Safety Interlocking as a Distributed Mutual Exclusion Problem

Alessandro Fantechi[1] and Anne E. Haxthausen[2]

[1] DINFO, University of Florence, Firenze, Italy
alessandro.fantechi@unifi.it
[2] DTU Compute, Technical University of Denmark, Lyngby, Denmark
aeha@dtu.dk

Abstract. In several large scale systems (e.g. robotic plants or transportation systems) safety is guaranteed by granting to some process or physical object an exclusive access to a particular set of physical areas or objects before starting its own action: some mechanism should in this case *interlock* the action of the former with the availability of the latter. A typical example is the railway interlocking problem, in which a train is granted the authorisation to move only if the tracks in front of the train are free. Although centralised control solutions have been implemented since decades, the current quest for autonomy and the possibility of distributing computational elements without wired connection for communication or energy supply has raised the interest in distributed solutions, that have to take into account the physical topology of the controlled areas and guarantee the same level of safety. In this paper the interlocking problem is formalised as a particular class of distributed mutual exclusion problems, addressing simultaneous locking of a pool of distributed objects, focusing on the formalisation and verification of the required safety properties. A family of distributed algorithms solving this problem is envisioned, with variants related to where the data defining the pool's topology reside, and to how such data rules the communication between nodes. The different variants are exemplified with references to different distributed railway interlocking algorithms proposed in the literature. A final discussion is devoted to the steps needed to convert the proposed definitions into a generic plug-and-play safety-certified solution.

1 Introduction

The current quest for autonomy of cyber-physical systems and the possibility of distributing computational elements without wired connection for communication or energy supply has raised the interest in distributed software solutions in which several computational elements cooperate to guarantee global properties. In the case of safety-critical systems, mastering the complexity of distributed solutions so to guarantee that safety is maintained is a hard task.

In this paper we address a particular class of safety-critical cyber-physical systems, showing how a systematic adoption of known distributed algorithms and of formal specifications can help to master the complexity.

In several large scale systems (e.g. robotic plants or transportation systems) safety is guaranteed by granting to some process or physical object an exclusive access to a particular set of physical areas or objects before starting its own action: some mechanism should in this case *interlock* the action of the former with the availability of the latter. A typical example is the railway interlocking problem, in which a train is granted the authorisation to move only if the track segments in front of the train are free.

The problem resembles a classical mutual exclusion problem: there are several active, or moving, physical objects (called from now on *processes*), that compete for the exclusive access for one or more free areas, which are actually shared *resources*.

Centralised solutions for this problem maintain the state of all shared resources, receive access requests and grant the exclusive access to the requesting process only if all the requested resources are free. Each resource can therefore have state = {available, requested by P_i, accessed by P_i} .

Notice that, since we are actually dealing with physical systems, the state of the resource has to reflect the actual state of a physical object: this may require that the *accessed* state is actually split in a state in which the resource is locked (meaning that the request by P_i has been met), and one in which the resource is physically *visited* by the process (see Fig. 1). Furthermore, the *requested* state may include not only a check that the related physical object is free, but also a command to the object to physically prepare it to be available to be visited, and a check that it is actually prepared, and this may take quite a long time. The subsequent states as well require some interaction with the physical object. Since a resource may be engaged in the *requested* state for long, concurrent requests by other processes should be served in the meanwhile. Atomicity of the treatment of a request is therefore guaranteed by denying requests of an already requested resource by other processes.

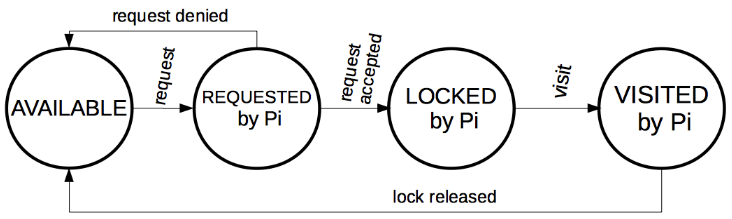

Fig. 1. States of a shared resource (*node*)

To guarantee safety of an interlocking system built according to this principle it is enough to prove that in any case two different processes cannot visit simultaneously the same resource, that is, any resource is exclusively locked by a single process. Putting this in temporal logic (CTL), it is sufficient to verify for each resource, and for each $i \neq j$ the formula $AG \sim (R\_visited\_by\_P_i \wedge R\_visited\_by\_P_j)$.

Not a big task for a model checker, if the principles above are expressed in a single finite-state model that takes into account the actual topology of the controlled areas. However, the experience with railway interlocking systems says that when several trains (processes) may require tens of track circuits and points, out of a pool of some hundreds, the combinatorial combination of the possibilities produces a state space explosion problem. This problem asks for suitable abstraction or compositional techniques, and for the power of recently available SAT and SMT-solvers to verify safety of the largest systems of realistic size [28, 9].

