Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2008

A Resilient SIL 2 Driver Machine Interface for Train Control Systems

Andrea Ceccarelli[1] Istvan Majzik[2] Danilo Iovino[3] Fausto Caneschi[4]
Gergely Pinter[2] Andrea Bondavalli[4]
1- University of Florence, Viale Morgagni 65, I-50134, Firenze, Italy
andrea.ceccarelli@unifi.it
2- Budapest University of Technology and Economics, Magyar Tudosok krt. 2,
Budapest, Hungary, majzik@mit.bme.hu, pinter@mit.bme.hu
3- Ansaldo Segnalamento Ferroviario S.p.A., Via Volvera 50, Piossasco (TO), Italy
iovino.danilo@asf.ansaldo.it
4- ISTI-CNR, Via Moruzzi 1, I-56234, Pisa, Italy
fausto.caneschi@lecitconsulting.it, bondavalli@unifi.it

Abstract

In railway train-borne equipment, the Driver Machine Interface (DMI) acts like a bridge between the train driver and the onboard automatic train control system (European Vital Computer, EVC). While the DMI is required to operate in a critical context, current DMIs have no safety requirements. This implies that the EVC may automatically stop the train whenever the DMI is suspected to misbehave, leading to delay of the train, inconvenience for passengers and consequent possible profit loss. For these reasons a DMI with higher safety requirements is worth to be taken into account, even if it implies higher costs. The SAFEDMI European project aims at developing (i) a DMI at Safety Integrity Level 2 (SIL 2) using off-the-shelf components and a simple hardware architecture to reduce costs, and (ii) a SIL 2 wireless communication support for maintenance. This paper describes the architecture of a DMI which satisfies these objectives. The main hardware and software characteristics will be shown, including the proposed error detection techniques and the related fault handling (characterized by a new operational mode that allows DMI to restart silently, thus reducing unexpected train stops).

1 Introduction

Two main components of railway train-borne equipment are the European Vital Computer (EVC, in the European Train Control System) and the Driver Machine Interface (DMI). The EVC is the main core of the on-board automatic train control system: it supervises the movement of the train and sends information to the DMI. The train is supervised using information received both from the "eurobalises" (transmitters located along tracks) and from the Radio Block Center (RBC) through a GSM-Railway network. The DMI acts like a bridge between the train driver and the EVC. It communicates with the EVC as a slave; it shows (using both audio and video devices) EVC messages and information to the driver, and communicates inputs from the driver to the EVC. The train driver interacts with the DMI using the DMI's LCD screen, audio devices and keyboard (or touchscreen); the train driver performs the two key roles of (i) information sink for EVC originated information, which is displayed by the DMI, and (ii) command source for commands to be delivered to the EVC by means of the DMI.

The EVC is a safety-critical component, its safety being assessed according to Safety Integrity Level 4 (SIL 4) as prescribed by the related CENELEC standard [5]. This means that the tolerable hazard rate per hour (THR) is required to be between 10^{-9} and 10^{-8}. So far, the DMI has not been considered a safety critical component. However, a DMI is required to operate in a critical context, which is becoming even more critical because of the increasing demand for speed and railway line capacity. Current DMIs are classified at SIL 0, which means that they have no safety requirements, and the EVC does not rely on their safe behaviour. This way, the EVC may automatically stop the train whenever the DMI is suspected to misbehave, leading to inconveniences and delays, thus reducing the quality of the service offered to the passengers. These are the main reasons why Railway Authorities of many EU countries have begun to require DMIs that feature at least SIL 2 (i.e., $10^{-7} \leq THR \leq 10^{-6}$). It has to be stressed that currently no SIL 2 train-borne DMIs are available.

Another limitation of current DMIs is that they need to be physically accessed in order to perform a configuration or even a software update. The introduction of wireless communication technologies is considered to be the best solution to allow quick and easy DMI configuration and software/firmware upgrade without mechanical operations. Wireless technologies not only supply a faster way for maintenance: they also offer the possibility to set up Maintenance Centres, which can perform the update of the DMI software and collect information (e.g., periodical diagnostic information) sent by the DMI.

