2011 International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE)
December 16-18, Changchun, China

A New Voting Strategy in Diverse Programming for Railway Interlocking Systems

Mustafa Seçkin DURMUŞ[1], Oytun Eriş[1], Uğur YILDIRIM[1] and Mehmet Turan SÖYLEMEZ[1]
Control Engineering Department
Istanbul Technical University
Maslak, Istanbul, TURKEY
e-mail: {durmusmu, erisoy, yildirimu, soylemezm}@itu.edu.tr

Abstract — **The main issue in controlling systems such as nuclear power reactors or railway systems is to provide safety at the highest level where risk ratio is high and sometimes small errors might result with death of several people. In order to improve reliability and keep safety of such systems at the required level by decreasing common cause failures at the same time Diverse programming (which is synonymous with N-version programming) technique can be used. In this study an example railway interlocking architecture is explained and a new voting strategy is also proposed.**

Keywords; Diverse Programming, Safety-Critical Software, Railway interlocking systems.

I. INTRODUCTION

Risk is defined as "combination of the frequency, or probability and the consequence of a specified hazardous event" by CENELEC (European Committee for Electrotechnical Standardization) [1-2]. Since physical components do not have zero failure rates and error is unavoidable when there are humans in charge, there is no such thing as zero risk [3].

To minimize the level of risk and possible errors (or common cause failures) by improving reliability at the same time, in other words, to ensure safety of the systems some design and implementation methods and techniques are determined for safety-critical systems. Common cause failures can be regarded as failures that are linked due to some dependency and therefore occur simultaneously or at least within a short time interval [3].

As an example, in order to provide safety and reliability CEI EIC 61508 standard is developed as an umbrella standard for functional safety requirements of all kinds of electrical, electronic and programmable electronic safety-related devices and particularly for railway systems EN 50126 (where the specification and demonstration of Reliability, Availability, Maintenance and Safety (RAMS) analysis are described), EN 50128 (software development requirements for railway applications) and EN 50129 (safety related electronic systems for signalling) are developed [1], [2], [4].

These standards also results with a very important definition known as Safety Integrity Level (SIL). SIL is the probability for the system to execute the safety functions required in all specified input conditions within a specified time interval [5]. SIL is examined in two different categories as Software SIL (classification number that determines the techniques those have to be applied to reduce software faults to an appropriate level) and System SIL (number that determines the required rate of confidence that a system satisfies its especially determined safety features) [1], [2].

For railway interlocking systems the SIL level have to be at 3 [4], in other words, the interlocking system has to work minimum 1000 years without falling into failure state and to satisfy this safety integrity level some methods and techniques for modelling the system components (e.g. for fixed-block railway signalization systems signal lights, track circuits, switches and level crossings can be considered as components of signalization) are also determined in related safety standards. More details on components of railway signalization, modelling of these components and software development process can be found in [6-9].

In this study, recommended steps for developing a safety-critical software by CENELEC standards (via using V-model) are described in detail, software architecture of a sample railway interlocking system is explained and a new voting strategy is also proposed while taking decisions where controllers work in a diverse manner (in parallel).

The organization of this paper is as follows: In section 2 requirements of CENELEC for safety-critical software development are explained and our proposed voting strategy is explained in section 3. Conclusions and future aspect are given in section 4.

II. DEVELOPING SAFETY CRITICAL SOFTWARE

To develop safety-critical software for railway interlocking systems, steps those are defined in EN 61508-3 (V-model) which given in figure 1 should be followed.

A. Software Safety Requirements

These specifications are determined mainly by customer (for our study regulations of Turkish State Railways are primarily considered) and general safety rules (e.g. when a rotational mechanism is working keep all persons away).

978-1-4577-1701-7/11/$26.00 ©2011 IEEE

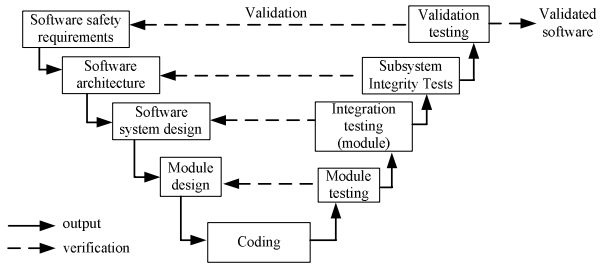

Figure 1. V-model.

B. *Software Architecture*

In order to meet suggestions of EN 50128, software designers should select proper methods (or techniques) for satisfying required SIL level. In order to satisfy the requirements of having software with SIL 3 level a sample selection is shown in figure 2.

