Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2015) 23: 1507 – 1521

doi:10.3906/elk-1306-228

A new bitwise voting strategy for safety-critical systems with binary decisions

Mustafa Seçkin DURMUŞ*, Oytun ERİŞ, Uğur YILDIRIM, Mehmet Turan SÖYLEMEZ
Control Engineering Department, Electrical and Electronics Faculty, İstanbul Technical University, İstanbul, Turkey

Received: 26.06.2013 • **Accepted/Published Online:** 09.10.2013 • **Printed:** 28.08.2015

Abstract: The main issue in controlling safety-critical systems such as nuclear power reactors or railway interlocking systems is to provide high safety and reliability where the risk ratio is at the highest level because small errors might result in hazardous accidents (e.g., death or injury of many people). The N-version programming technique, where N-different modules run in parallel, can be used to improve the reliability and safety of such systems at the desired safety level. Decisions of N-different modules are then evaluated by another component, usually known as the voter, using different voting strategies. In the current study a bitwise voting strategy to evaluate module decisions that are based on safe-states of variables is proposed and possible synchronization problems between the modules are determined. Sequence diagrams and solutions for synchronization problems are also explained.

Key words: Bitwise-voting, N-version programming, programmable logic controllers, railway interlocking systems, safety-critical software

1. Introduction

The "combination of the probability of occurrence of harm and the severity of that harm" is defined as risk in [1]. Since physical components do not have zero failure rates and error is unavoidable when there are humans in charge, there are no applications with a zero risk. Several design and implementation methods are defined by the related standards for safety-critical systems to minimize the level of risk and possible faults. For instance, IEC 61508 (the umbrella standard) is the functional safety standard for all electrical, electronic, and programmable electronic safety-related systems. Several standards have been developed for different industrial applications under this umbrella standard: IEC 61511 for the process industry; EN 50126, EN 50128, and EN 50129 standards for railways; IEC 61513 for the nuclear industry; and IEC 62061 for machinery [2].

These standards bring out a widely known definition named safety integrity level (SIL), which is a discrete level for specifying the safety integrity requirements of the safety functions to be allocated to the Electrical/Electronic/Programmable electronic safety-related systems [1]. SIL definition is made in [3] for Software SIL as "a classification number that determines the techniques those have to be applied to reduce software faults to an appropriate level" and for System SIL as "a classification number that determines the required rate of confidence". As an example, for a SIL 3 system in high demand mode of operation or continuous mode of operation [1], the average frequency of a dangerous failure of the safety function per hour (failure rate - λ) is between 10^{-8} and 10^{-7} [4]. The corresponding value of the mean time to failure (MTTF) is roughly between 1000 and 10,000 years. In other words, a SIL 3 system is expected to work between 1000 and 10,000

*Correspondence: durmusmu@itu.edu.tr

years without falling into a hazardous state. Several methods are recommended for the software development process in order to satisfy the desired safety integrity level. Some examples of railway interlocking software designs, including modeling of the railway signaling components (signals, track circuits, etc.), can be found in [5–7].

The requirements for satisfying desired SIL lead designers to use predefined specific methods and techniques in both software and hardware development. Hardware solutions that have the appropriate SIL can be produced using certified commercial off-the-shelf (COTS) products, whereas the software development process requires much more concentration and work.

Several software architectures for developing safety-critical software are recommended by [3]: defensive programming, failure assertion programming, fault tree analysis, and diverse programming. Diverse programming (or N-version programming) is one of the most widely used architectures where N-different software versions are running in parallel.

N-version programming is introduced by [8] and [9] to improve the reliability and to mask possible software errors. The main concept of N-version programming is to develop N-different algorithms that have the same input-output specifications built up by N-different (ideally non-interacting) workgroups. In other words, during the execution of a program N-version programming detects residual software faults to prevent the system from getting into fatal failures and provides high reliability for execution continuity [3]. The outputs of these N-different software versions are then sent to another unit known as the voter. The voter can be in hardware, software, or both, depending on the application requirements [10]. The voter receives outputs of N-different modules and compares them depending on the voting strategy. Several voting algorithms (or strategies) and their comparisons can be found in the literature [11–18].

If the safe-state of the system is not determined (or if the system is not fail-safe), generalized voting strategies [11,12] can be used, where generally N is chosen as 3 (triple modular redundancy (TMR)) [19–21]. A possible TMR architecture is illustrated in Figure 1. In such topologies, the voter usually applies the majority (or a weighted average) of the decisions given by the modules.

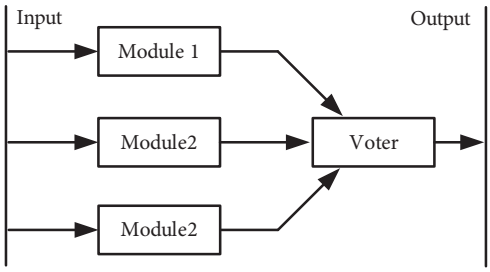

Figure 1. The architecture of TMR.

For fail-safe systems, where safety is the most important concern, a complete agreement of the modules can be sought before getting into an "unsafe" state. Hence, usually N is chosen as 2 [3]. The system gets into safe-state and produces predetermined fail-safe output when there is a disagreement between the modules. The use of such a strategy is usually encountered in high level safety systems. However, the main disadvantage of this strategy is a considerable reduction in system availability since disagreements between modules, which result in system locks, can happen frequently.

While comparing the incoming responses from the modules, two possible strategies exist: either all decisions are compared as a whole, or every decision bit can be compared separately. If the decisions are

compared as a whole, the system gets into the safe-state (all bits are set to their safe values) when there is an incompatibility between the decisions. In contrast, only the bits for which incompatibilities exist are set to their safe values in bitwise comparison of the decisions. In this sense, the bitwise voting strategy based on safe-states of variables can provide higher safety in comparison to the bitwise TMR architecture, while providing higher availability rates in comparison to the strategies that take the decisions as a whole.

