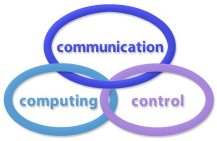
INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
Online ISSN 1841-9844, ISSN-L 1841-9836, Volume: 17, Issue: 5, Month: October, Year: 2022
Article Number: 4444, https://doi.org/10.15837/ijccc.2022.5.4444



CCC Publications

Identifying Alterability States of a Single Track Railway Line Control System

S. Martinez, S. Collart-Dutilleul, P. Bon

Sébastien Martinez, Simon Collart-Dutilleul, Philippe Bon
Université Gustave Eiffel
Laboratoire ESTAS, Campus de Lille
20, rue Élisée Reclus, 59650 Villeneuve d'Ascq, France
name.surname@univ-eiffel.fr

Abstract

In the context of automation and deployment of computer based control systems, a specific application on French railway line is proposed on low traffic single track railway lines. The issue of updates requires thorough consideration. In the case of low traffic single track railway lines, handling the removal of a shunting track, which role is to allow trains to circulate in both directions of a same line, the issue of timing the update to the control system is particularly critical. Indeed, a wrongly timed update could lead to a deadlock, while one or more trains are expected to travel while respecting safety constraints on the blocked infrastructure.

This paper studies the application of works from the field of dynamic software updating, specifically the works of Panzica La Manna et al. [12]. Using their results on a graph based model of a single track rail line, it identifies alterability states that ensure safety constraints are respected at all times without causing deadlocks. These results are then used to discuss the pertinence of using concepts from dynamic software updating in the context of railway systems.

Keywords: Railway systems, single track railway lines, alterability, graph based modeling

1 Introduction

French railway traffic is mostly controlled by electromechanical systems and human operators. Maintenance of the former is expensive and error-prone as few verification tools allow easy and reliable analysis of such systems. Human operators are more adaptable and can be relied on in unforeseen situations. However, assigning many operators to low traffic lines could be an unsuited option for economic purpose and for humane purpose as operating traffic can be a monotonous and laborious task. As an alternative, several projects such as LCHIP [4] study the automation of traffic control systems using computer based systems. This enables the usage of formal specification languages and formal verification tools for proving the safety of railway traffic. An example of formal specification language popular in the french railway community is B Method [1].

In the context of studying traffic control automation, system updates raise an issue. Current organization assigns human operators with the task of replacing electromechanical systems whenever

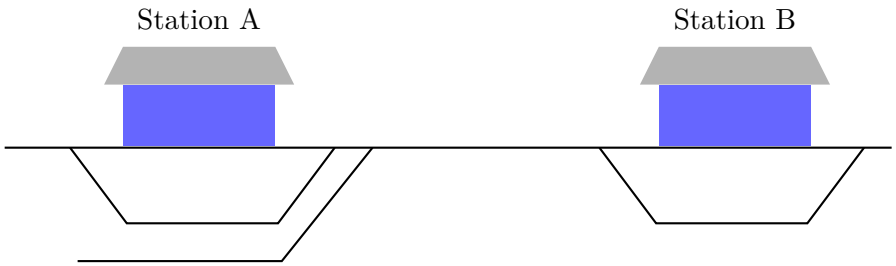


Figure 1: A single track rail line

they are no longer suited to the traffic organization. In case of malfunction in control systems, in railway lines or when the traffic is simply reorganized, human operators handle traffic control until control systems are repaired or updated. The flexibility offered by computer based solution would make these situations easier to manage and require less human work. Developing a software update and having its safety formally proved is easier and less expensive than developing and verifying a new electromechanical control system.

This paper studies the case of single track lines in France. On such lines, trains coming from both directions share the same track. To allow upstream and downstream traffic, stations have shunting tracks and/or deviation tracks where a train can be stationed while another train passes through the single track. That organisation concerns over 3000 kilometers of lines in France and is justified by the low traffic on theses lines. The low traffic also justifies the interest for automation as it would reduce the need for human operator intervention. Moreover, such kind of lines are common in Europe. Investments in their automation would rapidly be cost-effective.

Figure 1 presents an example of single track line joining two stations. Station A has a shunting track and a deviation track while station B only has a deviation track. Considering two trains going respectively from station A to B and from station B to A, the deviation and shunting tracks are essential to safely handle traffic on the line. One of the trains need to stay on a shunting or a deviation track to allow the passage of the other train. Removing a shunting or a deviation track from station A or station B would considerably impact the organization of traffic on the line.

