Interlocking control by Distributed Signal Boxes: design and verification with the SPIN model checker

Stylianos Basagiannis[1] Panagiotis Katsaros[1] Andrew Pombortsis[1]

[1] *Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece*
{basags,katsaros,apombo}@csd.auth.gr

Abstract. Control systems are required to comply with certain safety and liveness correctness properties. In most cases, such systems have an intrinsic degree of complexity and it is not easy to formally analyze them, due to the resulting large state space. Also, exhaustive simulation and testing can easily miss system errors, whether they are life-critical or not. In this work, we introduce an interlocking control approach that is based on the use of the so-called *Distributed Signal Boxes (DSBs)*. The proposed control design is applied to a railway-interlocking problem and more precisely, to the Athens underground metro system. Signal boxes correspond to the network's interlocking points and communicate only with their neighbor signal boxes. Communication takes place by the use of rendezvous communication channels. This design results in a simple interlocking control approach that compared to other centralized solutions produces a smaller and easier to analyze state space. Formal analysis and verification is performed with the SPIN model checker.

KEYWORDS: interlocking control, safety, distributed control, model checking

1 Introduction

Interlocking control aims to prevent certain operations from occurring, unless preceded by certain events. Although interlocking control may be used as a general signaling technique in the design of e.g. telecom network management systems, the term usually refers to a range of vehicular traffic control applications. Interlocking systems have been mainly developed and studied in the field of railway traffic control, where their task is to prevent trains from colliding and derailing, while at the same time allowing their movements.

Whether interlocking systems are integrated in life-critical control systems or not, they are required to comply with certain safety and liveness correctness properties. This fact elevates formal modeling and model checking to the number one concern in the design and development of real-scale interlocking systems.

However, in most cases, these systems have an intrinsic degree of complexity and it is not easy to fully analyze them, due to the resulting large state space. Usually, the control logic of the interlocking is not the single design concern that has to be checked with respect to the required safety and liveness properties. Complete system designs

have to include also an adequate representation of the communication between the system's components, as well as, the applied fault tolerance approach.

This work introduces the control logic of a new interlocking approach that is based on the use of the so-called *Distributed Signal Boxes* (DSBs). The proposed interlocking control is applied to a real-scale system model and more precisely to the recently build Athens underground metro network. Signal boxes correspond to the network's interlocking points (stations) and communicate only with their neighbors. Communication takes place by the use of rendezvous communication channels. The proposed interlocking control invests on design simplicity and avoids proprietary concept definitions and proprietary system requirements. It is not difficult to be generalized in networks with arbitrary topologies and compared to other centralized solutions produces a smaller and easier to analyze state space, as well as, improved scalability prospects. Formal analysis and verification is performed with the SPIN model checker.

Section 2 surveys recent research in interlocking control and attempts a comparison with the proposed solution. Section 3 introduces our interlocking approach and the use of the so-called *Distributed Signal Boxes* (DSBs). Section 4 refers to the formal verification of the proposed interlocking control. The paper ends with a discussion on our work's potential impact and comments interesting future research prospects.

2 Related work

In related work, interlocking control is mainly studied in the context of railway signaling systems. The work published in [1] points out the lack of precise concept definitions and the lack of overall system requirements. The author proposes an approach to formalize the principles and the concepts of interlocking systems in VDM. However, he focuses on the Danish interlocking systems and underlines that interlocking systems from other countries may be different.

In [2], the authors analyze the safety of a real computer interlocking system, for the control of railway stations. The system's architecture is based on redundancy and is composed of a central nucleus connected to peripheral posts for the control of physical devices. A formal model of the system's safety logic was developed in Verus ([3]), a tool that combines symbolic model checking and quantitative timing analysis. The model was checked with respect to a number of safety and liveness properties that were included in the initial system's specifications. The safety logic of the same system was also modeled in [4] and [5], where the authors used the SPIN model checker ([6]) to analyze all system's functions that may be requested by an external operator.

SPIN was also used in [7] where the authors present a model of the same system and validate certain safety properties, in the presence of Byzantine system components or of some hardware temporary faults.