For instance, in a majority voting strategy with three modules, the voter accepts the decision whenever two modules accept it even if these modules give wrong decisions. A more realistic example can be given for railway interlocking systems. While a train is moving on a railway track containing a switch, the switch must be kept in its position. In the TMR architecture with a majority voting strategy the switch position can be changed if two modules accept it, which may cause derailment of the train. However, in the proposed bitwise voting strategy if one of the modules denies the position change, the switch position will not change due to the safe-state of the switch position variable.

A possible disadvantage of the proposed voting strategy is the need for keeping track of module synchronization. The main contribution of this paper is to determine some synchronization and deadlock problems between the modules and propose possible solutions to these problems. Some of the other possible problems such as communication breakdowns, communication delays, or hardware malfunctions are not investigated here.

This paper is structured as follows: in Section 2 the proposed voting strategy is explained, two possible synchronization problems are identified in Section 3, a case study about a route reservation procedure together with solutions to the identified problems is given in Section 4, and the paper ends with the conclusion and possible future works.

2. Voting strategy based on safe-states

The safe-state of a variable can be considered as a state that prevents the whole system from falling into a dangerous situation due to that variable. The proposed voting strategy (hence, the proposed voter) demands the complete agreement of all modules to set a variable into its "unsafe" state. Figures 2 and 3 illustrate the general framework and the structure of the controller, respectively. In this study it is assumed that the voter and the modules are chosen as fail-safe programmable logic controllers.

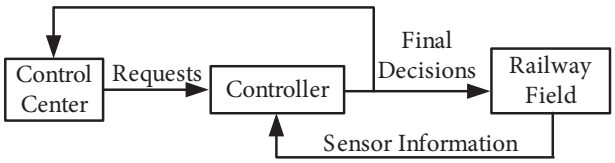

Figure 2. The general control framework.

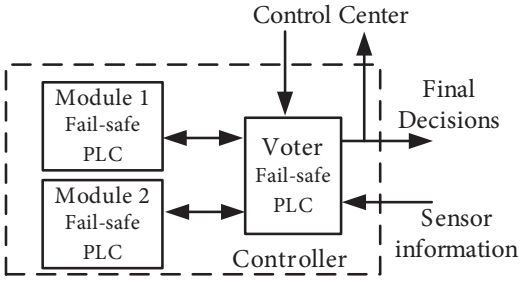

Figure 3. The structure of the controller (the voter and the modules).

The control center provides the user interface through which all field components can be controlled and monitored. It is obvious from Figure 3 that the voter also works as a communication unit between the field, the control center, and the modules (the modules do not interact with each other and the control center directly). A fail-safe communication protocol such as a safe-ethernet must be used for communication between all components (except for the communication between the voter and the control center) and the field. When a request is received from the control center (e.g., reserve route x or move switch 1 to its normal position), the voter sends this request to the modules concurrently and waits for their reply. After the voter receives the responses of the modules, it compares the responses according to a table where the safe-state of each variable is defined. The safe-state of each variable is determined at the "Software Safety Requirements Specification" level in the V-model [22] as an initial step before developing safety-critical software. N-different workgroups then develop their software using these specifications. Typically, voters are designed as simple as possible to reduce the probability of possible faults in the voter. In this study some simple but effective logic is also added to the voter as explained below.

The voter is designed as a state-independent machine. In other words, the decisions of the designed voter are independent from its previous decisions. This allows testing the voter for all possible inputs. In contrast, the modules are designed as state-dependent machines as they have to make decisions depending on previous inputs and states. In general, there can be an infinite number of possible evolutions of the states in such machines. Therefore, it is usually not possible to test the modules for all possible input combinations.

Frequently the modules do not send their decisions at the same time due to their cycle times and communication delays. In such cases, after the first decision is received from a module, the voter waits for a specified time, called consistency time, for the other module to give its decision. If the other module does not give any decision (no output to the voter), the voter produces an error called a consistency error. The voter keeps the corresponding output in its safe-state when there is an inconsistency. Moreover, the module that tries to drive a variable into its unsafe state cancels its decision upon receiving the consistency error for that variable.

The consistency time is usually determined depending on the cycle times of each module and expected communication delays. It is a nice practice to choose the consistency time greater than the sum of 2-fold of the communication time, the cycle time of the voter, and the module with a longer cycle time. Assume that the cycle times of the modules and the voter are 220 ms, 350 ms, and 100 ms, respectively. The communication between the voter and the modules is assumed as 300 ms. For this example the consistency time should better be selected greater than (600 ms + 100 ms + 350 ms) 1050 ms.

Similarly, when a module sends its decision after an incoming request from the voter, the module waits for a specified time called the synchronization time for the answer of the voter about its decision. If the module does not receive any answer back within this time, the module rejects the request and so does the voter. It is a nice practice to choose the synchronization time greater than the 2-fold of the sum of the communication time, the cycle time of the voter, the module with a longer cycle time, and the consistency time. For the given cycle time values in the previous paragraph, the synchronization time should better be selected greater than ((300 ms + 100 ms + 350 ms + 1050 ms) $\times$ 2) 3600 ms. The determination of the synchronization time also depends on the time constraints of the process under consideration.

The voter demands complete agreement of all modules before getting into an "unsafe" state [23,24]. In order to move from an unsafe sate to a safe-state, however, the decision of just one module towards this direction is enough. For instance, reserving a route for an incoming train needs more attention and the interlocking system has to check several conditions. Therefore, reserving a route demands a complete agreement of all modules.

Similarly, keeping a signal color red is safer than the other color information; thus, the signal color will always be red unless all modules produce the same color other than red.

If an incoming request is accepted or rejected, the voter sends proper signals to the field and informs the control center and the modules about the result. The voter also records discrepancies between the modules. The process between the voter and the modules can be considered as a kind of handshaking.