In this paper we suggest how a distributed formalisation of the interlocking problem can decompose this verification problem into manageable verification steps: the problem is formalised as a particular class of distributed mutual exclusion problems (Sections 2,3,4), addressing simultaneous locking of a pool of distributed objects, focusing on the formalisation and verification of the required safety properties. A family of distributed algorithms solving this problem is envisioned, with variants related to where the data defining the pool's topology reside, and to how such data rules the communication between nodes (Sect. 5). The different variants are exemplified with reference to different distributed railway interlocking algorithms proposed in the literature (Sect. 6). A final discussion is devoted to the steps needed to convert the proposed definitions in a generic plug-and-play safety-certified solution (Sect. 7).

2 Distributed Mutual Exclusion

In general, the Distributed Mutual Exclusion problem is typically characterised by the following statements:

- Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.
- Only one process is allowed to execute the *critical section*, that is, to access the shared resources, at any given time.
- In a distributed system there are no shared variables that can be used to implement mutual exclusion and semaphores.
- Message passing is the only means for exchanging information.

Either centralised or distributed Mutual Exclusion algorithms have typically to satisfy the following properties:

1. **Safety:** At any instant, only one process can execute the critical section.
2. **Liveness:** (absence of deadlock and starvation). Two or more processes should not endlessly wait for messages which will never arrive.
3. **Fairness:** Each process gets a fair chance to execute the critical section. Fairness generally means that the critical section execution requests are executed in the order of their arrival in the system.

Several Distributed Mutual Exclusion algorithms have been defined, especially in relation to distributed transactions, among which the most cited ones are Lamport's Algorithm [19], Ricart-Agrawala Algorithm [24], Maekawa's Algorithm [21].

Such algorithms actually guarantee safety, that is Mutual Exclusion, as obviously expected, and guarantee fairness and deadlock freedom at different degrees, with different performance parameters (number of messages, latency, throughput, response time).

3 The class of Distributed Mutual Exclusion problems of our interest.

In the case studied in this paper, we are interested in a Distributed Mutual Exclusion algorithm that primarily guarantees safety. Liveness and fairness are actually not a concern, since the focus is on guaranteeing that safety is not violated by multiple requests. If any process gets a request denial, it can just replay the request later: it is somehow assumed that this delay does not cause any major availability problem, because the normal interval between requests is largely greater than the time taken to accept or deny a request. If this assumption does not hold and hence availability becomes a problem, liveness and fairness should be then taken into consideration. This issue may impact on the definition of criteria to choose among different mutual exclusion algorithm variants (see Sect. 5), but we will not discuss it in details, leaving it to future work: the idea is that we concentrate on safety first, and then we will study availability and performability of the envisaged solutions.

We can recast the above problem as simultaneous locking of a pool of distributed nodes, in the following way:

- In this distributed setting, a physical resource is controlled by a dedicated computer, which is a *node* of a network. Hence, we will speak of nodes, rather than resources, from now on.
- A set of distributed nodes is *visited* by some computation *processes* (set of nodes N, set of processes P).
- In order to avoid conflicts between the computations of the processes, a process can request to exclusively lock a pool of nodes for an exclusive visit (pool of nodes $S \subseteq N$). We assume a predetermined set of possible pools $F_S \subseteq 2^N, \emptyset \notin F_S$, without loss of generality, since F_S can also be 2^N; a process request refers to a pool $S \in F_S$.
- In order to lock a pool of nodes, all nodes should be in (or should be brought to) a state in which they are available to be locked.
- If some node of the pool is not available, the lock request is denied.
- Otherwise, if all nodes are available, the lock is granted, and the process can start the visit of the pool.
- The lock on a node is singularly released after the process has declared to have finished visiting that node[3].

[3] This feature allows for partial release of the pool of nodes, at the advantage of other processes that want to request those nodes, so increasing availability.

Fig. 1 gives an abstract view of the states of a shared resource, that is, of a node; note that there may also be concrete transient states induced by the locking algorithm, such as "requested but not available".

This Distributed Mutual Exclusion problem is actually a simplified case of the general one presented in Sect. 2. Indeed, it can be reduced to a *Distributed Transaction problem (Distributed Atomic Action)*: in this problem, a set of nodes performs a distributed action, and the decision whether the action is committed has to be agreed among all the participants: if they do not agree, the action is aborted and the participants roll back to their previous state, so that either the distributed action has been fully performed, or it has not at all. In our case, the distributed action is the reservation of the requested pool of nodes.

4 2PC protocol for Distributed Mutual Exclusion

Algorithms to solve the Distributed Transaction problem have been defined since long time; the most popular one is the Two Phase Commit protocol (2PC) [13, 20].