In [8], the authors introduce a model for the interlocking of a particular track layout that is the one used by an Australian railway operator. Interlocking control is coded in the so-called control tables and the described analysis aims to find erroneous or in-

complete entries in the used tables. Modeling and safety checking is performed with the NuSMV model checker, but in earlier works the same group used also a Communicating Sequential Processes (CSP) approach and the Failure Divergence Refinement (FDR) model checker.

The work published in [9] reports the safety checking of the Line Block interlocking system that also adopts a centralized design approach. The overall control strategy runs on a Central Control Unit that communicates with a number of Peripheral Control Units (PCUs). PCUs are expected to drive particular interlocking system components and detect external events.

There is only one attempt known to us, for the development and verification of a distributed interlocking system. In that work ([10]), the authors note that today's centralized interlocking systems are far too expensive for small or possibly private networks. They propose to distribute the tasks of train control, train protection and interlocking over a network of cooperating components, using the standard communication facilities offered by mobile telephone providers. Their approach is based on the use of the so-called switch boxes, which locally control the point where they are allocated. Train engines are carriers of train control computers, which collect the local state information from switch boxes along the track to derive the decision whether the train may enter the next track segment.

In contrast to the forenamed solution, our approach is based on signal exchanges between the Distributed Signal Boxes (DSBs). There is no need of a mobile communication medium, which in any case requires security and reliability mechanisms that are unnecessary for systems transmitting signals over wires. DSBs communicate only with their neighbors. This design principle results in a general peer-to-peer signaling approach, possible to be applied in a wide range of interlocking problems, other than the typical railway traffic control applications (see for example [11]).

3 Distributed Signal Boxes

DSBs are allocated to the network's interlocking points. Interlocking points communicate only with their corresponding DSB and DSBs communicate only with their neighbor DSBs. Communication takes place by the use of rendezvous communication channels. The proposed interlocking control has been successfully checked in network topologies that include two different types of communication links and more precisely the ones shown in Figure 1.

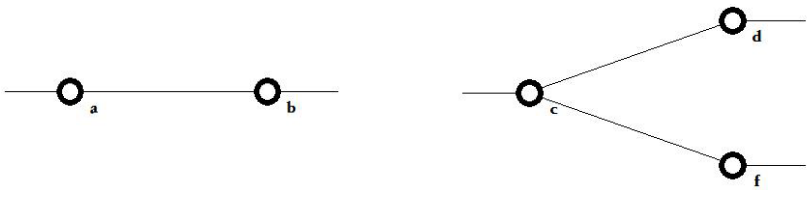

Figure 1. DSBs communication links

In a given network that includes one or both types of communication links, interlocking control requires exchange of two distinct messages, which in fact are used as control flags of the network's traffic. These messages control the network resources and distribute them among the entities that request them. For the track layout of the recently build Athens underground metro (Figure 2) we also call these two messages *signals*. Signals control movement and track allocation by exchanging their locations in a series of consecutive DSBs communications. Interlocking points (stations) cannot communicate with their neighbors by direct communication links, since this is possible only through their corresponding signal boxes.

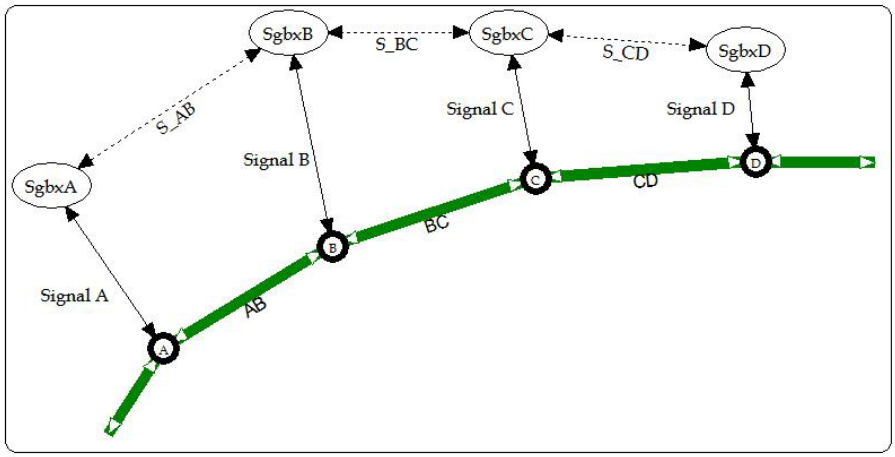

Figure 2. DSB communication for the track layout of the Athens underground metro system