This process of handshaking brings up the issue of synchronization between the modules, which is one of the main problems related with the N-version programming. Normally, the voter sends the requests synchronously to the modules. However, the modules receive the requests at different times depending on their cycle times and communication delays. This may lead a module not to produce an output in time or even fall into an irrelevant state if the design has not been done carefully. In other words, synchronization problems between the two modules can occur. To prevent this kind of situation the voter sends proper signals after it receives a response from a module. This allows re-synchronizing of the other module that has not produced the expected response in time. There are also some other solutions to such problematic cases. One possible solution is to wait for both modules to recover themselves (each module waits until the end of the synchronization time for the other module). Another possible solution is to operate the modules in an asynchronous manner by using safety precautions as suggested in this study.

3. Synchronization problems

A problematic situation can arise if the modules can move into different states depending on the order of two incoming events. Consider the case illustrated in Figure 4, where the state of each module changes from state 0 to state 1 by the event e_1 and to state 2 by the event e_2. If there is a possibility for the modules to receive the events in different order, the modules can move into separate states. This kind of problem will be called a type 1 problem.

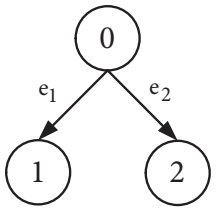

Figure 4. Type 1 problem.

A possible solution to this type of problems is letting each module change its state as shown in Figures 5a or 5b. This will resolve the problem since the order of occurrence of events e_1 and e_2 do not change the resulting state of the modules. It should be noted, however, that in this situation, if event e_2 changes the state of the modules from state 1 to state 2 and event e_1 changes the state of the modules from state 2 to state 1 (see Figure 5c), the modules may move into different states depending on the events they receive.

Sometimes the appearance and then the disappearance of a signal occur in a very short period of time. This situation may lead one of the modules to change its state while the other does not. As illustrated in Figure 6, the occurrence of event e_1 for a very short time leads one of the modules to state 1 while the other module waits in state 0. In such a case the modules divaricate at the end of the synchronization time. This case arises when event e_1 depends on the internal states of the module (such as time-out situations). This kind of problem will be called a type 2 problem.

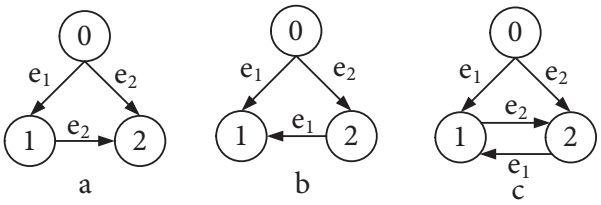

Figure 5. A situation where type 1 problem does not cause any failure (a, b), another situation where type 1 problem causes a failure (c).

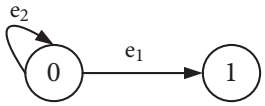

Figure 6. Type 2 problem.

A case study including the safe-state table and the comparison of decisions as well as possible solutions is given in the next section.

4. Case study: route reservation procedure for railway signaling systems

A detailed illustration of a railway signaling system is given in Figure 7. All requests and monitoring of the field are realized by the control center. The interlocking system is responsible for taking decisions by comparing the requests of the control center and the actual situation of the railway field. As mentioned earlier, an incoming request from the control center prompts the voter to send this request to the modules for evaluation. After evaluation, each module sends its decision back to the voter. The voter makes the final decision by considering both module decisions and the safe-state of the request. The voter then sends its final decision both to the control center and the modules. If the request is accepted, the voter also sends its final decision to the related field components.

As can be observed from Figure 7, it is assumed that trains coming from direction A can move to direction B or direction C. Specifications of these two routes are given in Table 1.

Table 1. Specification table.

Route	Switch Position	Outputs
A-B	Normal	Q_switch_A_normal
		Q_signal_Green
A-C	Reverse	Q_switch_A_reverse
		Q_signal_Yellow

According to Table 1, to reserve route A–B, the switch has to be in normal position, and when the route is reserved the entrance signal shows the green aspect. After the route reservation, it is forbidden to move the switch until a train has passed or the route is cancelled from the control center. A possible data sequence diagram for the reservation of route A–B is given in Figure 8.

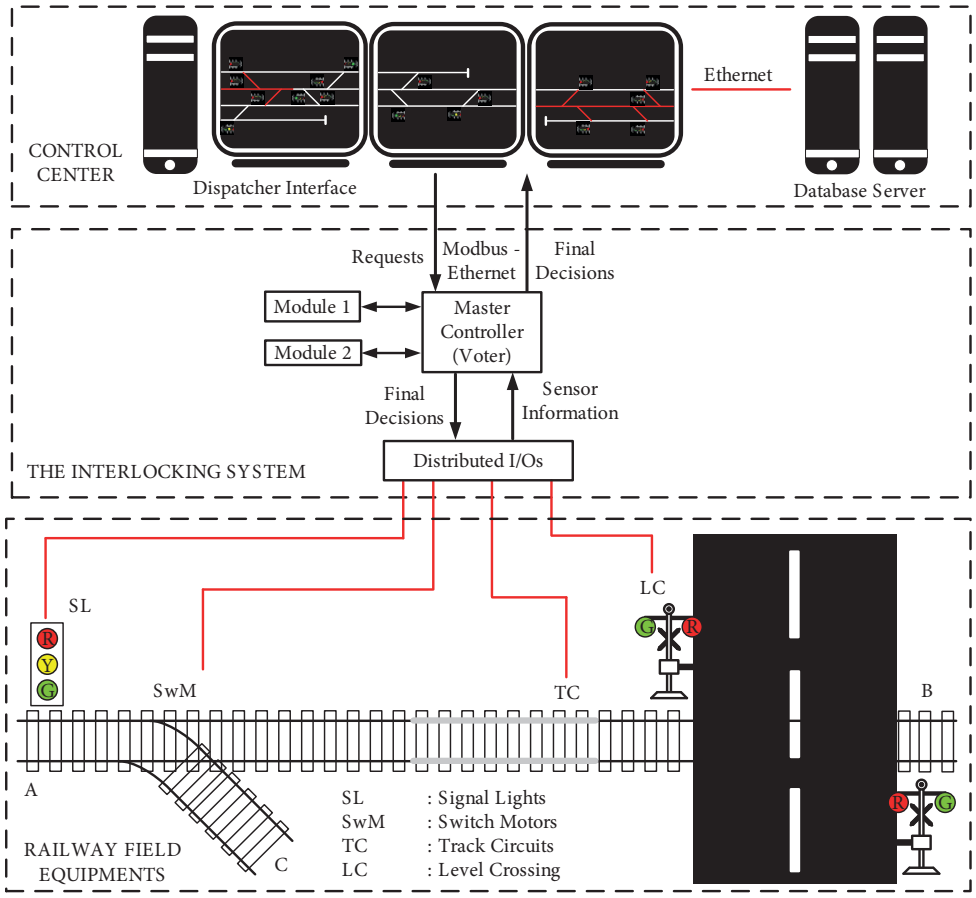

Figure 7. The general framework of a railway signaling system.

