Formal Verification of Network Interlocking Control by Distributed Signal Boxes

Stylianos Basagiannis[1] and Panagiotis Katsaros[2]

[1] United Technologies Research Center, Cork, Ireland
BasagiS@utrc.utc.com
[2] School of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
katsaros@csd.auth.gr

Abstract. Interlocking control prevents certain operations from occurring, unless preceded by specific events. It is used in traffic network control systems (e.g. railway interlocking control), piping and tunneling control systems and in other applications like for example communication network control. Interlocking systems have to comply with certain safety properties and this fact elevates formal modeling as the most important concern in their design. This paper introduces an interlocking control algorithm based on the use of what we call Distributed Signal Boxes (DSBs). Distributed control eliminates the intrinsic complexity of centralized interlocking control solutions, which are mainly developed in the field of railway traffic control. Our algorithm uses types of network control units, which do not store state information. Control units are combined according to a limited number of patterns that in all cases yield safe network topologies. Verification of safety takes place by model checking a network that includes all possible interconnections between neighbor nodes. Obtained results can be used to generalize correctness by compositional reasoning for networks of arbitrary complexity that are formed according to the verified interconnection cases.

1 Introduction

In the past, interlocking control was mainly developed and studied in the context of railway signaling, where its task is to prevent trains from colliding and derailing, while at the same time allowing their movements. Our view is that interlocking control is a mean for synchronizing exclusive access to distributed network resources (network segments) and its application extends beyond this of railway signaling. Interlocking control is also used in piping and tunneling control systems and may be involved in other applications like for example network management systems [2].

In this work, we introduce a distributed control algorithm with network control units that do not store information related to the algorithm's state. This option eliminates the intrinsic complexity of other solutions that are mainly centralized and the complexity of the few distributed approaches with control units that maintain state. More precisely, as shown in [19], algorithmic verification of interlocking safety properties is an extremely complex task, due to the state space explosion involved. Typically, the internal state of the analyzed system has 2^n possible configurations, where n is the number of components such as interlocking points, signals, etc. with which the system is built. Our contribution is summarized in the following:

– The Distributed Signal Boxes (DSBs) algorithm is verified within the SPIN model checker [14]. Interlocking safety is verified in a network formed by combining all possible interconnections between neighbor nodes. The control units of our algorithm can be composed only in the ways tested in this small network.
– Interlocking logic of the control units is decoupled from the network topology and this eliminates the need to locally store information related to the algorithm's state. Although network routing is not within the scope of our algorithm, we assume non-deterministic routing as an abstract modeling approach for verifying all routing possibilities in a network node [16]. Interlocking safety is provided as a network service, irrespective of the operation control commands.

For more complex networks, one can use the compositional verification technique for synchronous message passing [18] that decomposes the verification problem into correctness properties for smaller networks. Thus, we avoid the risk of interlocking schemes that cannot be fully analyzed, due to their large state space.

A preliminary version of the algorithm was presented in [3], where it was applied to a simple railway-interlocking problem, which did not include all the cases of node interconnections that are covered in present article. Section 2 presents the considered network interconnection cases and their corresponding DSBs connectivity. Section 3 introduces the algorithm and the obtained SPIN model-checking results. Finally, in Section 4 we review the related works and we compare them with the proposed approach. We conclude with comments on the potential impact of the presented work.

2 Network interlocking nodes and distributed signal boxes

The messages for the control of a single node that connects multiple network segments depend on the node's interconnection with neighbor nodes. In this section, we establish the terminology used to describe a general interlocking problem and we introduce the different cases of nodes' interconnection and the corresponding DSBs topologies.

Definition 1. *Nodes define the ends of interconnected network segments, which are distributed network resources. They cannot communicate with each other. A node – depending on the number of interconnected segments – controls access to at least one resource, i.e. access to the resource(s) is only possible through the controlling node.*

Figure 1 illustrates a simple network consisting of interconnected nodes X, A, B, C, D and Y. Node A controls access to resource $R(AB)$, but direct communication with node B is not possible.

Definition 2. *Each node is connected with a DSB and communicates with it through a synchronous signal channel. The $DSBs$ corresponding to a pair of connected nodes communicate with each other. Since nodes cannot directly contact, they manage the controlled resources only by messages to their corresponding $DSBs$.*

In Figure 1, node A and $DSBoxA$ are connected by the signal channel *NodeAtoDSBoxA*. As a consequence of Def. 2, $DSBoxA$ exchanges messages with $DSBoxB$.

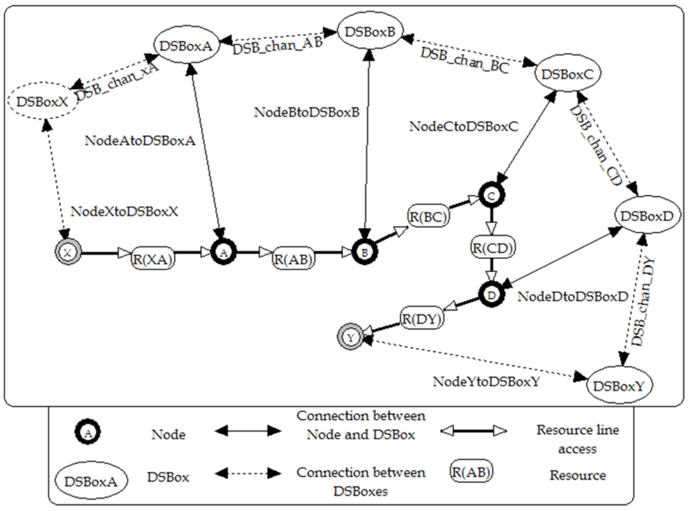


Fig. 1. One-to-one node interconnection and the DSBs topology

Definition 3. *When a moving entity requests access to a network resource, it can be granted only by the controlling node of this resource. All entities that request access move in a given direction.*