TECHNIQUE/MEASURE	Ref	SWS IL0	SWS IL1	SWS IL2	SWS IL3	SWS IL4
1. Defensive Programming	B.15	-	R	R	HR	HR
2. Fault Detection and Diagnosis	B.27	-	R	R	HR	HR
-------						
5. Failure Assertion Programming	B.25	-	R	R	HR	HR
6. Safety Bag Techniques	B.54	-	R	R	R	R
7. Diverse Programming	B.17	-	R	R	HR	HR
--------						
16. Fault Tree Analysis	B.28	R	R	R	HR	HR
Requirements						
1. Approved combinations of techniques for Software Safety Integrity Level 3 and 4 shall be as follows:						
a) 1, 7 and one from 4, 5 or 12						
b) 1, 4 and 5						
c) 1, 4 and 12						
d) 1, 2 and 4						
e) 1 and 4, and one of 15 and 16						

Figure 2. Some suggestions of Table A.3 from EN 50128 (R-Recommended, HR-Highly Recommended).

The aim of *Defensive Programming* is to detect abnormal control flow, data flow or data values during their execution and react to these in a predetermined and acceptable manner [2]. In other words, designer has to check several extra control points to verify the validity of results and (or) variables of the program. For example, movement of a switch on an occupied railway block must not be allowed. Such simple general rules have to be checked independently, regardless of the program, before sending output signals to the field.

In *Failure Assertion Programming*, the main idea is to check the initial conditions for validity before execution of a command and check all results after the execution of a command. For example, position indication of a switch has to be checked by the software before and after a movement request to ensure correct functioning of the switch. Similarly, If software receives red and green indications from a signal light at the same time, this has to be caught by the interlocking software as a failure.

The aim of *Diverse Programming* (or N-version Programming [10]) is to detect and mask software design faults during execution of a program in order to prevent safety critical failures of the system and continue operation with high reliability. In this technique N independent groups develop N different software with the same specifications using different techniques. If the result is considered to be valid, the result is transmitted to the computer outputs. The N versions can run in parallel on separate computers, alternatively all versions can be run on the same computer and the results subjected to an internal vote. Different voting strategies [11] can be used on the N versions depending on the application requirements. More explanation about this topic and a new voting strategy will be given on next section.

C. *Software System Design & Module Design & Coding*

In this step designers have to choose appropriate techniques to satisfy SIL 3 requirements. Semi-formal methods like Automata or Timed Petri Nets can be chosen to model the system and then converting these models to programmable logic controller (PLC) codes (ladder logic or function block diagrams) at the coding part can be given as an example (different workgroups can work on the same problem in different places, by using different modelling and design techniques).

D. *Module Testing & Integration Testing & Validation*

After the development of software by different workgroups firstly single modules (each program developed by each team) have to be tested separately. *Functional and black box testing*, *Performance testing* can be given as an example to these tests. If modules passes this separate testing step successfully then integrated system tests are performed by combining these single N modules (N-versions). Software safety validation is done to ensure that the integrated system satisfies the specified requirements of software safety at the specified SIL. To achieve this step successfully Interlocking Test Program (ITP) is developed by Control Engineering Department of Istanbul Technical University. All testing steps can be realized manually or automatically by using this programme.

III. VOTING STRATEGY

Voting algorithms can be classified as *generic*, *hybrid* or *purpose-built voters* with respect to their functionality [11]. Several algorithms including Fuzzy logic voting [12], weighted average voting [13], triple redundant systems [14] and many more are proposed in the literature.

In this study a new voting strategy is proposed to consider the results of N-versions. The idea behind our voting strategy mainly relies on *safe-states* of every variable. *Safe-state* of a variable can be defined as preventing the whole system not to fall into a failure situation due to that variable.

For fixed-block railway interlocking systems all information is binary and because of this communication of controllers is carried out at bit-level. To understand the communication structure between controllers and the railway field architecture of interlocking system is given in figure 3 (For our architecture N is chosen as 3.

Interlocking PLC's are chosen as fail-safe Commercial-off-the-shelf (COTS) products which constituted via converting related models to PLC codes [7], [8]. As mentioned before, recommended semi-formal methods (Automata and Timed Petri Nets) [2] are used for modelling and these models are converted to PLC codes as function block diagrams (FBD) which is an recommended IEC 61131-3 programming language and highly recommended in [2].