This paper considers the case of a single track line as depicted on figure 1, on which traffic is controlled by computer based systems and addresses the issue of updating the control system to adapt to the suppression of a shunting or a deviation track. The control system should be updated in a manner that disrupts traffic as little as possible while strictly respecting security constraints of railway traffic. To achieve that objective, the paper bases itself on results from the domain of Dynamic Software Updating (DSU), especially the paper from Panzica La Manna et al. [12]. DSU studies ways of applying updates to software systems without disrupting their services and Panzica La Manna et al. gave criteria for safely updating control subsystems of cyberphysical systems.

The following of the paper is organized as follows : Section 2 presents related works and summarizes the results of Panzica La Mana et al. used in this paper. Section 3 details a single track line model used as a case study by the paper and section 4 details a graph-based model of the control system for that case study. It details how the control system can be safely updated. Section 5 discusses the results of its previous section and concludes the paper.

2 Related Works

Low traffic lines have singular design requirements, as investments has to be tailored to the financial potentialities associated to the considered infrastructure. “Cost reduction on aging infrastructures” throughout Europe was studied in a H2020 European project named NETIRAIL[1]. In France, the Nextregio project lead by IRT Railenium focused on the regional lines with *SNCF Réseau* as main user. Its main object of study was the issue of single track railway lines. The diversity in terms of needs must be handled with the existing infrastructures, inherited from history: they are using

[1] http://netirail.eu (last visited on March 24th 2021)

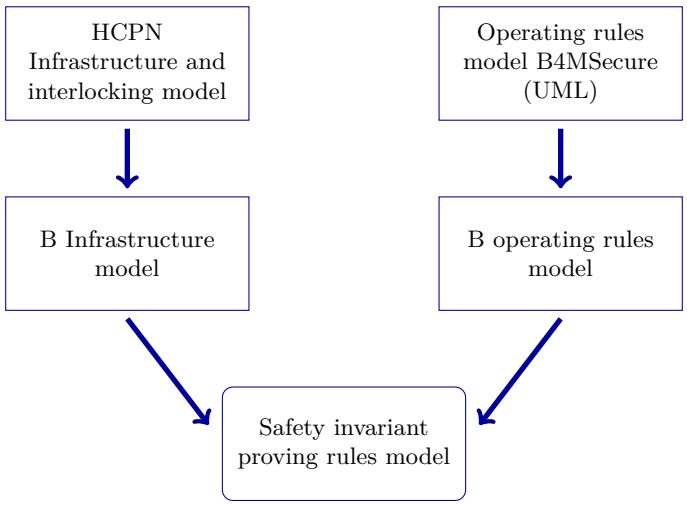


Figure 2: Global framework of the approach (courtesy of Antoine Ferlin et al. [5])

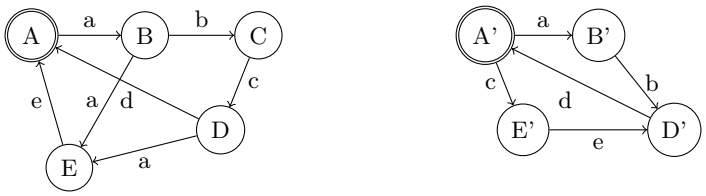
various technologies with various level of automation. In the Nextregio project, building blocks of the abstract architecture are based on functions expressed by the railway infrastructure manager in the document expressing its needs [2]. The design work consists in refining these elementary functions and combining them in order to achieve production goals. As depicted in figure 2, using the RBAC Profile and the B4MSecure Tool, an Event B model is produced, containing the functional specification and the safety restrictions [13]. Therefore, all subsystems have to provide the same functions, but the constraints and the means to achieve the functions are quite different from a subsystem to another. In other words, functions and states are similar to those of the high-speed lines studied in the Perfect project [3, 14], but they are broken out in different manners, depending on the level of automation of the line.

In the low traffic lines, the solution must be tailored, considering that infrastructure cannot be modified while allowing low exploitation costs. The use of low cost safety compliant programmable platforms seems a well-adapted architecture. They would use generic hardware components and their embedded software could be adapted in order to handle the diversity in low traffic lines.

In this context, the LCHIP Project was launched with the objective of using formal methodologies of specification in order to support the analysis, verification and code generation of railway interlocking systems. The main proposition is the use of low cost integrated chips allowing the above services, while running software component controlling local automatisms. However, using low cost components (like the Microchip PIC32 of the Lchip project) may lead to security issues. For instance, it is easier to reverse engineer compiled software as the MIPS architecture has a small instruction set. Some obfuscation techniques may be used to mitigate that weakness [10]. Study of the security and the updating of low cost computer-based automatism in the railway industry is recent. The Lchip technology was patented worldwide[2] and knows no equivalent so far [8]. As a result, the work presented in this paper cannot be compared to related industrial technologies.