Definition 4. *Interlocking control synchronizes requests generated by entities, for exclusive access to the controlled resource(s).*

Nodes are represented by *control processes* that accept as input an entity arrival, exchange messages with their corresponding $DSBs$ and subsequently release the moving entity, when possible, thus grantng access to the requested resource. Entity arrivals trigger a message dispatch to the node's DSB and upon receipt of the reply the entity is released. Thus, control processes do not need to store information for the algorithm's state, since this state is communicated to the network instantly. Within the SPIN model-checker, we assume synchronous communication, which is modeled by rendezvous communication channels [15]. This specification assumption keeps our model computationally tractable, since we avoid asynchronous communication that would increase interleaving between the modeled processes. Applicable implementation alternatives include all modern time-triggered communication options, with safety critical features (e.g. the TTP/C and the FlexRay protocols) that are often used in distributed embedded systems. However, we do not address issues related to implementation details like for example how to guarantee atomic message dispatch, since we are only interested to verify the correctness of our algorithm.

We have identified three types of resource interconnection namely, the *one-to-one link*, the *one-to-many split link* and the *many-to-one join link*. One-to-many and many-to-one links require synchronization between the $DSBs$ of the nodes in the many side. Figure 1 introduced an example with one-to-one resource interconnections, where for some entity that occupies resource $R(XA)$ node A grants access to resource $R(AB)$, if it is not occupied by another entity. The figure shows the implied $DSBs$ topology and represents a part of Athen's underground metro network, which was used in [3] for introducing a first version of our algorithm.

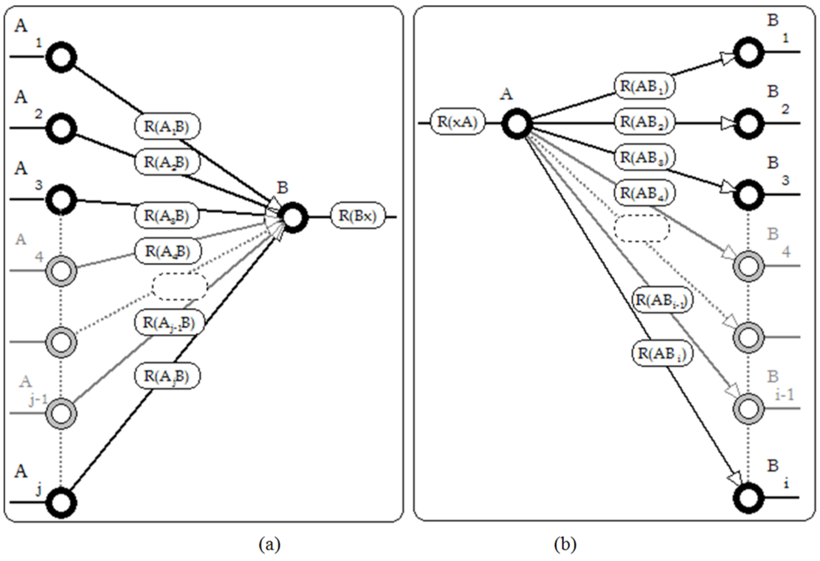


Fig. 2. (a) Many-to-one join node interconnection and (b) one-to-many split node interconnection

Figure 2a presents the many-to-one join node interconnection, where a number of network resources, say $R(A_1B), R(A_2B), ..., R(A_jB)$ is connected through some node B to a single resource shown as $R(Bx)$. Access to the controlled network resource $R(Bx)$ is performed by the synchronous exchange of control messages transmitted between: (i) nodes $A_1, A_2, \ldots, A_j$ and their corresponding $DSBs$, (ii) neighbor $DSBs$ (e.g. the $DSBs$ of nodes A_1 and B), (iii) node B and its corresponding DSB and (iv) for synchronizing the $DSBs$ of nodes $A_1, A_2, \ldots, A_j$ in the many side.

Figure 2b shows the one-to-many split node interconnection, where some network resource, say $R(xA)$ is connected to i network resources denoted by $R(AB_1), R(AB_2), \ldots, R(AB_i)$. We already pointed out that for verifying all routing possibilities for a passing entity, we include all possible entity routing decisions, i.e. a non-deterministic selection of the requested controlled resource (symmetrically, in Figure 2a we assume non-deterministic selection between the entities waiting in the many side). Access to the requested resource is regulated by the synchronous exchange of control messages transmitted between: (i) nodes $B_1, B_2, \ldots, B_i$ and their corresponding $DSBs$, (ii) neighbor $DSBs$ (e.g. the $DSBs$ of nodes A and B_1), (iii) node A and its corresponding DSB and (iv) for synchronizing the $DSBs$ of nodes $B_1, B_2, \ldots, B_i$.