Communication and Decision Making Unit (CDMU) is another fail-safe PLC that works as a main controller [9], [15] (and also a voter). CDMU is responsible for all

communication between all components given in figure 3. CDMU receives incoming commands (e.g. route requests or switch movement requests) and sends them to interlocking PLCs in parallel. If it is accepted after voting CDMU sends these commands to railway field and informs traffic control center about the result. CDMU also records mismatches between interlocking PLCs and errors.

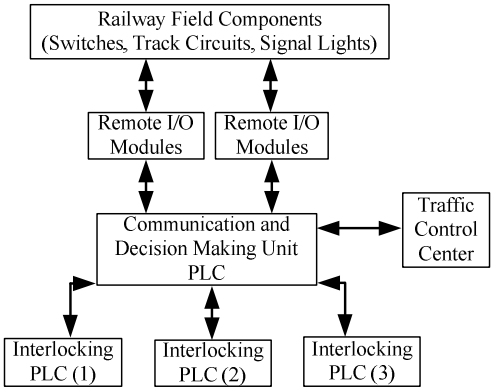

Figure 3. Architecture of interlocking system.

Since our voting strategy mainly relies on *safe-states* of every variable, CDMU has to contain safe-state information of all variables. An example table is given on table 1.

TABLE I. SAFE STATES OF SOME VARIABLES USED IN FIXED-BLOCK RAILWAY INTERLOCKING

Variable that has Logic "1" Safe-State	Variable that has Logic "0" Safe-State
Switch movement is blocked	Switch movement is unblocked
Q_Turn_on_red_light	Q_Change_switch_position
Route request is accepted*	Route request is accepted*

These variables are determined "Software Safety Requirements" phase in the V-model.

CDMU compares results of interlocking PLC's according to table 1. For instance, in order to block a switch movement (when a blocking command is sent from traffic control center) CDMU blocks the switch movement with just one agreement of any interlocking PLC whereas CDMU demands complete agreement of all interlocking PLCs (all 3 agree) for unblocking the movement of switch because keeping a switch movement blocked (its safe state is "1") is more safer than keeping it unblocked. The decision strategy is given in figure 4 and figure 5.

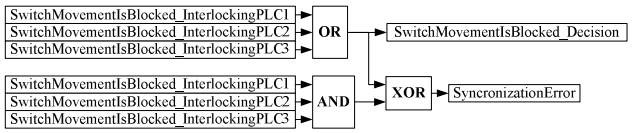

Figure 4. Blocking a switch movement.

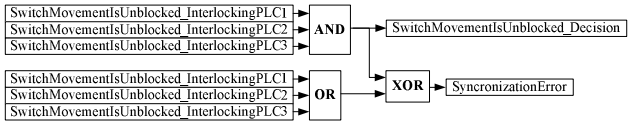

Figure 5. Unblocking a switch movement.

After first decision is received from an interlocking PLC to an incoming signal CDMU waits along a specified *synchronization time* for the other interlocking PLCs to give their decisions. If any of them do not give any decision (no output) to CDMU then this error is considered as *Synchronization Error* and CDMU cancels the request and informs traffic control center and the other interlocking PLCs which gave their decisions before.

In slightly different way, to move a switch from one position to another CDMU demands complete agreement of all interlocking PLCs (all 3 agree) because keeping a switch locked in a position is more safer than moving it from one position to another (its safe state is "0") and CDMU does not move the switch with just one agreement of any interlocking PLC.

However within the scope of *Defensive Programming* movement of a switch on an occupied railway block must not be allowed (see figure 6) or when a switch is begin to change its position (after agreement of all interlocking PLCs) even if one of interlocking PLC changes its decision (agreement of interlocking PLCs fails) CDMU has to keep moving of switch until it reaches to new position (because leaving a switch in the middle is forbidden by the laws of Turkish State Railways).

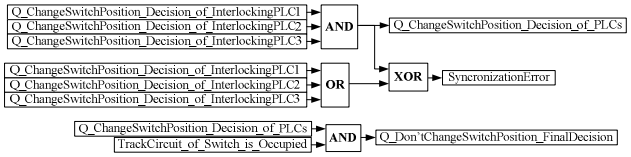

Figure 6. Prevention of switch movement when its track circuit is occupied.

In addition to these, there can be some variables with two safe-states. Safe-state of *Route request is accepted* variable in table 1 is "1" *before route is reserved* and it becomes "0" *after route is reserved*. Unlike the above explanations, CDMU demands complete agreement of all interlocking PLCs (all 3 agree) in order to accept a route request from traffic control center and after route reservation is done, CDMU again waits for complete agreement to cancel this variable.

