: Each train is composed of least one assembly (α_5). Each assembly has its unique identification number (α_6).

Formal Model:

```
scheme TRAIN_S =
    extend TIME_DATE_S with
    class
        type
            TR,
            A,
            TRNo,
            TRNa,
            AID,
            AT == el_loko | di_loko | cargo_car
            POS == fir | mid | las | sol | non

        value
            obs_TrnNa : TR → TRNa,
            obs_TrnNo : TR → TRNo,
            obs_Asml : TR → A*,
            obs_AId : A → AID,
            obs_AType : A → AT,
            obs_Km : A × DT → KM,
            Position : TR × A → POS
            Position(trn, a) ≡
                case obs_Asml(trn) of
                    ⟨a⟩ ⌢→ sol
                    ⟨a⟩⌢ asl → fir
                    asl ⌢ ⟨a⟩ ⌢→ las,
                    asl   ⟨a⟩   asm′ → mid
                    _ → non
                end

        axiom
            (α5) ∀ trn : TR •
                    len obs_Asml(trn) ≥ 1
            (α6) ∀ a : A • ∼∃ a′ •
                    a ≠ a′ ∧
                    obs_AId(a) = obs_AId(a′)
    end
```

2.4 Traffic Schedule

Traffic schedules together with network topologies and train descriptions are the main inputs into our application.

Narrative

Each railway company which operates trains needs to deal with schedules (SCH) from which traffic schedule (TS) can be extracted. Traffic schedules assign journeies (J) to each train number and date . A journey is a sequence of rides (R). A ride is composed of departure time and station, arrival time and station, and the train, that serves the ride. Sequence of rides served always by the same assembly is called an assembly roster or an assembly plan (AP).

The function (APlan) extracts the assembly plan for a given assembly identification from given traffic schedule and in a given time interval. Function (AIDL) returns a list of assembly identifications from a given ride. Function (ActAsms) extracts the set of assemblies which are active according to a given traffic schedule in a given time interval.

There are other axioms. Each traffic schedule has at least one journey (α_7). In each ride, the arrival station is different from the departure station and the arrival time is “later” than the departure time (α_8). The train number is the same for all rides of a journey. The arrival station of any ride in a journey is equal to the departure station of the next ride in that journey (α_9).

Formal Model:

```
scheme SCHEDULE_S =
    extend TRAIN_S with
    class
        type
        SCH,
        TS = TRNo →m (DT →m J),
        J = R*,
        R = (DT × S) × (S × DT) × TR,
        AP = R*

    value
        obs_TraSCH : SCH → TS,

        APlan :
            TS × AID × (DT × DT) → AP
        APlan(ts, aid, (t, t′)) as rl post
            ∀ i : Nat • {i, i+1} ⊆ indx rl ⇒
                aid ∈ elems AIDL(rl(i)) ∧
                aid ∈ elems AIDL(rl(i+1)) ∧
                {rl(i), rl(i+1)} ⊆ JSet(ts) ∧
                DepS(rl(i+1)) = ArrS(rl(i)) ∧
                t ≤ DeptT(rl(i)) <
                    DepT(rl(i+1)) ≤ t′,

        AIDL : R → AID*
        AIDL(r) ≡
        let(_, _, _, _, trn) = r,
            a = obs_Asml(trn) in
        ⟨aid | aid in
        ⟨obs_AId(hd a)..obs_AId(a(len a))⟩⟩
        end

        ActAsms : TS × (DT × DT) → A-set
        ActAsms(ts, (t, t′)) ≡
            {a | a : A •
                len APlan(ts, a, (t, t′)) > 0},

        JSet : TS → J-set,
        JSet(ts) ≡
            ∪ {rng tn | tn : (DT →m J) •
                tn ∈ rng ts}

        DepS : R → S
        DepS(r) ≡
            let (_, s, _, _, _) = s in s end,

        DepT : R → DT
        DepT(r) ≡
            let (t, _, _, _, _) = r in t end,

        ArrS : R → S
        ArrS(r) ≡
            let (_, _, s, _, _) = r in s end,

        ArrT : R → DT
        ArrT(r) ≡
            let (_, _, _, t, _) = r in t end,

    axiom
        (α7) ∀ ts : TS • card JSet(ts) ≥ 1,
        (α8) ∀ r : Ride •
            let ((dt, ds), (ast, ast), _) = r
            in ds ≠ ad ∧ dst < ast end,
        (α9) ∀ j : J, i : Nat •
            {i, i+1} ∈ inds j ⇒
            let (t1, _, s, t2, trn) = j(i),
                (t1′, s′, _, t2′, trn′) = j(i+1)
            in
            obs_TrnNa(trn)=obs_TrnNa(trn′)∧
            obs_TrnNo(trn)=obs_TrnNo(trn′) ∧
                    t1<t2≤t1′<t2′ ∧ s = s′
            end
end
```