Updating a system of distributed cyber physical systems stands the issue of global system homogeneity. If the system is distributed, it means that the physical parts are linked to local hardware and software components that are connected with others entities using telecommunications technologies. As a consequence, any change in the local knowledge may alter its ability to respect the global operating protocol that makes it able to behave as a whole. For example, telecommunications delays prevent instantaneous global modifications, allowing only successive local modifications. Between the first and the last local modifications the system is in a transitory state that may not comply with operating protocols. In the case of railway systems, operating protocols include safety rules that must be respected at any moment, leading to the identification of *forbidden states* (*e.g.* states that could lead to collisions). Local approaches are not suited in that case, and more global approaches (*e.g.* defining local management rules of a railway station) should be studied. Such approach can benefit

[2] https://www.erts2020.org/uploads/presentation-pdf/fr2-a1-lchip.pdf



Older version Newer version

Figure 3: Updating a cyberphysical system

of works from the domain of dynamic updates of software systems.

Dynamic Software Updating (DSU) addresses the application of updates on software systems during their execution, without disrupting the services they provide. Specific DSU mechanisms are used to modify the code and the state of running programs, migrating them from their outdated version to their newest version. After being dynamically updated, software systems appear as though their were running in their newest version from the beginning. A key issue of dynamic updating is the timing of the updates. Applying an update at a wrong time can lead to an inconsistent state where out of date parts of the updated program can call incompatible updated parts. Such behaviour may cause errors or even crash the running program. Updates need to be applied when the updated program is in a state of *alterability*, as defined in [11]. Several alterabilty criteria (*i.e.* conditions on the program state guaranteeing that state is a state of alterability) are defined in the literature. The most common is the quiescence of updated components [7]. According to Kramer et al. a software system's component can be safely updated if that component is not beeing used and will not be used in a near future.

In the context of cyberphysical systems (*i.e.* systems composed of software subsystems and physical subsystems), alterability of the system needs to consider the physical state of the system as well as its logical state (*i.e.* the state of its software subsystems). Horiuchi et al. [6] proposed methods for handling movement discontinuity when handling dynamic updates on moving robots. One method is based on the interpolation of functions from the older and the newer version to smooth the discontinuity caused by updating. That method modifies the behaviour of the system to ensure alterability. Another method proposed by Horiuchi et al. is to postpone the update until the system is in a state where movement discontinuity is not a problem (*e.g.* the robot is at a stable position) or until the system is in a state that can be reached in both older and newer version, ensuring absence of discontinuity.

This paper is based on the works of Panzica La Manna et al. [12] which proposed criteria for safely updating control subsystems of cyberphysical systems. Modelling systems as cyclic oriented graphs with labelled transitions, they identified states where the system is alterable. They defined an updatable state as a state which history in the original graph (*i.e.* all possible sequences of transitions leading to this state from the initial state in the original graph) are valid in the updated graph (*i.e.* each of these sequences of transitions can be fired from the initial state in the updated graph) and are completed by a same sequence of transitions in the updated graph (*i.e.* all the sequences of transitions from the history of the considered state have the same future in the updated graph).

They also define weakly updatable states as states that can be reached from a single non-initial updatable state and which history contain the history of the original updatable state (*i.e.* every sequence of transitions that leads from the initial state to the original updatable state can be fired from it and lead to the weakly updatable state). As for regular updatable states, the future of every sequence of transitions that lead to the weakly updatable state from the original updatable state must be the same in the updated graph. Weakly updatable states do not ensure that updates will necessarily be safe. Before applying an update at a weakly updatable state, one must verify that the continuation of the execution in the newer version will stay compliant with the specification of the newer version.

Figure 3 presents an example of update applied to a oriented graph with labelled transitions.

According to Panzica La Manna et al. states B and C are updatable. They are respecively accessed from the initial state by the sequences of transitions a and a,b in both older and newer versions. State D is not updatable as it is not reachable from the initial state by the sequence of transitions a,b,c in the newer version. State E is weakly updatable because it can be reached from B by the sequence of transitions a which is the same sequence of transitions by which the updatable state B is reached from the initial state in both older and newer version. Provided firing transitions b then d when the system is in state E complies with the new specification, the control system can be safely updated at state E. The definition of updatable states given by Panzica La Manna et al. is used in the following sections of this paper to identify states where the controlling systems of single track lines can be safely updated.

