# An Automatic SPIN Validation of a Safety Critical Railway Control System

S. Gnesi, G. Lenzini
IEI - CNR
Area della Ricerca di Pisa - S. Cataldo
Viale Alfieri 1 , 56010 Pisa, Italy
{gnesi, lenzini}@iei.pi.cnr.it

D. Latella
CNUCE - CNR
Area della Ricerca di Pisa - S. Cataldo
Viale Alfieri 1, 56010 Pisa, Italy
d.latella@cnuce.cnr.it

C. Abbaneo
Ansaldobreda Segnalamento Ferroviario (ASF)
Via dei Pescatori 35, Genova, Italy
cabbaneo@asf.ansaldo.it

A. Amendola, P. Marmo
Ansaldobreda Segnalamento Ferroviario (ASF)
Via Argine 425, Napoli, Italy
{amendola, marmo}@asf.ansaldo.it

## Abstract

*This paper describes an experiment in formal specification and validation performed in the context of an industrial joint project. The project involved an Italian company working in the field of railway engineering, Ansaldobreda Segnalamento Ferroviario, and the CNR Institutes IEI and CNUCE of Pisa. Within the project two formal models have been developed describing different aspects of a safety-critical system used in the management of medium-large railway networks. Validation of safety and liveness properties has been performed on both models. Safety properties have been checked primarily in presence of Byzantine faults as well as of silent faults embedded in the models themselves. Liveness properties have been more focused on a communication protocol used within the system. Properties have been specified by means of assertions or temporal logical formulae. We used* PROMELA *as specification language, while the verification was performed using the verification tool suite* SPIN.

## 1 Introduction

The increasing request for safety and better performance in automatic management of modern railways has forced the introduction of sophisticated dependable, computer-based, control systems [3]. Such systems have an intrinsic degree of complexity and require innovative validation techniques during design and developing phases. Traditional methodologies, such as testing and simulation, could be insufficient when applied to this kind of systems. Exhaustive testing is usually impossible because of the high number of runs to be analyzed, while simulation can provide useful information only on a limited set of sequences. An alternative approach is the use of Formal Methods (FM) to deal with these problems.

In the last decade many industries, like Ansaldobreda Segnalamento Ferroviario (ASF), started pilot projects [11, 15, 16, 6] directed to evaluate the impact of FM on their production costs. As a result, positive experiences [4, 7] have shown how, for railway control systems, it has been possible to formalize significant models and to perform validation using model checking [8, 17, 9] approaches. In some cases industries developed their own validation environment, as the LIVE of ASF [1].

In this paper we describe the results of a real project jointly carried out by ASF and the CNR Institutes IEI and CNUCE, in the context of the Pisa Dependable Computer Center of Consorzio Pisa Ricerche. The project consisted of two distinct parts: (a) designing a formal model of a critical control system; (b) verifying specific *safety* properties under the hypothesis of Byzantine behavior [14] of one of the system components, and verifying *liveness* properties of a dependable communication protocol used within the system. In this paper we focus on the general structure of the formal validation effort, while particular modeling strategies can be found in [10]. Industrial choices internal to ASF induced us to use PROMELA [12] as formal language and and SPIN [13] as model checker. In fact, SPIN was already used within ASF in successfully verifying safety properties of different parts of the system [2].

The paper is organized as follows: in Section 2 we briefly and informally describe the system and all its component units; in Section 3 we introduce the framework of our formalization work; in Section 4 and Section 5 we explain the formal models used and the properties verified on them, and

we discuss some significant result; finally in Section 6 we critically summarize on the whole experience.

## 2 System Description

The railway system considered in this work is a programmable centralized control system developed by ASF, specifically designed to manage a medium/large railway network. The system is composed by some control posts, from which *critical* commands are composed, and by *Peripheral Control Units*(PCUs) which in turn execute them. These commands are critical because their execution affect machineries such as railway semaphores, rail points, or level crossings. For this reason particular attention has been reserved to guarantee the safety of the system in case of fault-silent and Byzantine faults in some of the system components.

A control hardware, called *Safety Nucleus* (SN) has been specifically designed for control and safety purposes. It monitors on the state of the system and tries to discover a faulty component. Its architecture is based on a triple modular redundant [18] configuration of computers running different versions of the same program. It has to face also with internal consensus problems, but hardware constraints make the implementation of the classical solution of consensus in distributed architecture problem (i.e. the Byzantine Generals Problem [14, 5]) impossible. So, SN guarantees system safety not by looking for consensus, but excluding a faulty component or forcing the whole system in a safe shutdown.

