Domain engineering of railroad systems

source material:
Dines Bjørner et al

presentation:
Tero Hasu
tero@ii.uib.no

INF329 course

19 March 2012
source material

  - particularly Chapter 2
Dines Bjørner

- famous in the formal methods community
- current focus areas: (1) domain engineering, (2) requirements engineering, and (3) software design methods
- behind RAISE (Rigorous Approach to Industrial Software Engineering)
  - RAISE Specification Language (RSL) and tools
- www2.imm.dtu.dk/~db/
A (somewhat cyclic) definition:

**Definition**
An application (or business) domain: a universe of discourse, an area of human and societal activity, ...
www.railwaydomain.org

"Because we need a grand challenge project in order to gather enough momentum to make progress along the road to industrially scalable and useful, integrated formal techniques."
“basic railway domain model”

- the physical structure of railways
  - an *intrinsic facet* of the railway domain (?) ♠
- as collections and compositions of “railway nets, lines, stations, tracks (rail) units, and connectors”
  - components which “can be physically demonstrated”
  - but abstracting away a number of physical attributes
  - “top-down” description—decreasingly composite
examples of attributes abstracted away

- of rail units
  - length
  - topology, i.e. the three-dimensional layout of a unit, including “tilting” of rails in curves, etc.
  - context: on a bridge, in a tunnel, along a platform, along a quay, etc.
a domain description

- An informal narrative describing a domain, and a mathematical text formalising the description.

- Problem: Our use of natural language is very flexible. We hardly notice as we slip from one mode of description to another.
  - might encounter serious problems in formalising an informal description
  - no single specification language can cater for all modes: functional, imperative, logical, temporal, concurrency (with events and behaviors)
a natural language description of railway nets

1. A railway net consists of one or more lines and two or more stations.
2. A railway net consists of rail units.
3. A line is a linear sequence of one or more linear rail units.
4. The rail units of a line must be rail units of the railway net of the line.
5. A station is a set of one or more rail units.
6. The rail units of a station must be rail units of the railway net of the station.
7. No two distinct lines and/or stations of a railway net share rail units.
8. ...
RAISE specification language (RSL)

- supports different specification styles
  - algebraic or model-oriented
  - applicative or imperative
  - sequential or concurrent
- supports modular specifications
- does not cater for "true" concurrency or time
a mathematical description of railway nets

- sorts
  - abstract types—no specified structure
  - respectively: network, line, station, track, rail unit, connector

**type** N, L, S, Tr, U, C
example: railway net representation and constraints

1. A railway net consists of one or more lines and two or more stations.
2. A railway net consists of rail units.

value
1. \( \text{obs}_Ls : \mathbb{N} \rightarrow \text{L-set} \)
1. \( \text{obs}_Ss : \mathbb{N} \rightarrow \text{S-set} \)
2. \( \text{obs}_Us : \mathbb{N} \rightarrow \text{U-set} \)

axiom
1. \( \forall n : \mathbb{N} \bullet \text{card } \text{obs}_Ls(n) \geq 1 \)
1. \( \forall n : \mathbb{N} \bullet \text{card } \text{obs}_Ss(n) \geq 2 \)
example: linear sequence predicate

17. A linear sequence of (linear) rail units is a non-cyclic sequence of linear units such that neighbouring units share connectors.

value

\[
\text{lin\_seq: } U\text{-set} \rightarrow \text{Bool} \\
\text{lin\_seq}(us) \equiv \\
\forall u:U \bullet u \in us \Rightarrow \text{is\_Linear}(u) \land \\
\exists q:U^* \bullet \text{len } q = \text{card } us \land \text{elems } q = us \land \\
\forall i:\text{Nat} \bullet \{i, i+i\} \subseteq \text{inds } q \Rightarrow \exists c:C \bullet \\
\quad \text{obs\_Cs}(q(i)) \cap \text{obs\_Cs}(q(i+1)) = \{c\} \land \\
\text{len } q > 1 \Rightarrow \\
\quad \text{obs\_Cs}(q(i)) \cap \text{obs\_Cs}(q(\text{len } q)) = \{\} 
\]
example: well-formed route subtype

(A route is a sequence of pairs of units and paths...)

