Design-Time Railway Capacity Verification using SAT modulo Discrete Event Simulation

Bjørnar Luteberget / Koen Claessen / Christian Johansen

Chalmers, 8 Oct, 2018

UiO : University of Oslo

RailCOMPLETE

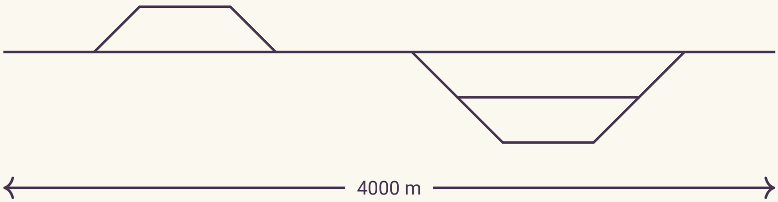
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Agenda, Gothenburg

1. Overview of capacity verification work with Koen, Christian.
2. Work in progress: synthesis/optimization of signalling designs. Suggestions?

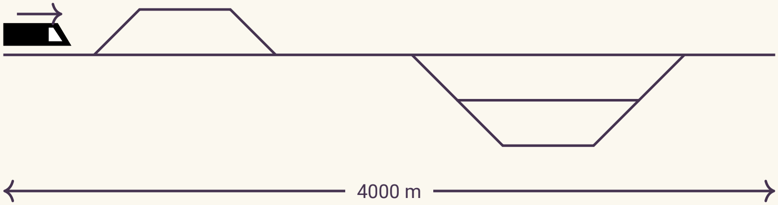
Railway control systems

Constructing a new railway line starts with a track plan:



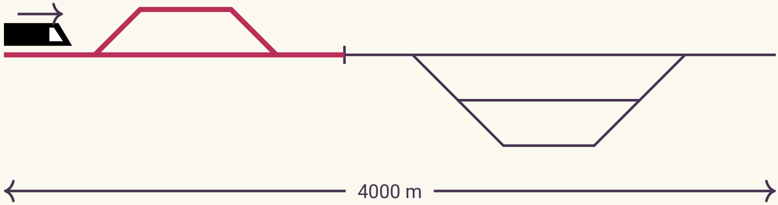
Railway control systems

Constructing a new railway line starts with a track plan:



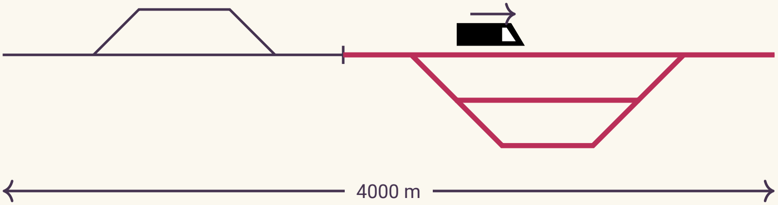
Railway control systems

By adding detectors, we can allocate smaller pieces of tracks to the train:



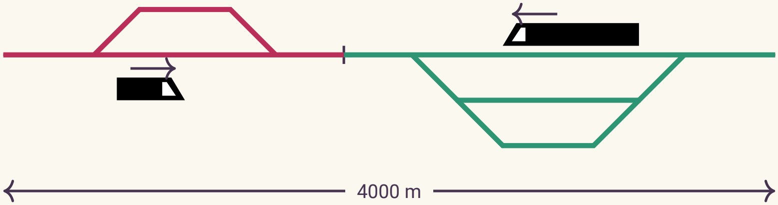
Railway control systems

By adding detectors, we can allocate smaller pieces of tracks to the train:



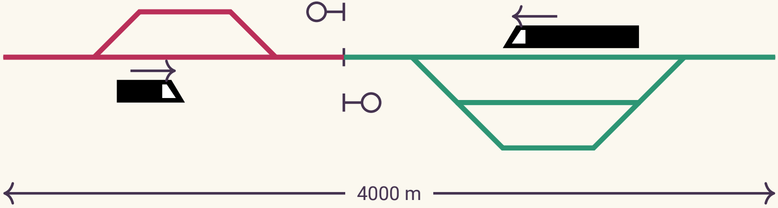
Railway control systems

Now, other trains can occupy different sections.



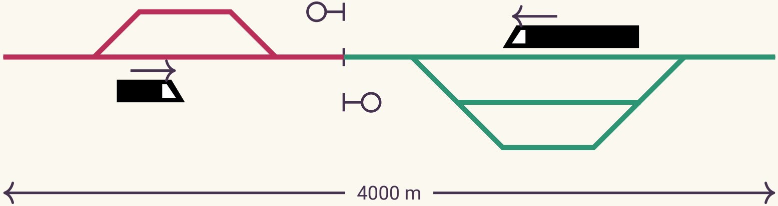
Railway control systems

We add signals to indicate to drivers when they can proceed.



Railway control systems

This situation is in principle safe, but is it a good design?



Requirements

Railway contruction projects have many clear requirements:

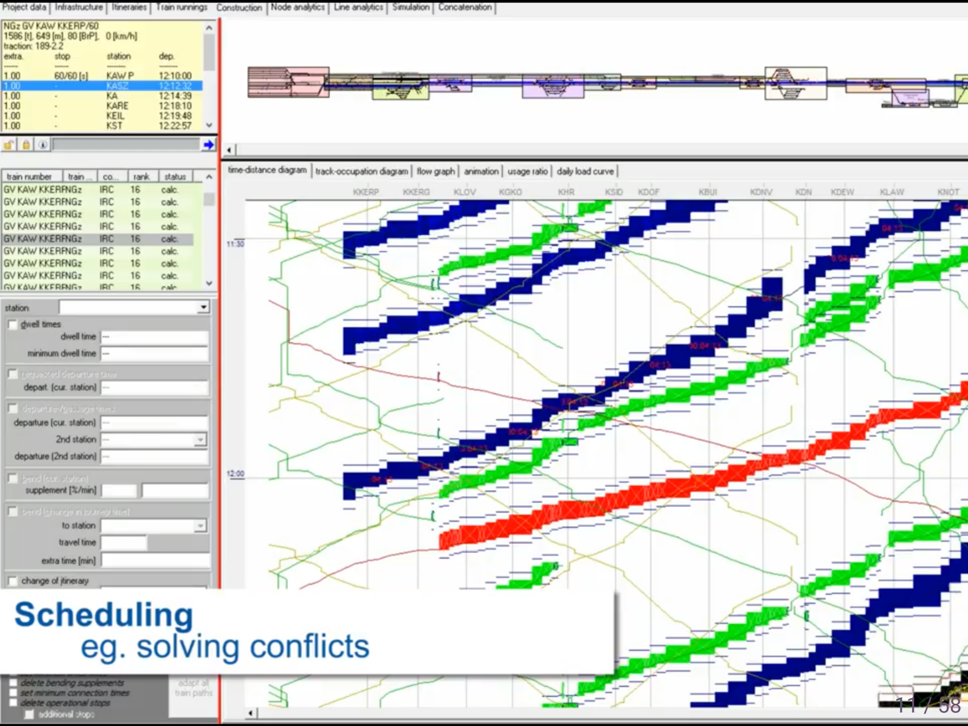
- static
- written down in regulations, specifications