3 Shunting lines on single track rail lines

In France, over 3000 kilometers of railway lines are single track. Trains coming from both directions of such lines share the same track, requiring specific consideration to avoid collisions. To address that issue, stations placed on such lines have deviated paths or dedicated shunting tracks as shown on figure 1. Because only one train at a time can travel on the rail track, one of the train needs to be temporarily removed from the track to allow passage of the other train. Shunting and deviation tracks are used for that purpose.

Stations, and therefore shunting and deviation tracks, can be separated by a distance of up to 20 kilometers. In the case of a routing error allowing two trains to travel in opposite directions between two stations, the only solution is to have one of the trains travel backwards to its last encountered station. Such maneuver is slow and requires the intervention of more human operators for safety reasons. In the worst cases the train may have to travel for 20 kilometers in such a fashion. Traffic on single track rail lines includes passenger trains and freight trains. The latter usually travelling at lower speed and having more complex shunting procedures, routing errors may cause extended delays, seriously disrupting rail services.

Shunting and deviation tracks may be unavailable due to maintenance operations or because their location is attributed to the construction of a new building. In such situations, automatic routing systems are locally replaced by human routing agents. Designing automated routing systems able to adapt to the suppression of shunting and deviation lines could make the mobilisation of human routing agents unnecessary or at least, make their tasks easier. Designing such a routing system raises the issue of transitioning from regular routing (when shunting and deviation tracks are available) to special routing (with unavailable shunting and/or deviation tracks). For obvious reasons, the routing system should operate the transition while the removed track is still physically available, otherwise it could send a train to a non-existing track. This condition is not sufficient to ensure a safe transition. If handled improperly, that transition may cause problematic situations requiring human intervention. For example, if the routing system transitions to special routing while a train is waiting on a shunting track, it would *forget* that train. Human intervention would then be necessary to insert that train back into the traffic.

The rest of the paper considers the case depicted on figure 1. Stations A and B are joined by a single track line. Both stations A and B have a deviation track and only station A has a shunting track. Trains can enter and leave a deviation track without having to go backwards. Trains using the shunting track need to travel backwards either for entering the track or for leaving it. Because travelling backwards on the main tracks implies strict safety protocols including the mobilization of personnel, we consider the shunting track as only accessible when travelling forward. Trains would have to travel backwards only when leaving the shunting track, which requires less strict safety protocols. We consider two trains respectively name $t1$ and $t2$, going from station A (respectively from station B) to station B (respectively to station A). A real-life situation would have stations A and B connected to other stations and several trains coming through the two considered stations. For simplicity purpose and without loss of pertinence for the objective of this paper, we only consider the two stations but consider that after reaching their destination the trains will turn around (which rarely happens in reality) and go back to their origin station. Such consideration approximate the passage of many

trains on the line.

4 Model and Updatable states

In order to use the results of Panzica La Manna et al. [12], we model the case depicted on figure 1 as an oriented graph with labelled transitions. That graph is obtained from the parallel composition of several smaller graphs modelling the different components of the single track line. We use the same composition rules as defined by Panzica La Manna et al.

All graphs are presented in the Finite State Process (FSP) notation and we use the LTSA tool [9] for calculating graph composition and checking for deadlocks.

4.1 Case Study modelling

The model of this case study needs to consider safety constraints. To avoid collision, two trains cannot travel in opposite directions on a same track. Safety protocols of railway traffic require that two trains cannot travel through a given track section at the same time. We add the constraint that trains cannot enter the shunting track of station A if they are travelling from station A to station B. We enforce these constraints in our model by making illegal transitions (*i.e.* transitions that would infringe the safety rules) impossible in the graph model.