## 3 Formal Specification and Verification

We developed two PROMELA models, called respectively TMR and TMR-PCU, each of them describing different views of the SN-PCU system. In particular:

1. the TMR model describes in detail the triple modular configuration of SN. The TMR model has been developed to verify safety properties of the triple modular redundant mechanism of the SN in presence of Byzantine behavior of one of its components;

2. the TMR-PCU model describes in detail the SN-PCU communication protocol, and the PCU architecture. Aspects of the SN behavior not related to the communication protocol have been left out. The TMR-PCU model has been developed to verify liveness properties of the SN-PCU protocol, and safety properties when hardware silent faults in the communication occur in the communication media or in some of the PCUs.

## 4 The TMR model

The general architecture of TMR (Fig. 1) consists of: (1) three identical *central modules*, called A, B and C, implementing the triple modular redundancy; (2) a module called *exclusion logic*, devoted to check the consistency of the three modules; (3) the PCUs, composed by $n$ control units (in our study we have considered $n = 2$); (4) communication *channels* between the modules (three symmetric channels), the modules and the exclusion logic (three symmetric channels), and the modules and the PCUs (a single bus); Our PROMELA model reflects quite faithfully this general

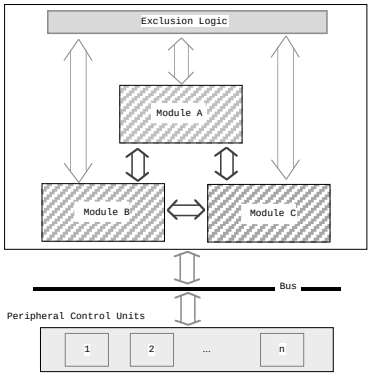


**Figure 1. The TMR architecture**

architecture.

The behavior of a module consists of a repeated sequence of *phases*. During each phase a central module runs local computations or communicates with other components of the system. In particular, in a *synchronization* phase each module sends to and receives from the other two modules, with time-out, a synchronization message. All of the communications in the ACC are synchronous with time-out: a time-out expiration is interpreted by the receiver as a sign of inactivity, while two time-outs force the receiver to a safe shut-down. In a *data exchange* phase each module broadcasts its local state, while in a *distributed voting* phase it votes on the information received. In a *communication with the exclusion logic*, the result of the voting is sent to the exclusion logic which can disconnect a module resulting in disagreement. Finally, in a *communication with the PCU*, a module communicates with the PCUs.

In developing the PROMELA code we had to solve the problem of simulating a time-out in the communications. In fact PROMELA does not deal with time. As a general solution we defined a particular EMPTY message, whose presence in a channel must be interpreted, by the receiver, as absence of any message it was waiting for. The *send* action changed too: it has been implemented as a non deterministic choice between either transmitting the “real” message or transmitting the EMPTY message. In the following we

report a synthesis of the PROMELA code implementing the synchronization phase for the module C[1].

```
/***      in the global environment        ***/

#define EMPTY 0 // the empty message
#define SYNCH 1 // the synchronization message
[..]

activeA=1;  /* state of A, in C viewpoint     */
sentA = 0;  /* flag "sent" (to module A)      */
recvA = 0;  /* flag "received" (from module A)*/

do
/* communication with A */
 :: (!sentA) ->
     if
     // send the synch (if A is active)
     :: true -> outA!(SYNCH && activeA);
     // send the empty message
     :: true -> outA!(EMPTY);
     fi;
     sentA = 1;
 :: (!recvA && inA?[synA]) -> inA?synA;
        recvA = 1;
/* set the activity state of A */
      if
      :: synA == SYNCH -> activeA = 1;
      // time-out implies not activity
      :: else -> activeA = 0;
      fi;
/* communication with B */
 [ ... the same for B ... ]

 :: (sentA && sentB && recvA && recvB) -> break;
od;

/* eventually safe shutdown*/
if
/* if the other modules are not active */
:: !activeA && !activeB ->
       \\ global state of module C
       global_activeC = 0;
       goto SHUTDOWN
:: else -> skip
fi;
```

The PROMELA code implementing the other phases is similar to the one of synchronization, except for the type of messages involved or for some local computation.

More interesting is the implementation of a Byzantine behavior. In this context, Byzantine behavior is to be intended as in Lamport et al. [14]: a Byzantine module runs the same algorithm as a loyal module, but it can arbitrarily fail in executing it, and in particular it may send wrong messages, or send no message at all. In this interpretation all the communication phases have been realized also in a *Byzantine* version, where a communication error in sending a message may occur. In the following we report the PROMELA code corresponding to the implementation of a Byzantine communication in the synchronization phase of module C:

```
 /* communication with A */
 :: (!sentA) ->
     if
     /* send the synch (if A is active) */
     :: true -> outA!(SYNCH && activeA);
     // send the wrong message */
     :: true -> outA!(-SYNCH);
     // send the empty message
     :: true -> outA!(EMPTY);
     fi;
     sentA = 1;
```

[1] We have omitted all the atomic directives and other strictly implementation details.