For a two-to-two resource interconnection by a single node, a suitable solution can be developed by decomposing the problem into two distinct two-to-one interconnection cases and by intermixing the algorithm's logic accordingly. This means that a synchronization message non-deterministically selects, which destination resource from the two-to-one interconnection cases will be occupied. A many-to-many resource interconnection is implemented as a complex with a many-to-one join link attached to a one-to-many split link.

3 Network interlocking nodes and distributed signal boxes

Figure 3 shows the $DSBs$ topology for a typical one-to-one resource interconnection, as well as the exchanged messages guaranteeing exclusive access to the controlled resources. Our algorithm encompasses:

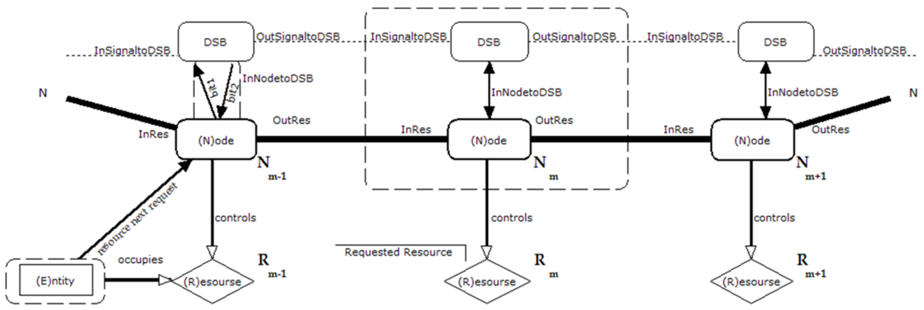


Fig. 3. DSBs and message communications for an one-to-one resource interconnection

- the control processes for the shown interlocking nodes (N),
- the resources (R), with each of them being controlled by some node (N),
- the DSBs control processes, where each DSB corresponds to some node (N),
- the moving entities (E) that request access to the available resources (R) and
- the messages between nodes and $DSBs$, and those exchanged between $DSBs$.

Resource allocation is established by using two message types, namely $bit1$ and $bit2$. These messages do not carry any information; they block or release processes, but none is dedicated to a specific role throughout the whole algorithm logic.

```
1 proctype Node(chan inRes,
2                   outRes,
3                   inNodetoDSB)
4 {
5 byte entity;
6      do
7      :: inRes?entity;
8         inNodetoDSB!bit1;
9         inNodetoDSB?bit2;
10        outRes!entity;
11     od
12 }
```

Fig. 4. The Node control process of the DSBs algorithm

The algorithm is introduced in PROMELA, the input language of SPIN. In Figure 3, let us assume that an entity E occupies R_{m-1} and requests access to R_m, which is controlled by N_m (line 7 of $proctype\ Node$ in Figure 4 is enabled for N_m). N_m then sends

$bit1$ to its corresponding DSB (line 8) through the signal channel $inNodetoDSB$ that synchronizes the two processes. We distinguish two different possibilities:

- Resource R_m is available, which means that the DSB of node N_m has already sent message $bit2$ to its corresponding node (line 18 of $proctype\ DSBox$ in Figure 5 for the node N_m). Message $bit2$ is received by node N_m that subsequently provides access to resource R_m (lines 9, 10 of $proctype\ Node$ for N_m). The DSB of node N_m is then blocked waiting for message $bit1$ in the communication channel $outSignaltoDSB$ that synchronizes it with the DSB of node N_{m+1} (line 19 of $proctype\ DSBox$ for the node N_m).
- Resource R_m is currently occupied by another entity. In this case, there is no message $bit2$ in the $inNodetoDSB$ signal channel for node N_m and this blocks the requesting entity from accessing R_m (line 9 of $proctype\ Node$ for N_m). The DSB of N_m also waits for message $bit1$ (line 19 of $proctype\ DSBox$ for N_m). The expected message will be received when the DSB of node N_{m+1} controlling the requested resource R_{m+1} will send $bit1$ (line 21 of $proctype\ DSBox$ for N_{m+1}) thus indicating that R_{m+1} can be used by the entity that currently occupies R_m. Upon receipt of $bit1$ by the DSB of N_m (line 19 of $proctype\ DSBox$) the message $bit1$ in the signal channel $inNodetoDSB$ is consumed (line 20 of $proctype\ DSBox$) and subsequently a $bit1$ message is dispatched to the DSB of node N_{m-1} (line 21 of $proctype\ DSBox$ for N_m). Then, the DSB of node N_m sends $bit2$ to the signal channel $inNodetoDSB$ that synchronizes it with its corresponding node (line 18 of $proctype\ DSBox$ for N_m) and this message releases the requested resource R_m (lines 9, 10 of $proctype\ Node$ for N_m).

```
13 proctype DSBox(chan inSignaltoDSB,
14                     outSignaltoDSB,
15                     inNodetoDSB)
16 {
17      do
18      :: inNodetoDSB!bit2;
19         outSignaltoDSB?bit1;
20      :: inNodetoDSB?bit1;
21         inSignaltoDSB!bit1;
22      od
23 }
```

Fig. 5. The DSBox control process of the DSBs algorithm

Figure 6 shows the message communications for a network with a two-to-one join node connected to a node one-to-two. The control processes for these two nodes differ from the control process of a simple node (Figure 3) in the use of *one more synchronous message exchange for every pair of adjacent resources in the many side*. This additional message is necessary, in order to synchronize concurrent requests for access to the same resource coming from the many side of the nodes.