The system's network topology is progressively formed through a step-by-step introduction of new interlocking points (stations), in a ring or a tree-based structure (refer to Figure 1). For every station, we assign the tracks that connect it with its neighbors, from now on called *tunnels*, as well as, the communication channel (e.g. Signal A) used to exchange messages with the corresponding signal box. The topology of the modeled network depends on a mutually consistent declaration of the neighboring stations, for all stations that are included in the network.

The proposed interlocking control mechanism (Figure 3) is coded in only three procedures. One of them refers to the control logic of the network's stations. Another one specifies the control logic of the network's signal boxes and the last one inserts a request for one of the network's tunnels. The last mentioned procedure is used to initialize the model with the required number of trains.

The network perceives an initial train entrance to one of its tunnels by having required the relevant procedure to not proceed to its execution, up to the reception of the expected *msg2* signal. This control signal will be generated by the tunnel's entrance DSB process and as a result it will release the train to enter to the tunnel. The same train will be set again to a stop-wait state as a result of control signals exchanged between the network's DSBs and the station, where the train has arrived. In Figure 3 we

used the SPIN's PROMELA language syntax to show a graphical representation of the described control signal exchanges.

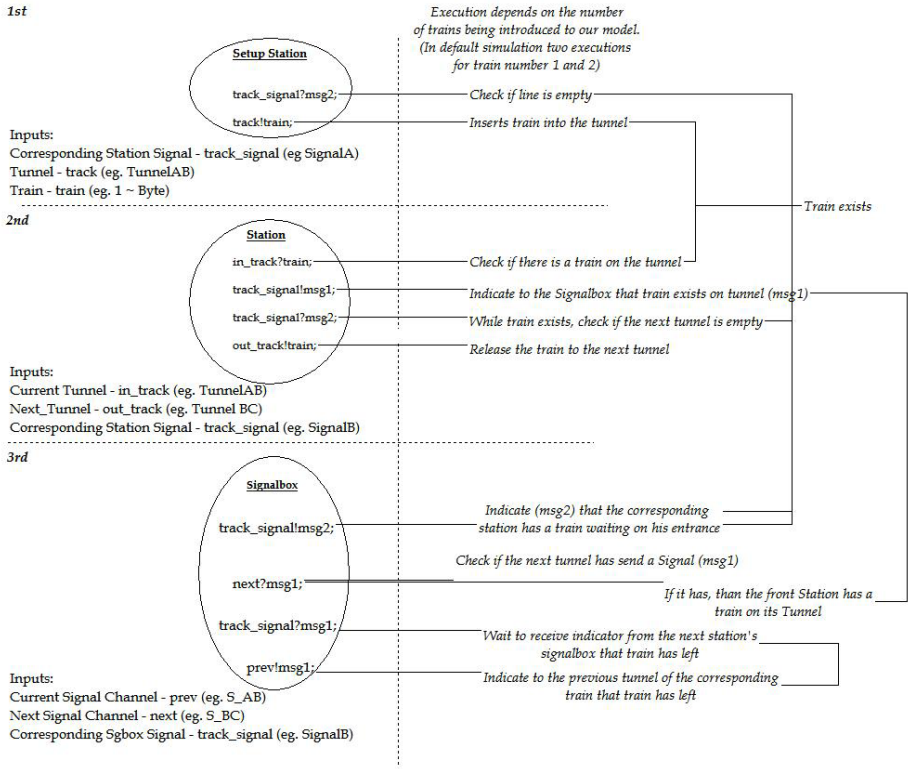

Figure 3. The DSBs based interlocking mechanism

4 Model building and model checking correctness properties

In this section, we present a model building and model checking approach for the described interlocking control mechanism. We aim to prove that the proposed DSBs based solution meets certain safety and liveness properties, which are a requisite for its deployment to real-scale interlocking problems. We decided to use the SPIN model checker ([12], [13]) for the following reasons:

- SPIN has been successfully used in simulating, verifying and finding errors (counter examples) in a wide range of concurrent software systems.
- SPIN is a versatile model checker that provides support to report all detected deadlocks and livelocks, potential race conditions, as well as, possibly unwanted situations regarding the relative speeds of the concurrent processes. It makes possible to express and model check the expected safety and liveness properties as Liner Temporal Logic (LTL) formulae.