The final decision is always made by the voter and the voter sends this decision to the modules, the control center, and the field (if necessary). As mentioned earlier, the voter also takes into account the safe-state of each variable while making its decisions. The safe-states of the variables of concern are given in Table 2.

Table 2. The safe-states of the variables.

Variable that has Logic "1" Safe-State	Variable that has Logic "0" Safe-State
Route request accepted*	Route request accepted*
Route is reserved*	Route is reserved*
Cancel route reservation	Q_switch_normal_pos**
Route request denied	Q_switch_reverse_pos**
Route cancellation is denied	Q_signal_Green
Switch movement is denied	Q_signal_Yellow
Q_signal_Red	Route is cancelled

*Before a route reservation the safe-states of these two variables are assumed to be "0", whereas the safe-states of these variables are assumed to be "1" after a route is reserved. For example, before the reservation of a route the voter demands full agreement from the modules about "route request accepted bit". After the reservation the voter keeps this bit in "1" as long as at least one module keeps it in "1".

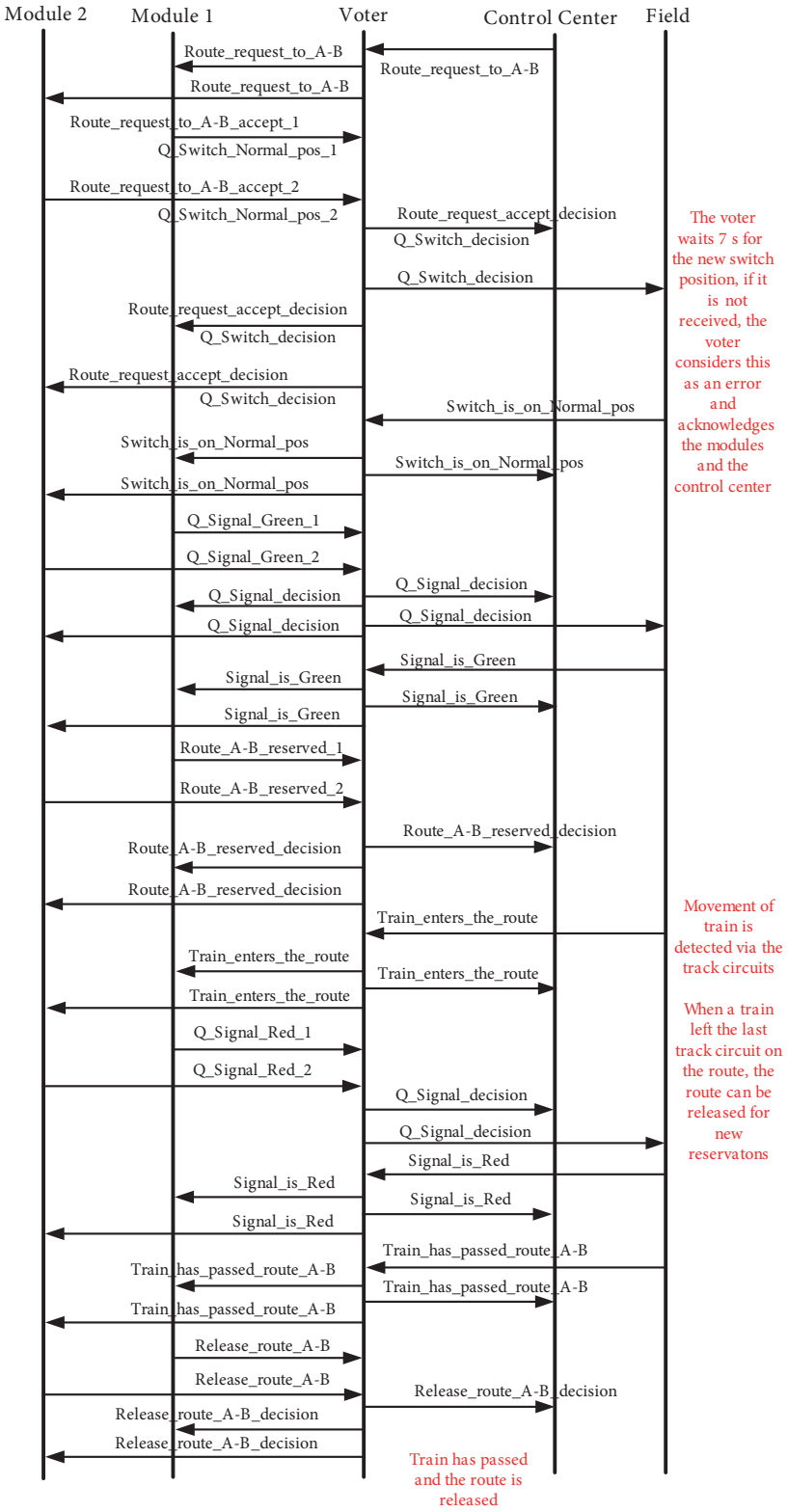

Figure 8. The sequence diagram for reservation of route A–B.