### 4.1 Formal Verification TMR

In this section we list some of the properties verified on the TMR model and the related results. We used either LTL formulae, or PROMELA *assertions*[2]. We used assertions for those properties that could be expressed as an invariant on all the run sequences. In the following we assumed that only one module might show a Byzantine behavior.

**(TMR1)** *After a communication phase it is always true that if two modules do not receive any reply from the third module, this latter module will be eventually disconnected by the exclusion logic.*

**(TMR2)** *After a communication phase, it is always true that if one module does not receive any reply from the other two modules, it will switch eventually in a safe shut-down state.*

**(TMR3)** *After a distributed voting phase, it is always true that if two modules, in reciprocal agreement on the global state knowledge, recognize that a third module is not in agreement with them, this latter module will be eventually disconnected by the exclusion logic.*

All the previous informal properties have been formalized with different LTL formulas with the following common structure:

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

where `p`, `q` and `r` are formalized as predicate on variables.

**(TMR4)** *After a communication phase, every module has sent and received a message (eventually the empty message) from the other modules.*

```
assert{(recvB+recvC==2) && (sentB+sentC==2)}
```

The previous assertion has been placed after each communication phase.

**(TMR5)** *If a module is in safe shut-down state then necessarily the other two have caused a time-out in a previous communication phase.*

```
assert{activeB + activeC == 0}
```

[2] An assertion in PROMELA is a statement including a boolean expression, which is evaluated each time the statement is executed. If the expression evaluates to `false` a violation of the correctness requirement is reported.

The previous assertion has been placed after the SHUTDOWN entry label. The verification runs have been performed using the compiler options of SPIN: COLLAPSE (CO) to compress the state vector and MA to obtain a minimal automaton encoding. In Table 1 we report on the results of the verifications.

**Table 1. Verification results on the TMR model.**

| *property* | *state vector* | *depth* | *result* |
|---|---|---|---|
| TMR1 | 192 | 5266 | success |
| TMR2 | 192 | 5266 | success |
| TMR3 | 196 | 45273 | fail |
| TMR4 | 188 | 297515 | success |
| TMR5 | 188 | 6808 | success |

### 4.2 Discussion

We briefly discuss the result about the property TMR3. In analyzing the counter example we noticed that the Byzantine module C caused one of the loyal module to be disconnected by the exclusion logic. In fact module C, not participating in a communication with one module, makes that module believe that module C is not active. Successively in the distributed voting the loyal module is found in disagreement, and then disconnected by the exclusion logic. This is a typical disagreement situation due to Byzantine behaviors. Discussing with ASF, we realized that in the real system there were further control mechanism (e.g. Control Redundant Check on all the messages) that guarantee anyway the safety in similar situation at a detail level not possible in our model.

## 5 The TMR-PCU model

The TMR-PCU describes in detail the SN-PCU communication protocol, and the PCU behavior. Target of the protocol is the delivery of critical commands also in presence of faults in the communication media. A scheme of TMR-PCU architecture is reported in Figure 2. We want to stress: (1) the three identical *central modules*, called *module* $A$, $B$ and $C$, implementing part of the SN; (2) the PCUs composed by $n$ control units (in our study we considered $n$=2), each constituted by two computers, we call A and B; (3) the *interconnections* among the modules (three symmetric channels), the ones between the modules and the PCUs (two busses). With the TMR-PCU model we were interested to verify:

1. *liveness properties* of SN-PCU communication protocol in an error-free environment hypothesis. This protocol is implemented as a distributed algorithm designed to assure a cyclic use of the busses and a cyclic selection of two modules demanded to send the commands.
2. *safety properties* of SN-PCU communication protocol in case of some hardware faults. In particular we were interested in silent faults in the interconnection busses and in the computers A and B of a peripheral unit.

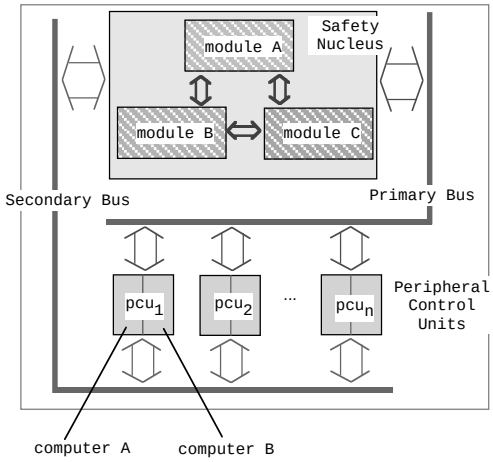


**Figure 2. The TMR-PCU architecture**

We now briefly describe the protocol run by a central module and the one run by a peripheral unit, giving the PROMELA code where significant.

