EG UK Computer Graphics & Visual Computing (2014)
Rita Borgo, Wen Tang (Editors)

Visualising Timed CSP Train Simulations for Capacity

Lloyd Roberts, Mike Smith, Faron Moller, and Markus Roggenbach,

Swansea University, Wales, UK

1. Context

Overcoming the constraints on railway capacity caused by nodes (stations and junctions) on the rail network is one of the most pressing challenges to rail industry. In 2007, the UK governmental White Paper "Delivering a Sustainable Railway" [DT007] stated: *"Rail's biggest contribution to tackling global warming comes from increasing its capacity."* High capacity, however, is but one design aim within the railway domain. Railways are safety-critical systems. Their malfunction could lead to death or serious injury to people, loss or severe damage to equipment, or environmental harm.

In a recent publication [IMNR12], in cooperation with our industrial partner, Siemens Rail Automation UK, we developed an integrated view of rail nodes within which capacity can be investigated and enhanced without compromising safety. The underlying formal models are written in the process algebra Timed CSP. Model-checking allows to analyse the models for safety and capacity. Simulation with Timed CSP-Simulator [FAGNR12] allows to validate the models and to demonstrate effects such as bottlenecks in the design of a rail node. Justifiable assumptions simplify the modelling. These include: trains move at constant speed, acceleration and breaking happens immediately, trains are shorter than tracks.

2. Objective

Timed CSP-Simulator provides a history of the system run of a rail node such as the double junction shown in Figure 1. The events of such a run indicate that, e.g., the front of a train has moved from one track to another (`moveff.DP.DR`), or that time has passed (`TimeEvolution: 2 in (0..2]`), see Figure 2. In such a run, time evolves from bottom (first event) to top (last event). In the run shown in the Figure, first the controller sets signal `S19` to green, then a train moves its front from track circuit `Entry3` to track circuit `DP`, etc.

Our objective is to visualise such simulation histories in order to make Timed CSP train simulations easily accessible to rail practitioners.

3. Method

To this end, we utilise established visual representations for rail network dynamics such as "train trajectories", see Figure 3, and the "control room view", see Figure 4.

For both approaches it was necessary first to prune Timed CSP-Simulator's history (e.g., remove the information that an internal state change has happened) and then to re-arrange is slightly, see Algorithm 1 for a sample.

3.1. Train trajectories

Train trajectories are a common tool in planning rail operation, e.g., in timetabling (e.g. [Hua06]) or minimising fuel costs (e.g. [LHHR13]), or approximate algorithmic rail simulations as in the SafeCap platform [ILR13]. Train trajectories plot the travelled distance over time, see Figure 3. We implemented train trajectories using the JFreeChart library [jfr].

Horizontal sections of a trajectory represent dwelling time of a train, i.e., lost capacity; the horizontal distance between train trajectories of trains on the same route is reciprocal to used capacity.

The advantage of train trajectories is that they summarise one run in one visualisation. It is their nature that they project a multi-dimensional rail yard down to a single distance. As a consequence they have the disadvantage that they show trains to be in the same "position", i.e., possibly colliding with each other, even when these trains are safely passing by, see in Figure 3, time 110. As they also abstract from other rail elements such as points or signals, they are not suitable to represent railway safety conditions. However, they are ideal for studying capacity.

3.2. Control room view

The *control room view*, see Figure 4, is close to what the controller sees of a rail yard in a fully implemented system. It is an animation compiled out of one simulation, where the user chooses how fast time shall evolve, proportional to time