**In addition to the module decisions, the voter also checks the states of some other bits for the final decision. For instance, if the track circuit of a switch is busy, the voter does not change the position of the switch even if both modules agree, and once the movement of the switch is started, due to safety-requirements, the voter does not leave the switch in a middle position even if both modules stop sending switch movement commands.

4.1. Decision making

As mentioned above, the voter demands complete agreement of the modules for route reservation or moving the switch from one position to another. In other words, reserving a route is much safer than not reserving it, and keeping a switch in a position is much safer than moving it. These are illustrated in Figures 9 and 10, respectively.

According to the definition of defensive programming in [4] (where detection of abnormal control flow, data flow, or data values during their execution is determined and software reacts to these problems in a predetermined and acceptable manner), the voter sometimes can make decisions with respect to the decisions of the modules. For the railway signaling system given in Figure 7, when a train is moving on a railway area containing a switch, the interlocking system must not change the position of the switch even if both modules are in agreement to change the position of the switch. This is illustrated in Figure 11.

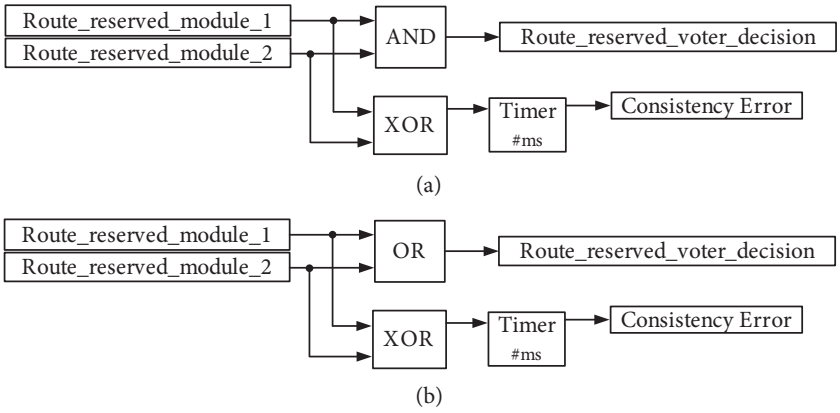

Figure 9. Route reservation: (a) before reservation; (b) after reservation.

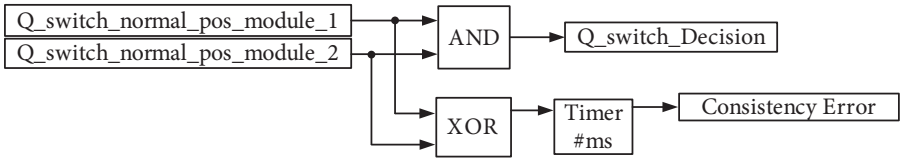

Figure 10. The switch movement decision.

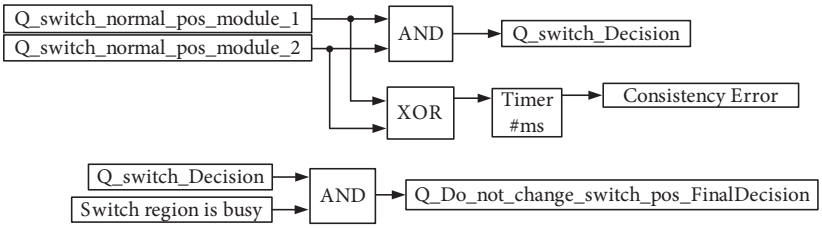

Figure 11. Prevention of the switch movement when its block is occupied.

Likewise, when a switch has begun to change its position following the complete agreement of all modules, the voter has to keep moving the switch until it reaches its new position even if one or all of the modules change(s) its/their decision. Leaving a switch in the middle position is forbidden by the operational conditions and safety-requirements of almost all railways in the world.

4.2. Solutions for type 1 and type 2 problems

A type 1 problem can be encountered while reserving a route. A route cancellation request can be made at any time instance, assuming that a route cancellation request is made from the control center after the modules