This paper describes the hardware and software architecture of a SIL 2 DMI (from now on we will call it SAFEDMI). This component is the result of activities related to the SAFEDMI European project, whose main objectives are designing and developing a SIL 2 DMI system able to perform safe wireless communication to support easy configuration, updating and diagnostic operations. An additional challenge of the project is to use off-the-shelf components (COTS) and a simplified hardware architecture, so to reduce the costs introduced by the safety requirements. The development of a DMI at a higher safety integrity level is in fact characterized by high costs caused by the more redundant architecture of the DMI (including extra hardware and software components to assure safe behaviour), and a more rigorous development process. One of the main factors to reduce the costs is the elaboration of a proper architecture that addresses the safety issues of a reasonable SIL.

Special attention has been devoted to the increase of the availability of train missions. With SIL 0 DMIs the train is stopped as soon as the EVC suspects the misbehaviour of the DMI: if, for example, there is a late reply to a message, the EVC cuts the DMI-EVC connection and activates the train brakes to stop the train. To re-start the train, the driver has to switch manually the EVC-DMI connection from the failed DMI to a second spare DMI located on the train. The SAFEDMI architecture features a special operating mode that enables to automatically restart the DMI without providing notification to the EVC, thus allowing the system to recover from transient faults, so to increase the availability of the train mission. The envisaged solution is that, when a fault is detected, the DMI does not enter immediately into a "safe" mode, which would prevent it to operate, rather it goes into a "suspect" mode. When in the suspect mode, the DMI tries to re-start, for one or more times, depending on a proper mechanism [1], [12]. If the detected fault is a transient one (or if there has been an incorrect fault detection) and the EVC does not query the DMI for critical acknowledgements during the restart period, the DMI will be able to become fully operative again without interrupting the train mission. Note that this new operating mode is fully compliant with the current European standards ([15], [14]) about EVC-DMI communication protocols and requirements.

In this paper we describe the realization of the SAFEDMI hardware and software architecture and the related mechanisms to guarantee safety requirements, while we will not deal with the wireless communication protocols. The paper is organized as follows: in Section 2 we present the main SIL 2 requirements that motivated the SAFEDMI architectural choices, in Section 3 we describe an overview of the system and its environment, in Section 4 we show the operational modes, in Section 5 we introduce software and hardware architectural choices and in Section 6 we present the error-detection techniques and fault handling mechanisms adopted in various software modules to satisfy the safety requirements. Concluding remarks are in Section 7.

2 SIL 2 Requirements

In this section we briefly describe the main requirements presented in the CENELEC standards that need to be satisfied to develop a SIL 2 compliant component. In particular, in this section we present the main hardware requirements [5] and the main software requirements [2]. The complete SAFEDMI hardware and software architecture has been developed keeping in mind the need to satisfy these requirements.

2.1 Hardware requirements

EN 50129 requires assurance that no single random hardware component failure mode is hazardous. This fail safety property can be achieved by (i) composite, (ii) reactive or (iii) inherent fail-safety techniques.

Composite fail-safety means that each safety-related function is performed by at least two independent items and non-restrictive activities are allowed only if the necessary number of items agree. According to the initial goals of the SAFEDMI design, especially considering the requirement to reduce the number of hardware components in the system, this alternative was not chosen.

Reactive fail-safety assures safe operation by proper detection and negation of hazardous faults that occur in a single item. However, the fault detection function is regarded as a second item that shall be independent in order to avoid common-cause failures. Independence could be lost by physical internal/external influences, or functional internal/external influences.

Inherent fail-safety is considered if all non-negligible failure modes of a single item are nonhazardous. Inherent fail-safety can be used for certain functions (e.g. to enforce shutdown). Justification shall be provided for any failure mode which is considered to be incredible.

EN 50129 contains recommended architecture-related techniques for SIL 2: both “single electronic structure with self tests and supervision” and “single electronic structure based on reactive fail-safety” are valid alternatives, as well as “single electronic structure based on inherent fail-safety”. Moreover, the following techniques are highly recommended: (i) dynamic fault detection: on-line dynamic testing should be performed to check the proper operation of the safety-related system and provide an indication to the operator; (ii) program sequence monitoring: temporal or logical monitoring of the program sequence and indication of faults to the operator; (iii) measures against voltage breakdown, voltage variations, overvoltage, low voltage and temperature increase (to detect overtemperature).