The PROMELA code in Figure 7 introduces the control process for the shown two-to-one join node that connects the network resources specified by the $inRes1$ and $inRes2$ parameters to the network resource specified by the $outRes$ parameter. For a single pair of adjacent resources in the many side, we need only one synchronization channel named here $synchronizerA$ and one additional message that we call $bit3i$.

Besides the use of the synchronizer, the algorithm's logic is essentially the same with the logic shown in Figure 4, apart from the fact that we use now two signal channels named $inNode1toDSB$ and $inNode2toDSB$ that synchronize the two-to-one join node with its corresponding DSB. Finally, the process logic for the two-to-one join node addresses the requirement for non-deterministic selection between concurrent requests of controlled resources. Figure 8 provides the process logic for the symmetric case of the one-to-two node shown in Figure 6.

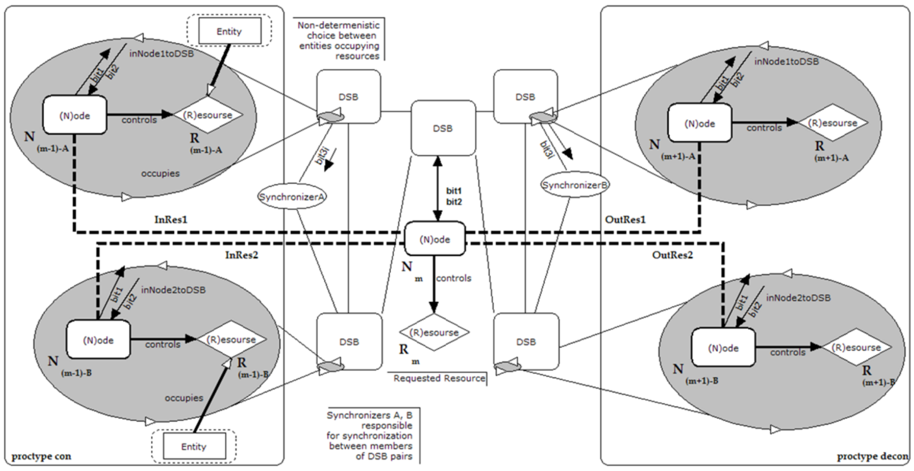


Fig. 6. DSBs and message exchanges for a two-to-one join node connected to a one-to-two node

```
proctype con ( chan inRes1,
                    inRes2,
                    inNode1toDSB,
                    inNode2toDSB,
                    outRes)
{
    byte entity;
    do
    :: inRes1?entity;
       inNode1toDSB!bit1;
       inNode1toDSB?bit2;
       synchronizerA?bit3i;
       outRes!entity;
       synchronizerA!bit3i;
       inRes2?entity;
       inNode2toDSB!bit1;
       inNode2toDSB?bit2;
        outRes!entity;
    :: inRes2?entity;
       inNode2toDSB!bit1;
       inNode2toDSB?bit2;
        synchronizerA?bit3i;
        outRes!entity;
        synchronizerA!bit3i;
        inRes1?entity;
        inNode1toDSB!bit1;
        inNode1toDSB?bit2;
        outRes!entity;
    od
}
```

Fig. 7. Algorithm's logic for two-to-one resource interconnection

```
proctype decon (chan inRes,
                     inNode1toDSB,
                     inNode2toDSB,
                     outRes1,
                     outRes2)
{
    byte entity;
    do
    :: inRes?entity;
       inNode1toDSB!bit1;
       inNode1toDSB?bit2;
       synchronizerB?bit3i;
       outRes1!entity;
       synchronizerB!bit3i;
       inRes?entity;
       inNode2toDSB!bit1;
       inNode2toDSB?bit2;
        outRes2!entity;
    :: inRes?entity;
       inNode2toDSB!bit1;
       inNode2toDSB?bit2;
        synchronizerB?bit3i;
        outRes2!entity;
        synchronizerB!bit3i;
       inRes?entity;
        inNode1toDSB!bit1;
       inNode1toDSB?bit2;
        outRes1!entity;
    od
}
```

Fig. 8. Algorithm's logic for one-to-two resource interconnection

The complete PROMELA code for the small network of Figure 9 includes all types of possible interconnections and at the same time gives us the opportunity to verify safety and to study existing possibilities for deadlock, livelock or other violations of progress. Resource $Rsrc34$ is connected to resource $Rsrc41$. $Node(1)$ represents a one-to-two split link to the resources $Rsrc12$ and $Rsrc13$ and $Node(3)$ is a two-to-one join link to the resource $Rsrc34$.