2.5 Maintenance

Narrative:

We extend the general model of railway network. First we define different types of maintenance (MT). Some possible maintenance types can be:

Regular operation check. Each engine and carriage — according to given rules and safety regulations — must be checked regularly. There is a limited number of stations where this maintenance can be made (usually just one for each train type).

Inside cleaning is the most common maintenance operation for passenger carriages. It can be done at nearly every station, without additional shunting demands and costs.

Outside cleaning is also common maintenance, but usually not all stations in the network have required equipment.

Diesel engine refuel and Water/sand/oil refill are other examples of maintenance types.

We next define maintenance plans. They, (MNTPLAN,) are lists of actions (ACTION), which are temporally ordered. These action could be: 'Working Ride' (WR), 'Empty Ride' (ER), and 'Maintenance Check' (MNT).

Each assembly type has defined certain maintenance types that have to be done (REQMNT). There are also given upper limits (MNTLIM) for each assembly and maintenance type. These limits are given either in or kilometer or in time intervals. According to the position of a given assembly (in a train) and of its assembly type, we can find out how difficult it is to exchange the assembly in the train with another assembly of the same type (EXDIF). Each station in the set has defined costs (COST) and required time for each maintenance type (MNTDUR).

For each assembly and maintenance type one can observe where and when that assembly was last maintained according to that type. Different topologies and shunting possibilities of each station allow or does not allow exchange of two assemblies within certain time limits. This time need not be the same for nights and for days.

The functions below are explained, ie., narrated, after their definition.

Formal Model:

```
    extend SCHEDULE_S with
    class
        type
            IMP,
            DIF,
            COST,
            MT ==
                regular_check | out_clean |
                in_clean | diesel_fuel,

            ACTION == WR | ER | MNT,

            WR = R,
            ER = R,
            MNT = DT × S × DT × MT,

            MNTPLAN = ACTION*,
    REQMNT = AT $\overrightarrow{m}$ MT-set,
    MNTLIM =
        (AT × MT) $\overrightarrow{m}$ (TI | KMS)
    MNTIMP = (AT × MT) $\overrightarrow{m}$ IMP,
    MNTDUR = AT $\overrightarrow{m}$ (MT $\overrightarrow{m}$ (S $\overrightarrow{m}$ TI)),
    EXDIF = (AT × POS) $\overrightarrow{m}$ DIF

value
    max_imp : IMP,
    req_mnt : REQMNT,
    mnt_lim : MNTLIM,
    mnt_imp : MNTIMP,
    mnt_dur : MNTDUR,
    exc_dif : EXDIF,

    obs_LastMnt: A × MT → (TI|KMS),
    obs_MinExDTime: S × DT→ TI,
    obs_MinExNTime: S × DT → TI,
    obs_ExDCost: S × DT $\xrightarrow{\sim}$ COST,
    obs_ExNCost: S × DT $\xrightarrow{\sim}$ COST,
    obs_MntCost: N×AT×MT×S×DT$\xrightarrow{\sim}$COST,

    $\underline{<}$ : IMP × IMP → Bool,
    $\underline{<}$ : DIF × DIF → Bool,

    RemDist : A × MT × DT → KMS
    RemDist(a, mt, t) ≡
        obs_LastMnt(a, mt) +
        mnt_lim(obs_AType(a), mt) −
        obs_Km(a, t),

    RemTime : A × MT × DT → TI
    RemTime(a, mt, t) ≡
        obs_LastMnt(a, mt) +
        mnt_lim(obs_AType(a), mt) − t,

    MStas : N × AT × MT → Sta-set
    MStas(n, at, mt) ≡
        {s | s : Sta •
            s ∈ obs_Sta(n) ∧
            s ∈ dom mnt_dur(at)(mt)},

    MTypes : N × S × AT → MT-set
    MTypes(n, s, at) ≡
        {mt | mt : MType •
            mt ∈ dom mnt_dur(at) ∧
            s ∈ dom mnt_dur(at)(mt) ∧
            s ∈ MStas(n, at, mt)},

    isMPos : N × AP × AT × MT → Bool
    isMPos(n, p, at, mt) ≡
        ∃ s : S, i : Nat •
            s ∈ MSta(n, at, mt) ∧
            {i, i+1} ⊆ inds p ∧
            s = ArrS(p(i)) ∧
            ArrT(p(i)) +
                mnt_dur(at)(mt)(s) <
                DepT(p(i+1)),

    isInSta: AP × S × DT → Bool
    isInSta(p, s, t) ≡
        ∃ i:Nat •
            {i, i+1} ⊆ inds p ⇒
            ArrT(p(i)) ≤ t ≤
            DepT(p(i+1)) ∧
            s = ArrS(p(i)) = DepS(p(i+1)),

    DisToMS:
        N × AP × AT × MT $\xrightarrow{\sim}$ (TI|KMS)
    DisToMS(n, p, at, mt) ≡
        if DepS(hd p) ∈ MStas(n, at, mt)
        then 0
        else
            RideDis(hd p) +
            DisToMS(n, tl p, at, mt)
        end,

    MntUrg : A × MT × DT → IMP
    MntUrg(a, mt, t) ≡
        if RemDist(a, mt, t) ≤ 0
        then max_imp
        else
            mnt_imp(obs_AType(a), mt) /
            RemDist(a, mt, t)
        end

    TMax : TS → DT
    TMax(ts) as tmax post
        ∀ j : J, i : Nat •
            j ∈ JSet(ts) ∧ i ∈ inds j ⇒
                ArrT(j(i)) < tmax

axiom
    (α10) ∀ i : IMP • i $\underline{<}$ max_imp,
    (α11) ∀ n : N, at:AT •
            ∃ s : S, mt : MT ⇒
            s ∈ obs_S(n) ∧
            mt ∈ MTypes(n, s, at) ∧
            s ∈ MStas(n, at, mt),
    (α12) ∀ at : AT, mt : MT •
            mt ∈ req_mnt(at) ⇒
            0 < mnt_imp(at, mt) ≤
                max_imp,
    (α13) ∀ at : AT, mt : MT •
            mt ∉ req_mnt(at) ⇒
                mnt_imp(at, mt) = 0,
    (α14) ∀ at : AT •
            exc_dif(at, sol) $\underline{<}$
            exc_dif(at, las) $\underline{<}$
            exc_dif(at, fir) $\underline{<}$
            exc_dif(at, mid),
end
```