The standard mentions, in the case of reactive fail-safety, the following fault detection techniques: (i) encoding, (ii) multiple computation and comparison, and (iii) continual testing.

2.2 Software requirements

EN 50128 recommends a set of techniques and measures to be implemented in the architecture design phase. The main proposed techniques related to fault detection and handling are summarized in Table 1. In case of SIL 2, one or more of these techniques shall be selected to satisfy the requirements.

Table 1. Recommended techniques for fault detection and handling

Recommended Techniques	Description
Defensive programming	Detect anomalous control flow, data flow or data values
Fault detection and diagnosis	Check for erroneous states based on a principle of redundancy and diversity on the physical, logical, functional or external levels
Error detecting codes	Detect errors in sensitive information by e.g. Hamming, cyclic or polynomial codes.
Failure assertion programming	Detect residual software design faults during the execution of a program by checking pre-conditions and post-conditions of operations
Safety bag	Protect against residual specification and implementation faults by an independent external monitor responsible for checking that the main system performs safe (not necessarily correct) actions and preventing that an unsafe state is entered
Diverse programming	Detect residual software design faults by using variants that were designed and implemented independently
Memorising executed cases	Force the system to a safe state if an unlicensed path (which is not registered as fault-free) is detected; the licensed paths are stored as a reference

3 System overview

As previously described, the SAFEDMI operates in an environment composed of (i) the *EVC*, (ii) the *train driver* (TD) and (iii) the *Maintenance Centre* (MC).

The SAFEDMI itself is composed of three elements: the DMI, the Bridge Device (BD) and the peripheral devices. The DMI is the core of the SAFEDMI. The DMI manages the following three communication activities: (i) it handles communications with the EVC, (ii) it handles communication with the BD, and (iii) it handles information I/O from/to the peripheral devices. DMI generates safe visual and audio outputs based on inputs received from the EVC, and communicates to the EVC the inputs received from the train driver. Moreover, it communicates with the MC using the BD as a wireless bridge to perform software update, configuration, or diagnostic operations. Note that communication between the DMI and the BD shall satisfy safety requirements of communication in closed networks [3].

The BD has a wireless interface and handles safe wireless communications in open networks, thanks to a Safety Layer that guarantees connection establishment, data transfer and connection release that fulfil the safety requirements for wireless communication stated in [4].

The BD features also an Ethernet interface to handle wired communications: in fact there is the possibility to connect manually to the BD using an Ethernet port in order to execute maintenance operations when wireless communication is not available. More information about BD and communication protocols (both towards DMI and MC) are provided in [11].

Peripheral devices are those SAFEDMI devices that allow interaction with the driver: these components are the LCD screen, the keyboard and the audio devices. Optionally the keyboard can be replaced by a touchscreen.

In Figure 1 the whole environment, the system elements and their interconnections are described.

4 SAFEDMI overall behaviour

The SAFEDMI has five operational modes: *start-up*, *configuration*, *normal*, *suspect* and *safe* mode. Figure 2 presents these operational modes and how they are interrelated. These modes originate mainly from the SAFEDMI functional requirements with the only addition of the "suspect" mode which has been inserted to increase mission availability.

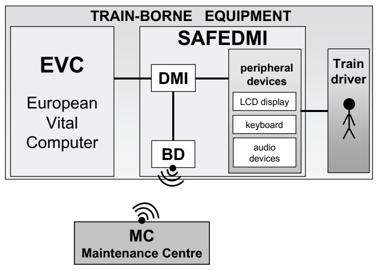

Figure 1. The SAFEDMI system and its environment

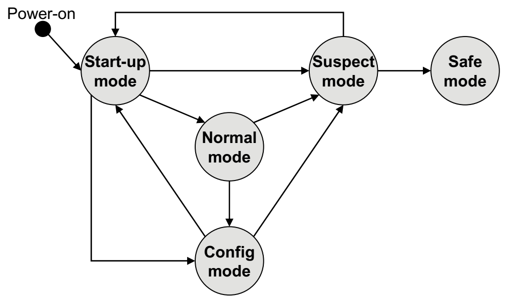

Figure 2. SAFEDMI operational modes

In *Start-up mode* the initialization procedure and the thorough testing of all devices are performed. Diagnostic functionality is also available. If a fault happens, a transition to the suspect mode is performed. Otherwise, the normal or configuration mode is entered, depending on the inputs received from the Maintenance Centre.