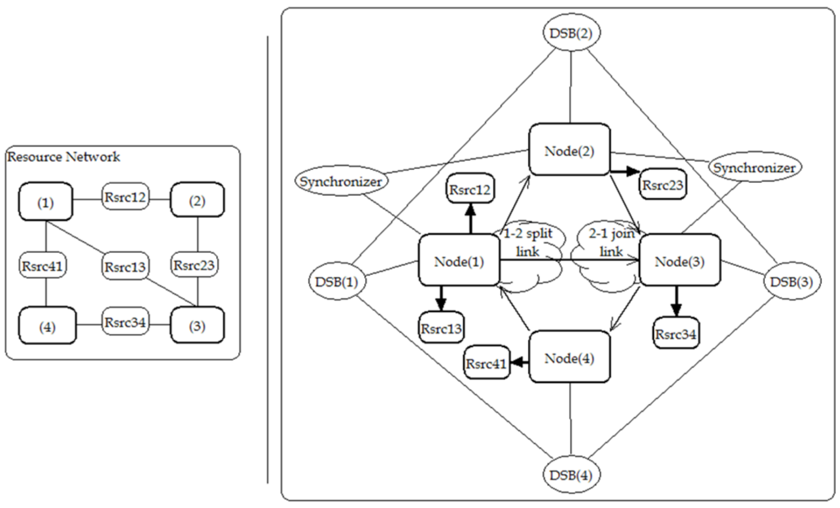


Fig. 9. A network of resources and the associated DSBs topology

3.1 Safety verification

The basic safety property for the DSBs interlocking control is expressed by the monitor assertion of Figure 10, which is used to check that *"in all reachable states, at most one entity occupies any resource"*. We utilize the predefined boolean function

$$nfull(q) = (len(q) < QSZ)$$

for testing that a network resource represented by channel q is occupied by a number of entities $(len(q))$ less than the number QSZ that represents violation of safety.

In order to detect violation of exclusive access to any resource, we set $QSZ = 2$. If it is possible to reach a state where for some resource, say $Rsrc$, holds $len(Rsrc) = QSZ$, then the model checking output reports an error (assertion violation).

For the network shown in Figure 9, the initial conditions guaranteeing constant protection involve the notion of what we call *network section*. A network section is defined based on some basic graph-theoretic terms: the undirected graph representing the resource network at hand is *connected*, if every pair of distinct nodes in the graph can be connected through some path. A *node cut set* (also known as vertex cut) of a connected network is a set of nodes, whose removal renders the network disconnected.

Definition 5. *A network section is defined over a node cut set with one-to-many and/or many-to-one nodes. It is represented by a biconnected subnetwork, whose nodes are given as the superset of the node cut set, i.e. a subnetwork that is not broken into disconnected networks by deleting any single node and its incident resources.*

Definition 6. *A closed chain of occupied resources is given as a cycle of occupied resources where all entities request some resource that is already occupied by another entity in the same cycle. Such a chain may be extended into multiple network sections. Every single network section is characterized by the* ***minimum required number of entities for a closed chain of occupied resources*** *in the overall resource network.*

```
proctype Mntr_assertion()
{
        do
        :: assert(nfull(Rsrc12)
               && nfull(Rsrc23)
               && nfull(Rsrc34)
               && nfull(Rsrc41)
               && nfull(Rsrc13))
        od
}
```

Fig. 10. Safety assertion for the DSBs interlocking control of the network of Figure 9

Predicate 1 *Under the following initial conditions, we verified that $DSBs$ interlocking control guarantees non-blocking execution and safety in all reachable states:*
- *At least one entity occupies a resource that is not in a separated network section.*
- *For all possible cycles, there is at least one network section, where the initial number of entities is less than the minimum required number of entities for a closed chain of occupied resources.*

If for example the network of Figure 9 is initialized with two entities occupying both $Rsrc34$ and $Rsrc41$ and another entity occupying some resource in the network section that is highlighted in Figure 11, then the second condition of *Predicate 1* implies that the $DSBs$ algorithm cannot guarantee non-blocking execution. This configuration opens a possibility for a closed chain of occupied resources.

However, in networks where moving entities represent traffic (e.g. in railway) the initial conditions of *Predicate 1* represent a marginal restriction. Usually, the number of moving entities within the individual network sections is very small compared to the minimum required number of entities for a closed chain of occupied resources.

In our case, non-blocking execution and safety with three entities is guaranteed only when at most one of the resources $Rsrc34$, $Rsrc41$ is initially occupied. In total, we model checked 4 valid initial configurations with three entities in the resource network of Figure 9 and 7 valid initial configurations with two entities. When having four entities in the network, the initial conditions of *Predicate 1* are not fulfilled.

Figure 12 provides representative model checking results for one of the valid initial configurations with three entities. The model checking output reports no assertion violation or invalid end states - deadlock - (errors: 0) in all reachable states (1697 stored states, when applying partial order reduction for state space pruning).

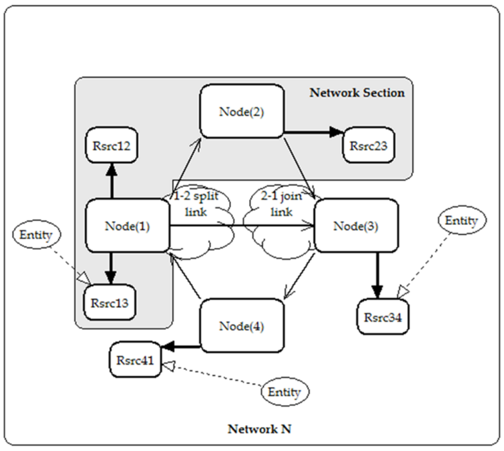


Fig. 11. Initial configuration that implies a closed chain of occupied resources

DSBs interlocking control is a compositional control algorithm. Control processes for the interconnected nodes can be composed as shown in the discussed resource network, such that the resulting communication by synchronous message passing provides the safety guarantees of interest. Model checking of large-scale networks with other topologies and node combinations fails to scale up well, since the state space that has to be explored can grow exponentially in the number of the implied control processes.