Figure 4 details the graphs modelling the different components of the single track line in the FSP syntax. Each subfigure details the graph of a component of the whole system (*e.g.* station, train, track). States of the graphs are written in uppercase and transitions between theses states are indicated by an arrow (−>) symbol preceded by their names written in lowercase. The | symbol separates possible transitions that can be fired from the defined state. Figure 4a details the graph modelling station A. Its initial state STATIONRAIL models the station having its shunting and deviation tracks free and having a train on the station main track. Through the transition deviation (respectively shunt), the station changes its state to having its main track free and its deviation (respectively shunting) track occupied. Transition leavedeviation (respectively leaveshunt) corresponds to a train leaving the deviation (respectively shunting) track and occupying the main track. Transitions enter and leave correspond respectively to a train entering and leaving the station. Station A is at maximum occupancy when trains occupy both shunting and deviation tracks as well as the main track. The graph modelling station B is similar to the graph modelling station A excepted it does not have transitions shunt and leaveshunt and their associated states. When composed together, the graphs modelling station A and station B have respectively their transition labels prefixed by a. and b., to ensure the stations can change state independently. The graphs modelling the stations only enforce the constraint disallowing two trains to occupy a same track section at the same time. The shunting track, the deviation track and the main track are three different sections and only when a section is free can it be occupied by a train. The graphs do not enforce the constraint requiring trains to be travelling from station B to station A for them to enter the shunting track. Indeed, station graphs have no memory of the direction of the trains. That constraint is enforced in the train graphs.

Figure 4b details the graph modelling the trains. It has the transitions of both stations plus transitions switchtoA and switchtoB respectively corresponding to the train changing its direction to *towards station A* and *towards station B*. The graph enforces the *cannot enter shunting track travelling backwards* constraint as the a.shunt transition is only available when the trains direction is *towards A*. Two instances of that graph are created and composed with the station graphs. The first instance modelling train $t1$ has all its transitions prefixed by t1. and has TRAINatAtoB as initial state. The second instance modelling train $t2$ has all its transitions prefixed by t2. and has TRAINatBtoA as initial state. When composed with the station graphs, common labels to train graphs and stations graphs (*e.g.* leave or shunt) are synchronized. That means that when the transition labelled t1.a.leave is fired in the $t1$ train graph, transition a.leave is fired in the station A graph.

Figure 4c details the graph enforcing that a train cannot enter the track joining station A and station B (considered as a single track section) if another train is already travelling on it. It ensures that any train entering the track leaves it before another train enters it. A train enters the track

```
STATIONRAIL = (deviation -> STATIONDEVIATION
        |shunt -> STATIONSHUNT
        |leave -> STATION),
STATION = (enter -> STATIONRAIL),
STATIONDEVIATION = (leavedeviation -> STATIONRAIL
        |enter -> STATIONRAILDEVIATION),
STATIONSHUNT = (leaveshunt -> STATIONRAIL
        |enter -> ASTATIONRAILSHUNT),
STATIONRAILDEVIATION = ( leave -> STATIONDEVIATION
        | shunt -> STATIONDEVIATIONSHUNT),
STATIONRAILSHUNT = ( leave -> STATIONSHUNT
        | deviation -> STATIONDEVIATIONSHUNT),
STATIONDEVIATIONSHUNT = (leaveshunt -> STATIONRAILDEVIATION
        | leavedeviation -> STATIONRAILSHUNT).
```

(a) Station A graph

```
TRAINatAtoB = (a.deviation -> TRAINatADEVIATIONtoB
        |a.leave -> TRAINatLINEtoB),
TRAINatADEVIATIONtoB = ( a.leavedeviation -> TRAINatAtoB),
TRAINatLINEtoB = (b.enter -> TRAINatBtoB),
TRAINatBtoB = (b.deviation -> TRAINatBDEVIATIONtoB
        |switchtoA -> TRAINatBtoA),
TRAINatBDEVIATIONtoB = (b.leavedeviation -> TRAINatBtoB),
TRAINatBtoA = (b.deviation -> TRAINatBDEVIATIONtoA
        |b.leave -> TRAINatLINEtoA),
TRAINatBDEVIATIONtoA = (b.leavedeviation -> TRAINatBtoA),
TRAINatLINEtoA = (a.enter -> TRAINatAtoA),
TRAINatAtoA = (a.deviation -> TRAINatADEVIATIONtoA
        | a.shunt -> TRAINatASHUNT
        | switchtoB -> TRAINatAtoB),
TRAINatADEVIATIONtoA = (a.leavedeviation -> TRAINatAtoA),
TRAINatASHUNT = (a.leaveshunt -> TRAINatAtoA).
```

(b) Train graph

```
TRACK = ( t1.a.leave -> t1.b.enter -> TRACK
        | t1.b.leave -> t1.a.enter -> TRACK
        | t2.b.leave -> t2.a.enter -> TRACK
        | t2.a.leave -> t2.b.enter -> TRACK).
```