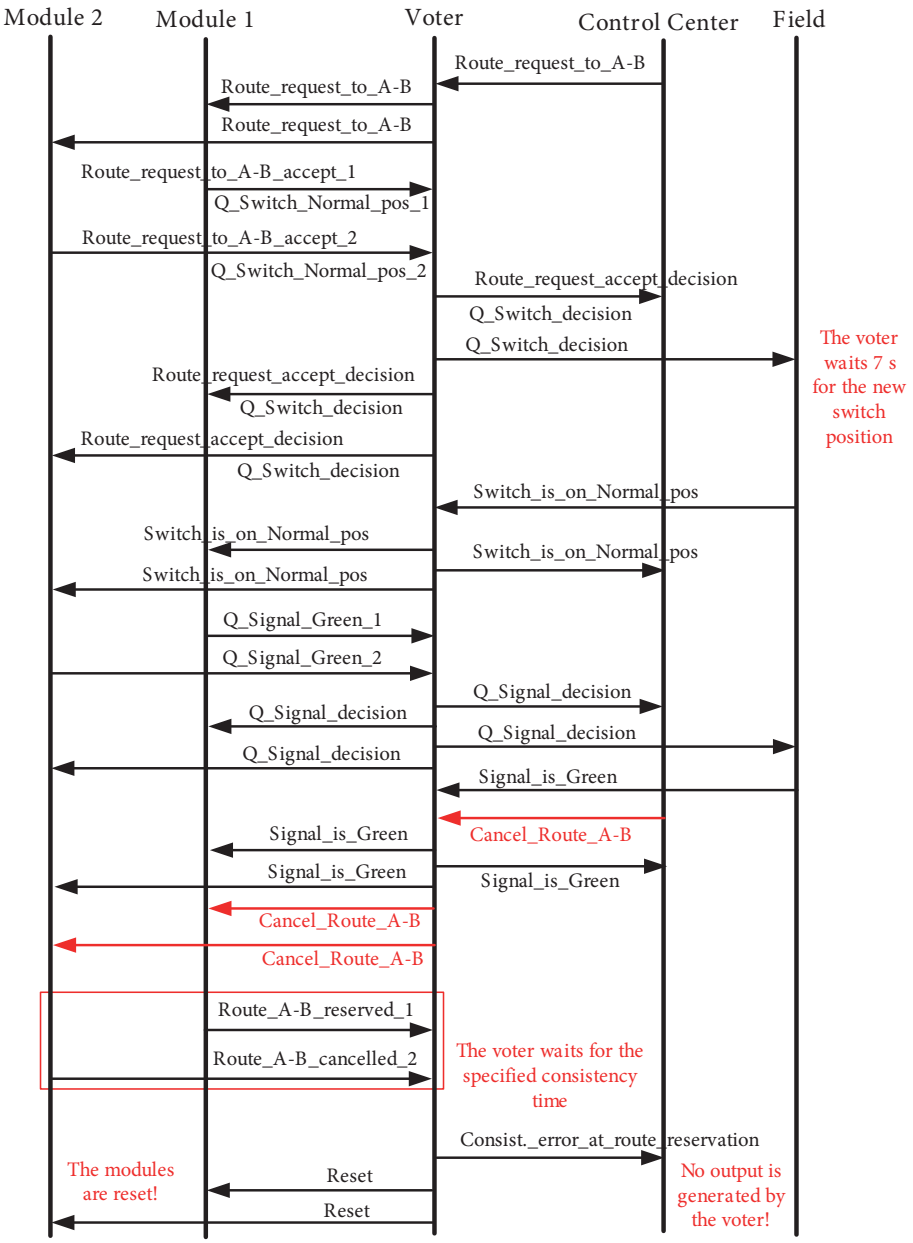

Figure 12. An example sequence diagram for a type 1 problem.

accept the route request and send proper signals to the voter. If Module 1 has a shorter cycle time, it receives the switch position indication and reserves the route before Module 2. If Module 2 receives the route cancellation request in this interval, it can cancel the route request before the route is reserved. Parallel running modules divaricate to different states. A sequence diagram of this example can be seen from Figure 12 (the red rectangle shows the different responses of the modules).

A possible solution for a type 1 problem is to convert the state diagram given in Figure 4 to Figure 5a or 5b. However, it is not always possible to convert a model to another one. An alternative way is to add extra states to synchronize the modules as given in Figure 13.

Before passing to state 2 from state 0, an extra state is added and named 2A. In this solution the modules have to pass to state 2A before passing to state 2; they can only pass to state 2 by event e_{2G}, which happens when all modules are in state 2A. If a module passes to state 2A and receives a signal from the voter indicating that the other module passed to state 1 (denoted by event e_{1G}), it also passes to state 1 from state 2A.

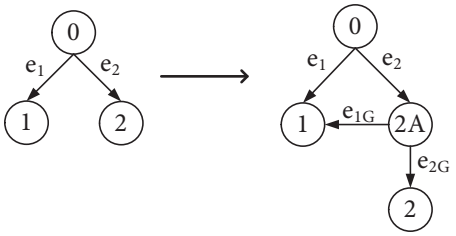

Figure 13. A possible solution for a type 1 problem.

This solution can be applied for the sequence diagram given in Figure 12. If a module accepts a request, the voter waits until the end of the consistency time for the response of the other module and does not send any incoming cancellation request to the modules. This is illustrated in Figure 14.

A type 2 problem can be encountered when detecting the faults of the railway field components. Assume that when a field component malfunction has occurred, the module with a shorter cycle time detects this fault first and moves into failure state. In this state the module will reject all incoming requests related to the faulty field component, whereas the other module (with a longer cycle time) might not catch this fault. In short, parallel running modules can divaricate to different states. An example sequence diagram for this case is given in Figure 15.

The first solution proposed for this kind of problem is to take back the module that passes to state 1 into state 0 again by event e_2. The second proposed solution is to take a module into state 1 from state 0 after receiving a signal from the voter indicating the passing of the other module into state 1 (denoted by event e_{1G}). These solutions are illustrated in Figures 16a and 16b, respectively.

By using Table 2 and the above definitions, the type 2 problem given in Figure 15 can be solved as follows: if a module detects a failure, it sends an identification signal to the voter about the failure. When the voter receives such a failure signal from any module, the voter sends proper signals to both modules. This signal enables the other module to detect the failure. Therefore, all incoming requests from the control center related to this faulty component will be rejected by both modules. This is illustrated in Figure 17.

5. Conclusion

Diverse programming, or N-version programming, provides high reliability and is highly recommended as software development architecture by the safety-related CENELEC standards to satisfy the required safety