We now explain the above planning functions.

(`RemDist`) and (`RemTime`) calculate remaining distance to the maintenance of a given type either in kilometers or in time interval, for a given assembly at a given time. (`MStas`) yields the set of stations that are maintenance stations for a given assembly and maintenance type, and in a given network. (`MTypes`) yields all maintenance types, which a given assembly can undergo at a given station. (`isMPos`) checks if there is a maintenance opportunity in a given plan, for given assembly and maintenance types. (`isInSta`) checks if, according to a given plan, a given assembly is at a given station and at a given time. (`DisToMS`) espresses the "distance" to a maintenance opportunity, in a given plan for a given assembly and maintenance type. (`MntUrg`) expresses the importance of undergoing maintenance of a given type at a given time for a given assembly. (`TMax`) espresses the total time horizon of a given traffic schedule.

3. PLANNING

Every day, last-moment changes and updates must be applied to the previously planned rolling stock roster. In this section we describe two basic functions for rolling stock maintenance routing: Generation of changes to a rolling stock roster and application of changes to a roster.

3.1 Generation of Rolling Stock Roster Changes

Given a rolling stock roster and a net we can express sets of necessary rolling stock roster changes. An example of rolling stock roster for several consecutive days is shown in figure 1.

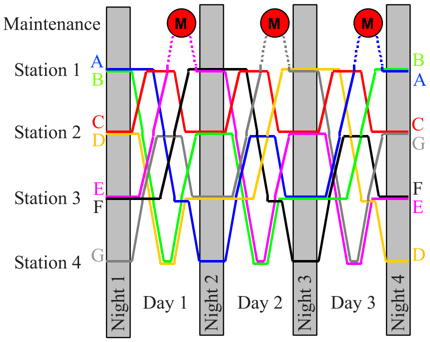

Fig. 1: The Original Plan

Each change set is composed from three different types of changes. They are called: Day Change, Night Change and Empty Ride.

Day Change is composed of two assemblies, of a station and a time, where and when the interchange between these two assemblies takes place. Day change may occur when the first assembly is an 'urgent' assembly (needs to undergo maintenance check in couple of days) and the second assembly has a maintenance station in the plan, and when both assemblies are in the same station at the same time, during a day time, and there is enough time to interchange them.