- SPIN model specification is expressed in PROMELA, a high-level specification language that provides built-in support for rendezvous, as well as, buffered message passing between the modeled processes.

4.1 Model structure and implementation

In Figure 4 we provide a view of the Athens underground metro network. There are three lines that intersect each other in four different stations. Each train moves within a certain line, but lines operate in different levels and do not interfere with other lines. We focus on modeling the station topology of Line 1 as a set of bi-directional interconnected tunnels. The used tunnels are shared by the trains moving in this line independently of the train movements in Lines 2 and 3.

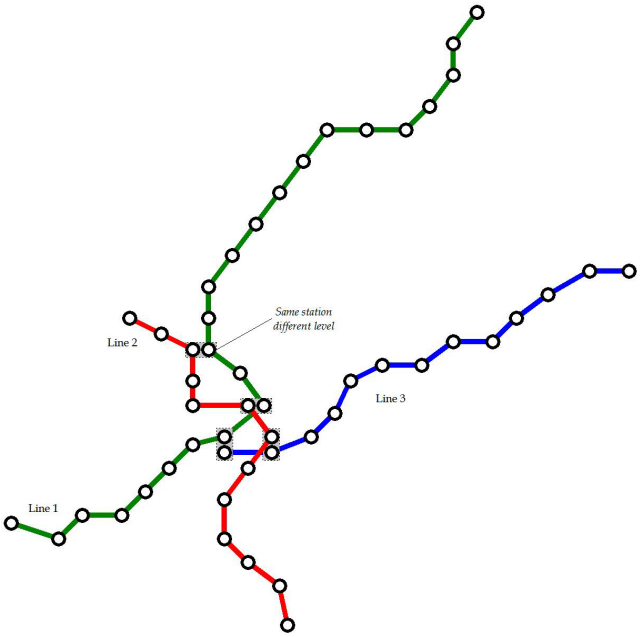

Figure 4. Athens underground metro network

Figure 5 shows the DSBs placement and the derived signal box connectivity for the entire Athens underground metro network. We assume installation of separate interlocking points within terminal stations, where the trains change direction. These additional interlocking points are called *line switching points* and come together with their corresponding signal box.

Line 1 model structure is based on the signal box topology of the stations shown in Line 1. DSBs communicate with their corresponding stations through channels of rendezvous communication that are represented as channels of size zero, in order to not store any signal (`chan Signalx` declarations of Figure 6). When a train arrives to a station this station's DSB communicates with its neighbor DSBs by synchronized

channels of communication (`chan S_xy` declarations of Figure 6), in order to check the availability of the outgoing tunnel. As we already noted, Line 1 (and all other lines) consists of a set of bi-directional tunnels where each direction route is represented by a separate communication channel of size 2 (`chan Tunnelxy` and `chan Tunnelyx` declarations of Figure 6).