4.1 Classical 2PC protocol

As the name says, the protocol works in two *phases*, according to the following steps for locking a pool of nodes S:

- Commit request phase (or Prepare phase)
  - The coordinator (a specially selected node in S) sends a *query to commit* message to all participants (all other nodes in S) and waits until it has received a reply from all participants.
  - Each participant replies with an *agreement* message or an *abort* message (an abort message may be due to the explicit denial to commit or the expiration of a timeout on the execution of an action or on a communication).
- Commit phase - *Success*
  - If the coordinator received an *agreement* message from all participants during the commit-request phase:
    - The coordinator sends a *commit* message to all the participants.
    - Each participant sends an *acknowledgement* to the coordinator.
- Commit phase - *Failure*
  - If any participant sends an *abort* during the commit-request phase (or the coordinator's timeout expires):
    - The coordinator sends a *rollback* message to all the participants.
    - Each participant rolls back and sends an *acknowledgement* to the coordinator.

This algorithm requires $4M$ messages, with $M+1$ nodes in S, and assumes that point to point communication is available, although broadcast communication from the coordinator can reduce the overall number of messages. The algorithm is fail-safe w.r.t. communication failures, in the sense that commit cannot be wrongly reached if communication fails somewhere.

4.2 Linear 2PC protocol for Distributed Mutual Exclusion

In this variant, participants are linearly ordered and each participant communicates with the previous and with the next participant. In the first phase, the coordinator makes the request to the first element of the pool, and each participant propagates the request to the next node in the list. In the second phase, the last participant replies OK if it is ready to commit, and the OK message is propagated backwards to the other participants; on its reception the first node delivers the OK message to the requesting process. If any of the nodes decides to abort, it propagates the abort messages in both directions. This algorithm needs - in the success case - only $2M$ point to point messages, and is hence favoured by a linear topology of the communication network.

4.3 Formalisation of the Linear 2PC protocol

Already [26] presented a formal verification that 2PC was able to guarantee commit only if all nodes had reached the commit point and no reason for aborting the protocol was raised. This is what suffices for safety certification.

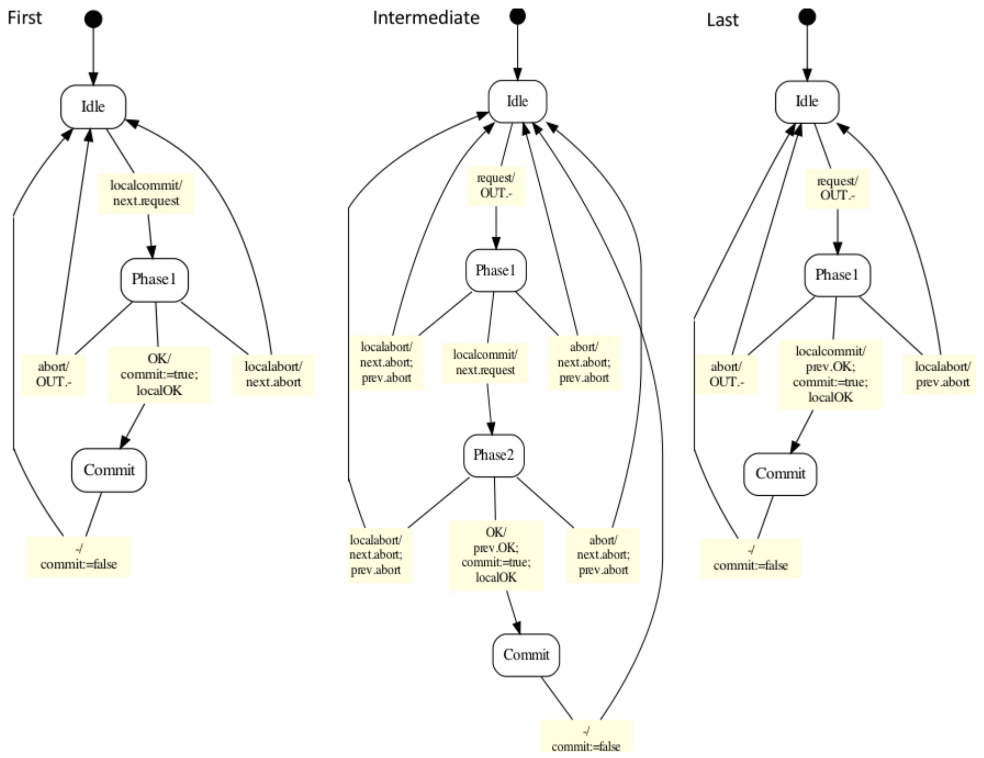
First
Idle
localcommit/ next.request
Phase1
abort/ OUT.-
OK/ commit:=true; localOK
localabort/ next.abort
Commit
-/ commit:=false
Intermediate
Idle
request/ OUT.-
Phase1
localabort/ next.abort; prev.abort
localcommit/ next.request
abort/ next.abort; prev.abort
Phase2
localabort/ next.abort; prev.abort
OK/ prev.OK; commit:=true; localOK
abort/ next.abort; prev.abort
Commit
-/ commit:=false
Last
Idle
request/ OUT.-
Phase1
abort/ OUT.-
localcommit/ prev.OK; commit:=true; localOK
localabort/ prev.abort
Commit
-/ commit:=false