In *Configuration mode* DMI-specific safe software upload and installation (e.g., change of some DMI procedures) or software configuration (e.g., change of parameters as timeouts or periodic test frequency) are performed. These operations are carried out by means of wireless communication and while the train is at a standstill. After a successful configuration session, a restart of the SAFEDMI is needed.

In *Normal mode* the DMI produces graphical and audio information to support train driving, as well as it acquires drivers commands (i.e., it implements the functional interface between the driver and the EVC). Moreover, periodic testing activities are performed and a diagnostic functionality is available. Both in Normal and Configuration modes, whenever a fault is detected, a transition to the suspect mode is forced. Safety requirements include that the information delivered by the SAFEDMI to the other entities shall be safe: in case no safe information can be delivered then the SAFEDMI goes into a suspect mode.

Suspect mode is a transitory mode that allows to go back to start-up mode, to perform attempts to restart. Using a specific mechanism [1], [12], it is possible to set a maximum number of allowed restart attempts. Moreover, since a re-start of the DMI may take a few minutes, a maximum time to restart the DMI is set. This mechanism allows to increase the *availability* of train missions, since DMI can automatically restart from transient faults and become fully working again, thus avoiding that the EVC activates the brakes and interrupts the train mission. While DMI is restarting, the EVC fully handles the train; EVC tolerates a temporarily absence of the DMI and does not activate brakes only if critical information from the DMI are not required during the restarting period: consequently safety of the whole mission is not affected by this DMI behavior. Note that not every DMI fault is processed in this way: faults that can be immediately recognized to be non-transient force a transition to the safe mode.

Safe mode is entered when a malfunctioning is detected and attempts to restart the DMI have failed. This mode prevents further operations of the DMI. In this mode the LCD backlight lamps are switched off, and the keyboard and EVC communications are disabled. Only diagnostic activities are allowed by means of the wireless interface. When such a mode is reached the driver has to perform a failover procedure: it disconnects the failed DMI from the EVC and it manually connects the EVC to another spare DMI available on train cab.

5 Architecture design

In this section we describe the DMI hardware and software architecture, in terms of its COTS components.

5.1 DMI hardware architecture

Regarding the hardware, the SAFEDMI has a non-redundant COTS architecture. It has a permanent storage to store log information and a switch to cut off the communication towards the EVC and to switch off the LCD lamp whenever required. It has a thermometer to allow temperature monitoring. The DMI interacts with the EVC using a communication bus like a PROFIBUS or an RS485, while it communicates with the BD using an Ethernet or an RS232 channel.

The hardware architecture is kept simple, and composed of COTS components, thus reducing costs. Absence of hardware redundancy imposes a more sophisticated software architecture, since safety requirements need to be accomplished only by software mechanisms.

5.2 DMI software architecture

For each operational mode of the SAFEDMI (normal, start-up, configuration, suspect and safe mode), we defined a set of "modules", or "classes of objects". When the SAFEDMI transits from a mode to another, an object factory performs the following operations: (i) deactivates the modules that were used in the old mode; (ii) activates the modules that will be used in the new mode.

Note that even if the names of the modules in different modes are identical, they are considered as different entities. Instances of each module may be present or not and may differ in behaviour according to the particular mode in which the SAFEDMI is currently acting. The only exception is the Global Monitoring module (see below), which is always the same process (or set of processes) that possibly configures itself depending on the current mode. The software objects (or modules) identified in the SAFEDMI system are *Global Monitoring*, *Checks and Tests*, *Operations*, *Communications*, and *I/O Manager*.