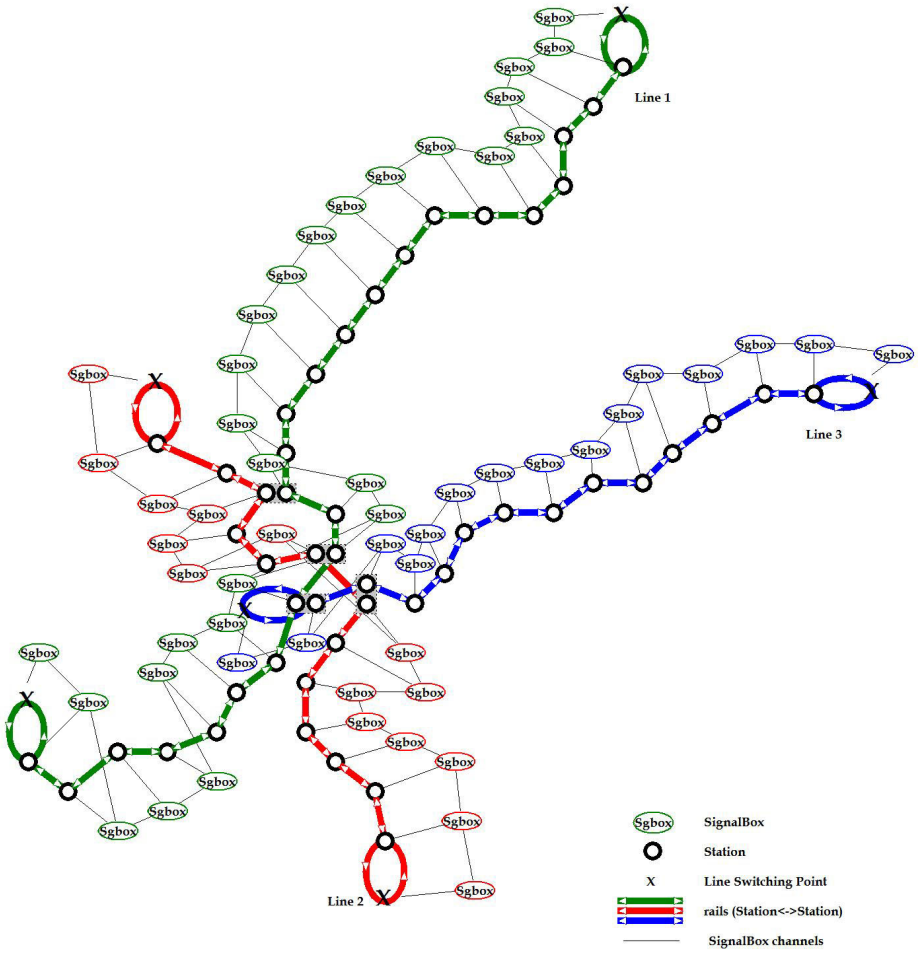

Figure 5. The DSBs based interlocking mechanism for the Athens underground metro network

Tunnels and signal boxes interconnection is specified by the `run Setup`, the `run Station` and the `run Signalbox` procedure calls shown in Figure 8. The code of the forenamed procedures is given in Figure 7.

For a given station, say X, a train arrival to it (`proctype Station`) causes the dispatch of `msg1` to its signal box. X's signal box forwards `msg1` to the signal box of the previous station (`proctype Signalbox`). Regarding X's interlocking control, on reception of `msg2` the train enters into the next tunnel (`proctype Station`).

```
/* Channels between Stations (Tunnels) */

chan TunnelAB=[2] of {byte};
chan TunnelBC=[2] of {byte};
chan TunnelCD=[2] of {byte};
chan TunnelDE=[2] of {byte};
.............................
chan TunnelXSw1=[2] of {byte};
.............................
chan TunnelBA=[2] of {byte};

/* Channels between Stations and Signalboxes */

chan SignalA=[0] of {mtype};
chan SignalB=[0] of {mtype};
chan SignalC=[0] of {mtype};
chan SignalD=[0] of {mtype};
.............................
chan SignalX=[0] of {mtype};

/* Channels between the Signalboxes */

chan S_AB=[0] of {mtype};
chan S_BC=[0] of {mtype};
chan S_CD=[0] of {mtype};
chan S_DE=[0] of {mtype};
.........................
chan S_XSw1=[0] of {mtype};
.........................
chan S_BA=[0] of {mtype};
```

Figure 6. PROMELA declarations of tunnels, DSB to station and DSB to DSB channels

```
proctype Setup(chan track,track_signal;byte train)
{
track_signal?msg2;
track!train;
}

proctype Station(chan in_track,out_track,track_signal)
{
byte train;
do
 :: in_track?train;track_signal!msg1;
    track_signal?msg2;out_track!train;
od
}

proctype Signalbox(chan prev,next,track_signal)
{
do
 :: track_signal!msg2;next?msg1;
 :: track_signal?msg1;prev!msg1;
od
}
```