In Figure 2 assembly C is exchanged at station 1 with assembly G — designated, thus, to reach the maintenance station in two days.

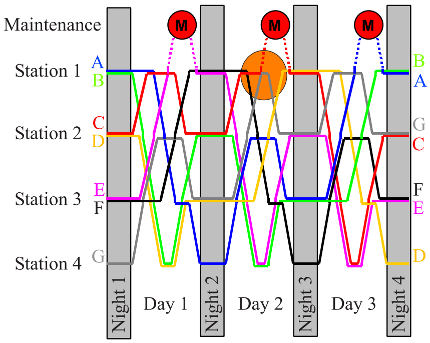

Fig. 2: Example of 'Day Change'

Night Change is quite similar to the day change. The main difference is in the time when this change is applied. Night changes may occur when the first assembly is an 'urgent' assembly and the second assembly has a maintenance station in the plan, and when both assemblies are in the same depot or station during the same night. It is the least expensive way in which to add maintenance into the assembly plan.

In Figure 3 assembly D is exchanged at station 3 with assembly G — designated, thus, to reach the maintenance station the second night.

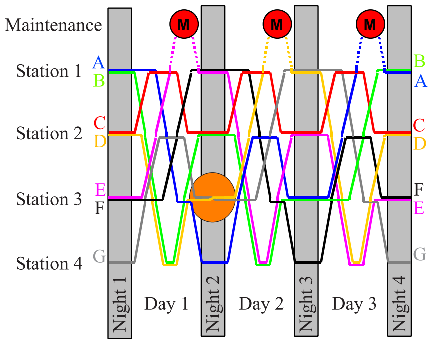

Fig. 3. Example of 'Night Change'

Empty Ride is the last possible change that can be applied to the rolling stock roster. This change must be applied, when there is no day or night change possible (ie., when there is no such situation in the rolling stock roster where two assemblies are in the same station and time during their operations). In that case, two additional rides have to be added into the plan. An 'Empty Ride' is composed of two assemblies and two additional rides for these assemblies. In general, an ‘Empty Ride’ is always possible, but it has the highest cost.

In Figure 4 assembly B is routed from station 2 to station 3 and assembly G in opposite direction.

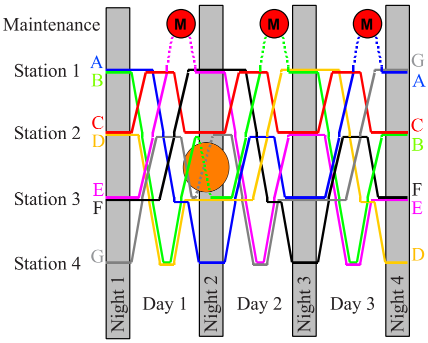

Fig. 4. Example of Empty Ride

Formal model:

```
scheme CHANGES_S =
    extend MAINTENANCE_S
    class
        type
            C == DC | NC | ER
            DC == mkDC(a1:A,a2:A,t:DT,s:S)
            NC == mkNC(a1:A,a2:A,t:DT,s:S)
            ER == mkER(a1:A,a2:A,r1:R,r2:R)
            CS = C-set

        value
            NPlan:
                TS × C × (DT × DT) → A
            NPlan(ts,c,(t,t′)) ≡
                APlan(ts,UAID(c),(t,CT(c)))
                  APlan(ts,SAID(c),(CT(c),t′))

             UAID: C → AID
             UAID(c) ≡
             case c of
                 mkDC(a,_,_,_) → obs_AsmID(a),
                 mkNC(a,_,_,_) → obs_AsmID(a),
                 mkER(a,_,_,_) → obs_AsmID(a)
             end

             SAID: C → AID
             SAID(c) ≡
             case c of
                 mkDC(_,a,_,_) → obs_AsmID(a),
                 mkNC(_,a,_,_) → obs_AsmID(a),
                 mkER(_,a,_,_) → obs_AsmID(a)
             end

             CT: C → DT
             CT(c) ≡
             case c of
                 mkDC(_,_,t,_) → t,
                 mkNC(_,_,t,_) → t,
                 mkER(_,_,r,_) →
                     let ((_,_),(_,t),_)=r
                     in t end
             end

             CS: C → S
             CS(c) ≡
             case c of
                 mkDC(_,_,_,s) → s,
                 mkNC(_,_,_,s) → s,
                 mkER(_,_,r,_) →
                     let ((_,s),(_,_),_)=r
                     in s end
             end

             MinExT: C → TI
             MinExT(c) ≡
             case c of
                 mkDC(_,_,t,s) →
                     obs_MinExDTime(s,t)
                 mkNC(_,_,t,s) →
                     obs_MinExNTime(s,t)
                 mkER(_,_,r,_) →
                     let ((_,s),(_,t),_) = r
                     in obs_MinExDTime(s,t)
                     end
             end
    end
```

