Formally Unfolding Interlocking Architectures
MFES Project 13/14

Jorge L. Lopes[1] and Paulo R. Silva[1]

Universidade do Minho pg25285@alunos.uminho.pt pg25332@alunos.uminho.pt

Abstract. The answer to the question "Which is the best architecture for an interlocking system ?" is difficult, since the most common, the geographic and centralized architectures have both advantages and disadvantages. The system implemented by Efacec follows the geographic design. Thus, the pros and cons of this architecture are well identified, while regarding the centralized one they are not so clear. The aim of this project is to model centralized interlocking architectures for a better understanding and comparison with other variants.

1 Introduction

1.1 The Problem

Railways control system and interlocking. Interlocking is a small part of railways world, however it is the one that provides safety and allows us to step into a train feeling safe. It works like a function that guarantee that all of the allowed train movements occur safely. For this reason, and being aware the railway control and signalling is in fact a great critical system, interlocking comes as a crucial part of it regarding safety. Therefore, the matter is considered as one of the biggest challenges of Computer Science, specially of the formal methods.

In more technical terms, interlocking ensures a set of special safety invariants, and regarding interlocking systems, safety means, among others, collision-free systems and no derailment. On the other hand, interlocking system has also to guarantee liveness properties to prevent deadlock. Following are some basic notions, namely elements, which will help clarifying better the interlocking policies:

1. Tracks are split into *track sections* or just *sections*, provided of some technology capable of making the controller knows if it is occupied or free.
2. Points are elements that are positioned in tracks and have two configurable states: *Normal* and *Reverse*, being the normal position the one that allows trains to move straight ahead and the other one to change into another line.
3. A signal is what we would call semaphore, and helps controlling train movements by either forbidding it or allowing it. In this approach, signals have only two aspects: green and red. Usually they are positioned between two sections.
4. A route consists in consecutive track sections which starts and ends at signals. This is the main element of the interlocking. It both deals with and relates with other routes to check their availability.

In Fig 1 we can find a brief example of how does interlocking work. For instance, a train following a route that begins in *signal* $S1$ and ends in $S3$ will have to stop at $S1$, because it is its home signal, and when the interlocking ensures that *point* $P1$ is set in *normal position* - which will allow straight movement - and locked, will then the green aspect be shown in $S1$. Moreover, the tracks *a_TAA*, *a_TAB* and *a_TAC* are also locked for exclusive use of the train, being the latter representing overlap [1].

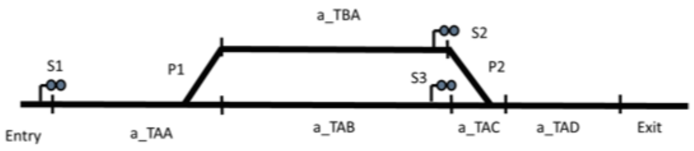


Fig. 1: Example of Track Layout

These principles are generic among the several types of interlocking systems. Which means that no matter what kind of interlocking we are referring, they remain valid and useful in all of them. From the existent architectures of interlocking that are used, those ones focused in this project were the geographical and centralised ones, in order to help answering the question *Which is the best architecture for an interlocking system?*, proposed by EFACEC's representatives.

Centralized and Geographic Interlocking Accordingly to the requirements of our industrial supervisors, the main goal is to explore the difference between the centralized and geographical architectures, regarding its maintainability and flexibility of implementation. It is important being aware there are many aspects in which both architectures can differ, and attached to that, those advantages and disadvantages can change. For this project, however, answers were achieved considering a centralised implementation and a previous study of geographic architecture.

Geographic Interlocking This architecture, which is widely used for electronic and relay interlocking, consist in define relations between each element belonging to a route with its neighbours. When a particular route is to be set , its path is checked through this topological relations, i.e, the first signal will *ask* to the next element if it is free and unlocked, and so on until the last. Then the process is inverted but this time they will either be set in their correct positions and lock or remain as they were, regarding their previous condition. An example of its implementation, using the layout presented in Fig 1 is the following: if route $S1$ to $S2$ is requested, the elements being part of it, namely $S1$, *a_TAA*, $P1$ and *a_TBA* will be *asked* if it is possible to grant the route. Since

[1] Safety margin beyond a stop signal

they have built-in that route logic, they know its needs. Then they provide the answer to the controller.