The *Global Monitoring* is one of the main software components of the DMI. Satisfaction of safety requirements is strictly dependent on the behaviour of Global Monitoring. In fact, it performs the role of an execution monitor, a software watchdog, a log manager, and a diagnostic manager. It acts as a global scheduler, which handles all activities executed on the DMI, implements the scheduling rules, and allows or stops the execution of each activity. It recognizes the conditions that cause an operational mode change of the DMI and performs the operations needed to reconfigure the system when a mode change happens (e.g., apply a testing policy, perform an operation, suspend or restart a thread). At mode change, the global monitoring activates the mode change procedure, which contains a set of static tables containing rules for the deactivation and activation of the modules (these tables enlist the processes to kill, the new processes to start, and the necessary memory pointers). Moreover, the Global Monitoring checks that the configuration of the system is always the one expected in the current operational mode. As far as the safety related mechanisms are concerned, in this module both control flow monitoring and a software watchdog are implemented. Lastly, the

Global Monitoring provides diagnostic and logging services (e.g., logging faults and safety-critical conditions). Details on the safety related mechanisms of this component are presented in Section 6.

The *Checks and Tests* module has the role of performing checking and testing activities. It is subdivided into two sub-modules, the Tests module and the Checks module. The former module contains all the algorithms required to test the devices (e.g., CPU, RAM and video pages), and provides services to set the testing policy; the latter module implements the mechanism of data acceptance, credibility checks and comparison of multiple computations, and it carries out periodic checks (e.g., LCD backlight lamp and temperature checks).

The *Operations* module contains all the software objects involved in the implementation of the functional requirements of the operational modes: the most important ones are visualization procedures and audio emissions, conditions that trigger an operational mode change, procedures to perform an operational mode change. This module handles the interactions with the driver in terms of graphic display arrangements and audio signal emissions according to [6].

The *Communications* module handles the communications towards EVC and BD; it provides services for channel opening and closing, data packet sending and receiving and channel status checking. These services are positioned in the application layer of the communication protocol stack. Lower layers' communication protocols are described in [11].

The *I/O Manager* module contains all the drivers of the SAFEDMI hardware devices (e.g. LCD backlight lamps, audio device, keyboard, thermometer etc.).

Further details on these modules can be found in [10].

Regarding lower level software, the operating system is VRTX [8] (Versatile Real Time Executive), a scalable COTS OS with real-time features which allows to integrate application-specific system-level software (e.g., specific system call handlers and routines) with the kernel.

6 Fault detection mechanisms and the supporting technologies

According to the *reactive fail safety* principle adopted in the SAFEDMI project, the efficient fault detection plays a crucial role. Detected faults in the Normal and Config operational modes force a transition to the Suspect mode which then leads either to the Start-up mode or to the Safe mode. This kind of fault handling ensures safety in case of detected faults either by a proper retry mechanism or a by fail-stop behaviour, while the undetected faults contribute to hazards.

6.1 The applied fault detection techniques

According to the safety standard EN 50129, the potential hazardous effects of *systematic faults* (especially software design faults) are to be handled primarily by means of fault prevention and removal during development, i.e., the quality and safety management conditions corresponding to SIL 2. Fault detection addresses *random faults* that include both permanent and transient faults.

In the Start-up mode *permanent hardware faults* are addressed by efficient test procedures with high fault coverage (in this off-line mode non-intrusiveness and small overhead are not required). In detail, (i) RAM and flash memory are tested by march test algorithms [16] (similar tests are applied in the case of the Video Page); (ii) CPU is checked by a test suite addressing faults in the registers, internal buses, arithmetic and logic unit, and instruction decoding; (iii) ROM testing is based on checksum validation; (iv) testing of the peripheral devices (keyboard, LCD lamp, audio device) relies on the assistance of the driver.

In Normal and Config operational modes on-line (run-time) fault detection is applied.

For detecting latent *random permanent faults* in the hardware components periodic testing is performed. In these operational modes the execution time of the tests is an important issue, so that fast (but less efficient) versions of the test procedures applied in the Start-up mode were developed (eg., a simpler version [13] of *March C- test* [16] is applied). Considering passive objects (files, configuration data) an integrity checking procedure is adopted.