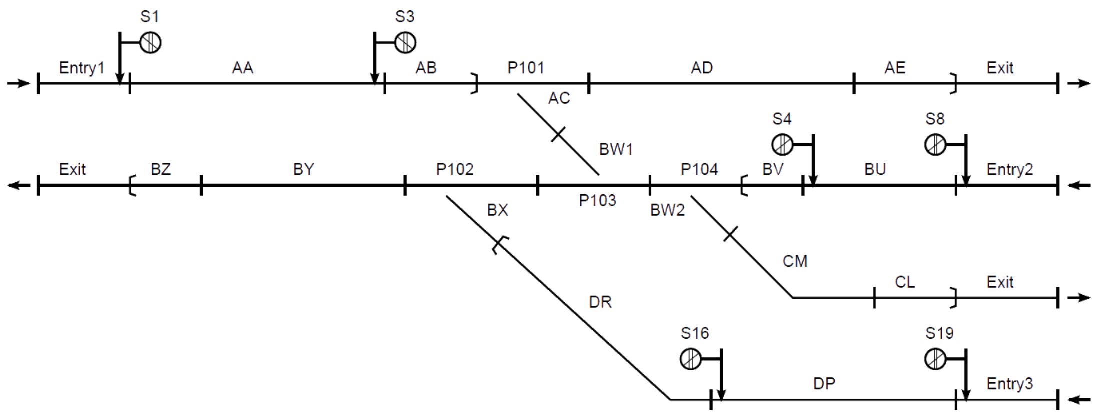


Figure 1: *Trackplan of a Double Junction*

evolution in the simulator. The control room view shows the given trackplan, where the actual track length is abstracted from. Into this trackplan, trains are shown at discrete positions, following our modelling approach. A train either occupies one track, i.e., both its front and its rear are on this track, or it occupies two consecutive tracks, i.e., its front and rear are on different tracks. We implemented this visualisation using the Visual Studio library [vs], see Figure 5 for a code snippet.

Long times of a train remaining in one position represent lost capacity or long distances to travel. Taking the animation at one point of time, the horizontal distance between trains is reciprocal to used capacity.

An advantage of the control room view is that it allows one to see the whole rail yard at a glance, however, only at one point in time. Typical safety properties such as that there is no collision in a run are visible in the animation.

3.3. Tool integration

Train trajectory visualisation has been fully integrated with Timed CSP-Simulator. This turned out to be slightly cumbersome: Timed CSP-Simulator's GUI is written in Tcl/Tk [tcl], the visualisation package was written in Java, and the Java Tcl package did not support a seamless integration. In the end, we used an Tcl "exec" command to start the visualisation via a call to the operating system.

4. Results

Our industrial partner Siemens Rail Automation UK has seen both visualisations, experimented with them, and evaluated them to be a step in the right direction. Interestingly enough, they have a slight preference for the control room view, though train trajectories are the by far more popular visualisation approach in railway optimisation. However, rail industry traditionally has its focus on safety, which is "invisible" with train trajectories. It is ongoing work to develop our visualisation approaches towards accepted tools in the field.

5. Novelty

The general topic underlying this work is how to support the adoption of formal methods within industry, see e.g. [JR14]. While formal methods meanwhile can cope with the analysis of many safety-critical systems and can be cost-effective [Bar11], cryptic notation often hinders their uptake in industrial practice. Here, visualisation can be of much help, e.g., to represent counterexamples (e.g. [AST11]), formal specifications (e.g. [Til03], simulations (e.g. in the tool UPPAAL [upp]).

The speciality of our approach is that we faithfully translate our formal model back into notation of the application domain – thus closing the notational gap between the formal method Timed CSP and the application domain of railway control systems.

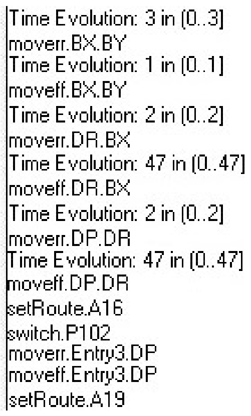
References

[AST11] ABDELHALIM I., SCHNEIDER S., TREHARNE H.: Towards a practical approach to check UML/fUML models consistency using CSP. In *Formal Methods and Software Engineering*, Qin S., Qiu Z., (Eds.), vol. 6991 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 33–48. 2

[Bar11] BARNES J. E.: Experiences in the industrial use of formal methods. In *AVoCS'11* (2011), Romanovsky A., Jones C., Bendiposto J., Leuschel M., (Eds.), Electronic Communications of the EASST. 2

[DT007] Department of Transport. Delivering a Sustainable Railway. White Paper CM 7176, 2007. 1