Centralized In this architecture, also known as tabular interlocking, all possible routes are defined in a control table, indicating exactly which elements belong to the respective route and in which position. This kind of interlocking is provided with all logic and when a route has to be set, it will check weather the elements are available or not. Using the same example - route $S1$ to $S2$ - the procedure is different. There is a control table where the elements are up-to-date, and the the ones above-mentioned will have a mark saying if they can or cannot be set for that respective route. Once they are available, the interlocking mark them again as locked for the route requested.

1.2 Approaching the problem

Efacec proposal consists in a project to implement an interlocking system based on a centralised architecture. The goal of this project is to raise the knowledge about this architecture, in order to obtain a more detailed information concerning its advantages and disadvantages.

The first steps in this project were given in order to understand the problem. The research was done towards formal methods approaches [GHMR13], allowing to face the problem accordingly to the course. The notions of interlocking and safety requirements were understood based on what is explained in *Interlocking Principles* from *Railway Signalling* & *Interlocking* [TV09]. The chapter gives full description of interlocking policies covering every aspect in which safety faults can occur and how to avoid them. It also provides brief descriptions of interlocking architectures, which is intended to fill the needs to proceed with the project.

In terms of modelling, unlike the formal concepts of interlocking, the situation differs. Since Railway's Safety System is faced as a great challenge of the computer science, the subject is an historical target of formal methods. Furthermore, such researches concern several areas within formal methods, where many techniques and tools are used and a variety of results is achieved. Hansen [Han98] suggests a formal validation of models by verification and simulation using VDM. JR Abrial derived a large number of different requirements for railway system in [Abr10]. Winter model checks interlocking systems using the process algebra CSP [Win02]. However, one of the most interesting approaches is the one made in Swansea University, combining techniques of state-oriented and event-oriented based approaches. Moreover, one of the implementations of this project is inherited from that approach, that was widely studied and the authors were asked to provide what, in our opinion, is a fantastic open source tool to Railway Verification [JTT+13]. Notwithstanding, for some good reason, that tool was never provided, but the work continued to proceed. Meanwhile, the external supervisors proposed an implementation in *SCADE*, which led to another stage in the research. In [GHMR13] was founded an article named Verification of solid state interlocking [JLM+14] which gives links for certain interesting approaches. It is notorious the formal work developed using this technology, even consider several results studied upon *Lustre*, which is the language underlying SCADE Suite tool. Andrew Lawrence's thesis gives a detailed description of how to implement and verify railways interlocking in SCADE [LSLS10], that was of great help.

Although a direct approach in interlocking architectures was highly desirable, it seems there is no such documentation, neither research concerning the subject. Nevertheless, the implementation was made taking centralised architecture into account, in a way that makes possible to one relate both designs.

1.3 Outline

Section 1 introduces introduces interlocking context and explains the approach that was taking, considering another researches. In Sec 2 the technique CSP||B [MNR$^+$12b] is detailed and is given a full description of the model. Then in Sec 3 is summarized a research made in SCADE language since the attempt to implement it was not completed. Section 4 concludes the project by telling what was achieved and what was learned during this project. There is yet a Authors' Note to explain what could have been done better.

2 CSP||B

2.1 CSP||B and ProB

At this point, our interest relies on the analysis and simulation of a centralised architecture, developed under the combined technique CSP||B using ProB [MNR$^+$12b, LB08]. In order to achieve that, the approach followed was a Double Junction Case Study [MNR$^+$12a] allowing to have a good starting point on modelling, by being a mathematical language very familiar in this school. Despite mathematical formalism, the models are of easy-understanding and verification, helping us to successfully model it, due to our aforesaid background, verifying and demonstrate specifications. The article led to others, where was found detailed descriptions of how to model the components under the technique. Firstly, it was an understanding process of such techniques, which took its time but was successfully concluded. It describes a representation of a railway system behaviour very close to reality , consisting in a controller, which is modelled as a process in CSP [Dav13, Hoa85], requesting operations to interlocking and the latter communicating with the equipment to provide correct answers for the request, represented by B-Machines [ST05]. The double behaviour described suggests a mixed approach, where the events play a role for the behaviour of the system encoded through CSP processes, whereas the static part consists of data describing all model-specific elements, namely of a representation of the track plan and of several relations derived from the control tables, and furthermore, for every state has valid properties regarding safety that must be verified. Event-Based and State-based approaches applied solely can in fact allow to model a railway control system. However, combine this mixed technique is highly desirable [MNR+12a]. Through the literature, it is easy to understand that it is possible to build a generic modelling architecture, i.e, the CSP process Controller fitting several layouts, and for some particular cases, the machines fits as well, although for simulation effects, the latter becomes very concrete when provided with the respective layout data. Therefore, a generic model of the controller is presented in CSP, although interlocking is dependent of the layout, being necessary to change the topological data.