Compositional reasoning shifts the burden of verification from the network level to the subnetwork level, so that a global safety property for the network is established by composing together independently verified subnetwork properties like the one proved in the examined network. The closest compositional reasoning approach in the related bibliography is the *assumption–commitment* (A-C) method that was first proposed by [17] and that was subsequently developed in [18] into a sound and semantically complete proof method. This method integrates the use of inductive assertions and the proof method of [1] for verifying synchronous distributed message passing systems.

An A-C correctness formula has the form:

$$< A, C >: \{\phi\}P\{\psi\}$$

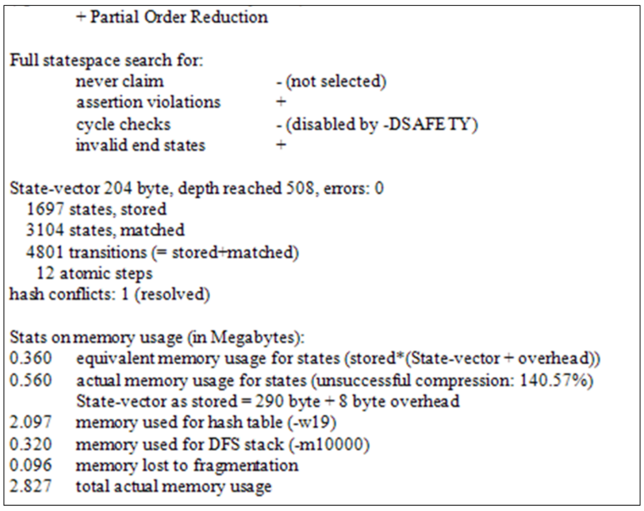
where P denotes a (PROMELA) program and A, ϕ, ψ, C represent predicates. We require that A and C predicates involve values that do not depend on the values of any program variables. A valid $A - C$ formula has the following meaning:

If ϕ holds in the initial state, in which P starts its execution, then

- C holds initially, and C holds after every communication provided A holds after all preceding communications, and
- if P terminates and A holds after all communications then ψ holds in final state.

3.2 Model checking resource occupancy and availability

Under the conditions of *Predicate 1* for the network of Figure 9 and for infinitely many requests for any network resource, we verified that there is always some entity that temporarily acquires and subsequently releases this resource.



```
        + Partial Order Reduction

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

State-vector 204 byte, depth reached 508, errors: 0
   1697 states, stored
   3104 states, matched
   4801 transitions (= stored+matched)
     12 atomic steps
hash conflicts: 1 (resolved)

Stats on memory usage (in Megabytes):
0.360   equivalent memory usage for states (stored*(State-vector + overhead))
0.560   actual memory usage for states (unsuccessful compression: 140.57%)
        State-vector as stored = 290 byte + 8 byte overhead
2.097   memory used for hash table (-w19)
0.320   memory used for DFS stack (-m10000)
0.096   memory lost to fragmentation
2.827   total actual memory usage
```

Fig. 12. Model checking DSBs interlocking control safety for the network of Figure 9

The aforementioned progress property guarantees availability of the network's resources for infinitely many uses and excludes the possibility for an entity to hold the occupied resource forever. For model checking this property we developed an appropriate formulation in the Linear Propositional Temporal Logic (LTL) of the SPIN model checker. The LTL operators used are the following:

$<> x$	eventually
$[]x = \neg <> \neg x$	always
$\rightarrow$	logical implication

The recurrence formula $[](<> p)$ asserts that in an infinite sequence of states the proposition p occurs infinitely many times [15]. For any network resource $Rsrcij$ we consider the propositions

$$\#defineR\_f(len(Rsrcij) == 0)$$
$$\#defineR\_o(len(Rsrcij) == 1)$$

corresponding to states of free or occupied resources. The checked correctness property is then expressed as

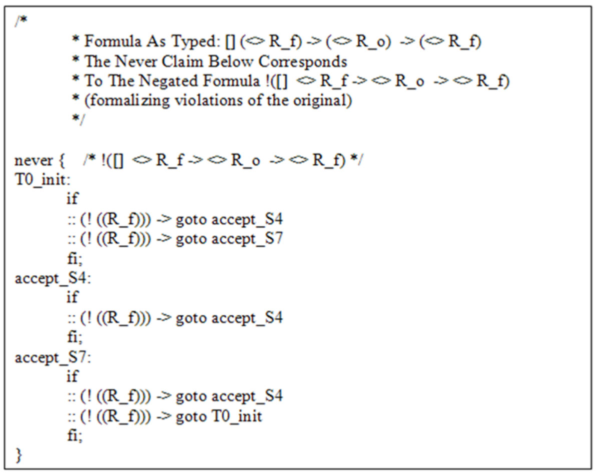
$$[](<> R\_f) - > (<> R\_o) - > (<> R\_f)$$

where $\rightarrow$ is left associative with higher precedence than $[]$. Thus, the formula is interpreted as $[](((<> R\_f) - > (<> R\_o)) - > (<> R\_f))$ and expresses the property:

"In an infinite sequence of system states a temporarily occupied resource becomes free infinitely often".

For the network of Figure 9 the discussed LTL formula was model checked in the 4 valid initial configurations with 3 entities, as well as in the 7 valid initial configurations with 2 entities. In these cases, SPIN generated the never claim (automaton in Figure 13) of the above formula and verified that the property holds in all possible executions.