[FAGNR12] FONTAINE M., ANDY GIMBLETT F. M., NGUYEN H. N., ROGGENBACH M.: Timed CSP Simulator. In *Proceedings of the Posters & Tool demos Session, iFM 2012 & ABZ 2012* (2012). 1



```
Time Evolution: 3 in [0..3]
moverr.BX.BY
Time Evolution: 1 in [0..1]
moveff.BX.BY
Time Evolution: 2 in [0..2]
moverr.DR.BX
Time Evolution: 47 in [0..47]
moveff.DR.BX
Time Evolution: 2 in [0..2]
moverr.DP.DR
Time Evolution: 47 in [0..47]
moveff.DP.DR
setRoute.A16
switch.P102
moverr.Entry3.DP
moveff.Entry3.DP
setRoute.A19
```

Figure 2: *A Timed* CSP*-Simulator history*

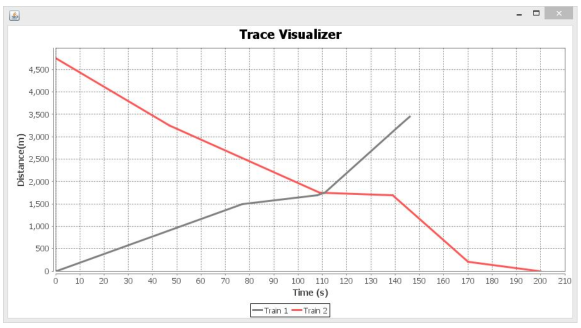


Figure 3: *Train trajectories*

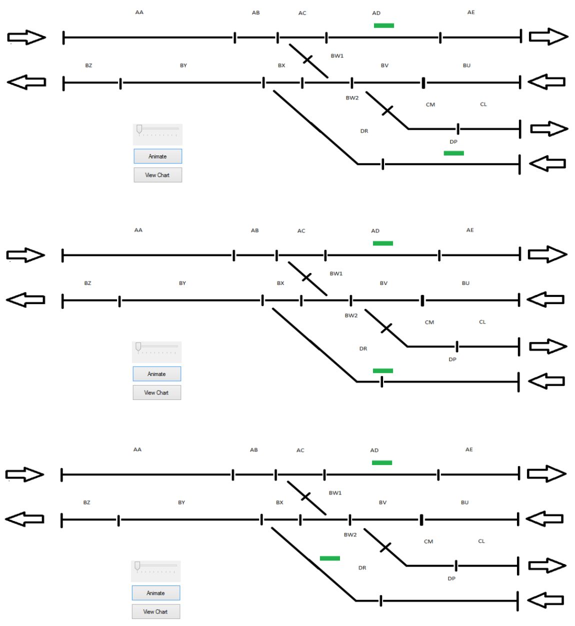


Figure 4: *Control Room View – 3 snapshots*

[Hua06] HUANG J.-Y.: Using ant colony optimization to solve train timetabling problem of mass rapid transit. In *JCIS* (2006), Atlantis Press. 1

[ILR13] ILIASOV A., LOPATKIN I., ROMANOVSKY A.: The safecap project on railway safety verification and capacity simulation. In *Software Engineering for Resilient Systems*, Gorbenko A., Romanovsky A., Kharchenko V., (Eds.), vol. 8166 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 125–132. 1

[IMNR12] ISOBE Y., MOLLER F., NGUYEN H. N., ROGGENBACH M.: Safety and line capacity in railways - an approach in Timed CSP. In *IFM* (2012), pp. 54–68. 1

[jfr] JFreeChart. http://www.jfree.org/jfreechart. 1

[JR14] JAMES P., ROGGENBACH M.: Encapsulating formal methods within domain specific languages: A solution for verifying railway scheme plans. *Mathematics in Computer Science 8*, 1 (2014), 11–38. 2

[LHHR13] LU S., HILLMANSEN S., HO T.-K., ROBERTS C.: Single-train trajectory optimization. *IEEE Transactions on Intelligent Transportation Systems 14*, 2 (2013), 743–750. 1