In railway systems safety is the most important issue where human life is in question so keeping the whole system in safe-state can be time consuming but prevents hazardous accidents. For example, if a switch malfunction occurred after the entrance of a train on a reserved route then all signal lights on the railway field have to show red (safe-state).

IV. CONCLUSIONS & FUTURE WORK

Diverse Programming or in other words N-version programming eliminates common cause failures, improves reliability and also recommended by the railway related CENELEC standards. In this study, a railway interlocking system is described where three controllers work diversely under the supervision of a main controller (CDMU) in a synchronous manner.

A new voting strategy is also proposed where *safe-state* of each variable is defined. CDMU compares results of each interlocking PLC and decides with respect to *safe-state* table.

Obtaining a general solution based on this voting strategy that is applicable to other systems, preventing controllers those work in parallel from deadlocks and

solutions to possible critical races (sometimes controllers can give different results to same input due to difference of cycle times of each controller) between controllers can be considered as future works.

ACKNOWLEDGMENT

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

REFERENCES

[1] IEC 61508-3, Functional Safety of Electrical/Electronic/Programmabel electronic safety-related systems, Part 3: Software requirements, 1997.

[2] EN 50128, Railway Applications, Communications, signalling and processing systems, Software for railway control and protection systems, 2001

[3] D. J. Smith and K. G. L. Simpson, "Functional Safety – A straightforward guide to applying IEC 61508 standard and related standards", 2nd ed. Elsevier Butterworth-Heinemann, 200 Wheeler Road, Burlington, 2004.

[4] M. T. Söylemez, M. S. Durmuş, U. Yıldırım, "Functional Safety Application on Railway Systems: Turkish National Railway Signalization Project", The 24th International Congress on Condition Monitoring and Diagnostics Engineering Management, COMADEM'11, Stavanger, Norway, 30 May - 1 June, 2011, pp 1683-1692.

[5] M. Spellemaeker, L. Witrant, "How to Determine the Safety Integrity Level (SIL) of a Safety System," Available online at: http://www.indsci.com/docs/Press/PIN_0907.pdf. (13.10.2011).

[6] M. S. Durmuş, U. Yıldırım, A. Kursun, M. T. Söylemez, "Fail-Safe Signalization Design for a Railway Yard: A Level Crossing Case", WODES'10, International Workshop on Discrete Event Systems, 30 August - 01 September, Berlin, Germany, 2010.

[7] M. S. Durmuş, U. Yıldırım, M. T. Söylemez, "Application of Functional Safety on Railways Part I: Modelling & Design", The 8th Asian Control Conference, ASCC'11, Kaohsiung, Taiwan, 15-18 May, 2011, pp 1090-1095.

[8] U. Yıldırım, M. S. Durmuş, M. T. Söylemez, "Application of Functional Safety on Railways Part II: Software Development", The 8th Asian Control Conference, ASCC'11, Kaohsiung, Taiwan, 15-18 May, 2011, pp 1096-1101.

[9] O. Eriş and S. Kurtulan, "Application of Functional Safety on Railways Part III: Development of a Main Controller", The 8th Asian Control Conference, ASCC'11, Kaohsiung, Taiwan, 15-18 May, 2011, pp 1102-1104.

[10] A. Avizienis, "The N-version Approach to Fault-Tolerant Software", IEEE Transactions on Software Engineering, Vol. SE-11, No. 12, pp. 1491-1501, 1985.

[11] G. Latif-Shabgahi, J. M. Bass and S. Bennett, "A Taxonomy for Software Voting Algorithms Used in Safety-Critical Systems", IEEE Transactions on Reliability, Vol. 53, No:3, pp. 319-328, 2004.

[12] G. Latif-Shabgahi and A. J. Hirst, "A Fuzzy Voting Scheme for hardware and Software Fault Tolerant Systems", Fuzzy Sets and Systems, Vol. 150, pp. 579-598, 2004.

[13] G. Latif-Shabgahi, "A Novel Algorithm for Weighted Average Voting Used in Fault Tolerant Computing Systems", Microprocessors and Microsystems, Vol. 28, pp. 357-361, 2004.

[14] A. Chakraborty, "Fault Tolerant Fail Safe System for Railway Signalling", Proceedings of the World Congress on Engineering and Computer Science, Vol. II, pp. 1177-1183, 2009.

[15] M. S. Durmuş, U. Yıldırım, O. Eriş, M. T. Söylemez, "Synchronizing Automata and Petri Net based Controllers", 7th International Conference on Electrical and Electronics Engineering, ELECO'11, Bursa, Turkey, 01 - 04 December, 2011. (*Accepted for oral presentation*).