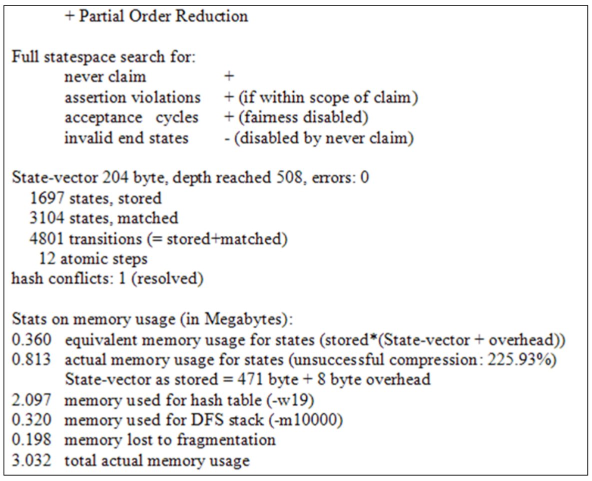
In SPIN, never claims specify either finite or infinite system behavior that should never occur. When a never claim is generated from an LTL formula, all its transitions are condition statements, formalizing atomic propositions on the global system state. SPIN checks infinite executions for the specified behavior. Execution of the claim starts at labeled statement $T0\_init$, where the conditions trigger transitions to the accept states, when the resource is occupied. Violation is detected as an acceptance cycle, i.e. if the resource remains occupied forever. If the resource is not occupied forever we do not have an acceptance cycle. Figure 14 reports representative results for one of the valid initial configurations with three entities, where we observe that SPIN performed a state space search for acceptance cycles. The shown output reports no errors (errors: 0).



```
/*
         * Formula As Typed: [] (<> R_f) -> (<> R_o) -> (<> R_f)
         * The Never Claim Below Corresponds
         * To The Negated Formula !([] <> R_f -> <> R_o -> <> R_f)
         * (formalizing violations of the original)
         */

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

Fig. 13. Never claim of LTL formula to model check that no entity occupies some resource forever



```
        + Partial Order Reduction

Full statespace search for:
        never claim             +
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 204 byte, depth reached 508, errors: 0
   1697 states, stored
   3104 states, matched
   4801 transitions (= stored+matched)
     12 atomic steps
hash conflicts: 1 (resolved)

