(Formally) Assessing Interlocking Policies

Paulo Silva Jorge Lobo

Department of Informatics
University of Minho

Thursday 3rd July, 2014

Outline

- Recall the previous milestones
- Remember interlocking architectures
- Job done since last milestone
- Our approaches
- Results & Conclusions

Aims

1. Rigorous modelling of interlocking policies
2. ... resorting to tool support for:
   - Simulation
   - Validation of properties
3. ... rendered in a domain-friendly way

In order to **raise the knowledge** regarding advantages and disadvantages of both *centralized* and *geographic* architectures

First Milestone

- Interlocking insight
- Railways control system architecture
- Rules and properties regarding safety and liveness
- Analyse the two architectures

Second Milestone

- Analyse track plan and organize its data
- Modelling in CSP||B
  - Extract the information from the track plan
  - Build the topology with respect to the extracted information
  - Simulation of railway system using CSP process to guide the railway evolution
  - Model Check
  - Present a demo in ProB

Third Milestone

- Re-modelling of CSP||B approach due to EFACEC's requirements, regarding
  - The locking of tracks before the route set
  - A specification of another *clear track* test before setting green aspect
- First approach to SCADE
  - Study the language
  - Try to find an architecture that fits the centralised design
  - Introduced the modularity of components

Recalling the concepts

Geographic Interlocking The elements has builted-in the rules of the respective routes associated and are placed along the path. In order to ensures the safety, the verification made through topological connection of the elements.

Centralised Interlocking The routes and the respective elements are related in a table. This allows the interlocking to check the availability of a route by consulting the table and verify the free elements.

CSP||B

Implementation successfull.

- Invariants verified
- Simulations succeeded(Demo 2nd milestone)
- Allows to observe trains and interlocking behaviour.

SCADE

Two attempts

- Didier Bert in *Building lustre control systems from b abstract machines: A case study* provide a technique to derive B specifications to low-level program written in LUSTRE[1]
- Andrew Lawrence research is focused on implementation of a railway interlocking using SCADE using a concret layout.

[1] Language supporting SCADE Suite Tool

SCADE

Following Andrew's research:

- Resorts to *state machines* to model the route elements
- Behaviour modelled by data flow (*boolean* data)
- Provide a node[2] RouteController that checks routes in conflict. This node filters the conflicting routes and allows only the non-conflicting routes to be set.

[2] Function, Procedure

However

The implementation was not successfull.
Consequently there was no simulation to achieve valid answers.

Comparing both approaches

Make generic

- CSP||B as a high level language.
- SCADE low level specification requires much more effort.

Code Generation

- CSP||B : abstract models more difficult to generate code. Several refinement steps are needed.
- SCADE : KCG provides a powerfull code generator, easy to work with.

Comparing both approaches

Models

- CSP||B despite being a mathematical language, it is of easy understanding
- SCADE is very verbose; the necessary use of many variables makes the reading much more difficult.

Best interlocking architecture?

Centralised Interlocking

- Provide better maintainability regarding small changes in track plans
- Implementation confined to a single place
- One implementation is easier extended to others

Best interlocking architecture?

Geographic Interlocking

- Inflexibility once is proved safe
- Autonomy
- If something fails, the worst case is no train is allowed to move.

(Formally) Assessing Interlocking Policies

Paulo Silva Jorge Lobo

Department of Informatics
University of Minho

Thursday 3rd July, 2014