Figure 7. DSBs interlocking control procedures

```
/* Introduces train 1 into TunnelBC */

run Setup(TunnelBC, SignalB,1);

/* Introduces train 2 into TunnelML */

run Setup(TunnelML, SignalM,2);

/* assertion of the problem */

run Monitor();

/* StationA with track_signal A etc*/

run Station(TunnelSwA,TunnelAB, SignalA);
run Station(TunnelAB,TunnelBC, SignalB);
run Station(TunnelBC,TunnelCD, SignalC);
run Station(TunnelCD,TunnelDE, SignalD);
.....................................
run Station(TunnelNM,TunnelML,SignalM);
run Station(TunnelML,TunnelLK,SignalL);
.....................................

/*Signalbox A, communication channels, track_signal A etc*/

run Signalbox(S_SwA,S_AB,SignalA);
run Signalbox(S_AB,S_BC,SignalB);
run Signalbox(S_BC,S_CD,SignalC);
run Signalbox(S_CD,S_DE,SignalD);
.............................
run Signalbox(S_NM,S_ML,SignalM);
run Signalbox(S_ML,S_LK,SignalL);
.............................
```

Figure 8. Initiating the DSBs interlocking control model

However, `msg2` cannot be received if the signal box is still blocked, waiting for the dispatch of `msg1` from the next station's signal box (`proctype Signalbox`). Accordingly, this depends on the availability of X's outgoing tunnel, that is, it is possible only if another train has already left the next station or only upon the departure of that train from the next station (`proctype Station`).

Figure 8 shows the PROMELA code for a model instance with 2 trains (two `Setup` procedure calls). It is not difficult to verify the expected safety and liveness properties for an arbitrary number of trains. A detailed operational view of the proposed interlocking control for a Line 1 segment is given in Figure 9.

4.2 Model checking safety and liveness correctness properties

The basic safety correctness property refers to the possibility of collision between the two trains in one of the tunnels' routes shown in Figure 9. The developed model makes this possible by having declared all tunnels' routes as separate communication channels of size 2 (Figure 6). The proposed interlocking control aims to prevent the two trains from occupying both channel positions at the same time. Correctness with respect to the forenamed safety property is checked by the following assertion:

ASR1: "*A tunnel route can only be occupied by one train at a time*"

Assertion *ASR1* is included in the code of procedure `Monitor` (Figure 10). Model checking is activated by the `run Monitor()` procedure call of Figure 8. In the performed full state space search of Figure 11 an assertion violation would be reported as error, but the obtained results prove the safety correctness of the proposed interlocking control (`errors: 0`). The reported number of states refers to a real-scale application of our solution to a network of 26 interlocking points, 2 directions and 2 trains moving in both directions of the network's interconnected tunnels.