However, requirements on traffic, delays,

- dynamic
- not formalized, trial and error, miscommunication

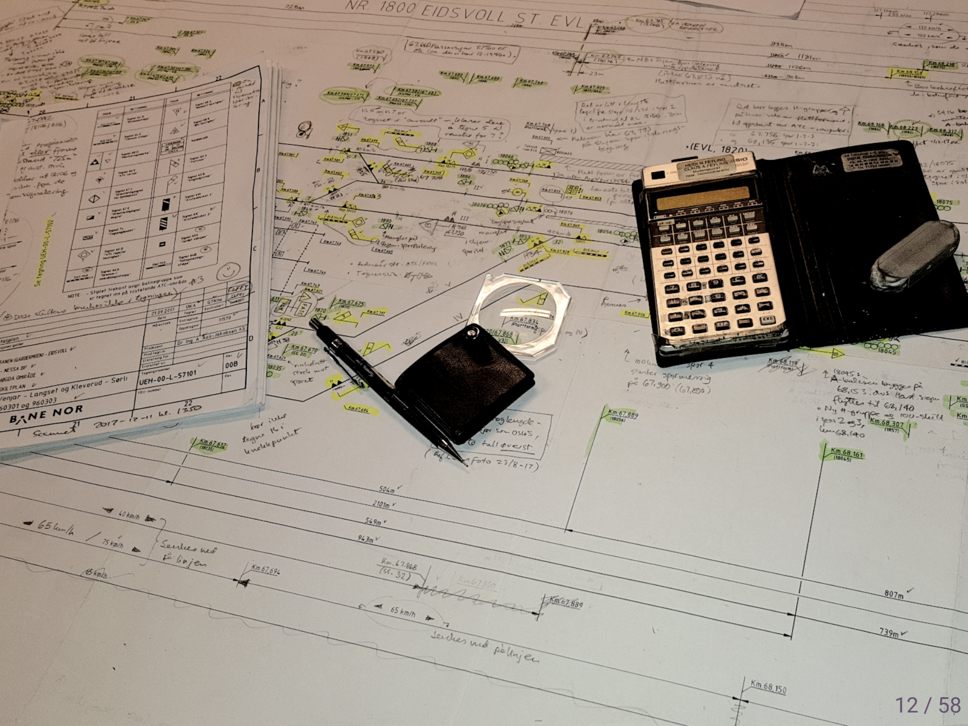
Two solutions in practice:

1. Whole-network time table analysis: a whole discipline in itself – complicated theory and software
2. Manual, ad-hoc analysis: varying quality, little documentation, low repeatability.

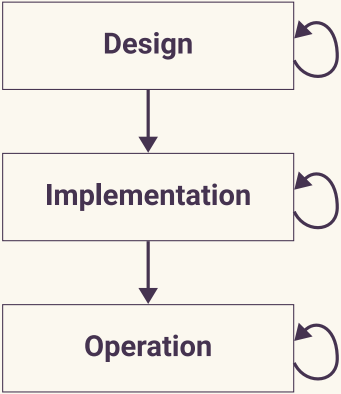


Scheduling

eg. solving conflicts


NR. 1800 EIDSVOLL ST. EVL
(EVL, 1820)
UEH-00-L-57101
BANE NOR
CASIO

Design-implementation-operation

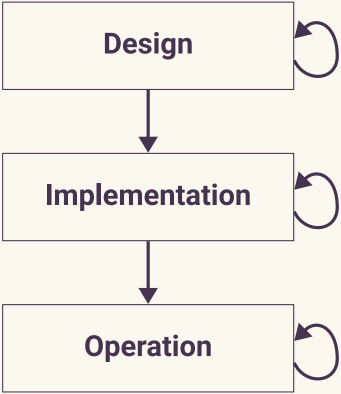


?

Formal methods for verifying correctness (safety).

Railway optimization for network-wide timetables.

Design-implementation-operation



Agile, fast verification methods with suitable, small specifications.

Formal methods for verifying correctness (safety).

Railway optimization for network-wide timetables.

Specification capture

Railway engineers gave us examples of performance properties that governed their designs.

Typical categories:

1. Running time (get from A to B)
   - Similar to a simulation test, but smaller specification.
2. Frequency (several consecutive trains)
   - Route trains into alternate tracks.
3. Overtaking
   - Let one train wait on a side track while another train passes.
4. Crossing

Capacity specifications

Local requirements suitable for construction projects.

- Operational scenario $S = (V, M, C)$:
- Vehicle types $V = \{(l_i, v_i^{\text{max}}, a_i, b_i)\}$, defined by length, max velocity, max accel, max braking.
- Movements $M = \{(v_i, \langle q_i \rangle)\}$, defined by vehicle type v and ordered sequence of visits $\langle q_i \rangle$.
  - Each visit $q_i = (\{l_i\}, t_d)$ is a set of alternative locations l_i and an optional dwelling time t_d.
- Timing constraints $C = \{(q_a, q_b, t_c)\}$ which orders two visits and sets a maximum time from the first to the second $t_{q_a} < t_{q_b} < t_{q_a} + t_c$. The maximum time constraint can be omitted ($t_c = \infty$).

Constraints

Verification of these specifications would involve finding satisfying train trajectories and control system state.

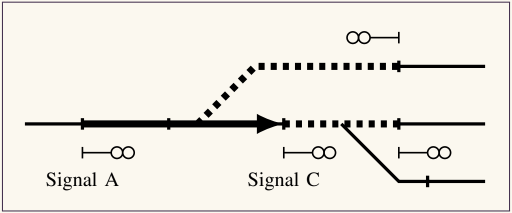
Also, constrained by:

- 1 - Physical infrastructure
- 2 - Allocation of resources (collision safety)
- 3 - Limited communication
- 4 - Laws of motion

Constraints (2) Allocation of resources

Avoiding collisions by exclusive use of resources.