(c) Track graph

```
TIMETABLE = (t2.b.leave -> t1.a.deviation-> t2.a.enter -> t2.a.shunt ->
    t1.a.leavedeviation -> t1.a.leave -> t2.a.leaveshunt -> t1.b.enter ->
    t1.b.deviation -> t2.switchtoB -> t2.a.leave -> t2.b.enter -> t2.switchtoA ->
    t2.b.leave -> t1.b.leavedeviation -> t2.a.enter -> t1.switchtoA -> t1.b.leave ->
    t2.a.deviation -> t1.a.enter -> t1.a.shunt -> t2.a.leavedeviation ->
    t2.switchtoB -> t2.a.leave -> t1.a.leaveshunt -> t1.switchtoB -> t2.b.enter ->
    t2.switchtoA ->  TIMETABLE).
```

(d) Timetable graph

Figure 4: Model of the initial control system

```
STATIONRAIL = (leave -> STATION),
STATION = (enter -> STATIONRAIL).
```

(a) Station B graph

```
TRAINatAtoB = (a.deviation -> TRAINatADEVIATIONtoB
                            |a.leave -> TRAINatLINEtoB),
TRAINatADEVIATIONtoB = ( a.leavedeviation -> TRAINatAtoB),
TRAINatLINEtoB = (b.enter -> TRAINatBtoB),
TRAINatBtoB = (switchtoA -> TRAINatBtoA),
TRAINatBtoA = (b.leave -> TRAINatLINEtoA),
TRAINatLINEtoA = (a.enter -> TRAINatAtoA),
TRAINatAtoA = (a.deviation -> TRAINatADEVIATIONtoA
                            | a.shunt -> TRAINatASHUNT
                            | switchtoB -> TRAINatAtoB),
TRAINatADEVIATIONtoA = (a.leavedeviation -> TRAINatAtoA),
TRAINatASHUNT = (a.leaveshunt -> TRAINatAtoA).
```

(b) Train graph

```
BOCCUPIED = (b.leave -> BFREE),
BFREE = (b.enter -> BOCCUPIED
        | a.leave -> BFREE).
```

(c) Correction graph

```
TIMETABLE = (t2.b.leave -> t1.a.deviation-> t2.a.enter -> t2.a.shunt ->
    t1.a.leavedeviation -> t1.a.leave -> t1.b.enter -> t1.switchtoA ->
    t1.b.leave -> t1.a.enter -> t1.a.deviation -> t2.a.leaveshunt ->
    t2.switchtoB -> t2.a.leave -> t1.a.leavedeviation -> t1.switchtoB ->
    t2.b.enter -> t2.switchtoA ->  TIMETABLE).
```

(d) Timetable graph

Figure 5: Model of the updated control system

whenever its leaves one of the station (*i.e.* transitions of the form $*$.leave are fired). That train leaves the track by entering the other stations, (*i.e.* the corresponding *enter* transition is fired).

The parallel composition of the station graphs, of the train graphs and of the track graph results in a global graph with one hundred states that models all the legal behaviours of our single track line example. It enforces the safety constraints presented at beginning of the subsection and the LTSA tool analyzed it as safe from deadlocks. However, it does not model a complete control system as it does not give specific instructions on how and when the trains should travel. We compose that global graph with the graph from figure 4d that models a timetable instructing the movements of the trains. The timetable graph is cyclic to model the passage of several trains through the stations and could model a daily timetable or a weekly timetable. It is composed with the global graph while adding all transitions labels from the global graph to its dictionary. As a consequence no transition from the global graph can be fired unless it can be fired in the timetable graph. The resulting graph is the restriction of the global graph to the behaviour of the timetable graph. Because the timetable does not lead to illegal situation, the resulting graph is identical to the timetable graph. If the timetable graph allowed illegal situation, the result of its composition with the global graph would be a different graph with at least one ERROR state, corresponding to illegal states of the system.

We consider the removal of the deviation track of station B. The control system requires updating to take that removal into account. The global graph needs to be updated to model all the new possible behaviour and enforce the safety rules and a new timetable must be defined. Figure 5 presents the modifications of the global graph and the new timetable graph using the FSP syntax as in figure 4. Figure 5a shows the updated graph for station B. It now only has transitions leave and enter. Figure 5b shows the updated train graph where transitions b.deviation and b.leavedeviation where removed. The track graph is not modified in the new version. Composing the updated station, train and track graph

results to a graph containing a possible deadlock. Indeed, if a train is at station B and a train is on the track joining station A and station B, going towards station B, the global graph does not allow the train on the track to enter station B and does not allow the train at station B to leave the station. This would correspond in reality to a situation where human intervention is required to have the train on the track safely travel backwards to station A. To avoid such situation, we define the graph of figure 5c allowing trains to leave station A only when station B is free.