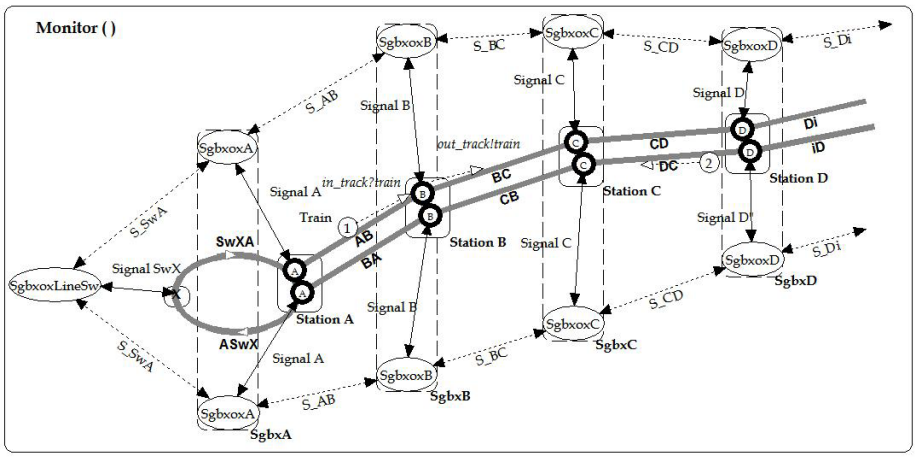

Figure 9. Operational view of DSBs based interlocking control applied to a line segment

```
proctype Monitor()
{
do
 :: assert(nfull(TunnelSwA)&&nfull(TunnelAB)&&nfull(TunnelBC)
&&nfull(TunnelCD)&&nfull(TunnelDE)&&nfull(TunnelEF)
&&nfull(TunnelFG)&&nfull(TunnelGH)&&nfull(TunnelHI)
&&nfull(TunnelIJ)&&nfull(TunnelJK)&&nfull(TunnelKL)
&&nfull(TunnelLM)&&nfull(TunnelMN)&&nfull(TunnelNO)
&&nfull(TunnelOP)&&nfull(TunnelPQ)&&nfull(TunnelQR)
&&nfull(TunnelRS)&&nfull(TunnelST)&&nfull(TunnelTU)
&&nfull(TunnelUV)&&nfull(TunnelVW)&&nfull(TunnelWX)
&&nfull(TunnelXSw1)&&nfull(TunnelSw1X)&&nfull(TunnelXW)
&&nfull(TunnelWV)&&nfull(TunnelVU)&&nfull(TunnelUT)
&&nfull(TunnelTS)&&nfull(TunnelSR)&&nfull(TunnelRQ)
&&nfull(TunnelQP)&&nfull(TunnelPO)&&nfull(TunnelON)
&&nfull(TunnelNM)&&nfull(TunnelML)&&nfull(TunnelLK)
&&nfull(TunnelKJ)&&nfull(TunnelJI)&&nfull(TunnelIH)
&&nfull(TunnelHG)&&nfull(TunnelGF)&&nfull(TunnelFE)
&&nfull(TunnelED)&&nfull(TunnelDC)&&nfull(TunnelCB)
&&nfull(TunnelBA)&&nfull(TunnelASw))
od
}
```

Figure 10. Safety assertion ASR1

```
Full statespace search for:
never claim         - (not selected)
assertion violations+
cycle checks        - (disabled by -DSAFETY)
invalid end states- (disabled by -E flag)

State-vector 1588 byte, depth reached 2189, errors: 0
  141241 states, stored
  280500 states, matched
  421741 transitions (= stored+matched)
     102 atomic steps
hash conflicts: 20022 (resolved)

Stats on memory usage (in Megabytes):
225.421 equivalent memory usage for states (stored*(State-vector + overhead))
132.292 actual memory usage for states (compression: 58.69%)
State-vector as stored = 929 byte + 8 byte overhead
2.097 memory used for hash table (-w19)
0.320 memory used for DFS stack (-m10000)
0.605 memory lost to fragmentation
134.104 total actual memory usage
```

Figure 11. Full state space results for assertion ASR1

In Line 1 of the Athens underground metro network, due to its track layout we expect that each train loops through the line's interlocking points (stations) and there are no unreachable stations. To prove this expectation we verify that in an infinite run the train eventually passes through declared tunnel routes (and corresponding stations). This requirement is specified by the following liveness property:

LVN1: "*For any tunnel route, as soon as a train loops in its line, this train will eventually pass through it*"

We express the forenamed property by the following LTL (Linear Temporal Logic) formula

```
[] (<> p) -> (<> q) -> (<> p)
```

with the symbol definitions

```
#define p (len(TunnelAB)==0)
#define q (len(TunnelAB)==1)
```

and the used temporal operators defined as follows:

`<>x = TRUE U x`	*eventually*
`[] x = ¬ <> ¬x`	*always*
`->`	*logical implication*

The recurrence formula `[] (<> p)` asserts that in an infinite sequence of states `p` occurs infinitely many times. If so, the train should eventually pass through the

checked tunnel route. SPIN generates the never claim (finite automaton shown in Figure 12) of the LVN1 formula and checks if the expected property holds for all executions. As a result we get that *LVN1* is valid.

```
never {    /* !(( []  (<>  p ) -> (<>  q) -> (<>  p))) */
T0_init:
if
:: (! ((p))) -> goto accept_S4
:: (! ((p))) -> goto accept_S7
fi;
accept_S4:
if
:: (! ((p))) -> goto accept_S4
fi;
accept_S7:
if
:: (! ((p))) -> goto accept_S4
:: (! ((p))) -> goto T0_init
fi;
}
```

Figure 12. Never claim for the *LVN1* formula

5 Conclusion

In this paper we introduced an interlocking control that is based on a peer-to-peer signaling approach, between the so-called Distributed Signal Boxes and the network's interlocking points. Compared to all other centralized solutions, the proposed interlocking control does not depend on proprietary concept definitions that refer to railway interlocking systems, invests on design simplicity, provides improved scalability prospects and avoids the problem of the single point of failure.