Fig. 2. The Linear 2PC protocol: behaviours of the participating nodes

In order to discuss how a compositional formal verification of safety can be conducted, we show in Figure 2 a simplified formalisation of the nodes of the Linear 2PC protocol, by means of UML Statecharts, representing respectively the *First* node, any *Intermediate* node and the *Last* node of the linear sequence; the Statecharts have been drawn by the UMC tool [5]. The charts show that any Intermediate node goes in state *Phase1* when it receives a *request message* from the previous node, and propagates the message if the node is locally ready to commit. In state *Phase 2*, it waits for the *OK* message from the next node in order to reach the *Commit* state. The node rolls back to the initial state in case of a local abort decision or an abort message from an adjacent node. The *First* and *Last* nodes act similarly upon loosing the communication with the previous or the next node, respectively. The input from the physical environment of the node is abstracted by the incoming *localcommit* and *localabort* actions; the latter abstract communication timeouts as well. Moreover, to keep them simple the shown charts do not model the *release* feature, just exhibiting an unconditional return to the initial state after the *Commit* state. UMC allows any number of *Intermediate* objects to be instantiated, connected in a linear list by means of the *prev* and *next* variables; UMC provides the capability to perform model checking on the modelled network of nodes.

The safety property we are interested to prove can be expressed as: the *First* node reaches the *Commit* state only if all the nodes have locally committed. This can be directly proved on a model consisting of $n+2$ nodes (*First, Last, n Intermediate* nodes), but when n is already in the order of ten, the state space explosion problem makes the verification time too long to be practical.

We can however decompose this proof noting that it is actually enough to prove that each node can reach its own *Commit* state only if the next one has reached the *Commit* state. This amounts to discharge the following proof obligations:

– locally prove, for each type of node, that reaching the *Commit* states is always preceded by the local commit and by (for the *First* and the *Intermediate* nodes) the reception of the *OK* message from the next node;
– locally prove, for the *Last* and *Intermediate* nodes, that sending the *OK* message to the previous node is always preceded by the local commit;
– prove that the communication means does not forge fake *OK* messages (a received *OK* message has always been sent by the next node).

The first two items above can be easily proved locally for each node. Actually the authors have proved them by model checking for the Statecharts shown in Figure 2 by means of UMC: the property to be proved has been expressed as a CTL universally quantified "precedes" formula – e.g. the first property above for the *Last* node is: `not E [ not (localcommit) U Commit]`. The last item above is actually a security assumption over the communication between nodes.

A similar principle can be used to prove safety of the release features when included in the model, that is, to prove that reserved nodes cannot be released before they have been visited.

While safety is easily assured by employing 2PC, proving liveness and fairness would need to take into account several factors we do not address here, such as synchronous or asynchronous of communication, communications faults, ordering of messages, modelling of timeouts, distinguishing successive requests to the same pool, etc.

5 Distributed Mutual Exclusion Variants

The topology of the pool of partners engaged in the 2PC protocol can change at every new invocation of the protocol, since the requesting process may differ, and it might request to lock a different pool of nodes. Different distributed mutual exclusion algorithms can be envisioned, with variants related to the topology of the pool, to where the data defining the pool's topology reside, and how such data rules the communication between nodes. For example, when applied to mutual exclusion of a pool of nodes, the Linear 2PC protocol assumes the knowledge of the linear sequence of nodes of the pool: in particular the formalisation of Linear 2PC provided above assumes that each node can send/receive messages to/from the next and previous elements of the pool. But the list of nodes could also be passed along with the *request* message from the requesting process.

We identify three main variants:

- **Variant 1)** The Classical 2PC algorithm is adopted: the requesting process knows the set S of nodes in the pool and is able to broadcast the request to all the nodes in the pool. The nodes are able to reply to the requesting process.
- **Variant 2A)** The Linear 2PC algorithm is adopted: the pool of nodes S has a linear structure, that is, is composed by a list of nodes. The communication between nodes follows the order of the list. The requesting process knows the list S and sends its request with the list S to the first element of S, each element takes the next and the previous element from the list S and propagates the request, with the list S, according to 2PC: the OK messages are propagated backwards from the last node to the first, by using the knowledge of the previous element for each node. In the case abort messages are generated, they are propagated back and forth in a similar way.
- **Variant 2B)** The Linear 2PC algorithm is adopted, as in Variant 2A: the pool of nodes S has a linear structure, and communication between nodes follows the order of the list. Each node has the knowledge of the previous and next elements for any pool $S \in F_S$ to which it is participating, that is, it knows the adjacent nodes in the pool's topology for each pool to which it belongs. The requesting process sends its request with the requested pool identifier S to the first element of S, and propagates the request, with the S pool id, according to 2PC. Adjacency may be related to physical adjacency or connection between the physical elements controlled by the nodes. Routing mechanisms common to communication networking may be used in each node to determine the next node to which propagate the request, and hence this variant may include limited local rerouting features for availability.