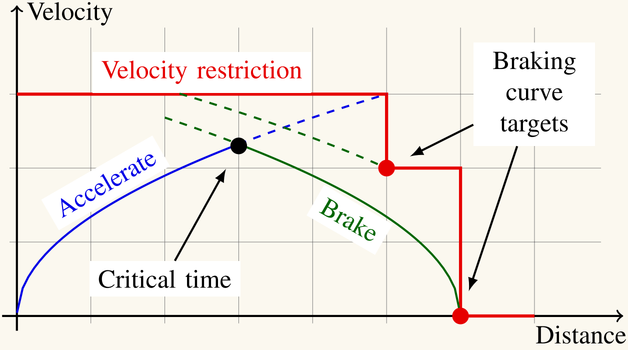
An elementary route is the smallest unit of resources that can be allocated to a train.



Constraints (4) Laws of motion

Trains move within the limits of given maximum acceleration and braking power. Train drivers need to plan ahead for braking so that the train respects its given movement authority and speed restrictions at all times.

$$v - v_0 \leq a\Delta t, \qquad v^2 - v_i^2 \leq 2bs_i.$$



Automated verification

Design-time capacity verification amounts to planning in a mixed discrete/continuous space.

Some suggestions:

- PDDL+, a planning domain description language for modelling mixed discrete-continuous planning domains.
- SMT with non-linear real arithmetic.
- dReal: δ-complete decision procedures for first-order logic formulas over the real numbers.

Using these tools/techinques and straight-forward modeling did not make our problem manageable on relevant scales.

Dispatch vs. driver

Split the planning work into two separate points of view:

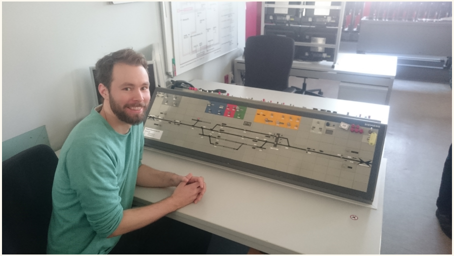
Dispatcher

Train driver

Dispatch vs. driver

Split the planning work into two separate points of view:

Dispatcher

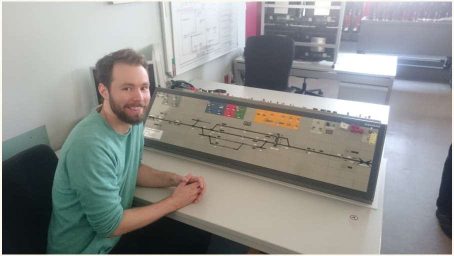


Train driver

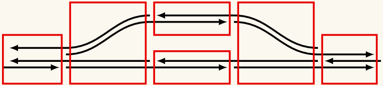
Dispatch vs. driver

Split the planning work into two separate points of view:

Dispatcher



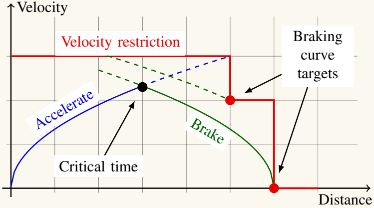
↓



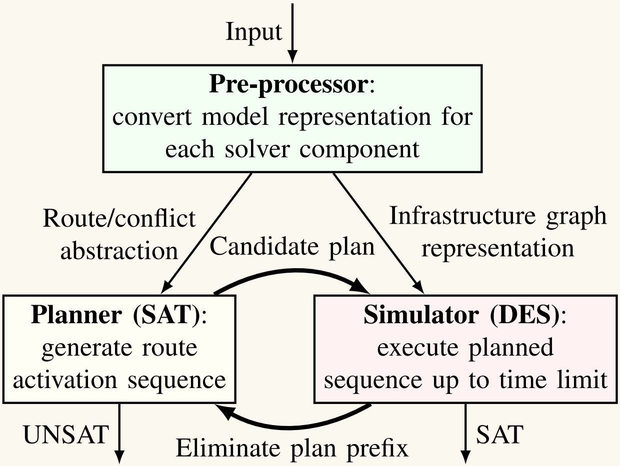
Elementary routes and their conflicts

Train driver

↓

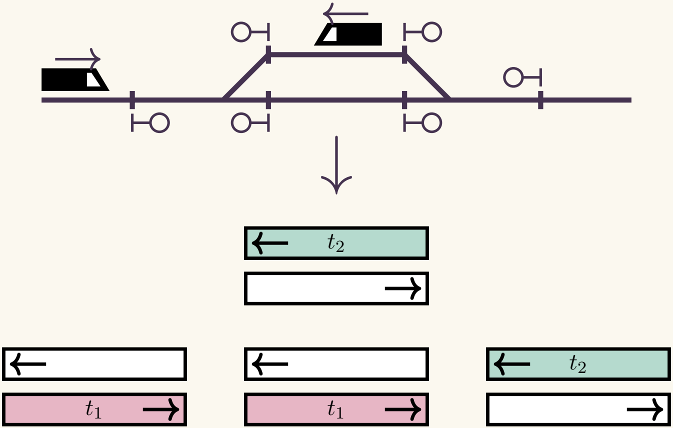


Solver architecture



SAT encoding of dispatch planning

General idea: represent which train occupies which elementary route in each of a sequence of steps.



SAT encoding

Planning as bounded model checking (BMC). Build planning steps as needed using incremental SAT solver interface.

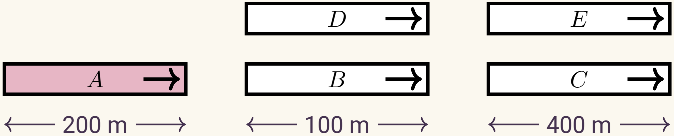
Movement correctness:

- Conflicting routes are not active simultaneously $\text{conflict}(r_1, r_2) \Rightarrow o^i_{r_1} = \text{Free} \vee o^i_{r_2} = \text{Free}$.
- Elementary route allocation is consistent with train movement: $(o^i_r \neq t \wedge o^{i+1}_t = t) \Rightarrow \bigvee \{o^{i+1}_{r_x} = t \mid \text{route}(r_x), \text{entry}(r) = \text{exit}(r_x)\}$

Satisfy specification:

- Visits happen in order (timing requirement is measured on simulation).

Freeing



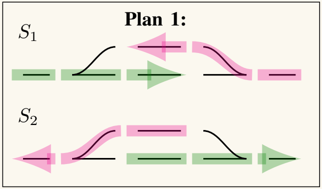
If A holds a train t of length 200.0 m, freeing A is constrained by:

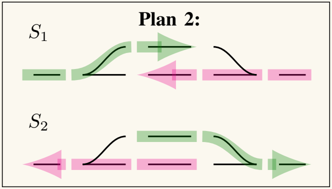
$$A^i \Rightarrow \left(A^{i+1} \vee (B^i \wedge C^i) \vee (D^i \wedge E^i)\right).$$

Eliminate equivalent solutions

- Can free $\Longrightarrow$ must free
- Can allocate $\Longrightarrow$ must allocate
- Exception to allocation: deferred progress
  a train may waiting for a conflict to be resolved, even if the conflict starts in the future.