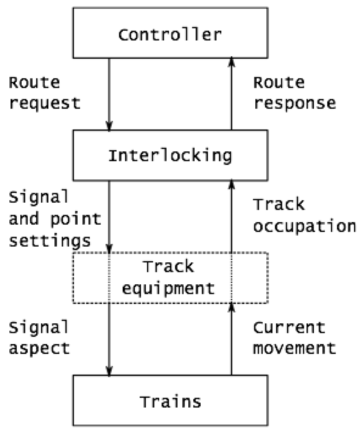


Fig. 2: Railways Behaviour

Figure 2 was adopted from the literature to depicts the architecture. The controller, as a regular and real controller, will send requests for routes and control movements of the trains, regarding authorisations by the interlocking, since they communicate directly through channels - bidirectionally - which means that a boolean answer is given when controller sends a request.

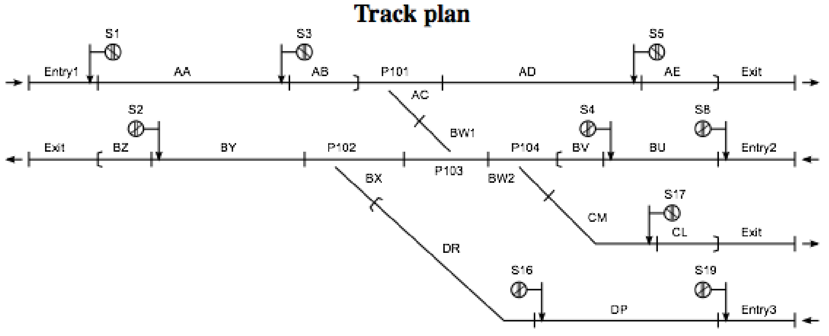
I n order to that answer be valid, interlocking will have to check and verify the current state of the railway system, and for that, it monitors the track equipment, which in turn will depend on train movements, and with respect to its defined rules, will answer either positively or not. These defined rules can change regarding the layout. It is a matter of interest to analyse since the subject Centralised *Vs* Geographic relies very much on it, concerning implementation.

The above-mentioned concrete part of a railway leads to a data-rich specification, that is why the double junction case study was chosen. To achieve such specification, the track plan is a crucial part for one to model a railway system. Usually it comes provided with a *control table* and a *track layout*, being the layout a design of the combined routes, and the control table the element that describes the inherent rules.

2.2 Model description

Figure 4 above depicts the layout of the double junction plus the control table and release table. The layout provides us with a lot of topological information concerning this specific example. As we may see, it consists in 22 tracks, 9 signals, and 4 points. Moreover, it tells us which track section follows the previous one, the sections where points are positioned as well as the signals. The studied approach attached four more signals in order to guard entry points, but nevertheless they are still home signals of the respective routes. These *entry* and *exit* points allow a train to disappear and appear again in the model. Otherwise once we travel to a route end the simulation will be over. Both tables in Fig 4 tells us how to control signals and points, regarding the requested

Track plan



Control table

Route	Normal	Reverse	Clear
3A	P101		AB, AC, AD, AE
3B		P101 P103 P104	AB, AC, BW1, BW2, CM, CL
4A	P101* P102 P103 P104		BV, BW2, BW1, BX, BY, BZ
16A		P102	DR, BX, BY, BZ