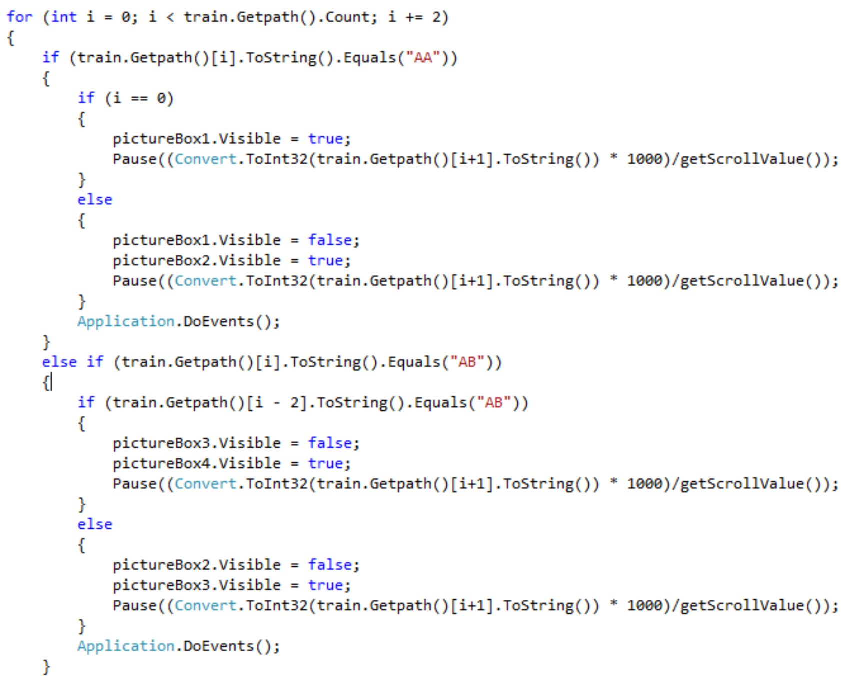
[tcl] Tcl developer site. http://www.tcl.tk. 2

[Til03] TILLEY T.: Towards an fca based tool for visualising formal specifications. In *Using Conceptual Structures: Contributions to ICCS 2003* (2003), Shaker Verlag, pp. 227–240. 2

[upp] Uppaal. http://www.uppaal.org. 2

[vs] Visual Studio. http://www.visualstudio.com. 2

Origin Lloyd Roberts and Mike Smith developed the reported visualisations as their final year projects at Swansea University.



```
for (int i = 0; i < train.Getpath().Count; i += 2)
{
    if (train.Getpath()[i].ToString().Equals("AA"))
    {
        if (i == 0)
        {
            pictureBox1.Visible = true;
            Pause((Convert.ToInt32(train.Getpath()[i+1].ToString()) * 1000)/getScrollValue());
        }
        else
        {
            pictureBox1.Visible = false;
            pictureBox2.Visible = true;
            Pause((Convert.ToInt32(train.Getpath()[i+1].ToString()) * 1000)/getScrollValue());
        }
        Application.DoEvents();
    }
    else if (train.Getpath()[i].ToString().Equals("AB"))
    {
        if (train.Getpath()[i - 2].ToString().Equals("AB"))
        {
            pictureBox3.Visible = false;
            pictureBox4.Visible = true;
            Pause((Convert.ToInt32(train.Getpath()[i+1].ToString()) * 1000)/getScrollValue());
        }
        else
        {
            pictureBox2.Visible = false;
            pictureBox3.Visible = true;
            Pause((Convert.ToInt32(train.Getpath()[i+1].ToString()) * 1000)/getScrollValue());
        }
        Application.DoEvents();
    }
```

Figure 5: *Animation using Visual Studio*

```
while Text file is not completely read do
    Read the current line;
    if Line contains a rear move, and the train is not at the exit then
        Add a new track section to the current train at the current time of the system;
    else if The current line contains a front move and the train is at the entry. then
        Add the current train to a list of completed trains;
        Create a new train, with an ID 1 greater than the previous train;
    else
        Try and parse the current time as an integer, and add this to the current system time;
    end
end
```

Algorithm 1: *Pruning Timed* CSP*-Simulator's history*