  Crossing example: **exactly two** solutions:





Discrete event simulation

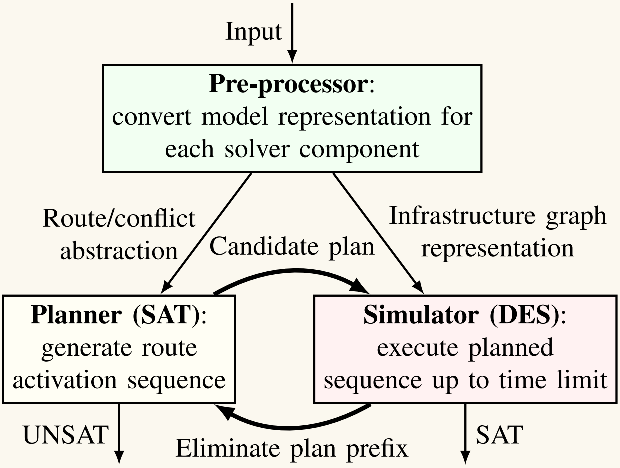
Initialize a world, and let processes mutate the world coordinated by a global clock.

- Scheduler: priority queue of events, ranked by time.
- ```
  enum PState { Finished, Wait([EventId]) }
  trait Process<T> {
   fn resume(&mut self, s:&mut Sim<T>) -> PState; }
  ```
- Observable values fire events when changed.

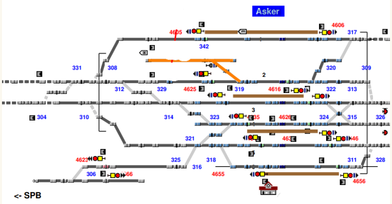
Railway simulation uses the following processes:

- Elementary route activation (subproc.: turn switch)
- Resource release (observe detectors)
- Train driver (observe signals, choose accel/brake).
  ```

Solver architecture



Case studies



Infrastructure	Property	Result	n_{DES}	t_{SAT}	t_{DES}	t_{total}
Simple (3 elem.)	Run.time	Sat.	1	0.00	0.00	0.00
	Crossing	Unsat.	0	0.00	0.00	0.00
Two track (14 elem.)	Run.time	Sat.	1	0.01	0.00	0.01
	Frequency	Sat.	1	0.01	0.00	0.01
	Overtaking 2	Sat.	1	0.00	0.00	0.01
	Overtaking 3	Unsat.	0	0.01	0.00	0.01
	Crossing 3	Unsat.	0	0.01	0.00	0.01
Kolbotn (BN) (56 elem.)	Run. time	Sat.	2	0.01	0.00	0.02
	Overtake 4	Sat.	1	0.05	0.00	0.06
	Overtake 3	Unsat.	0	0.05	0.00	0.06
Eidsvoll (BN) (64 elem.)	Run. time	Sat.	2	0.01	0.00	0.02
	Overtake 2	Sat.	1	0.08	0.00	0.08
	Crossing 3	Sat.	1	0.04	0.00	0.04
	Crossing 4	Unsat.	0	0.21	0.00	0.21
Asker (BN) (170 elem.)	Overtaking 2	Sat.	1	0.20	0.00	0.21
	Overtaking 3	Unsat.	1	0.73	0.00	0.74
	Crossing 4	Sat.	0	0.75	0.00	0.77
Arna (CAD) (258 elem.)	Run. time	Sat.	1	0.02	0.00	0.04
	Overtaking 2	Sat.	1	0.50	0.00	0.51
	Overtaking 3	Sat.	1	1.43	0.00	1.45
	Crossing 4	Sat.	1	1.73	0.00	1.74
Gen. 3x3 (74 elem.)	High time	Sat.	1	0.01	0.00	0.01
	Low time	Unsat.	27	0.18	0.01	0.19
Gen. 4x4 (196 elem.)	High time	Sat.	1	0.01	0.00	0.03
	Low time	Unsat.	256	2.08	0.26	2.34
Gen. 5x5 (437 elem.)	High time	Sat.	1	0.06	0.00	0.09
	Low time	Unsat.	3125	38.89	4.35	43.24

TABLE I: Verification performance on test cases, including Bane NOR (BN) and RailCOMPLETE (CAD) infrastructure models. The number of elementary routes (*elem.*) is shown for each infrastructure to indicate the model's size. n_{DES} is the number simulator runs, t_{SAT} the time in seconds spent in SAT solver, t_{DES} the time in seconds spent in DES, and t_{total} the total calculation time in seconds.

Future work

- Improved abstraction refinement? Would need more difficult cases to solve.
- Support for turning trains and loops in the infrastructure.
- Interface to more comprehensive simulation software?
- Depends on feedback from engineers.

Future work

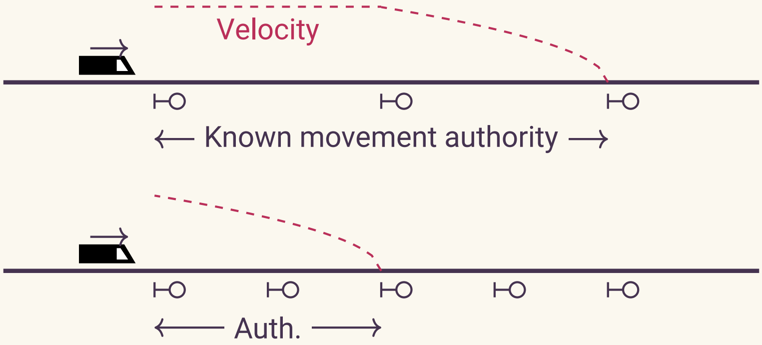
- Improved abstraction refinement? Would need more difficult cases to solve.
- Support for turning trains and loops in the infrastructure.
- Interface to more comprehensive simulation software?
- Depends on feedback from engineers.

- Fast and fully automatic verification could be a basis for **design synthesis**.

Design synthesis

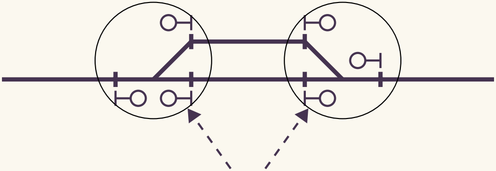
Create signal and detector placement for a given track plan.

- Schedulability: is the dispatch possible (add more signals and detectors)
- Timing: running time can become worse with more signals. Signal information only carries across two signals ("pre-signalling").



Idea for approach to synthesis/optimization

1. Maximal schedulability design.

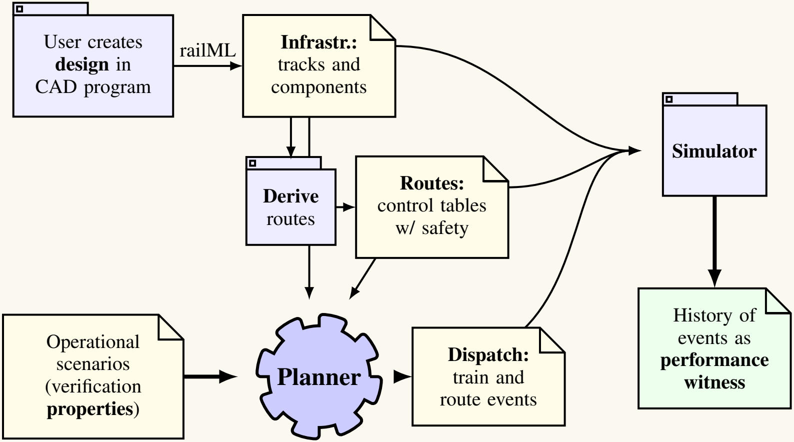


Guard every branch

2. Run dispatch planner to see which signals are not needed (optimization).
3. Add/remove signals on non-branching sections to improve timing.
4. Move signals locally to improve timing (local search).

Thanks for listening!

Solver architecture



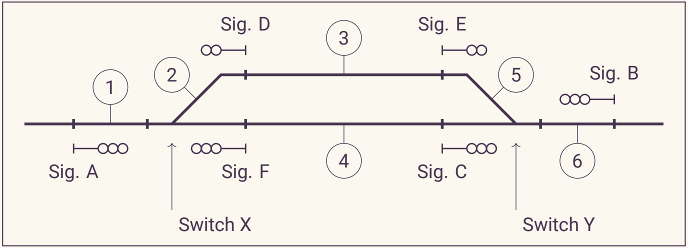
RailCons project: automated verification

Project objectives:

- Verify that railway signalling and interlocking designs comply with regulations.
- Provide tools which allow railway engineers to perform such verification as part of their daily routine ("lightweight verification").

> "Formal methods will never have a significant impact until they can be used by people that don't understand them."
>
> — (attributed to) Tom Melham

Models: railway signalling and interlocking designs



(a) Track and signalling component layout

Route	Start	End	Sw. pos	Detection sections	Conflicts
AC	A	C	X right	1, 2, 4	AE, BF
AE	A	E	X left	1, 2, 3	AC, BD
BF	B	F	Y left	4, 5, 6	AC, BD
BD	B	D	Y right	3, 5, 6	AE, BF

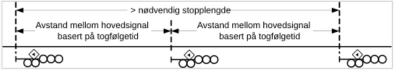
(b) Tabular interlocking specification

Properties: technical regulations

- In our case study: Norwegian regulations from national railways (Bane NOR)
- Static kind of properties, often related to object properties, topology and geometry (example on next slide)

https://trv.jbv.no/wiki/Signal/Prosjektering/Lyssignal

e) Dersom nødvendig stopplengde er lengre enn avstanden mellom to etterfølgende hovedsignal, skal det benyttes gjennomsignalering ved hjelp av ATC (Signal/Prosjektering/ATC), se Figur 7.



Figur 7: Forsignalplassering og gjennomsignalering

f) Et forsignal skal plasseres på foregående hovedsignals mast dersom avstanden mellom det tilhørende hovedsignalet og det foregående hovedsignalet er ≤ 2200 meter.

g) Mellom et forsignal og det tilhørende hovedsignalet skal det ikke plasseres andre hoved- eller forsignal.

h) Et forsignal skal plasseres slik at siktavstanden oppfyller kravene til enten “brutt sikt” eller til “ubrutt sikt” i Tabell 4:

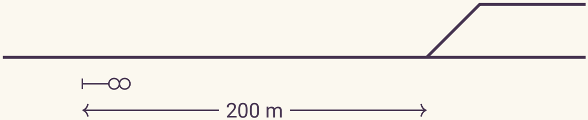
Tabell 4: Siktkrav til forsignal

Sikt	Strekningens høyeste tillatte kjørehastighet [km/h]																		
	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125	≥130
	Siktavstand [m]																		
Ubrutt	78	88	97	107	117	126	136	146	156	165	175	185	194	204	214	224	233	243	250

Properties: technical regulations

Example from regulations:

- A *home main signal* shall be placed at least 200 m in front of the first controlled, facing switch in the entry train path.

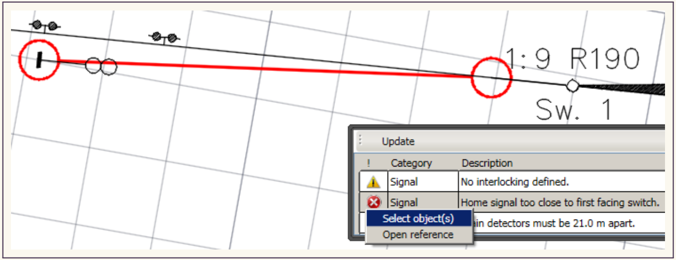


- Can be classified as follows:
  - Object properties
  - Topological layout properties
  - Geometrical layout properties
  - Interlocking properties

Datalog verification tool

- Prototype using XSB Prolog tabled predicates, front-end is the RailCOMPLETE tool based on Autodesk AutoCAD
- Rule base in Prolog syntax with structured comments giving information about rules

```
%| rule: Home signal too close to first facing switch.
%| type: technical
%| severity: error
homeSignalBeforeFacingSwitchError(S,SW) :-
    firstFacingSwitch(B,SW,DIR),
    homeSignalBetween(S,B,SW),
    distance(S,SW,DIR,L), L < 200.
```



Challenge: participatory verification

Challenge: Users (railway engineers) are not experts in verification techniques, so how can they

- build models of the systems to be verified?
- write properties in the verifier's input language?
- interpret the output of the verifier when violated properties are found?

Input to verification:

- Models: CAD extended with structured railway data (familiar to engineers, user-friendly)
- Properties: Datalog (unfamiliar to engineers, not user-friendly enough)

... consider another verification property input language?

REMU project – Chalmers/GU Gothenburg

REMU project: Reliable Multilingual Digital Communication –

- Goals (among others): grammar development, testing, analysis.
- Tools: Grammatical Framework – Programming language for multilingual grammar applications.

- Controlled natural language
  Controlled natural languages (CNLs) are subsets of natural languages that are obtained by restricting the grammar and vocabulary in order to reduce or eliminate ambiguity and complexity.

Grammatical Framework

Define domain model in an abstract syntax, define one or more mappings to text in a concrete syntax.
Abstract syntax:

- Domain-specific tree data structure for representing the desired content.