Another source of variability is that actual interlocking algorithms for Cyber-Physical Systems might require two rounds, each employing a 2PC protocol to complete the procedure. In the first round the pool of nodes is locked. In the second round commands are issued to physical objects associated to nodes to move to the desired state, and the acknowledge messages include the check that the physical nodes have actually reached the desired state. Only then the process can start the visit. This behaviour can be needed, e.g., for energy efficiency, because it avoids useless physical movements in case a reservation is aborted.

6 Distributed interlocking as Distributed Mutual Exclusion

Railway interlocking systems are those systems that are responsible to grant to a train the exclusive access to a *route*: a route is a sequence of track elements that are exclusively assigned for the movement of a train through a station or a network. Actually, railway interlocking systems are the most complex (in term of topological size and structure) instances of the safety interlocking concept defined above.

Granting to a train the exclusive access to a route typically means i) checking that the route is free from other trains, by means of track circuits or other presence sensors, ii) commanding points in their correct position, iii) checking that the points have actually reached the commanded position, and iv) setting the signals so to give the driver the permission to move. The instantiation of these generic rules on a station topology (made of the track layout and the set of routes) is usually defined in a data structure named *control table*, that is specific for the station where the system resides. The control table drives the subsequent development of a centralised interlocking system. In the usual meaning of railway interlocking, we intend therefore a system that simply receives requests of reservations, and grants reservations or not on the ground of safety rules, until the reservation has been fully used (the track is again free) or has been safely revoked. It is not a burden of the interlocking to look for alternative routes in case the requested one is busy, in order to optimise traffic throughput parameters, nor to guarantee that a train does not enter a not reserved track. These two functions are traditionally in charge of separate systems, namely Automatic Train Supervision (ATS) and Automatic Train Protection (ATP) respectively.

Centralised interlockings are complex and costly to design and especially to be certified against safety guidelines. The complexity is due to the need of verifying every possible conflicting combinations of different routes through the station: adopting model checking to verify the interlocking logic of large stations has indeed proved challenging [11, 28].

The distribution of the interlocking logic over a network of computing nodes, according to the spirit of cyber-physical systems, has also the side effect of partitioning the verification effort. According to what was said in Sect. 4, we can think to split the safety certification into simpler and repetitive (hence factorised) proofs that each node verifies the safety requirements, plus a security proof for the employed 2PC protocol. The idea of distributed interlocking has been proposed in several papers [2, 8, 15], where advantages and possible drawbacks of such a solution are discussed: in practice, preference is still given to centralised solutions, but this may change with the general trend to distribute intelligence.

In a distributed solution, track elements are directly controlled by a set of distributed communicating nodes: each node controls a given layout element. However, a route is still a global notion: a route has to be established by proper cooperation between the distributed elements. The communication among nodes follows the physical topology of the station/yard and a route is established by the status of the elements that lie along the route.

The following correspondence can be established to consider a distributed railway interlocking as an instance of the general distributed safety interlocking concept:

- Track circuit, point $\rightarrow$ Node.
- Route $\rightarrow$ Pool of nodes.
- Trains $\rightarrow$ Processes.
- A route is requested by a train $\rightarrow$ A process sends a request for locking a pool - including reserving track circuits and locking points in a specific position.
- A route is reserved for a train $\rightarrow$ Requested pool is locked - if track elements are free and points are positioned.
- A train occupies a track circuit or a point $\rightarrow$ Visit of a node.
- A train leaves a track circuit or a point $\rightarrow$ Release of a node.

A specific characteristic of railway interlocking is that nodes of a route are visited by the movement of the train along the route, hence are visited in a sequential predetermined way. As soon as a track circuit or a point is left by a train, it is available for possibly setting another route: this feature is called *sequential release*, a common feature not needed for safety (a route could also be collectively released when the visit of the last node has ended), but desired to improve availability. Another specific characteristic is that cancellation of an already reserved route may be asked (for example when a train is not able to leave a station due to a mechanical problem). Safe cancellation can be achieved in a similar way to safe reservation.