23. ...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.

\[
\text{type} \\
R = \{| \ r : R' \land \text{wf}_R(r) | \}
\]

axiom

\[
\text{wf}_R : R' \rightarrow \text{Bool} \\
\text{wf}_R(r) \equiv \text{len } r > 0 \land \\
\forall i : \text{Nat} \land i \in \text{inds } r \land \text{let } (u, (c, c')) = r(i) \land (c, c') \in \bigcup \text{obs}_\Omega(u) \land i + 1 \in \text{inds } r \Rightarrow \\
\text{let } (_, (c'', _)) = r(i + 1) \land c' = c'' \text{ end end}
\]
example: “does a line connect to a station” query

- LS_Connection is a partial function, with guards
- $N$ for railway net, $L$ for line, $S$ for station
- $U$ for rail unit, $C$ for connector
- a station is a set of units; cf. 5. obs_Us : $S \rightarrow U$-set
- note overload of obs_Us

value

\[
\text{LS\_Connection} : N \times L \times S \rightarrow \text{Bool}
\]

\[
\text{LS\_Connection}(n, l, s) \equiv \\
\exists u, u' : U \bullet u \in \text{obs}\_Us(l) \land u' \in \text{obs}\_Us(s) \land \\
\exists c : C \bullet \text{obs}\_Cs(u) \cap \text{obs}\_Cs(u') = \{c\}
\]

pre $l \in \text{obs}\_Ls(n) \land s \in \text{obs}\_Ss(n)$
modular RSL

- RSL supports scheme and class syntax
  - for structuring, i.e. breaking up a model into smaller parts
    - this opens up reuse possibilities
  - like concepts, with types, signatures, and axioms
  - a scheme declaration is parameterizable
  - a scheme may combine and extend others by adding types, signatures, and axioms
  - Examples: [extend] and [generics]
modular RSL versus UML

- suitable structuring of an RSL specification may make it amenable to “UML-ising”
  - not everything is expressible in UML, but makes for a more visual representation
  - may provide a useful “view” into certain aspects of the model
UML example

- **Example [uml]**
- **1.** A railway net consists of one or more lines and two or more stations.
  - can be modeled with UML *composition*
- **12.** A rail unit is either a linear unit, a switch, a simple crossover, or a switchable crossover.
  - can be modeled with UML *generalization*; unit is an *abstract class*
- **14.** A linear rail unit has two distinct connectors, a switch rail unit has three distinct connectors, crossover rail units have four distinct connectors, etc.
  - can be modeled with UML *associations*
dynamism

- a railway net would ideally be a *programmed, dynamic active system*
- less ideally (in the real world) it is a *dynamic reactive system*
- Bjørner et al primarily regard a railway net as “programmed”, assuming its managers are in control of its time-wise behavior
- a *managed rail net* has state
  - e.g. switches and signals have state
- “small” parts of a rail net may be undergoing change
  - e.g. new lines and stations being added, old ones removed or put under repair
  - $T \rightarrow N$
states of a rail unit

- either in stable, transition, or reconfiguration state
- it is assumed that durations can be observed
- stable state: determines how a train can move across a unit
- transition state: between stable states; a transition takes time
- reconfiguration state: change of (stable) state space
  - enable additional paths, or disable previously valid paths
  - units may have “dangling” connectors
formalisms for modeling time

- plain RSL
- Timed RSL (TRSL)
- Duration Calculus (DC)
no built-in way to model time

time can be modeled, but “not in general very satisfactorily”
  e.g. impossible to specify *timeout*
  Example [time]
Timed RSL (TRSL)

- an extension to RSL; only
  - type Time, an alias for the non-negative subtype of Real
  - a wait construct; takes an expression of type Time

```plaintext
sensor_state := high; wait δ; sensor_state := low
```

- implementing a timeout with wait and the external choice operator

```plaintext
normal? ; ...  
[]  
wait t ; abnormal!()
```
Duration Calculus (DC)

- well suited for (timed) requirements specifications
- example: any complete period with a high state must have a duration $\ell$ of at least $\delta$

$$\square\((\boxdot[sensor\_state = low] \land \boxdot[sensor\_state = high]) \land \boxdot[sensor\_state = low]) \Rightarrow \ell \geq \delta)$$
RSL, TRSL, and DC used together

1. specify un-timed properties in RSL
2. specify requirements for real-time properties in DC
3. add timing information to RSL, with TRSL extensions
4. verify that the TRSL specification satisfies the DC specification

▶ satisfaction verification typically through *abstract interpretation*

▶ by defining *operational semantics* for TRSL wrt DC
“all things railways”

- timetables (Chapter 3 of Towards a TRain book)
- rolling stock maintenance (Chapter 4)
- rostering (Chapter 5)
- station interlocking (Chapter 6)
- signalling on lines (Chapter 7)
- line direction agreement (Chapter 8)
route and timetable information might be published in some machine accessible/readable manner ♠
  to allow for implementation of services

suggested: use OWL (Web Ontology Language), a semantic markup language for specifying ontologies
  “An ontology formally represents knowledge as a set of concepts within a domain, and the relationships between those concepts.” (Wikipedia)

the Semantic Web idea/movement: publish information on the web in a structured form
  to enable complex queries, especially in combination with other information providers
  application-independence matches Bjørner’s thinking ♠

popular choice: scraping and web service APIs ♠
rolling stock maintenance

- “Rolling stock comprises all the vehicles that move on a railway. It usually includes both powered and unpowered vehicles, for example locomotives, railroad cars, coaches and wagons.” (Wikipedia)

- maintenance: regular checks, cleaning of carriages, refuelling, refilling supplies, etc.