The new global graph has fifty six states and has no potential deadlocks according to the LTSA tool. The previous timetable being obsolete, a new time table has to be defined, considering the removal of station B deviation track. Figure 5d details the new timetable graph we composed with the new global graph to obtain the new control system graph. As in the previous version, that graph is the restriction of the global graph to the behaviour of the new timetable graph. Its initial state is equivalent to the initial state in the first version (*i.e.* one train on the main tracks of each station). We chose to design the updated version with the same initial sate because we consider the new control system as managing traffic on the new line configuration as if it always were configured this way. Moreover, this choice makes it easier to find updatable state according to the criteria defined by Panizca La Mana et al.

4.2 Finding updatable states

The definition of updatable states as given by Panzica La Mana et al. Can be formally summarized in our context as :

Definition 1. *Considering a graph $G = (i, S, T, L)$ in which $i \in S$ is the initial state of G, S is the set of all its states, T is the set of all its transitions (expressed in the form of $s_1 \xrightarrow{l} s_2$ with $s_1, s_2 \in S$, $l \in L$) and L the set of all its transition labels (also called the dictionary of G), a state $s \in S$ is updatable regarding an update from G to $G' = (i, S', T', L')$ if any sequence of transitions leading from i to s in G can be fired from i in G' and be completed by a same sequence of transition in G'.*

Analyzing the two versions of the control system graph, we can find updatable states according to the criteria of Panzica La Mana et al. We compare the graphs from figures 4d and 5d and list the states that can be reached from the initial state through the same transitions in both graph. Obviously the initial state is one of these states. All states from the first[3] to the sixth are reachable in both versions through transitions t2.b.leave, t1.a.deviation, t2.a.enter, t2.a.shunt, t1.a.leavedeviation and t1.a.leave. All these states are reachable in the updated graph from the initial state through a single sequence of transitions. As a consequence, all possible sequences of transitions leading to one of these states in the updated graph from the initial state are completed by the same sequence of transitions. States zero to six from the original graph are therefore updatable.

The definition of weakly updatable states as given by Panzica La Mana et al. Can be formally summarized in our context as :

Definition 2. *a state $s \in S$ is weakly updatable regarding an update from G to G' if an updatable state $s_i \neq i$ can be found in S such that all sequences of transitions that lead from i to s_i can be fired from s_i and lead to s and all sequences of transitions leading to s from s_i are completed by a same sequence of transitions in G'.*

Analyzing the two versions of the control system graph, we can find no weakly updatable states. Indeed no state can be reached from the sixth first states (*i.e.* the only updatable states of the graph) through a sequence of transitions that begins with t2.b.leave (the first transition that leads to the first updatable state from the initial state). The linear nature of the control graphs resulting from the modelling of our case study make the appearance of weakly updatable states less probable.

5 Discussion and conclusion

After analyzing our model of the case study, we identified six states where the control system can be safely updated at the beginning of its cycle. If the control system is at a later state, we need for

[3]The initial state being enumerated as state zero, the first state is the state reached by firing the first transition

```
STATE15 = (t2.a.enter -> t2.a.shunt -> t1.b.leave
    -> t1.a.enter -> t1.a.deviation).
```

Figure 6: Example of transitional timetable

it to finish its current cycle before updating. This is not an issue when the removal of the deviation track of station B is planned. However if an unplanned event (*e.g.* an accident, exceptional weather conditions) makes that track unavailable, we may not have the time to wait for the completion of the control system cycle to update it. A solution would be to guide the execution of the control system to alterability, as proposed by Horiuchi et al. [6]. A specific transitional timetable, taking the special conditions into account, could be designed to lead the control system from its current non-updatable state to an updatable state. An example of such transitional timetable could be the one presented on figure 6, guiding the control system from its fifteenth state (non-updatable) to its fourth state (updatable). The transitional timetable would be composed with the newer version of the global graph, modified so that its initial state correspond to the fifteenth state of the original control system graph. The resulting transitional control system would be installed by the mean of a dynamic update. After reaching the last state of the transitional control system, another dynamic update would install the updated control system. In the case of these two updates, original and updated graphs have different initial states, disabling the usage of the results of Panzica La Mana et al. Other methods for ensuring the safe application of these updates would have to be studied.

Designing generic procedures for designing transitional timetables starting from any given non-updatable state is kept for future work as it may raise issues. For example, in our case study, if a train is stationed on the deviation track of station B, it is not possible to find a transitional timetable compliant with the new global graph. Possibility for safely using the older global graph must be studied and a refinement of the model presented in this paper would probably be needed.