Some proposed distributed railway interlocking algorithms are discussed in the following and use instances of the Distributed Mutual Exclusion variants shown in Sect.5:

- Variant 1) [15, 12]. The engineering concept was originally developed by INSY GmbH Berlin for their railway control system RELIS 2000 designed for local railway networks. In this solution, the train has an onboard computer with route information. Instead of signals, the computer gives Movement Authorities to the driver. The train broadcasts the request of a route to *distributed switch boxes* that control the track elements. This is actually a special case of Variant 1, since it does not require the locking of the complete route, before the train is allowed to move (*sequential locking*): it is as if the train route is divided into sub-routes, each just containing one track segment, and that the train then sequentially locks these small routes. The protocol implicitly includes sequential release. In [15] the concept has been formalised in the RAISE Specification Language, RSL [27], and the RAISE theorem prover was used for verification. In [12] an extension of RSL, called RSL-SAL [23] was used for the formalisation, and the formal verification was performed using the SAL symbolic model checker.

- Variant 1) US patent 8820685 B2 [22]. A controller onboard the train first identifies a group of resources permitting the vehicle to continue its mission, by querying a local database (which contains the data of the whole railway network) with the mission received from a regulating center. Although details of the communication protocol are not given, the onboard controller broadcast the locking request to the identified group of resources, and gives the consensus to move only when all the resources are locked in the desired state. Sequential release is considered as well.
- Variant 1) US patent 20120323411 A1 [18]. The concept is not much different from that of patent [22], with the added complexity that the reservation of a route is negotiated first with other trains as well, and the state of the wayside elements is also recorded at a central location as a back-up. Also in this case, details of the protocol are not given, but in reference to our scheme, the distributed protocol concerns the other processes as well, and the central location can be considered as a further node. This patent also includes higher level negotiation mechanisms on board trains to improve availability.
- Variant 2A) [10]. In this proposal, the linear 2PC is adopted. The information about the route to be reserved (that is, the list of nodes) is propagated to the nodes, from the first to the last node of the route: each node knows from this list its adjacent nodes in the route, with which it directly communicates. The concept has been modelled by UML Statecharts, using UMC for formal verification of safety properties.
- Variant 2B) [7]. Again, this proposal adopts linear 2PC. Each node is initialised with a table containing, for each route traversing the node, the adjacent elements with which it has to communicate. Only the route identifier is propagated along the locking request. The concept has been modelled by UML Statecharts, using UMC for formal verification of safety properties.
- Variant 2B) [4]. This paper formalises in SPIN an interlocking system, considered at the level of sections between stations of a metro line: the proposed interlocking model is shown, by model checking, to guarantee that two trains cannot enter the same section. Due to the linear topology of the line, the model is a direct instance of Variant 2B, and does not include the aborting possibility.

A few other attempts at distributing the interlocking logic in separate computations have been developed, starting from the so-called geographic approach [3, 6, 1], which encodes the interlocking logic in separate objects that each take care of the control of a physical element (point, track circuit, signal, . . .) by means of predefined composition rules, mimicking the topology of the specific layout, although the obtained control software is still centralised. In particular, [2] proposes to start from a Statechart geographic model that uses shared variables as a communication means between objects, and to allocate each object on a distributed node. The adoption of standard distributed consistency protocols guarantees that the exchange of information is the same of the full centralised model. However, this approach requires the safety proof of the centralised model, with no attempt to decompose it into simpler proofs. Similarly, in [16, 17] an overall Petri Net model of a distributed interlocking system is proposed, by connecting Petri Nets representing the behaviour of each node. Again, the analysis of the model does not employ any decomposition strategy.

Different criteria could be used in practice to choose among the variants; these include for example:

– replicating the network database onboard all trains running in a network can be practically done for a closed network, such as a metro network. Instead, in an open infrastructure, such that envisioned by European interoperability that foresee a train crossing many borders between national network, the size of the database and frequency of its updates would be very high: since these data are critical for safety, trains running with a previous release of the database (maybe due to poor communication) would become dangerous. It seems more reasonable that missions received by a train include a list of identifiers of routes to be followed in each traversed station, to be asked to a local, either distributed or centralised, interlocking system.
– on the other end, keeping route tables on the distributed trackside elements requires robust distributed initialisation, configuration and reconfiguration algorithms to maintain consistency [8].
– resilience to faults of single elements - in view of higher availability, which is one of the advantages of distribution - may require redundancy, replication of data and specific policies that could be favoured by one of the variants.
– another criterion pertains to energy efficiency and reliability of track machinery: if points are soon moved in an attempt to set a route that will fail due to conflicting requests, this may result in a lower reliability and energy waste.

The proposals according to Variant 1 show that moving the network map onboard the train may favour the moving of route decision on board as well: routes are currently predetermined in terms of a pool of elements, and allocated to trains in a centralised way (e.g. by an Automatic Train Supervision (ATS) system). Instead, routes could be dynamically generated in front of the train, allowing for last minute choice according to optimisation strategies computed on board. The push towards a *train-centric vs. infrastructure-centric* decision making is one of the challenges considered in the Multiannual Programme of the Shift2Rail Joint Undertaking Initiative [25].