Active *permanent faults* and *random transient faults* in the hardware components are detected on the basis of their *effects on software execution*. Considering the recommendations of the safety standards (see Section 2) we divided the techniques into three main categories as follows: the techniques “defensive programming” applied to detect anomalous data flow or data values, “failure assertion programming”, “safety bag”, “encoding” and “error detection codes” belong to the general category of *data acceptance/credibility checks*, as they focus on the faults in intermediate and output data (as the results of the computation to be checked). Similarly, “defensive programming” applied to detect anomalous control flow, “program sequence monitoring” and “memorising executed traces” belong to the general category of *control flow monitoring*, as they focus only on the faults affecting the control flow of the computation. *Multiple computation and comparison* is a generic technique efficient in the case of both data and control related faults (moreover, it can be combined with diverse programming to detect software design faults as well).

Data acceptance/credibility checks are efficient in the case of data processing functions but only if we are able to assign acceptance criteria to intermediate or output data. Control flow monitoring is efficient in control-related functions. However, its coverage is limited, as the result of the computation could be erroneous even if the control flow itself was correct, while data acceptance checks could also detect if the computation was affected by a control flow error. Multiple computation and comparison is costly in terms of time and memory overhead, thus it is used only when the other techniques are not sufficient. Time-out checking is applied when timing constraints can be defined.

According to these observations and the related analysis of the peculiarities of the different SAFEDMI modules (functions), the fault detection techniques presented in Table 2 were selected.

Table 2. Error detection techniques assigned to main SAFEDMI functions

SAFEDMI function	**Error detection technique**	**Specific considerations**
Tests and Checks	Control flow monitoring	No acceptance criteria can be given, but the results (and the related hazard) are sensitive to the missing or incomplete execution.
Global Monitoring	Control flow monitoring and time-out checking	The function is control-oriented (global monitoring is responsible for starting and suspending threads).
I/O Manager	Duplicated execution and internal comparison	Internal processing is sensitive to data related transient faults in the CPU, and no data acceptance criteria can be given. A specific case of the duplicated execution is the explicit acknowledgement procedure in the EVC-DMI communication protocol.
Operations: Configuration	Data acceptance/credibility checks	Error detection codes are used and integrity checking is performed on the transmitted files.
Operations: Visualization	Duplicated execution and visual comparison by the driver	Internal processing is sensitive to data related transient faults in the CPU (e.g., resulting in the presentation of incorrect icons on the display), and no data acceptance criteria can be given.
Operations: Audio management	Control flow monitoring	Audio outputs are not of vital importance thus the output checking of the correct audio signal is not performed.
Operations: Mode change procedures	Control flow monitoring and time-out checking	The functions are control-oriented and no data acceptance criteria can be given.

6.2 Tool support for implementing fault detection

Data acceptance/credibility checks are application-specific fault detection techniques that shall be explicitly addressed in the design phase of the corresponding modules. Similarly, duplicated execution needs a specific comparison method that shall be carefully designed. On the contrary, control flow monitoring is a mechanism that can be applied independently of the actual functionality of the checked module: the reference control flow can be extracted from the source code of the module and a re-usable checker component (often called a *watchdog processor*, WP) can check whether the run-time control flow corresponds to the reference control flow [7]. To facilitate the comparison, compact identifiers (so-called *signatures*) are assigned to states of the program, and the run-time sequence of these signatures is transferred to the WP. The WP checks whether the signatures are regularly received and represent a valid control flow allowed by the reference one. Signatures can be assigned hierarchically, i.e., at the level of *processes* (to check sequencing and scheduling), *functions* (to check the function calling hierarchy), or *statements* (to check syntactically allowed transitions).

The instructions for signature transfer can be inserted into the program source either manually (which is a tedious task) or by a *source code preprocessor*. The preprocessor implements a parser of the input programming language, assigns the signatures to program states (depending on the selected hierarchy level) and finally generates both the instrumented source and the reference information for the WP. In the SAFEDMI project we develop a watchdog preprocessor tool that inserts the signature transfer instructions systematically into the source code of C programs.

Another technology we propose supports defensive programming by generating the source code of *executable assertions* (predicates that are evaluated run-time within the application to detect the violation of invariants or other application-specific requirements). We automatically generate the assertion code on the basis of *high-level specification* in the form of temporal logic formulas interpreted on UML statechart models [9].