Our case study considered two timetables and found several updatable states. Selecting different timetable could have lead to finding more or less updatable states. For example, switching transitions t2.a.leaveshunt with t1.b.enter fired respectively from states six and seven in the initial timetable would have made the seventh state updatable. The choice of timetables has no impact on the point of this paper. The issue of identifying optimal updated timetables maximizing the number of updatable states is kept for future work.

Our work showed the pertinence of using the concept of alterability to address the issue of modifications on control signaling systems. It increased the interest for the deployment of computer based systems for managing traffic on single track rail lines by increasing the degree of automation. Reconfiguration of the line can be safely and automatically handled. Further work on the issue of updating control systems would increase automation. As a result, operating costs on single track rail lines would decrease and human workers conditions would improve. Another perspective of our work is that through the use of modelling and of formal specification, the safety of the control systems could be formally proved, increasing safety of traffic. Moreover, in today's world where updates are the cornerstone of systems safety and security, each update applied on the control system could be checked to ensure that their application guarantees the safety of passengers.

References

[1] J. R. Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, New York and NY and USA, 1996.

[2] R. Ben-Ayed, S. Collart-Dutilleul, and E. Prun. Formal method to tailored solution for single track low traffic french lines. *International Railway Safety Council 2016*, 10 2016.

[3] Rahma BEN AYED, Simon Collart-Dutilleul, Philippe Bon, Akram Idani, and Yves Ledru. B Formal Validation of ERTMS/ETCS Railway Operating Rules. pages p124–129, June 2014.

[4] Clearsy. *Low Cost High Integrity Platform*, 2016 (accessed November 25, 2020).

[5] Antoine Ferlin, Rahma Ben-Ayed, Pengfei Sun, Simon Collart-Dutilleul, and Philippe Bon. Implementation of ertms: A methodology based on formal methods and simulation with respect to french national rules. *Transportation Research Procedia*, 14:1957 – 1966, 2016.

[6] Eiichi Horiuchi, Osamu Matsumoto, and Noriho Koyachi. Safety issues in nonstop update of running programs for mobile robots. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 01 2005.

[7] Jeff Kramer and Jeff Magee. The evolving philosophers problem: Dynamic change management. *Software Engineering, IEEE Transactions on*, 16:1293–1306, 12 1990.

[8] Thierry Lecomte, David Déharbe, Denis Sabatier, Étienne Prun, Patrick Péronne, Emmanuel Chailloux, Steven Varoumas, Adilla Susungi, and Sylvain Conchon. Low cost high integrity platform. *CoRR*, abs/2005.07191, 2020.

[9] Jeff Magee, Jeff Kramer, Robert Chatley, Sebastian Uchitel, and Howard Foster. *Labelled Transition System Analyser*, 2006 (accessed November 25, 2020).

[10] Sébastien Martinez, Dalay Israel de Almeida Pereira, Philippe Bon, Simon Collart-Dutilleul, and Matthieu Perin. Towards safe and secure computer based railway interlocking systems. *International Journal of Transport Development and Integration*, 4:218–229, 07 2020.

[11] Sébastien Martinez, Fabien Dagnat, and Jérémy Buisson. Pymoult : On-Line Updates for Python Programs. In *ICSEA 2015 : 10th International Conference on Software Engineering Advances*, pages 80 – 85, Barcelone, Spain, November 2015.

[12] V. Panzica La Manna, J. Greenyer, C. Ghezzi, and C. Brenner. Formalizing correctness criteria of dynamic updates derived from specification changes. In *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 63–72, 2013.

[13] Bougacha Racem, Abderrahim Ait Wakrime, Slim Kallel, Rahma Ben Ayed, and Simon Collart-Dutilleul. A model-based approach for the modeling and the verification of railway signaling system. pages 367–376, 01 2019.

[14] Pengfei SUN, Simon Collart-Dutilleul, and Philippe Bon. A formal modeling methodology of the French railway interlocking system via HCPN. June 2014.

Journal's webpage: http://univagora.ro/jour/index.php/ijccc/

This journal is a member of, and subscribes to the principles of,
the Committee on Publication Ethics (COPE).
https://publicationethics.org/members/international-journal-computers-communications-and-control

Cite this paper as:

Martinez, S.; Collart-Dutilleul, S.; Bon, P. (2022). Identifying alterability states of a single track railway line control system, *International Journal of Computers Communications & Control*, 17(5), 4444, 2022.

https://doi.org/10.15837/ijccc.2022.5.4444