7 Certification

The certification of safety of a distributed interlocking system, according to what was discussed in Sect. 4, amounts to verify that each component locally complies with the standard communication protocol, plus the verification that the protocol does not forge messages. This makes the basis for a simpler and less expensive certification process. First, the safety distributed protocol should be formally verified once for all – this includes proper security measures against attacks. Assembling off-the-shelf plug-in controller elements, manufactured by different vendors, on top of this safety layer will automatically guarantee overall safety, if they are certified to comply with the standard interlocking protocol.

As we have seen, the verification of the safe behaviour of a node can be cheaply done by automated formal verification. One element that we have ignored so far is that the proofs envisaged in Sect. 4 assume the local knowledge of the previous and next element of the pool. The different variants have different views on how these data are available to the nodes: routing tables may be injected in the node at configuration (or reconfiguration) time, or routing information may arrive together with the locking request. Assuring that the data is always consistent with the physical track layout in each node becomes indeed the major certification effort. The possible application of static analysis techniques, such as those described in [14], is a promising research direction at this regard.

8 Conclusions

In this paper we have shown that *safety interlocking* can be seen as a particular class of *Distributed Mutual Exclusion* problems and consequently distributed algorithms solving this problem can be used for safety interlocking. We presented variants of such distributed algorithms and exemplified them with references to different distributed railway interlocking algorithms proposed in the literature. Finally, we discussed the steps needed to convert the proposed solutions into generic plug-and-play safety-certified solutions. Regarding the possible applications in the railway field, we believe that the achieved gains in the certification effort can significantly decrease costs in the production and deployment of interlocking systems, once a standard communication protocol is emerging: variants presented in this paper aims to be a first step in this direction.

Distributed safety interlocking systems may find application in any domain where safety depends on the guarantee that a set of objects is in a determined state. To our knowledge, however, the only example found in the literature is the one reported (with no details about the adopted algorithms) in [29], aimed to guarantee a safe access to a large physics experiment installation.

We have on purpose focused only on safety, mostly ignoring availability: the proposed protocols do not guarantee liveness and fairness under several conditions, and an accurate analysis of different factors (timing, fault models,. . .) affecting these attributes would be needed. In the case of railway interlocking, low availability can severely impact service performability. Given that safety is granted by principles like those put forward by this paper, distributed solutions can be adopted in practice only if sufficient availability is demonstrated, possibly employing quantitative analysis techniques, as suggested in [8].

References

1. FP7 Project INESS - Deliverable D.1.5 report on translation of requirements from text to UML. Tech. rep. (2009)
2. Banci, M., Fantechi, A., Gnesi, S.: The role of formal methods in developing a distribuited railway interlocking system. In: Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems, FORMS/FORMAT, Braunschweig, Germany. pp. 79–91 (2004)
3. Banci, M., Fantechi, A.: Geographical versus functional modelling by statecharts of interlocking systems. Electr. Notes Theor. Comput. Sci. **133**, 3–19 (2005). https://doi.org/10.1016/j.entcs.2004.08.055
4. Basagiannis, S., Katsaros, P., Pombortsis, A.: Interlocking control by distributed signal boxes: Design and verification with the SPIN model checker. In: Guo, M., Yang, L.T., Di Martino, B., Zima, H.P., Dongarra, J., Tang, F. (eds.) Parallel and Distributed Processing and Applications, ISPA. LNCS, vol. 4330, pp. 317–328. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
5. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: A state/event-based model-checking approach for the analysis of abstract system properties. Sci. Comput. Program. **76**(2), 119–135 (2011). https://doi.org/10.1016/j.scico.2010.07.002
6. van Dijk, F., Fokkink, W., Kolk, G., van de Ven, P., van Vlijmen, B.: EURIS, a specification method for distributed interlockings. In: Ehrenberger, W.D. (ed.) Computer Safety, Reliability and Security. LNCS, vol. 1516, pp. 296–305. Springer (1998). https://doi.org/10.1007/3-540-49646-7_23
7. Fantechi, A.: Distributing the challenge of model checking interlocking control tables. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, ISOLA. LNCS, vol. 7610, pp. 276–289. Springer (2012)
8. Fantechi, A., Gnesi, S., Haxthausen, A., van de Pol, J., Roveri, M., Treharne, H.: SaRDIn - A safe reconfigurable distributed interlocking. In: Proc. 11th World Congress on Railway Research, WCRR. Ferrovie dello Stato Italiane, Milano (2016)
9. Fantechi, A., Haxthausen, A.E., Macedo, H.D.: Compositional verification of interlocking systems for large stations. In: Cimatti, A., Sirjani, M. (eds.) International Conference on Software Engineering and Formal Methods, SEFM. LNCS, vol. 10469, pp. 236–252. Springer (2017). https://doi.org/10.1007/978-3-319-66197-1_15
10. Fantechi, A., Haxthausen, A.E., Nielsen, M.B.R.: Model checking geographically distributed interlocking systems using UMC. In: 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing, PDP. pp. 278–286 (2017). https://doi.org/10.1109/PDP.2017.66
11. Ferrari, A., Magnani, G., Grasso, D., Fantechi, A.: Model checking interlocking control tables. In: Schnieder, E., Tarnai, G. (eds.) Proc. Formal Methods for Automation and Safety in Railway and Automotive Systems, FORMS/FORMAT. pp. 107–115. Springer (2010)
12. Geisler, S., Haxthausen, A.E.: Stepwise Development and Model Checking of a Distributed Interlocking System - using RAISE. In: Peleska, J., Roscoe, B., Havelund,

