A fault tolerant architecture for brake-by-wire in railway cars

ROGER JOHANSSON

CHALMERS LINDHOLMEN UNIVERSITY COLLEGE
Göteborg, Sweden 2003

Report no.15

REPORT No.15

A fault tolerant architecture for brake-by-wire in railway cars

ROGER JOHANSSON[1]

Department of Electrical and Computer Engineering
Chalmers Lindholmen University College
roger@chl.chalmers.se

Chalmers Lindholmen University College

Göteborg, Sweden, December 2003

[1] This work has been conducted within the Centre of Excellence CHARMEC (CHAlmers Railway MEChanics) – a VINNOVA Competence Center under the “Programme area 4, SD3. Computer control of braking systems for freight trains”

A fault tolerant architecture for brake-by-wire in railway cars

ROGER JOHANSSON

Report – A fault tolerant architecture for brake-by-wire in railway cars
Report no. 15
ISSN 1404-5001

Chalmers Lindholmen University College
P.O. Box 8873
SE-402 72 Göteborg
Sweden
Telephone + 46 31 772 1000

Chalmers Lindholmen University College
Göteborg, Sweden, December 2003

Abstract

In this paper we will present a computer architecture suitable for distributed control systems where fault tolerance is desired. Today these are commonly referred to as "brake by wire" or "steer by wire" -systems. The architecture is designed for implementation mainly with standard components "off the shelf" (COTS). In particular there is only a comparable small device called FTCC (Fault Tolerant Communication Control) that requires extensive redundancy.

The FTCC is used to close control loops as tight to the controlled physical device as possible, gaining from the excess computing capacity that a distributed system offers but at the same time remove impact of increased fault intensity from an increased number of processing elements.

The architecture preferences applications where there is some kind of natural, inherent, redundancy. As a starting point, and a case, we consider a state of the art brake control system for railway vehicles. We recapture common computer architectures designed to handle safety critical applications and arrive at a feasible solution in the shape of a slightly modified distributed architecture. We then apply this revised distributed architecture and describe a revised brake control system.

The FTCC device has been implemented, however without redundancy, with standard VHDL-tools and tested in a simulator environment. Results are promising and indicates that the FTCC-device has a great potential in future "control-by-wire" designs.

Keywords:
Control by Wire, Inherent redundancy, Fault Tolerance, Hard Real-Time requirements

1. INTRODUCTION

For the last couple of decades, computers have been used for control of railway signals and for many years they have also been used on board trains. Railway signals and automatic train control place extreme demands on safety. Applications on board trains have not yet been subjected to corresponding demands, but as more and more functions are computerised, the need for reliability will increase. One example is train brakes. Today's method of braking a train set has proven highly reliable and served well for nearly a hundred years. However, this is a good case for where computers can bring increased performance and functionality. A distributed computer system can give shorter response times and better means of controlling braking processes than pneumatic systems. A computer controlled brake system, sometimes known as a "brake-by-wire" system will provide major advantages such as; Reduced wear out of wheels and rail due to more adaptive and economical brake functions, less mechanical/hydraulic/pneumatic components due to replacement of electronic parts, at the same time it will be possible to optimise maintenance intervals thanks to diagnostic functionality exercised by software.

The important issue when implementing safety critical functionality, such as a vehicle car brake function, is how to achieve a satisfactory level of safety and still using today's technology. In this paper we will present a conceptual computer architecture that is suitable for distributed control systems where fault tolerance is required. As a special important case we consider brake systems for rail vehicles such as lightweight trains and trams. This architecture can be easily implemented with standard components with a few extensions. The architecture has been primarily targeted for an intelligent brake system adapted for railway vehicles but should work for other control applications as well.

In particular the conceptual architecture is intended for applications where we might extract natural redundancy from the application. As examples of generalisation of the architecture we find an aircraft which might be controlled even in the absence of one control surface provided that the faulty surface has entered a fail-safe state. Another example is found in automotives, consider a car with four wheel steering, it can obviously be satisfactory controlled even in the case that one wheel steering fails again though provided that the faulty wheel has entered a fail safe state.

The paper is organised as follows; the remainder of this chapter is devoted to a short background and a brief review of related work. We will as an example describe a state of the art electromechanical brake system manufactured by SAB WABCO. We will summarize essential requirements on such a system. In chapter two we will describe conceptual fault tolerant designs known from a broad range of safety critical applications. We will then suggest a new architecture design with a minimum of impact on the current solution. The major invention with this new architecture is the introduction of a Fault Tolerant Control and Communication unit (FTCC). In chapter three we detail the FTCC and apply it within our revised architecture, on our generic brake system as presented in this chapter. Finally, chapter four offers discussions and conclusions.