** flank protection*

Release tables

P101	Occupied
3A	AD
3B	BW1
4A	BX

P102	Occupied
3B	CM
4A	BY
16A	BY

P103	Occupied
3B	CM
4A	BX

P104	Occupied
3B	CM
4A	BX

Fig. 3: Extracted scheme plan of Double Junction

route. As described in 1.1, a route consists in successive neighbours sections and starts and ends at signals. In addiction, Fig 4 also shows there are 11 routes, although $S1$, $S5$, $S8$, $S17$, $S19$ and $S2$, are simple routes which do not have any points within it and were set only to consider overlap and the borders of the layout.
When a route is to be set, interlocking checks the control table for points and tracks that have to be verified free and unlocked. On the other hand, it will have to consult the *releaseTable* to know exactly when to release a lock.

For instance, if route $A4$ (from $S4$ to $S2$) is requested, points $p101$, $p102$, $p103$ and $p104$ have to be set in *normal position*, and despite p101 does not affect directly the train direction, lock it in *normal position* avoid other trains to come from the track above. This situation is referred as **flank protection** and is one of the main principles regarding interlocking. It is very important being aware that the design of these tables are safety-critical, hence situations like flank protection have to be taken into account, in order to avoid collisions.

Figure 4 demonstrate the architecture of the model itself. The real elements of the track plan are defined in *Context.mch*, which is the less generic machine, since contains all data representing the layout, including points, tracks, sections and signals. In contrast, ReleaseTable as well as Topology and ControlTable, has properties about the track plan, defined through relational properties, such as injection or Cartesian product, and just have to be instantiated among layouts. The controller is the entity which models both controller and trains. Those elements run independently of each other, making use of the process algebra interleaving operator.

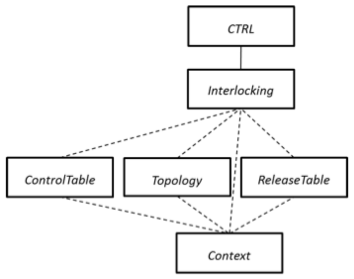


Fig. 4: Architecture

$$RW\_CTRL = \bigsqcap r : ROUTE \bullet (request!r?b \rightarrow RW\_CTRL)$$

RW_CTRL represents the controller. The operator non-deterministic indicates any route on the set *ROUTE* can be requested, being up to interlocking decide whether it is safe or not to accept.

```
$TRAIN\_CTRL(t, currp) =$
        $if(s = none \lor s = green)$
            $then(move.t.currp?newp \rightarrow$
                $(if(member(newp, EXIT))$
                    $then(exit!t!newp- > TRAIN\_OFF(t))$
                $elseTRAIN\_CTRL(t, newp))$
            $\square$
                $stay.t.currp \rightarrow TRAIN\_CTRL(t, currp))$
            $elsestay.t.currp \rightarrow TRAIN\_CTRL(t, currp)$
```

On the other hand, the trains are modelled to take internal decisions: if it stays or moves in front of a green signal or even when there is none at all, but it definitely has to stop in front of one showing the red aspect.

At this point the parallel composition in CSP||B is revealed when the controller in form of CSP processes performs events offered by *Interlocking.mch* as operations, which are defined in CSP as communicating channels between the machines and the processes. As so, the Interlocking operations only run by reacting to the controller requirements, representing in this way a centralised control logic, since it is all processed on this machine only.

The *ControlTable.mch* is very similar to the one in Fig 4, and contains the rules and relations between the points, routes and track sections, split into *normalTable*, *reverseTable*, and *clearTable*. Whereas the former ones determine the correct position for points concerning each one, the latter consists in a injective function that point out for each route the sections that must be clear and, thus, will be locked. Due to its injective nature, *clearTable* ensures that there are not two different routes made of the same track sections. The *Topology.mch* encodes the track plan design, helped by assumptions of the authors, such as a single line can only be travelled in one direction. It describes which signals are associated with the respective route, where they are, in which track belongs the points and the successor relation between tracks.