7 Concluding remarks

In current railway train-borne equipment, the DMI is a component with no safety guarantees. Safety of information visualized to the train driver, and safety of driver commands are completely demanded to the EVC. In such a context, it appears evident that the EVC cannot rely on the DMI and needs to be highly suspicious on the DMI behaviour. That is, as soon as the DMI is suspected to misbehave, the EVC commands a train stop. In current railway environment, where train speeds are continuously increasing, such a situation is no more acceptable: a DMI with *safety requirements* is necessary.

In this paper we have detailed the architecture design of a SIL 2 DMI (SAFEDMI), as a result of the activities of the SAFEDMI European project. The main challenge of the architecture development is laid on the decision to guarantee safety requirements only by software, in order to have a simplified hardware architecture and to reduce development costs: consequently our DMI is characterized by a non-redundant COTS hardware, where safety requirements are achieved exclusively by software. We focused even on train mission reliability, introducing a new operational mode (we called it *suspect mode*) that allows to perform automatic restart of the DMI, without the necessity to stop the train. It has to be noted that no train-borne DMIs with SIL guarantees are currently available in literature (nor even in the market) as well as no train-borne DMIs are currently able to perform attempts to self-restart. Last, we have provided the SAFEDMI with the ability to communicate to a bridge device to perform safe (SIL 2) wireless communication for software update or configuration activities.

We have logged in a traceability matrix [10] that we have satisfied the functional and related safety requirements. Such a list provides evidence that we have analyzed and proposed solutions to all requirements, to easily check that no inconsistencies between different techniques and choices have appeared, and it can be used as a starting point for subsequent validation activities (validation and quantitative evaluation activities are scheduled as upcoming tasks of the SAFEDMI project).

8 Acknowledgments

This work has been partially supported by the European Community through the SAFEDMI Project (Contract n. 031413).

References

[1] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni. Threshold-based mechanisms to discriminate transient from intermittent faults. *IEEE Transactions on Computers*, 49(3):230–245, 2000.

[2] CENELEC. *EN 50128 - Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems*, 2001.

[3] CENELEC. *EN 50159-1 - Railway applications - Communication, signalling and processing systems - Part 1- Safety related communication in closed transmission systems*, 2001.

[4] CENELEC. *EN 50159-2 - Railway applications - Communication, signalling and processing systems - Part 2- Safety related communication in open transmission systems*, 2001.

[5] CENELEC. *EN 50129 - Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling*, 2003.

[6] CENELEC. *CLC-TS 50459 - Railways Applications - Communication, signalling and processing systems - European Rail Traffic management system - Driver Machine Interface - Part 1, 2, 3, 4, 5 and 6*, 2005.

[7] A. Mahmood and E. J. McCluskey. Concurrent error detection using watchdog processors-a survey. *IEEE Trans. Comput.*, 37(2):160–174, 1988.

[8] Microtec. *VRTXsa Real-Time Kernel. Programmers Guide and Reference.* San Jose, CA.

[9] G. Pinter and I. Majzik. Automatic generation of executable assertions for runtime checking temporal requirements. In *HASE '05: Proceedings of the Ninth IEEE International Symposium on High-Assurance Systems Engineering*, pages 111–120, Washington, DC, USA, 2005. IEEE Computer Society.

[10] SAFEDMI. *Deliverable D2.1 - Architecture and Related Mechanisms*, 2007.

[11] SAFEDMI. *Deliverable D3.1 - Wireless communication architecture and protocols, Preliminary Version*, 2007.

[12] M. Serafini, A. Bondavalli, and N. Suri. Online diagnosis and recovery: On the choice and impact of tuning parameters. *IEEE Trans. on Dependable and Secure Computing*, 4(4):295–312, 2007.

[13] S. M. Thatte and J. A. Abraham. Test generation for microprocessors. *IEEE Trans. Comput.*, 29(6):429–441, 1980.

[14] UIC. *Subset 033 -rev. 2.0.0 - ERTMS-ETCS Class 1 - FIS for the Man-Machine Interface*, 2000.

[15] UIC. *Subset-026 - rev.2.2.2 - ERTMS-ETCS Class 1 - System requirements*, 2002.

[16] A. J. van de Goor. *Testing semiconductor memories: theory and practice*. John Wiley & Sons, Inc., New York, NY, USA, 1991.