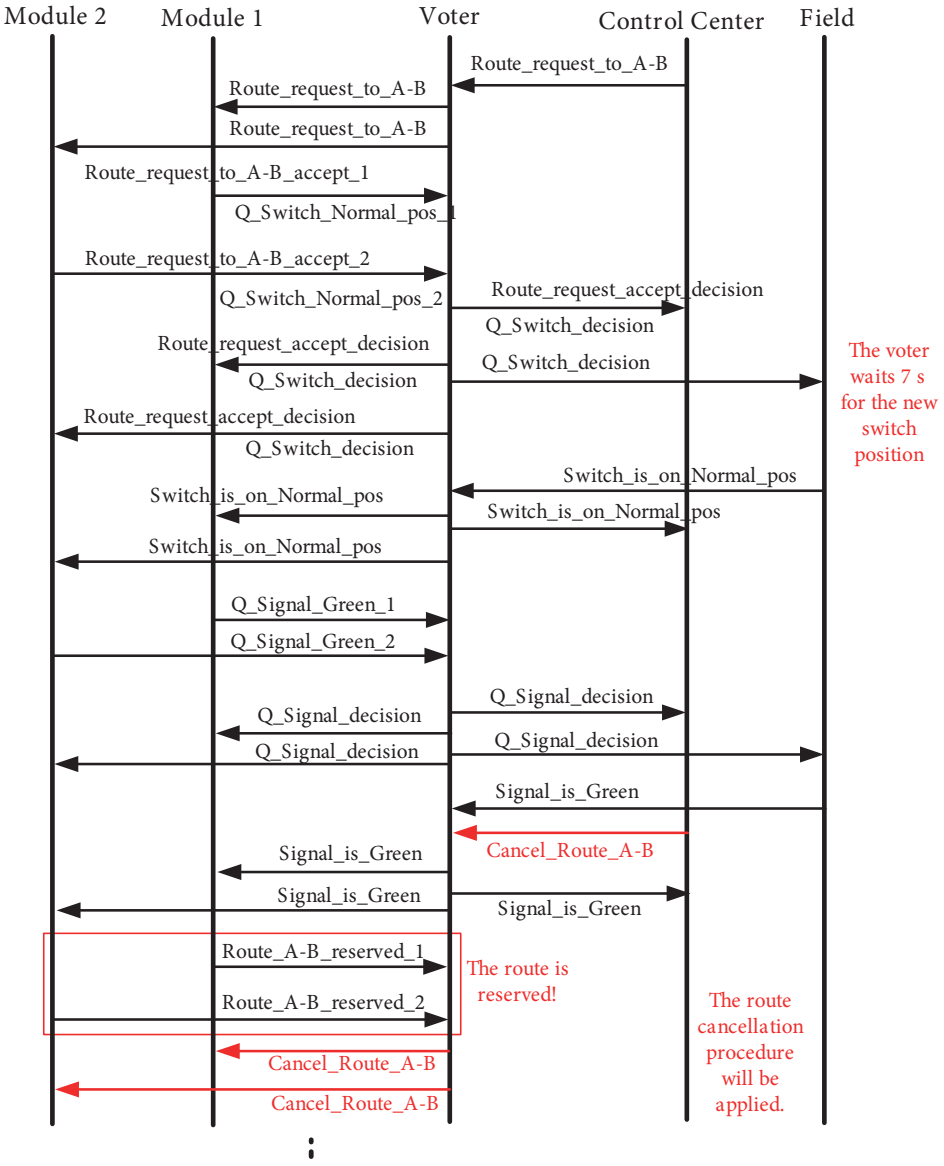

Figure 14. The sequence diagram of a possible solution for a type 1 problem.

integrity levels. The use of the diverse programming technique is also highly recommended for interlocking systems to achieve SIL 3 or SIL 4 software.

In the proposed control architecture N is chosen as 2 and these two parallel modules are supervised by a main controller called the voter. The voter takes into account the safe-states of each variable for making final decisions. Moreover, possible deadlocks and synchronization problems between the modules due to differences in their cycle times and design strategy are defined.

A case study of a railway signaling system is given to explain the solutions for the possible problems. A signaling system using this architecture has been realized in the Adapazarı region, Sakarya, Turkey. Future works can focus on the use of N-version programming with the proposed voting strategy in different safety-critical areas.

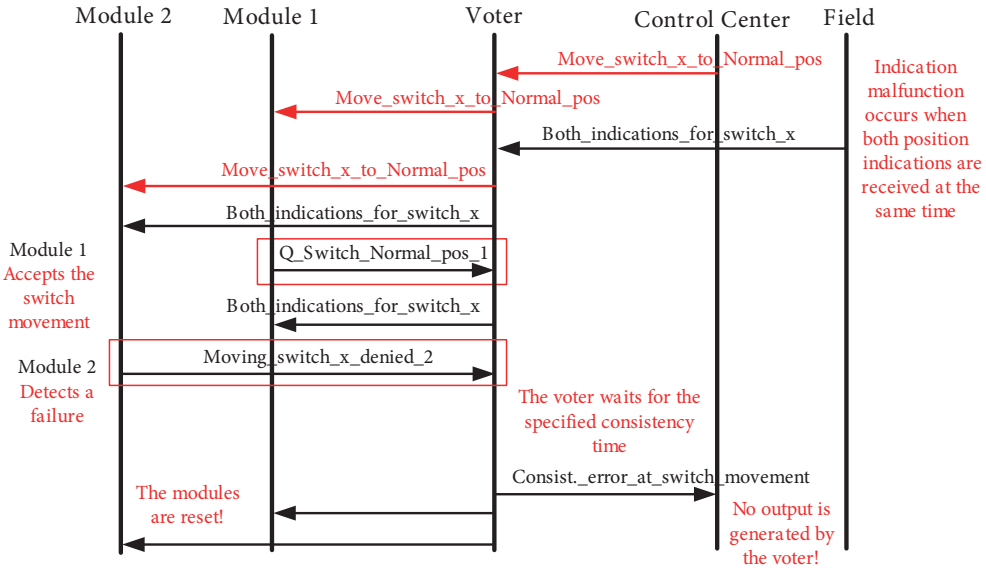

Figure 15. An example sequence diagram for a type 2 problem.

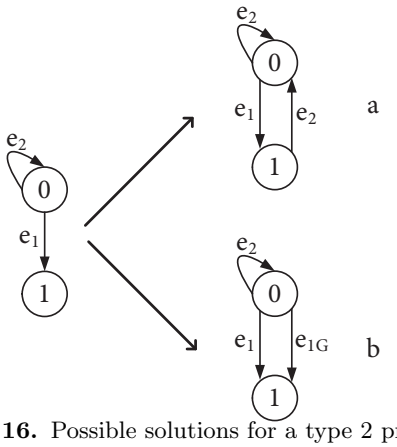

Figure 16. Possible solutions for a type 2 problem.