The *Interlocking.mch* is provided with several operations, hence is a dynamic type of machine, and the only one in this model. It *SEES* all the other machines abovementioned, and gathers their information. One particular aspect of this machine, which is determinant, is the relation *pos*, that models the train location on tracks. Since it is a partial injective function, it captures one safety requirement: no collisions; because the nature of an injective function does not allow behaviours such as two trains on the same track. Moreover, this function remains as invariant along with the evolution of the system. Therefore, any violation would be of short-notice. Furthermore, the allowed range in *pos* is contained in set *TRACK*, which does not have *nullTrack* as inhabitant, so derailments are also prevented. Interlocking also captures the status of each signal by a total injective function *signalStatus*, which demands that there are no signals without an aspect. Points are controlled within it using the set variables *normalPoints* and *reversePoints*, providing runtime information whether some point is set normal or reverse, and *currentLocks*, just like the name says, represent the points which are locked in each state. Also relevant is the *nextd* function that captures the information about successor tracks, regarding the position of train and the configuration of points within it, if there is any.

The model makes use of $B - Method$ [Abr05] feature INITIALISATION to set all the tracks to being empty, all signals to red, all points in normal position and no locks made. Operations in this model are within Interlocking.mch resorting to the information data among the other machines. There are the following 5 operations:

enter operation is a simple entering movement by an existent train which is not positioned on any track. Then updates the positions and also the set occupiedTracks and emptyTracks.

exit operation enables movements to pull off the tracks once the line has come to its end. Likewise enter operation, the same elements are updated.

nextSignal operation evaluates the signal on the track to either allow or forbid the train movement. The output is obtained by checking their existence on tracks and their aspect, and serve as input to the CSP process.

move , as the name suggests, provides updates to model the evolution of the train on tracks. In case of the destination track has a signal, its aspect is automatically set to red.

request operation is the main operation and the one controlling the interlocking system. When the controller sends out a request to the interlocking, it has a route as input and essentially the operation provides the correct response to the controller, which is the main goal of an interlocking system. In order to accomplish that, firstly it checks if the respective tracks belong to set *emptyTracks* and then it verifies the points unlocked. In both cases, whether if one point or section fails, the output will be *no* (or *false*), but if points are unlocked and all the tracks clear, then it proceed with the configuration for the route. In other words, it locks the points in correct position and add them to the currentLocks set, and also updates the topology since successors may have been changed by the points re-configuration, and only after check again that tracks cleared, set the signal to green. In addiction to what is described in the literature, and as demanded by our industrial partners, some features are added to the original model, namely *lockedTracks*. This feature allows the Interlocking not only to lock the points, but also to lock the sections, and with respect to that, beyond checking whether the tracks cleared or not, it also checks if they are locked for another route or if they are not.

```
$bb <-- request(route) =$
$PRE$
$route : ROUTE \quad THEN$
    $IF((clearTable(route) \subseteq emptyTracks))$
        $THEN$
        $LET$
            $unlockedPointsBE$
            $unlockedPoints = POINTS - ran(currentLocks)$
        $IN$
            $IF((normalTable[route] \subseteq normalPoints \cup unlockedPoints)$
                $\wedge(reverseTable[route] \subseteq reversePoints \cup unlockedPoints))$
                $THEN$
                $LET$
                    $np, rpBE$
                    $np = (normalPoints \cup normalTable[route]) - reverseTable[route]$
                    $rp = (reversePoints \cup reverseTable[route]) - normalTable[route]$
                $IN$
                    $normalPoints := np \parallel$
                    $reversePoints := rp \parallel$
                    $currentLocks := currentLocks \cup (route \lhd lockTable) \parallel$
                    $signalStatus(signal(route)) := green \parallel$
                    $bb := true \parallel$
                    $nextd := staticNext \cup dynamicNext[(np \times \{normal\} \cup rp \times \{reverse\})]$
                $ELSE$
                    $bb := false$
        $ELSE$
            $bb := false$
```

2.3 Safety requirements

In this section we describe the invariants which are used in the model, regarding two safety requirements: No collision and no derailment. Having these two properties verified through invariants will ensure that the track plan is well defined allowing only safe movements to happen during the simulation.