```
abstract ToyRailway = {
  cat Subject; Length; Restriction; Statement;
  fun Signal, Switch, Detector : Subject;
      LengthMeters : Int -> Length;
      GreaterThan, LessThan : Length -> Restriction;
      ObjectSpacing : Subject -> Subject -> Restriction
                    -> Statement; }
```

- Example phrase in abstract syntax:

```
ObjectSpacing Signal Switch (GreaterThan (LengthMeters 20))
```

Grammatical Framework

Concrete syntax:

- A mapping from the abstract syntax to text.
- Invertible, so a GF concrete syntax gives you a parser and a linearization (generator).

```
concrete ToyRailwayEng of ToyRailway = {
  lincat Subject = Str; Length = Str; (...)
  lin Signal = "signal"; (...)
      LengthMeters i = i ++ "m"
      GreaterThan l = "more than" ++ l
      ObjectSpacing o1 o2 r =
          "a" ++ o1 ++ "must be" ++ r
          ++ "from a" ++ o2; }
```

- Parse: “`a signal must be more than 20 m from a switch`”

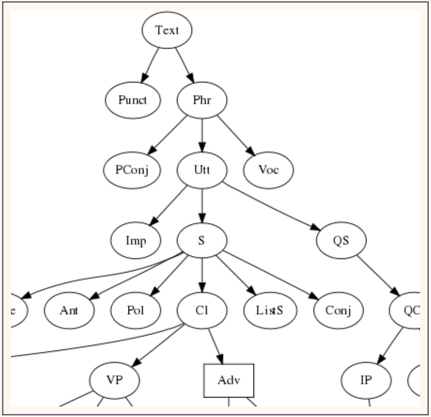
`ObjectSpacing Signal Switch (GreaterThan (LengthMeters 20))`

- Complexity and constraints of natural language quickly becomes infeasible to handle when the language grows...

Grammatical Framework's Resource Grammars

Comprehensive linguistic model of natural languages with a unified API for forming sentences.

- Parse/generate in 31 languages using a unified API.
- Ensures grammatical correctness of phrases using the type system.



API usage example:

```
OrientationAngleTo vec =
        mkCN (mkCN angle_N)
             (mkAdv to_Prep (mkNP the_Det vec));
```

Related work

Domain-specific languages for railway verification:

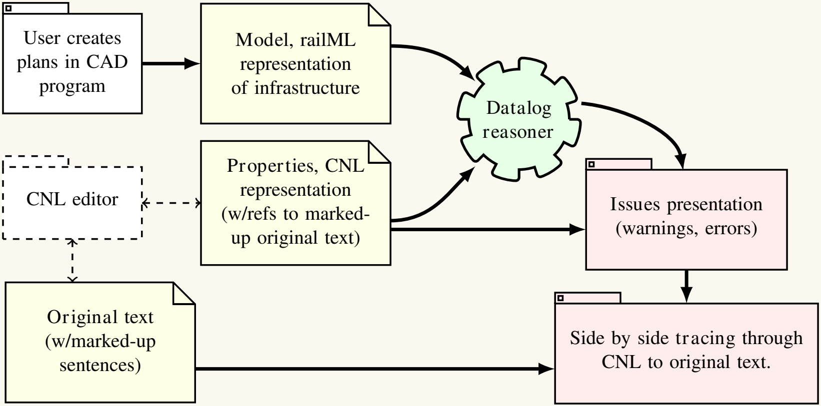
- Verification of implementation of railway control systems (Vu, Haxthausen, Peleska, 2014). Concise verification properties.
- Verification of railway layouts (James, Roggenbach, 2014). Focus on integrating domain modeling (UML) with verification, focus on control systems and fixed designs.

Controlled natural languages – formally defined restricted subsets of natural language – used for:

- Object Constraint Langauge, KeY reasoning about Java programs (Johannisson, 2007).
- Contract language $\mathcal{CL}$ (Prisacariu, Schneider, 2012) mapped into natural language and also diagrams (Camilleri, Paganelli, Schneider, 2014).
- Database queries for tax fraud detection (Calafato, Colombo, Pace, 2016).

Overview of approach

- Define a Controlled Natural Language as a high-level domain-specific language to write properties.
- Represent properties as rephrasing of natural language specifications (adds tracability of requirements)

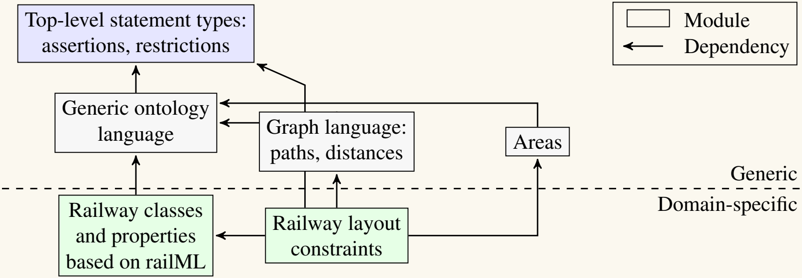


RailCNL: Language design

Top-level statements:

- Constraint: logical constraints, typically used by a Datalog reasoner to infer new facts.
- Obligation: design requirement, CAD model is checked for compliance.
- Recommendation: design heuristics, CAD model checked, but violations are shown as warnings, can be dismissed.

Modules:



RailCNL language design: ontology module

Statements about classes of objects and their properties and relations form a basis for for knowledge representation.

- Class names: "*signal*", "*switch*", ...
- Properties and values: "*color*", "*red*", "*200.0m*", ...
- Restrictions: Equality: "*A signal must have height 4.5m*".
- Relations name and multiplicity. "*A distant signal should have one or more associated signals.*"

Example 1 (Parse tree for an obligation statement.)

CNL: *A vertical segment must have length greater than 20.0m.*

AST:
```
OntologyRestriction Obligation
  (SubjectClass (StringClassAdjective "vertical"
      (StringClass "segment")))
  (ConditionPropertyRestriction (MkPropertyRestriction
      (StringProperty "length")
      (Gt (MkValue (StringTerm "20.0m")))))
```

RailCNL language design: graph module

For writing statements about the topology and geometry of objects' placement wrt. to railway tracks.

- Goal object: modifies a subject to optionally add orientation, direction, etc.
- Path restriction: combine subject, goal, and path condition. "*All paths from a station border to the first facing switch must pass an entry signal*".
- Distance restriction, see example:

Example 2 (Parse tree for a railway layout statement.)

CNL: *Distance from an entry signal to first facing switch must be greater than 200.0 m.*

AST:
```
DistanceRestriction Obligation
  (SubjectClass (StringClassAdjective "entry"
    (StringClass "signal")))
  (FirstFound FacingSwitch)
  (Gt (MkValue (StringTerm "200.0m")))
```

Tooling

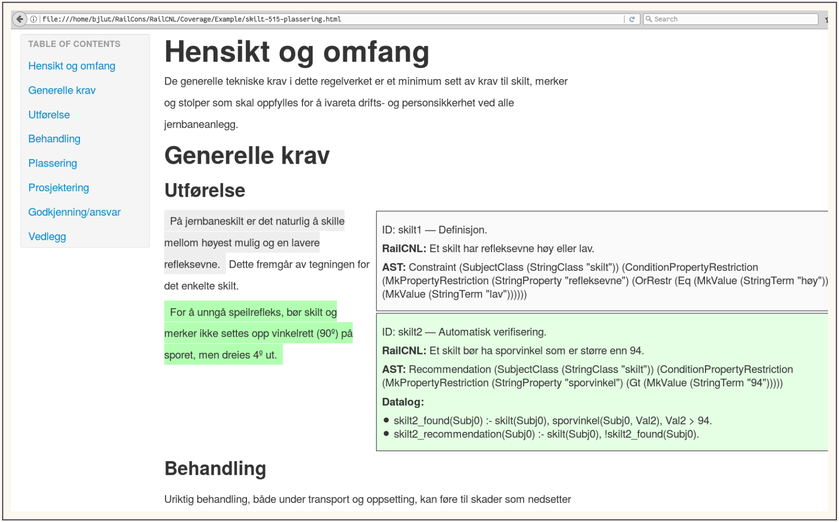
- The quality of the tool support influences the success of a domain-specific language for non-IT-experts. Textual input is a part of the overall user interface design.

Tool support for RailCNL:

- Paraphrasing view – present originals and CNL paraphrases side-by-side.
- Issues view – present verification errors in the CAD tool with links to the paraphrasing view.
- Editor – Text editor with support for writing (correct) CNL phrases.

Side-by-side CNL/original (paraphrasing view)

- Requirements tracing



file:///home/bjlut/RailCons/RailCNL/Coverage/Example/skilt-515-plassering.html

TABLE OF CONTENTS

Hensikt og omfang
Generelle krav
Utførelse
Behandling
Plassering
Prosjektering
Godkjenning/ansvar
Vedlegg

Hensikt og omfang

De generelle tekniske krav i dette regelverket er et minimum sett av krav til skilt, merker og stolper som skal oppfylles for å ivareta drifts- og personsikkerhet ved alle jernbaneanlegg.

Generelle krav

Utførelse

På jernbaneskilt er det naturlig å skille mellom høyest mulig og en lavere refleksevne. Dette fremgår av tegningen for det enkelte skilt.

ID: skilt1 — Definisjon.

RailCNL: Et skilt har refleksevne høy eller lav.

AST: Constraint (SubjectClass (StringClass "skilt")) (ConditionPropertyRestriction (MkPropertyRestriction (StringProperty "refleksevne") (OrRestr (Eq (MkValue (StringTerm "høy")) (MkValue (StringTerm "lav"))))))

For å unngå speilrefleks, bør skilt og merker ikke settes opp vinkelrett (90º) på sporet, men dreies 4º ut.

ID: skilt2 — Automatisk verifisering.

RailCNL: Et skilt bør ha sporvinkel som er større enn 94.

AST: Recommendation (SubjectClass (StringClass "skilt")) (ConditionPropertyRestriction (MkPropertyRestriction (StringProperty "sporvinkel") (Gt (MkValue (StringTerm "94")))))

Datalog:

- skilt2_found(Subj0) :- skilt(Subj0), sporvinkel(Subj0, Val2), Val2 > 94.
- skilt2_recommendation(Subj0) :- skilt(Subj0), !skilt2_found(Subj0).

Behandling

Uriktig behandling, både under transport og oppsetting, kan føre til skader som nedsetter

Issues view

- Backwards tracing – explanation of non-compliance

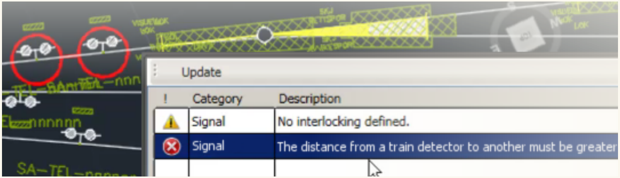
CAD program showing issues in layout plan

↕

CNL debug view paraphrased text and translations

↕

Original text highlighting source of paraphrased text



<u>ID: detector_1</u>

RailCNL: The distance from an axle counter to another must be larger than 21.0m.

AST: DistanceRestriction Obligation (SubjectClass (StringClassNoAdjective (String "axle_counter"))) (AnyFound (AnyDirectionObject SubjectOtherImplied)) (Gt (MkVal

Datalog: detector_1_start(Subj0, End, Dist) :- trainDetector(Subj0), next(Subj0, End

Placement and length

This section gives generalized rules for placement and length for train detection systems and its relationship to other infrastructure components. Detailed requirements are given in appendices.

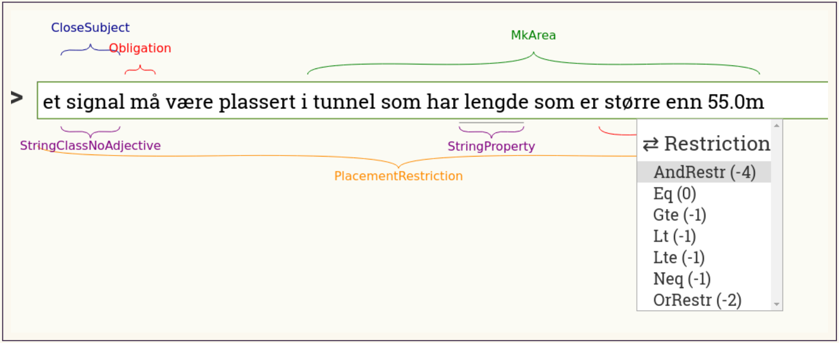
General

a) No detection sections shall be shorter than 21 meters.

b) No dead zone shall be longer than 3 meters.

Text editor CNL support

- Rule authoring tool – syntax checks, predictive parsing, chunked parsing, language exploration



Advantages

RailCNL as a front-end for property input for verification:

- RailCNL is domain-specific: tailored to Datalog logic and regulations terminology. Gives readability and maintainability.
- Resembles natural language – improves readability and engineer participation.
- Separate textual explanation (such as comments used in programming) are typically not needed.
- RailCNL statements are linked the original text. so that reading them side by side reveals to domain experts whether the CNL paraphrasing of the natural text is valid. If not, they can edit the CNL text.

Further challenges and future work

Participatory verification:

- RailCNL is a common language shared between programmers and railway engineers for verification work.
- CNLs are not a magical solution to end-user programming.
- DSLs evolve along-side the application.

Language:

- Structures in regulations that span several phrases/rules (scopes, exceptions) – represent on textual or GUI level?
- Macros – can users extend the language within the scope of their texts?

Tool support:

- Can railway engineers from other disciplines create their properties themselves, from scratch, with editor support?
- Is example-based and editor-supported language learning good enough?

Coverage

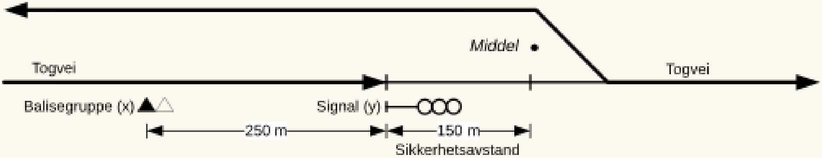
Classification for coverage analysis:

- Not relevant for verification, examples:
  Non-normative: *the technical qualities of the track construction ensure safe and efficient traffic, with the least possible environmental impact.*
  Non-checkable: *the tracks' construction must take into account the topography, soil, hydrology, climate, etc. of the location.*

- Out of scope for static analysis, examples:
  Construction: *Signs must have their original wrapping during transportation.*
  Operation: *A signal which cannot signal "stop" because of fault must be unlit.*

Coverage

- Not covered:
  - exceptions (awkward to write out all premises)
  - linguistically complex: *The safety zone (overlap) can be reduced to 200 m if the speed control system is designed such that the velocity at balise group (x) is not higher than 40 km/h when the signal (y) shows a "stop" aspect, and rolling stock will stop before the fouling point even when speed control communcation has failed in both the balise group and in the main signal.*



- Covered:
  - ontology, graph, areas, interlocking (targets), ...

Coverage statistics

Eng. discipline	**Chapter title**	Phrases	Normative	Relevant	Covered	**Coverage**
Track	Planning: general technical	140	74	74	70	95%
Track	Planning: geometry	278	157	152	119	78%
Signalling	Planning: detectors	144	106	35	21	60%
Signalling	Planning: interlocking	376	265	130	81	62%
Total		938	602	391	291	**74%**

Table 1: Coverage evaluation for a subset of Norwegian regulations. *Phrases* of the original text which could be classified as *normative* (i.e. applying some restriction on design) were evaluated for *relevance* to static infrastructure verification. The *coverage* is the percentage of relevant phrases expressible in RailCNL.

Participatory verification: experience from meetings between programmers and railway engineers

Positive:

- invites engineers to splitting hairs
  - discuss semantics of natural language
  - leads to discussion of interpretation of regulations
- example-based learning
  - explain and explore language with the editor
  - change names and values / copy-paste coding

Negative:

- total understanding of language is infeasible
  - extend language: ask for examples, not grammar

Datalog verification

- Datalog with negation (n.-as-failure) and arithmetic, implemented in e.g. XSB Prolog, RDFox, Soufflé.
- Prefer very fast (< 100 msec) re-evaluation integrated into CAD tool.
- Incremental Datalog approaches can exploit locality.

Railway construction process

1. Politicians allocate funds for new railways, upgrades or maintenance.

2. National railway administration define high level requirements, such as passenger/freight capacities, travel times, maintainability, etc.

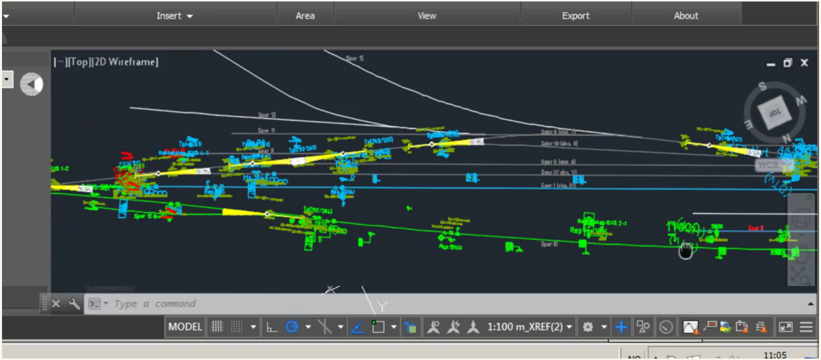
3. **Engineering companies work out the detailed plans and specifications of the upcoming construction project.**

4. Construction/implementation companies build the railway and implement control systems.

5. Finally, train companies can transport passengers and goods.

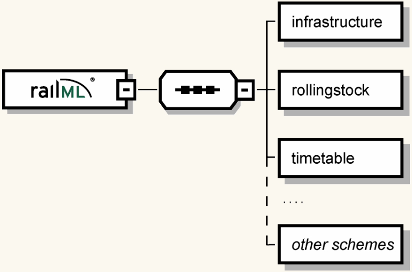
CAD programs in railway signalling

- Overview of a station, typically showing tracks and signalling system components (signals, signs, balises)



The railML XML standard data exchange format

- Thoroughly modelled infrastructure schema
- XML schema development by international standard committee