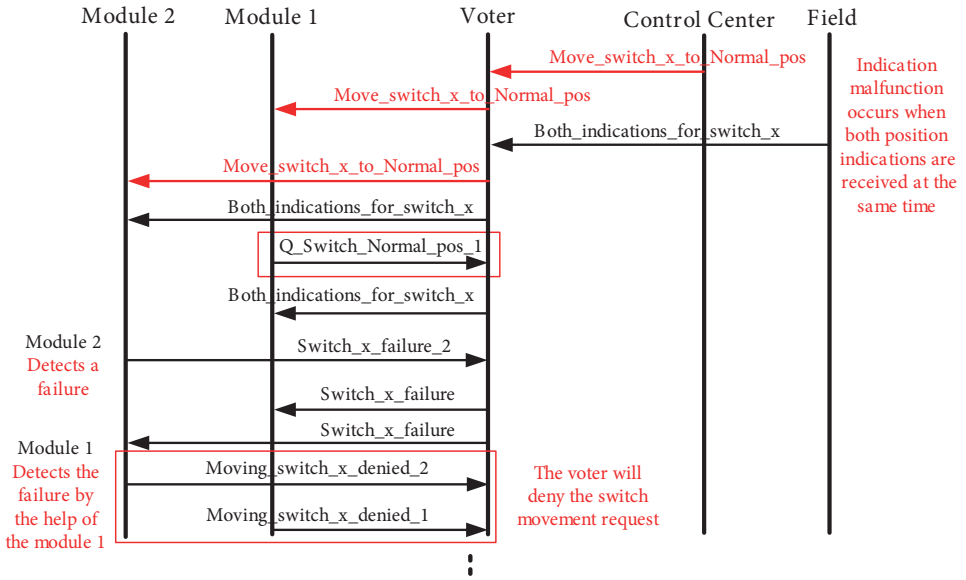

Figure 17. A sequence diagram of a possible solution for a type 2 problem.

Acknowledgment

This work is supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) project number 108G186 – The National Railway Signaling Project.

References

[1] IEC 61508-4. Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, Part 4: Definitions and Abbreviations, 2010.

[2] Smith DJ, Simpson KGL. Functional Safety–a straightforward guide to applying IEC 61508 standard and related standards. Burlington, MA, USA: Elsevier Butterworth-Heinemann, 2004.

[3] EN 50128. Railway Applications, Communications, Signalling and Processing Systems, Software for Railway Control and Protection Systems, 2001.

[4] IEC 61508-1. Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, Part 1: General Requirements, 2010.

[5] Durmuş MS, Yıldırım U, Söylemez MT. Application of Functional Safety on Railways Part I: Modelling & Design. In: IEEE 2011 8th Asian Control Conference; 15–18 May 2011; Kaohsiung, Taiwan: IEEE. pp. 1090–1095.

[6] Yıldırım U, Durmuş MS, Söylemez MT. Application of Functional Safety on Railways Part II: Software Development. In: IEEE 2011 8th Asian Control Conference; 15–18 May 2011; Kaohsiung, Taiwan: IEEE. pp. 1096–1101.

[7] Üstoğlu İ, Kaymakçı ÖT, Durmuş MS, Yıldırım U, Akçil L. Signaling System Design for Urban Transportation: The Case of İstanbul Esenler Depot. In: 9th Symposium on Formal Methods; 12–13 December 2012; Braunschweig, Germany: pp. 90–98.

[8] Avizienis A. Fault-tolerant systems. IEEE Transactions on Computers 1976; C25: 1304–1311.

[9] Avizienis A. The N-version approach to fault-tolerant software. IEEE Transactions on Software Engineering 1985; SE11: 1491–1501.

[10] Latif-Shabgahi G, Bass JM, Bennett S. A taxonomy for software voting algorithms used in safety-critical systems. IEEE Transactions on Reliability 2004; 53: 319–328.

[11] Lorczak PR, Cağlayan AK, Eckhardt DE. A Theoretical Investigation of Generalized Voters. In: 19th International Symposium on Fault-Tolerant Computing; 21–23 June 1989; Chicago, IL, USA: pp. 444–451.

[12] Gersting JL, Nist RL, Roberts DB, Van Valkenburg RL. A Comparison of Voting Algorithms for N-version Programming. In: 24th Annual Hawaii International Conference on System Sciences; 8–11 January 1991; Kauai, HI: pp. 253–262.

[13] Parhami B. Voting algorithms. IEEE Transactions on Reliability 1994; 43: 617–629.

[14] Mitra S, McCluskey EJ. Word-Voter: A New Voter Design for Triple Modular Redundant Systems. In: 18th IEEE VLSI Test Symposium; 30 April–04 May 2000; Montreal, Que: pp. 465–470.

[15] Latif-Shabgahi G, Bennett S, Bass JM. Smoothing voter: a novel voting algorithm for handling multiple errors in fault-tolerant control systems. Microprocessors and Microsystems 2003; 27: 303–313.

[16] Latif-Shabgahi G. A novel algorithm for weighted average voting used in fault tolerant computing systems. Microprocessors and Microsystems 2004; 28: 357–361.

[17] Latif-Shabgahi G, Hirst AJ. A fuzzy voting scheme for hardware and software fault tolerant systems. Fuzzy Sets and Systems 2005; 150: 579–598.

[18] Singamsetty PK, Panchumarthy SR. A novel history based weighted voting algorithm for safety critical systems. Journal of Advances in Information Technology 2011; 2: 139–145.

[19] Von Neumann J. Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. Automata Studies, Annual of Mathematic Studies. Princeton, NJ, USA: Princeton University Press, 1956.

[20] Lyons RE, Vanderkulk W. The use of triple-modular redundancy to improve computer reliability. IBM Journal of Research and Development 1962; 6: 200–209.

[21] Cooper AE, Chow WT. Development of on-board space computer systems. IBM Journal of Research and Development 1976; 20: 5–19.

[22] IEC 61508-3. Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, Part 3: Software Requirements, 2010.

[23] Durmuş MS, Yıldırım U, Eriş O, Söylemez MT. Synchronizing Automata and Petri Net based Controllers. In: 7th International Conference on Electrical and Electronics Engineering; 1–4 December 2011; Bursa, Turkey: pp. 386–390.

[24] Durmuş MS, Eriş O, Yıldırım U, Söylemez MT. A New Voting Strategy in Diverse Programming for Railway Interlocking Systems. In: IEEE International Conference on Transportation and Mechanical & Electrical Engineering; 16–18 December 2011; Changchun, China: IEEE. pp. 723–726.