Our control design is a general signaling technique that is possible to be applied in a wide range of control problems. We describe its application to the topology of the Athens underground metro network, but we have also verified its validity in networks that include a second type of DSB communication link and more precisely the 1-to-2-split communication link shown in Figure 1. DSBs correspond to the network's interlocking points and communicate only with them, as well as with their neighbor signal boxes. We proved that the proposed interlocking control averts the possibility of collision between two trains in one of the network's tracks and that each train will eventually pass through all tracks (and stations) of the line, where the trains move. Model checking was performed based on the use of the SPIN model checker. We noted that when the proposed control design is applied to a real-scale problem, compared to the published centralized solutions results in a smaller and easier to analyze state space.

As a first priority future research prospect we consider the introduction of a third signal exchanged between the network's interlocking points and DSBs. This will allow us to cover one more case of communication link and more precisely the 2-to-1-join communication link that is not tested so far. We also consider extending the proposed interlocking control, such as to include redundant standby DSBs and duplex

communication links between DSBs, as well as between DSBs and their interlocking points. In this way, we aim to develop a complete fail-safe control design, based on the signaling technique that we described in this paper.

References

[1] Hansen, K. M., Formalizing Railway Interlocking Systems, In Proceedings of the 2nd FMERail Workshop, 1998

[2] Garmhausen, V. H., Campos, S., Cimatti, A., Clarke, E., Giunchiglia, F., Verification of a Safety-Critical Railway Interlocking System with Real time Constraints, Science of Computer Programming, Vol. 36, No. 1, Elsevier North-Holland, 53-64, 2000

[3] Campos, S., Clarke, E., Minea, M., The Verus tool: a quantitative approach to the formal verification of real-time systems, In Proceedings of the Conference on Computer Aided Verification, 1997

[4] Cimatti, A., Giuchiglia, F., Mongardi, G., Romano, D., Torielli, F., Traverso, P., Model Checking Safety Critical Software with SPIN: an Application to a Railway Interlocking System. In Proceedings of the 3rd SPIN workshop, 1997

[5] Cimatti, A., Giunchiglia, F., Mongardi, G., Romano, D., Torielli, F., Traverso, P., Formal Verification of a Railway Interlocking System using Model Checking, Formal Aspect of Computing, Vol.10, No. 4, 361–380, 1998

[6] Holzmann, G. J., The Model Checker SPIN, IEEE Transaction on Software Engineering, Vol.5, No.23, 279–295, 1997

[7] Gnesi, S., Latella, D., Lenzini, G., Abbaneo, C., Amendola, A., Marmo, P., A Formal Specification and Validation of a Critical System in Presence of Byzantine Errors, In Proceedings of TACAS 2000, Lecture Notes in Computer Science 1785, 535–549, 2000

[8] Winter, K., Robinson, N. J., Modelling large railway interlockings and model checking small ones, In Proceedings of the 26th AustralAsian Computer Science Conference in Research and Practice in Information Technology, Adelaide, Australia, 309-316, 2003

[9] Hlavaty, T., Preucil, L., Stepan, P., Klapka, S., Formal methods in development and testing of railway interlocking systems, In Proceedings of the Conference on Intelligent Methods for Quality Improvement in Industrial Practice. Prague: CTU FEE, Department of Cybernetics, The Gerstner Laboratory, Vol. 1, 14-25, 2002

[10] Haxthausen, A. E., Peleska, J., Formal Development and Verification of a Distributed Railway Control System, IEEE Transactions on Software Engineering, Vol. 26, No. 8, 687-701, 2000

[11] Arozarena, P., Frints, M., Collins, S., Fallon, L., Zach, M., Serrat, J., Nielsen, J., Madeira: A peer-to-peer approach to network management, In Proceedings of the Wireless World Research Forum, Shanghai, China, April 2006

[12] The SPIN model checker official website, available at http://spinroot.com/

[13] Holzmann, G. J., Design and Validation of Computer Protocols, Prentice-Hall, 1991.