3.2 Generating changes

We now present the main function of this paper: (**gen_Changes**). This function expresses all possible sets of changes from a given net, traffic schedule and planning period. These change sets are limited by several constraints (ie., post conditions). All changes which are in the generated set must be possible, necessary and sufficient.

The ‘possible’ condition checks whether two assemblies are of the same type and whether these two assemblies are active assemblies in a given time period. Then it is checked if it is a case that both assemblies are in the same station at the same time for a long enough period according to their plans.

The change is ‘necessary’ when remaining distance to becomin an ‘urgent’ assembly is smaller than the distance to the maintenance station in the plan of this assembly.

The ‘sufficiency’ condition checks if an ‘urgent assembly’ can arrive at its maintenance station before exceeding its (time or kilometer) limit according to the new plan.

```
scheme PLANNING_S =
    extend CHANGES_S
    class
        type
        value

            gen_Chgs: N×TS×(DT×DT) ~→ CS-set
            gen_Chgs(n, ts, (t, t′)) as css
            pre
                ∃ a : A, mt : MT •
                 DisToMS(APlan(ts, obs_AID(a),
                     (t, t′)), obs_AType(a), mt)
                      > RemDis(a, mt, t)
            post
                ∀ cs : CS, c : C • c
                ∈ cs ∧ cs ∈ css
                ⇒ isPossible(ts, c, (t, t′)) ∧
                    isNecessary(ts, c, (t, t′)) ∧
                    isSufficient(ts, c, (t, t′))

                isPossible:
                    TS × C × (DT×DT) → Bool
                isPossible(ts, c, (t, t′)) ≡
                    obs_AType(UA(c)) =
                        obs_AType(SA(c)) ∧
                    {UA(c), SA(c)} ⊆
```

```
          ActAsms(ts, (t, t′)) ∧
     MntUrg(UA(c), mt, t′) >
          MntUrg(SA(c), mt, t′) ∧
     isInSta(APlan(ts, UAID(c),
          (t, t′)), CS(c), CT(c)) ∧
     isInSta(APlan(ts, UAID(c),
          (t, t′)), CS(c), CT(c) +
          MinExTU(c)) ∧
     isInSta(APlan(ts, SAID(c),
          (t, t′)), CS(c), CT(c)) ∧
     isInSta(APlan(ts, SAID(c),
          (t, t′)), CS(c), CT(c) +
          MinExTU(c)),


isNecessary:
     TS × C × (DT×DT) → Bool
isNecessary(ts, c, (t, t′)) ≡
     ∃ mt: M •
          DisToMS(APlan(ts,
          UAID(c), (t, t′)),
          obs_AType(UA(a)), mt) >
          RemaDist(a, mt, t),


isSufficient:
     TS × C × (DT×DT) → Bool
isSufficient(ts, c, (t, t′)) ≡
     ∀ mt: MT •
     mt ∈ req_mnt(obs_AType(a))
     ⇒
     DisToMS(NPlan(ts, c, (t, t′)),
     obs_AsmType(UA(c)), mt) ≤
     RemaDist(UA(c), mt, t)
end
```

3.3 Applying changes

Once a day, specified changes are applied to the rolling stock roster traffic schedule. We get a new traffic schedule, which is used as an input to next day's operations. There can be only one difference between the old and the new traffic schedule: some trains in the new schedule can be served by different assemblies of the same type. That means, that assembly plans are modified as shown on Figure 5.