K. (eds.) International Symposium on Formal Methods, FM. LNCS, vol. 10951. Springer (2018)

13. Gray, J.: Notes on data base operating systems. In: Operating Systems, An Advanced Course. LNCS, vol. 60, pp. 393–481. Springer-Verlag, London, UK (1978), http://dl.acm.org/citation.cfm?id=647433.723863
14. Haxthausen, A.E., Østergaard, P.H.: On the use of static checking in the verification of interlocking systems. In: Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications, Part II. LNCS, vol. 9953, pp. 266–278. Springer International Publishing AG (2016)
15. Haxthausen, A.E., Peleska, J.: Formal development and verification of a distributed railway control system. IEEE Trans. Softw. Eng. **26**(8), 687–701 (2000)
16. Hei, X., Takahashi, S., Nakamura, H.: Distributed interlocking system and its safety verification. In: Proc. of 6th World Congress on Intelligent Control and Automation. vol. 2, pp. 8612–8615. Dalian, China (2006). https://doi.org/10.1109/WCICA.2006.1713661
17. Hei, X., Ma, W., Gao, J., Xie, G.: A concurrent scheduling model of distributed train control system. In: Proc. IEEE Inter. Conf. on Service Operations, Logistics, and Informatics, SOLI. pp. 478–483 (2011)
18. Kanner, F.W.A.: Control of automatic guided vehicles without wayside interlocking, Patent US 20120323411 A1 (2012)
19. Lamport, L.: The implementation of reliable distributed multiprocess systems. Computer Networks **2**, 95–114 (1978). https://doi.org/10.1016/0376-5075(78)90045-4
20. Lampson, B., Sturgis, H.: Crash recovery in a distributed storage system. Tech. rep., Comput. Sci. Lab., Xerox Parc, Palo Alto, CA (1976)
21. Maekawa, M.: A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. ACM Trans. Comput. Syst. **3**(2), 145–159 (May 1985). https://doi.org/10.1145/214438.214445
22. Michaut, P.: Method for managing the circulation of vehicles on a railway network and related system, Patent US 8820685 B2 (2014)
23. Perna, J.I., George, C.: Model Checking RAISE Applicative Specifications. In: Proceedings of the Fifth IEEE International Conference on Software Engineering and Formal Methods, SEFM. pp. 257–268. IEEE Computer Society Press (2007)
24. Ricart, G., Agrawala, A.K.: An optimal algorithm for mutual exclusion in computer networks. Commun. ACM **24**(1), 9–17 (1981). https://doi.org/10.1145/358527.358537
25. Shift2Rail Joint Undertaking: Multi-annual action plan (November 2015), http://ec.europa.eu/research/participants/data/ref/h2020/other/wp/jtis/h2020-maap-shift2rail_en.pdf
26. Skeen, D., Stonebraker, M.: A formal model of crash recovery in a distributed systems. IEEE Trans. Softw. Eng. pp. 219–228 (1983)
27. The RAISE Language Group: C. George, P. Haff, K. Havelund, A. E. Haxthausen, R. Milne, C. B. Nielsen, S. Prehn, K. R. Wagner: The RAISE Specification Language. The BCS Practitioners Series, Prentice Hall Int. (1992)
28. Vu, L.H., Haxthausen, A.E., Peleska, J.: Formal modeling and verification of interlocking systems featuring sequential release. Science of Computer Programming (2016). https://doi.org/10.1016/j.scico.2016.05.010
29. Walz, H.V., Agostini, R.C., Barker, L., Cherkassky, R., Constant, T., Matheson, R.: Distributed supervisory protection interlock system SLC acceleration. In: Proceedings of the IEEE Particle Accelerator Conference: Accelerator Science and Technology. vol. 3, pp. 1928–1930 (1989). https://doi.org/10.1109/PAC.1989.72972