- *maintenance routing*: for types of maintenance not planned in advance for given rolling stock, modify plans to route rolling stock to maintenance stations in a timely manner, according to operating hours elapsed and kilometers travelled and associated limits

  - output: set of changes in rolling stock roster for the next few days (or all possible sets)
rostering

- **staff rostering**: ordering of *duties* (short-term working schedules) into *base rosters* (long-term working schedules), and assignment of specific staff members to rosters
  - hiring decisions can be made based on such staff planning

- based on a suitable formal model, from a given schedule, staff type, depot, and rules, can produce a set of rosters

  \[
  \text{gen\_sross: SCH} \times \text{StfTp} \times \text{Dep} \times \text{eRS} \rightarrow \text{Ros}
  \]
22. A route is a sequence of pairs of units and paths...
23. ...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.
station interlocking

- routes may (also) be described in terms of units, switches, signals, and interlocking tables
  
  - “A signal is a mechanical or electrical device erected beside a railway line to pass information relating to the state of the line ahead to train/engine drivers.” (Wikipedia)
  
  - “An interlocking is an arrangement of signal apparatus that prevents conflicting movements through an arrangement of tracks such as junctions or crossings.” (Wikipedia)
  
  - an interlocking table: for all routes of a station, can present valid interlockings (required setting—if any—for each switch and signal) as a table
    
    - Example [table]
station interlocking: modeling formalism

- capture interlocking requirements as *Petri nets*
  - consist of *places*, *transitions*, and *arcs*; places may contain *tokens*
  - are suitable for modeling and simulating concurrent behavior of distributed systems
  - have nondeterministic execution semantics

- e.g. ensure that a switch can only change state when no route requiring its current state is active
  - by having transitions require a certain number of tokens to fire, and by having open routes keep tokens away from switches

![Petri Net for a Switch](image1.png)
![Petri Net for a Signal](image2.png)

Fig. 6.2. (a) Petri Net for a Switch, (b) Petri Net for a Signal
interlude: Statecharts and Live Sequence Charts

- entities in the charts may be physical phenomena, processes, objects, etc.
- *Statecharts* (SC) are used to describe the sequences of states an entity may pass through in response to external stimuli (internal behavior)
- *Live Sequence Charts* (LSC) are used to specify sequences of communication—i.e. the protocol—between two or more entities (external behavior)
- in combination specify both internal and external behavior
signalling on lines

- high-speed trains cannot stop within a sighting distance of a signal—hence automatic signalling
- might model automatic line signalling as Statecharts (as opposed to Petri nets)
  - states for a line (agreed line direction): OpenAB, OpenBA, Close
  - states for a line *segment*: segFree, segOccupied (i.e., occupied by a train)
  - signal states: sigOnRed, sigOnGreen, sigOnYellow, sigOff
signalling on lines: example Statechart

**Fig. 7.3.** General State Charts for Automatic Line Signalling
line direction agreement

- safety property: two trains are not allowed to move in opposite directions on any railway line
- Line Direction Agreement System (LDAS) for two stations to agree on the direction of trains between them
- the externally visible behavior of an LDAS may be specified using Live Sequence Charts
  - entities: Station A (SA), LDAS, and Station B (SB)
line direction agreement: protocol

Fig. 8.1. Communication with LDAS
line direction agreement: external behavior

Live Sequence Chart
line direction agreement: internal behavior

Statechart
what can be done with a formal model

- queries
  - e.g., recall the LS_Connection function to compute whether a line connects to a station
  - e.g., compute possible changes to the rolling stock roster to meet maintenance requirements

- verification
  - e.g., check interlocking and the LDAS logic for correctness, for safety reasons

- documentation
  - particularly diagrammatic constructs might appeal to readers (Petri Nets, SCs, LSCs, UML diagrams, …)
further reading


- George and Haxthausen: The Logic of the RAISE Specification Language.
  - doesn’t cover scheme syntax or Timed RSL (TRSL)