The invariants *emptyTracks* and *occupiedTracks* demand that the considered tracks for train to travel are restricted to set *TRACK*. This ensures that a train will always travel within the defined layout topology, preventing derailment.

$$\begin{gathered} emptyTracks \subseteq TRACK \\ occupiedTracks \subseteq TRACK \\ emptyTracks \cap occupiedTracks = \emptyset \end{gathered}$$

nextd relation was updated in this project to a set of relations between *TRACK*. In the followed approach, this relation is defined as a set of partial injections, but this restriction does not allow travelling in both directions as it needs to happen in track $BW1$ and $BW2$.

$$nextd : TRACK \leftrightarrow TRACK$$

pos prevents both collision and derailment..

Variables *normalPoints* and *reversePoints* are defined in order that both are restricted to set *POINTS*. Moreover, through their intersection and reunion being empty and equal to the whole set, respectively, it prevents a point from being present in both set *normalPoints* and *reversePoints* at the same time, which can cause a malfunction in model and leading to derailment.

$$\begin{gathered} normalPoints \subseteq POINTS \\ reversePoints \subseteq POINTS \\ normalPoints \cap reversePoints = \emptyset \\ normalPoints \cup reversePoints = POINTS \end{gathered}$$

In order to set *currentLocks* be valid, it has to be confined to the previously defined *lockTable*, which is the correct locked points allowed with respect to the layout topology.

$$\begin{gathered} currentLocks : ROUTE \leftrightarrow POINTS \\ currentLocks \subseteq lockTable \end{gathered}$$

Property $not\ (collision \subseteq occupiedTracks)$ ensures that both $BW1$ and $BW2$, despite having double directions, are not allowed to occur simultaneously.

Moreover, ProB[MNR+12b, LB08, BCD+13] is capable of prove no invariant violations and deadlock freedom is also verified [Mor90].

Although there exist additional requirements [Abr10], the aim of the project does not rely exactly on the railway verification, but instead, as above-mentioned, on the comparison of interlocking architectures. Therefore, being aware of that, the important part is that there are some properties which does not instantiate to other track plans. Consequently, each railway model needs to be verified.

3 SCADE

3.1 Introducing SCADE Suite

SCADE In this section we present a not-so-practical approach to the *SCADE Suite tool*. It is a powerful tool widely applied in industry, in which many critical systems rely on.

SCADE moto - *Design*, *Verify*, *Generate* - characterises its versatility justifying its industrial use. Moreover, its reliability - Stalmark's algorithm is very explored and provide correctness in technique behind SCADE suite - makes the use of the tool very tempting to approach the embedded and critical systems. Furthermore, the graphical language tends to increase its user-friendliness.

3.2 Approaching

The intent concerning this approach is verify how well does *SCADE* Suite perform a well-formed design of centralised interlocking architecture, therefore it is intended to capture the maximum of the topology of the railway. Since *SCADE* is a low level language, it is not as easy as the previous approach to formalise about concrete properties of the railway domain. Nevertheless, the followed research models a railway system based on modularity, which is an interesting option since modularity allows the reuse of components, providing a kit from which it is possible to assemble the modules of well-chosen generic components. Consequently, it is possible to achieve a concrete level of topology.

In his research, Andy Lawrence models the elements of the railway using *state machines* and makes use of Caspi et al [CPHP87] work to model the train movements by using boolean data flows. These boolean data flows, which is a Lustre [CMSW99] feature to model synchronous systems, are used not only to represent movements but also signals and points' positions. The elements are modelled in a concrete fashion, otherwise it would be hard to distinguish each type of element.

In order to model a track section, it is used a node[2] which produces boolean variables as outputs, indicating whether the track is occupied or empty and if a crash happened. This process after the node has received boolean inputs, one of the inputs indicating if is a train entering, and three more to represent the signal aspect. There is two more track sections nodes: one similar to the above described but allowing a train to travelling in both ways and the other one to represent junctions on lines, modelled in such a way that the train changes its direction accordingly to the point's position.

The points are modelled in a node considering 4 possible states: normal and free, normal and locked, reverse and free, reverse and locked. The respective variables of the state machine are updated with respect to the current state. The node receives boolean inputs reflecting its state and if it is occupied, and then returns 4 boolean inputs indicating its current state.

On the other hand, the signals and its aspects are modelled separately, because signal works as a device to control its own aspect. This behaviour suggests that there is a inner logic in the elements, which in fact it is true.