```xml
<tracks>
  <track id="tr0" name="01">
    <trackTopology>
      <trackBegin id="x399" pos="0.000000" absPos="34
        <connection id="co399" ref="co397"/>
      </trackBegin>
      <trackEnd id="y151" pos="80.000000" absPos="346
        <connection id="co151_2" ref="co151_1"/>
      </trackEnd>
    </trackTopology>
    <trackElements>
      <speedChanges>
        <speedChange id="spu399" pos="0.000000" absPo
        <speedChange id="spd403" pos="30.000000" absP
        <speedChange id="spu405" pos="30.000000" absP
        <speedChange id="spd151" pos="80.000000" absP
      </speedChanges>
      <gradientChanges>
        <gradientChange id="gr399" pos="0.000000" abs
      </gradientChanges>
      <radiusChanges>
        <radiusChange id="ra399" pos="0.000000" absPo
      </radiusChanges>
      <platformEdges>
        <platformEdge id="pe399" pos="0.000000" absPo
      </platformEdges>
    </trackElements>
    <ocsElements>
      <signals>
        <signal id="si399" pos="0.000000" absPos="346
" code="6"/>
```

Datalog

- Basic Datalog: conjunctive queries with fixed-point operators (“SQL with recursion”)
  - Guaranteed termination
  - Polynomial running time (in the number of facts)
- Expressed as logic programs in a Prolog-like syntax:

$$a(X,Y) \mathrel{:-} b(X,Z), c(Z,Y)$$

$$\Updownarrow$$

$$\forall x, y : ((\exists z : (b(x,z) \land c(z,y))) \rightarrow a(x,y))$$

- We also use:
  - Stratified negation (negation-as-failure semantics)
  - Arithmetic (which is “unsafe”)