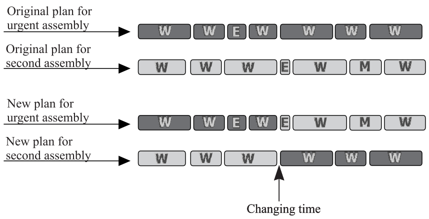
Original plan for urgent assembly
W W E W W W W
Original plan for second assembly
W W W E W M W
New plan for urgent assembly
W W E W E W M W
New plan for second assembly
W W W W W W
Changing time

Fig. 5: Plan modification

The correct solution is when all assemblies which require maintenance in the given period, after application of calculated changes in the new traffic schedule can reach the maintenance station in the remaining distance.

```
scheme UPDATE_S =
     extend PLANNING_S
     class
          value
               AppChgs: TS×CS×(DT×DT)→TS
               AppChgs(ts, cs, (t, t′)) as ts′
               post
                    ∀ c: C • c ∈ cs ⇒
                    let aid1 = UAID(c),
                         aid2 = SAID(c)
                    in
                         APlan(ts, aid1, (t, CT(c)))⌢
                         APlan(ts, aid2, (CT(c), t′)) =
                              APlan(ts′, aid1, (t, t′))
                         ∧
                         APlan(ts, aid2, (t, CT(c)))⌢
                         APlan(ts, aid1, (CT(c), t′)) =
                              APlan(ts′, aid2, (t, t′))
                    end
                    ∧
                    ∀ a: A, mt: MT, cs: CS •
                         a ∈ ActAsms(ts, (t, t′)) ∧
                         mt ∈ req_mnt(obs_AType(a)) ∧
                         MntUrgency(a, mt, t) > 0 ⇒
                              ∃ c: C •
                              c ∈ cs ∧ a = UA(c) ∧
                              DisToMS(APlan(ts′,
                                   obs_AId(a), (t, t′)),
                                   obs_AType(a),mt) ≤
                                        RemDis(a, mt, t)
     end
```

4. SUMMARY

A formal model of maintenance routing have been shown. The task was divided into two basic steps:

- generation of possible, necessary and sufficient changes of the traffic schedule
- application of these changes to the traffic schedule

We emphasize that we formally characterized schedules, assembly plans and changes in the plan to meet maintenance demands. On the basis of such formal space software we can now prescribe requirements.

In the future, this formal approach, we claim, should lead to deeper, better and easier integration of all railway optimization applications in all the tasks of railway planning, monitoring and control processes.

5. BIBLIOGRAPHY

We refer to two papers in these proceedings: (Strupchanska et al., 2003) and (Bjørner, 2003). In (Strupchanska et al., 2003) a model, very much along the lines of the present paper, is presented of (what ultimately becomes an operations research problem, namely) train staff (ie., crew) rostering. In (Bjørner, 2003) arguments are presented for developing software from domain descriptions, as here, via requirements, to software design.

References

Bjørner, D. (2000). *Formal Software Techniques in Railway Systems.* In Schnieder, E., editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik. Invited talk.[2]..

Bjørner, D. (2003). *New Results and Trends in Formal Techniques & Tools for the Development of Software for Transportation Systems — A Review.* In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems.* L'Harmattan Hongrie. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.

Bjørner, D. (2003–2004). *The SE Book: Principles and Techniques of Software Engineering*, volume I: Abstraction & Modelling (750 pages), II: Descriptions and Domains (est.: 500 pages), III: Requirements, Software Design and Management (est. 450 pages). [Publisher currently (March 2003) being negotiated].[3].

George, C., Haff, P., Haxthausen, A., Havelund, K., Milne, R., Nielsen, C.B., Prehn, S., and Wagner, K.R. (1992). *The RAISE Specification Language.* The BCS Practitioners Series. Prentice-Hall International.

Karras, P. and Bjørner, D. (2002). *Train composition and decomposition: From passenger statistics to schedules.* Technical report, Informatics and Mathematical Modelling, Building 322, Richard Petersens Plads, Technical University of Denmark, DK–2800 Kgs.Lyngby, Denmark.

Kroon, L. and Fischetti, M (2000). *Crew Scheduling for Netherlands Railways Destination: Customer.* ERIM Report Series: Research In Management, Netherlands.

Kroon, L. (2001). *Models for rolling stock planning.* Research report, Univ. of Utrecht, Netherlands.

Maróti, G. (2001). *Maintenance Routing.* Research report, CWI, Amsterdam and NS Reizigers, Utrecht, Netherlands. The EU IST Research Training Network AMORE: Algorithmic Models for Optimising Railways in Europe: `www.inf.uni--konstanz.de/algo/amore/`. Contract no. HPRN-CT-1999-00104, Proposal no. RTN1-1999-00446

Strupchanska, A. K., Pěnička, M., and Bjørner, D. (2003). *Railway staff rostering.* In FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems. L'Harmattan Hongrie. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.

[2] http://www.imm.dtu.dk/~db/documents/2ifacpaper.ps
[3] http://www.imm.dtu.dk/~db/TheSEBook