[2] In Lustre a node is equivalent to a function or procedure

Node route, in contrast with the previous elements, provide a specification accordingly to the control table, i.e, to a given route, the elements such as signal aspects and points positions are set with respect to the control table. The input entries provide information concerning the state of the system, as points, lights and routes set.

There is a additional node called RouteController which captures the routes that conflict with each other, completing the implicit information given in the control table. This node acts like a filter of route requests allowing only the non-conflicting routes.

In order to connect the elements, they must be instantiated to concrete elements in reference to a concrete layout. For instance, to model a train travelling from one track to its successor, the variable which represents the train leaving the first track - through the output of the track - will be the input representing the train entering in the next one. This is why is called data flow language. Therefore, to achieve a concrete model of a given track plan, all the elements have to be declared and instantiated in a single node, provided with all the necessary input to simulate a railway system.

This section presented summary of an external approach, as aforesaid, that served to acquire knowledge in this field using SCADE Suite. With respect to the problem itself, there is one point noteworthy: the design of the described model does not represent a geographic interlocking design, since the route logic is not embedded in all the elements. Moreover, each element has within it solely the necessary conditions to be updated correctly. However, it is not considered a centralised design also, because there is no such entity controlling the railways safety movements, but instead, the properties are verified within the nodes and in particular nodes serving that purpose as the RouteController node. Henceforth, there is no conclusion taken from this section regarding the comparison between both architectures.

4 Conclusion

This project has proved to be a challenge since the beginning. The question *Which is the best architecture for an interlocking system?* it is indeed difficult to answer and there are plenty of reasons to find it hard respond. Perhaps because there is none. The best design solution depends on the needs of the implementation and also on of the available technology. Moreover, the lack of definitions of what an interlocking architecture is makes it difficult to assess the effectiveness of which is the best architecture.

In our CSP||B approach, a centralised interlocking implementation is intuitive because the technique relies on a high level language. Moreover, the simulation in ProB allow to verify the system state in runtime. Thereby it is possible to check the train and interlocking behaviour in all states.

In this first approach, the centralised design is almost mandatory and it is also well ahead of the geographic architecture, since this technique was applied to other small examples of track plans changing only the data and topology of the track plans, which was possible to achieve by keeping the same assumptions and for the same requirements. This constitutes an advantage over the geographic interlocking. Another advantage was noticed when there was a change in the layout and all that was done was changing a particular property in invariants, without compromising the validity of the model.

On the other hand, there is not much to say regarding the SCADE approach. Despite learning the concepts and the language which supports the tool, the implementation is not completed. Despite this, the followed research shows that it is possible to implement an architecture very close to a centralised one, although there is not sufficient experience to make a comparison between both interlocking designs.

4.1 Authors' Note

This project contemplates a real world problem which is the railways control system. The subject gives rise to numerous papers and thesis, indicating its diversity and grandiosity. For the approach, this was an extra motivation. It is a regret that, in the last months, a more completed approach was not possible. The major cause of this sense of incompleteness dues to the limitation in the use of the SCADE Suite tool, since *eduroam* does not allow the connections needed to use it, and a lot of time was lost in order to understand that. Another issue is that it can only be used by one at the same time.

Despite these adversities, the feeling is that a lot was learned and allowed to brought together all that was lectured in *MFES*' modules.

5 Future Work

Definitely, a complete implementation of an interlocking control system in SCADE following a centralised design would be the next step that we should be looking forward to do, although it would not be easy since it is difficult to find sufficient conditions to work with SCADE Suite, as it was until now. Furthermore, an implementation regarding a geographic design could be helpful to unveil more details concerning the differences between both interlocking architectures.

In spite of these aspects being more outstanding , during the project a lot of ideas had come to our mind while the development was on process. Abstractions of models in CSP||B would be other feature to add, in order to optimize the system verification. A technique to translate CSP||B models to SCADE [Ber97, BGGL92] it would also be useful, and in spite of not being in the scoop of the proposal, would indeed improve the modelling as well as defining interlocking requirements.

6 Acknowledgement

The project's authors would like to thank to the following: João Martins and Hélder Azevedo, for being available to discuss and help whenever they were asked for; Prof. Luís Barbosa, who provided guidance and support since the first day, and for his patience.