### 5.1 A central module

Before communicating with the PCUs a module tries to infer information about the global state of the system. In particular, in a *synchronization* phase (the same of the TMR) a module tries to know the other modules activity state. This information is used in a distributed tournament procedure to decide which two modules have to be selected to send a message. In a *diagnostic* phase each module collects information about the global state of the PCU computers and of the busses. This information is used to decide which bus to use. Finally, in a *message elaboration* phase depending on the state of peripheral computers from a previous diagnostic, either the effective peripheral command, or a DIAGNOSTIC message is prepared.

### 5.2 A peripheral unit

In TMR-PCU the PCU model is realized in more detail. In the **decide the state** phase, a non-deterministic choice is made to decide on the functional state of the busses and of the computers A and B of the peripheral unit. In case of a state set to "fault" every communication resulted in a fault-silent behavior. Then each computer of each units waits for

a message from one of the busses. Then it replies with an acknowledgment to all the modules. In the following we report a synthesis of the PROMELA code:

```
/* recvA1,recvB1: #messages via bus1 */
/* recvA2,recvB2: #messages via bus2 */
/* stateBUS1, stateBUS2: state of bus1, bus2 */
/* stateA, stateB: state of computer A,B */
/* loop */
do
::
/* decide the state */
    if
/* fault in the 1st computer */
    :: stateA = 0
/* 1st computer is ok */
    :: stateA = 1
/* fault in the 2nd computer */
    [... the same for stateB ... ]
/* fault in the 1st bus */
    :: stateBUS1 = 0
/* 1st bus is ok */
    :: stateBUS1 = 1
/* fault in the 2nd bus */
    [... the same for BUS2 ...]
/* no fault */
    :: else -> skip
    fi;
RECEIVING:skip;
    i = 0;
   /* Promela channels defined */
   /* A1, A2 : computer A-BUS1, A-BUS2  */
   /* B1, B2 : computer B-BUS1, B-BUS2  */
    do
/* computer A receives from bus1 */
     :: !DONE && A1in?[PCU1, senderA1, msg] ->
        A1in?PCU1, senderA1, msg;
        if
/* if it is a diagnostic message */
        :: msg == DIAGNOSTIC -> skip;
/* if it is a command message */
        :: else -> msg[i] = msg; i++;
        fi;
/* acknowledgment to all the module */
        A1out!PCU1,A,(stateA && stateBUS1);
        A1out!PCU1,B,(stateA && stateBUS1);
        A1out!PCU1,C,(stateA && stateBUS1);
        recvA1++;
/* computer A receives from bus2 */
     [... the same using A2, stateBUS2,
              recvA2, senderA2 and stateA ..]
/* computer B receives from bus1 */
     [... the same using B1, stateBUS1,
              recvB1, senderB1 and stateB ..]
/* computer B receives from bus2 */
     [... the same using B1, stateBUS2,
              recvB2, senderB2 and stateB ..]
     :: DONE -> break;
    od;
RECEIVED: skip
/* endloop */
od;
```

### 5.3 Formal Verification on TMR-PCU

In this section we informally list some of the properties verified on the TMR-PCU model, and the most meaningful results.

**(PCU1)** *Correctness of the communication protocols, in absence of faults.*

We verified this properties checking for absence of deadlock. Here, with the term *correctness*, we mean correctness of the diagnostic test and of the tournament algorithm run by a module. In this case we slightly modified the PROMELA code of the PCUs in such a way to force a peripheral unit to receive messages according to the right cyclic use of the busses. An incorrect use of it by one of the central module would have caused a deadlock.

The following properties have been verified in presence of faults.

**(PCU2)** *When two or more modules are active each peripheral unit eventually receives exactly two messages, in a single loop.*

**(PCU2′)** *In presence of Byzantine errors in one module, when two or more modules are active each peripheral unit eventually receives exactly two messages, in a single loop.*

**(PCU3)** *When two or more modules are active each peripheral unit eventually receives exactly two message via different busses, in a single loop.*

**(PCU4)** *When two or more modules are active each computer of every peripheral units receives exactly one message, in a single loop.*

All the previous properties have been formulated with different LTL formulas with the following common structure:

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

where `p`, `q` and `r` are predicate on variables.