Stats on memory usage (in Megabytes):
0.360   equivalent memory usage for states (stored*(State-vector + overhead))
0.813   actual memory usage for states (unsuccessful compression: 225.93%)
        State-vector as stored = 471 byte + 8 byte overhead
2.097   memory used for hash table (-w19)
0.320   memory used for DFS stack (-m10000)
0.198   memory lost to fragmentation
3.032   total actual memory usage
```

Fig. 14. Model checking resource occupancy and availability for the network of Figure 9

4 Related Work

Research on interlocking control has been mainly advanced in the area of railway interlocking systems. Since the introduction of mechanical interlockings in late 1800s the control has been progressively centralized with fewer control centers, individually responsible for larger portions of networks. This trend continued with the advent of computer controlled signaling to the railway networks. In related works, the most widely studied railway signaling system is the Solid State Interlocking (SSI) [6]. Many railway operators have adopted such geographic-data-driven solid-state control units in their interlockings. In [9], the author proposes an approach to formalize the principles and the concepts of interlocking systems in VDM.

The work reported in [20] introduces a model for the interlocking of the network used by a local Australian railway operator. Interlocking control is encoded in control tables and the described analysis aims to find erroneous or incomplete entries in these tables. Modeling and safety checking is performed with the NuSMV model checker, but in earlier works the same group used a Communicating Sequential Processes (CSP) approach and the Failure Divergence Refinement (FDR) model checker.

The work in [13] reports the safety checking of the Line Block interlocking system that also adopts a centralized approach. The control strategy runs on a Central Control Unit that communicates with Peripheral Control Units (PCUs). PCUs are expected to drive particular interlocking system components and detect external events.

In [10], the authors focus on a 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 [4], a tool that combines symbolic model checking and quantitative timing analysis. In [8], the authors present a model of the same system and validate safety in the presence of Byzantine system components or of some hardware temporary faults. The safety logic of the same system was also modeled in [5], where the authors used the SPIN model checker to analyze all system's functions that may be requested by an external operator.

We already noted the fundamental differences of our algorithm compared to the mentioned approaches. First, in DSBs interlocking control, safety is decoupled from entity routing and is an integrated network service that works independently from operation control and geographic data for the network topology. Second, we adopt a communication-based network control approach that makes our solution similar to the following distributed interlocking control proposals found in the related bibliography.

In [11], 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 uses 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. However, mobile communication requires security and reliability provisions that in a large-scale network increase the cost, when compared to solutions that transmit signals over wires.

EURIS [7] is a modular specification method used to formulate distributed interlocking logics for railway yards. The EURIS architecture consists of a collection of generic blocks representing control units that communicate by means of data structures called telegrams. EURIS uses the notion of routes, i.e. sets of network segments for which a train is granted exclusive access to all of them atomically. The building blocks maintain a state and can also exchange telegrams with the logistic layer that incorporates the logic behind operation control. Safety guarantees can be analyzed only through the available interactive simulation facilities. Compared to EURIS, our approach intentionally avoids application-domain-dependent concepts and system requirements, since we aim in the development of a generic interlocking algorithm. In our case, control decisions are taken on the basis of exchanged messages between the control units that, as opposed to the EURIS building blocks, do not store state related to the algorithm's logic. Moreover, our solution is fully verified with respect to the required safety guarantees.

5 Conclusions and Future Work

In the last years, with the ever-increasing computing power of small and inexpensive computing devices, distributed interlocking control is a promising alternative towards reducing the complexity involved in the systems' design, and towards reducing the costs for installation and maintenance of the needed equipment. Most current interlocking control approaches are centralized and they are defined on the basis of geographic data and commands of the networks operation control. Usually, interlocking safety of centralized solutions cannot be fully verified, due to the state space explosion involved.

In response to these concerns, we introduced a distributed interlocking control algorithm, where control logic for guaranteeing safety is decoupled from the network topology data and the used control units do not store information related to the algorithm's state. The basic control function is based on what we call Distributed Signal Boxes that are attached to the networks interlocking nodes. The algorithm works on the basis of point-to-point communication between control processes. We described the message communications between nodes and DSBs and the message communications between neighbor DSBs, for a series of node interconnection cases. The initial conditions of the verified properties guarantee safety and progress for the considered network that included all described network interconnection cases.

Future research includes adaptation of appropriate architectural solutions (e.g. [12]) for control processing redundancy and communication redundancy towards implementing fail-safe DSBs control architectures. An important concern is to demonstrate the applicability of the compositional verification technique of [18] for synchronous distributed message passing systems. This will enable the verification of large-scale networks by decomposing the verification problem into the model checking of properties for smaller networks, which will be independently verified.

References

1. Apt, K.R., Francez, N., de Roever, W.P.: A proof system for communicating sequential processes. ACM Trans. Program. Lang. Syst. 2(3), 359–385 (Jul 1980)

2. Arozarena, P., Frints, M., Abad, D., Gonzlez Ords, J., Fallon, L., Zach, M., Nguyen Thi Van, H., Serrat Fernndez, J.: Madeira: A peer-to-peer approach to network management. In: Proc. of the Wireless World Research Forum (2006)
3. Basagiannis, S., Katsaros, P., Pombortsis, A.: Interlocking control by distributed signal boxes: Design and verification with the spin model checker. In: Parallel and Distributed Processing and Applications. pp. 317–328 (2006)
4. Campos, S., Clarke, E., Minea, M.: The verus tool: A quantitative approach to the formal verification of real-time systems. In: Computer Aided Verification. pp. 452–455 (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 Aspects of Computing 10(4), 361–380 (Apr 1998)
6. Cribbens, A.: Microprocessors in railway signalling: the solid-state interlocking. Microprocessors and Microsystems 11(5), 264 – 272 (1987)
7. van Dijk, F., Fokkink, W., Kolk, G., van de Ven, P., van Vlijmen, B.: Euris, a specification method for distributed interlockings. In: Computer Safety, Reliability and Security (1998)
8. 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: Tools and Algorithms for the Construction and Analysis of Systems. pp. 535–549 (2000)
9. Hansen, K.M.: Formalizing railway interlocking systems. In: Proc. of FME Rail Workshop #2. FME:Formal Methods Europe (1998)
10. Hartonas-Garmhausen, V., Campos, S., Cimatti, A., Clarke, E., Giunchiglia, F.: Verification of a safety-critical railway interlocking system with real-time constraints. Science of Computer Programming 36(1), 53 – 64 (2000)
11. Haxthausen, A.E., Peleska, J.: Formal development and verification of a distributed railway control system. IEEE Trans. on Software Engineering 26(8), 687–701 (2000)
12. Hecht, M., Agron, J., Hecht, H., Kim, K.H.: A distributed fault tolerant architecture for nuclear reactor and other critical process control applications. In: [1991] Digest of Papers. The 21st Int. Symposium of Fault-Tolerant Computing. pp. 462–498 (June 1991)
13. Hlavaty, T., Preucil, L., Stepan, P., Klapka, S.: Formal methods in development and testing of safety-critical systems : Railway interlocking system. In: Conference on Intelligent Methods for Quality Improvement in Industrial Practice. pp. 14–25 (2002)
14. Holzmann, G.J.: The model checker spin. IEEE Trans. on Software Engineering 23(5) (1997)
15. Holzmann, G.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional, 1st edn. (2011)
16. Jain, A., Nelson, K., Bryant, R.E.: Verifying nondeterministic implementations of deterministic systems. In: Formal Methods in Computer-Aided Design. pp. 109–125 (1996)
17. Misra, J., Chandy, K.M.: Proofs of networks of processes. IEEE Trans. on Software Engineering SE-7(4), 417–426 (July 1981)
18. Roever, W.P., S. de Boer, F., Hannemann, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J.: Concurrency Verification: Introduction to Compositional and Noncompositional Methods. Cambridge University Press, New York, NY, USA (2001)
19. Simpson, A.: Model checking for interlocking safety. In: Proc. of FME Rail Workshop #2. FME:Formal Methods Europe (1998)
20. Winter, K., Robinson, N.J.: Modelling large railway interlockings and model checking small ones. In: Proc. of 26th Australasian Computer Science Conference - Vol. 16. pp. 309–316. ACSC '03 (2003)