References

[Abr05] Jean-Raymond Abrial. *The B-book - assigning programs to meanings*. Cambridge University Press, 2005.

[Abr10] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[BCD$^+$13] Jens Bendisposto, Joy Clark, Ivaylo Dobrikov, Philipp Körner, Sebastian Krings, Lukas Ladenberger, Michael Leuschel, and Daniel Plagge. Prob 2.0 tutorial. In *Proceedings of the 4th Rodin User and Developer Workshop*, TUCS Lecture Notes. TUCS, 2013.

[Ber97] Didier Bert. Building lustre control systems from b abstract machines: A case study, 1997.

[BGGL92] Gerard Berry, Georges Gonthier, Ard Berry Georges Gonthier, and Place Sophie Laltte. The esterel synchronous programming language: Design, semantics, implementation, 1992.

[CMSW99] Paul Caspi, Christine Mazuet, Rym Salem, and Daniel Weber. Formal design of distributed control systems with lustre. In *in Proc. Safecomp'99*, pages 396–409. Springer-Verlag, 1999.

[CPHP87] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: A declarative language for real-time programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, pages 178–188, New York, NY, USA, 1987. ACM.

[Dav13] Tom Davies. *CSP Implementation Techniques – A Critical Analysis*. Swansea University, 2013.

[GHMR13] Stefan Gruner, Anne Elisabeth Haxthausen, Tom Maibaum, and Markus Roggenbach. *Towards a Formal Methods Body of Knowledge for Railway Control and Safety Systems: FM-RAIL-BOK Workshop 2013*. DTU Compute-Technical Report-2013. Technical University of Denmark, 2013.

[Han98] Kirsten Mark Hansen. Formalising railway interlocking systems. In *Department of Computer Science, Technical University of Denmark*, pages 83–94, 1998.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[JLM$^+$14] Phillip James, Andy Lawrence, Faron Moller, Markus Roggenbach, Monika Seisenberger, Anton Setzer, Karim Kanso, and Simon Chadwick. Verification of solid state interlocking programs. In Steve Counsell and Manuel Núñez, editors, *Software Engineering and Formal Methods*, Lecture Notes in Computer Science, pages 253–268. Springer International Publishing, 2014.

[JTT$^+$13] Phillip James, Matthew Trumble, Helen Treharne, Markus Roggenbach, and Steve Schneider. Ontrack: An open tooling environment for railway verification. In *NASA Formal Methods*, Lecture Notes in Computer Science. Springer, 2013.

[LB08] Michael Leuschel and Michael Butler. Prob: an automated analysis toolset for the b method. *International Journal on Software Tools for Technology Transfer*, 10(2):185–203, 2008.

[LSLS10] Andrew Lawrence, Monika Seisenberger, Andrew Lawrence, and Monika Seisenberger. Verification of railway interlockings in scade. In *AVOCS'10, Proceedings of the 10th International Workshop on Automated Verification of Critical Systems and the Rodin User and Develop Workshop*, pages 112–114. Springer, 2010.

[MNR$^+$12a] Faron Moller, Hoang Nga Nguyen, Markus Roggenbach, Steve Schneider, and Helen Treharne. Csp||b modelling for railway verification: The double junction case study. In *AVOCS'12*, 2012.

[MNR+12b] Faron Moller, Hoang Nga Nguyen, Markus Roggenbach, Steve Schneider, and Helen Treharne. Using prob and csp||b for railway modelling. In *Proceedings of the Posters & Tool demos Session, iFM 2012 & ABZ 2012*, 2012.

[Mor90] Carroll Morgan. Of wp and csp. In W.H.J. Feijen, A.J.M. van Gasteren, D. Gries, and J. Misra, editors, *Beauty Is Our Business*, Texts and Monographs in Computer Science, pages 319–326. Springer New York, 1990.

[ST05] Steve Schneider and Helen Treharne. Csp theorems for communicating b machines. *Formal Asp. Comput.*, 17(4):390–422, 2005.

[TV09] Gregor Theeg and Sergej Vlasenko. *Railway Signalling and Interlocking*. Eurailpress, 2009.

[Win02] Kirsten Winter. Model checking railway interlocking systems. *Aust. Comput. Sci. Commun.*, 24(1):303–310, January 2002.