In the Table 2 we report some significant results.

**Table 2. Verification results on the TMR-PCU model.**

| *property* | *state vector* | *depth search* | *output* |
|---|---|---|---|
| PCU1 | 352 | 44047 | success |
| PCU2 | 284 | 25465 | success |
| PCU2′ | 284 | 1295 | fail |
| PCU3 | 284 | 25465 | success |
| PCU4 | 284 | 405178 | success |

### 5.4 Discussion

We briefly discuss the result of property PCU2′. We wanted to prove safety properties of the tournament algorithm in the hypothetic situation of a persistent Byzantine module. We proved that a Byzantine behavior in the communication with the periphery phase, makes fail the tournament algorithm. Analyzing the counter-example, we noticed that three modules (and not two) send a message to the periphery. Indeed, in the real system the exclusion logic, which we have omitted in this model, should have been

forced the system to a safe shutdown before entering in the communication with the periphery phase.Indeed this is what happens in reality, as proved by ASF on the real system. With this result we have underlined the critical role of safety logic: if it should fail in disconnecting a Byzantine module before it enters into the communication with the periphery phase the tournament algorithm might be wrongly executed.

## 6 Conclusions

The work described in this paper, related to a industrial project, consisted of the verification effort performed on a safety-critical system developed by ASF. During the formal verification we found some erroneous situations that underline potentially weaknesses in the system itself, successively corrected.

Although briefly described in this paper, many formalization problems has been faced during the modeling phase, primarily due to the lack of any concept of time in the PROMELA language, and secondly to an inappropriate (with respect to our needs) treatment of the termination of a processes in its run time support. This obliged us to follow modeling choices that have had a substantial impact in the formalization effort and, indirectly, in the state dimension of the model. To overcome this last problem we needed to design *ad hoc* abstraction strategies. All these formalization issues can be found in a companion paper [10].

## 7 Acknowledgment

This work was partly supported by the CNR/GMD cooperation project DECOR and by Progetto speciale CNR "Strumenti Automatici per la Verifica Formale nel Progetto di Sistemi Software ".

## References

[1] A.Amendola, L. adn P.Marmo, and F.Poli. Experimental Evaluation of Computer-Based Railway Control Systems. In *Proc. of FTCS-27*, pages 380–384, 1997.

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

[3] A. Amendola. Dependability of Railway Control Systems. In *Proc. of FTCS-26 (Pannel)*, pages 150–155, 1996.

[4] C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, and D. Romano. A Formal Verification Environment for Railway Signaling System Design. *Formal Methods in System Design*, 2(12):139–161, 1998.

[5] W. Bevier and W. Young. Machine Checked Proofs of the Design and Implementation of a Fault-Tolerant Circuit. Technical Report NAS1-18878, NASA, 1990.

[6] A. Borälv. A Case Study: Formal Verification of a Computerized Railway Interlocking. *Formal Aspect of Computing*, 10(4):338–360, 1998.

[7] A. Cimatti, F. Giunchiglia, G. Mongardi, D. Romano, F. Torielli, and P. Traverso. Formal Verification of a Railway Interlocking System using Model Checking. *Formal Aspect of Computing*, 10(4):361–380, 1998.

[8] E. M. Clarke and E. A. Emerson. Design and Synthesis of Ssynchronization Skeletons using Branching Time Temporal Logic. In *Lecture Notes in Computer Science*, volume 131, pages 52–71. Springer-Verlag, 1981.

[9] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification. *ACM Transaction on Programming Languages and Systems*, 8(2):244–263, 1986.

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

[11] J. F. Groote, S. F. M. van Vlijemn, and J. W. C. Koorn. The Safety Guaranteeing System at Station Hoorn-Kersenboogerd in Propositional Logic. In *Proc. of 10th Annual Conference on Computer Assurance (COMPASS'95)*, pages 57–68, 1995.

[12] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentise Hall, 1991.

[13] G. J. Holzmann. The Model Checker SPIN. *IEEE Transaction on Software Engineering*, 5(23):279–295, 1997.

[14] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transaction on Programming Languages and Systems*, 4(3):382–401, 1982.

[15] P. G. Larsen, J. Fitzgerald, and T. Brookers. Applying Formal Specification in Industry. *IEEE Software*, 13(7):48–56, 1996.

[16] M. J. Morely. Safety-Level Communication in Railway Interlockings. *Science of Communication*, 29:147–170, 1997.

[17] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. *Lecture Notes in Computer Science*, 137:337–371, 1982. Proc. 5th International Symposium on Programming.

[18] N. Storey. *Safety Critical Computer Systems*. Addison-Wesley, 1996.