1.1 Brake system functions

The brake system provides several functions. Basic requirements depend on the vehicle type, i.e. demands on freight car brake systems differ significantly from demands on a Light Rail Vehicle (LRV) or tram. Although some parts of the following discussion do not really apply to freight cars, we do treat them here in order to keep our discussion as general as possible.

Service braking is frequently used by the driver in controlling the train. Service braking shall achieve specified levels of performance at any time.

Emergency braking may be initiated by the driver or, in some cases, even by a passenger in the case of extreme hazards. Emergency braking shall achieve a specified level of performance and a high level of integrity.

Security braking is a particular form of braking. Security braking is activated in the case of system failure within the ordinary brake system, thus it is a redundant back-up system. The security brake system is designed to apply maximum forces, so as to stop the car within the shortest possible

distance. Security brake systems are often implemented using mechanical or electro-mechanical devices. A major advantage with such solutions is the high reliability accomplished with these devices. The serious drawbacks are the large size and high costs associated with this back up. From the contents, it is obvious that security braking shall achieve a high level of integrity.

Holding brake maintains speed during downhill movements. The term is sometimes also used to denote a short duration brake used for ensuring against moving a vehicle once stationary, e.g. for un-load and load of passengers.

Parking brake should be able to hold a defined load on a defined gradient for an infinite time. It is intended for use while the train is stabled. The parking brake should be designed to ensure that it will automatically secure the train in the event of loss of emergency or service brake. On newly designed trains, the parking brake should apply automatically to ensure no movement of the train.

Wheel slide protection is fitted to optimize braking performance and to provide protection against wheel set damage e.g. during braking in poor adhesion conditions. Such systems are furthermore designed to minimize the braking force so as to achieve minimal practical stopping distance.

1.2 The SWEBC10 system design

The SAB WABCO "SWEBC10" is an electromechanical brake system for 4 axle vehicles. SWEBC10 is fault tolerant with *Fail Safe* properties. I.e. upon a system failure as signalled by the *Safety signal* line, or a failure in the vehicle *Brake Control Unit* (BCU), the safety loop will be asserted (meaning the current loop will be braked). This information will reach all brake actuators almost simultaneously and make them apply maximal pressure to activate the brakes. Thus the *Fail Safe* state is when maximum brake forces are applied to all axles (Figure 1.1).

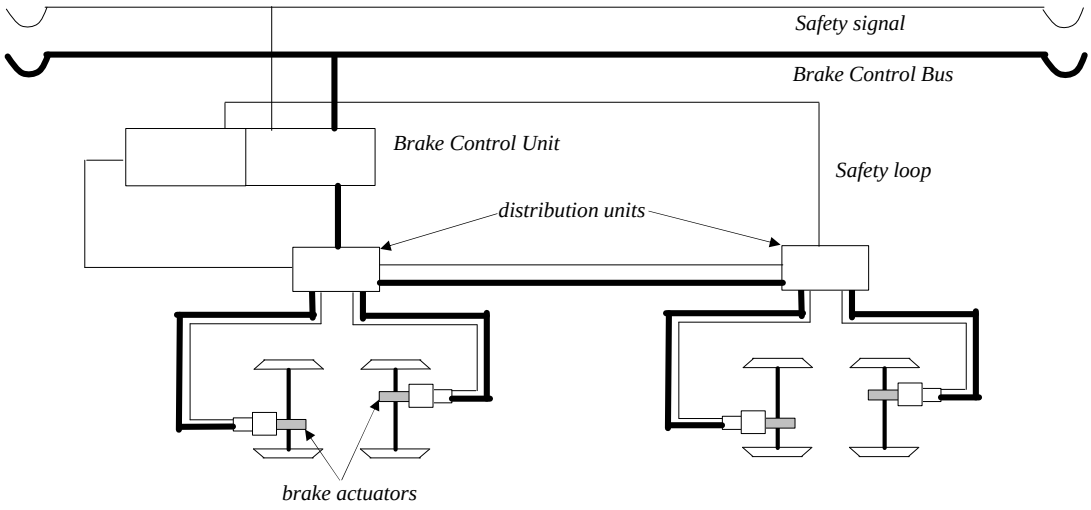

Figure 1.1: Electromechanical brake system for 4-axle vehicle (SAB WABCO)

1.3 The brake actuator

The physical brake actuator in SWEBC10 (see Figure 1.2 below), *Electronic Brake Control* (EBC) is an electro-mechanical device which controls a yoke, the device that finally applies pressure to the brake disc. The unit utilises two separate motors (M), the primary, which is used to control the yoke (apply and release mechanical force), the secondary which is used to conserve mechanical energy in a spring (S). In the case of a failure, the spring force is released through mechanical couplings (C). This force is sufficient to apply maximum pressure at the brake discs. During normal operation the primary motor is operated from the CPU which communicates with the brake control unit via a serial communication bus.

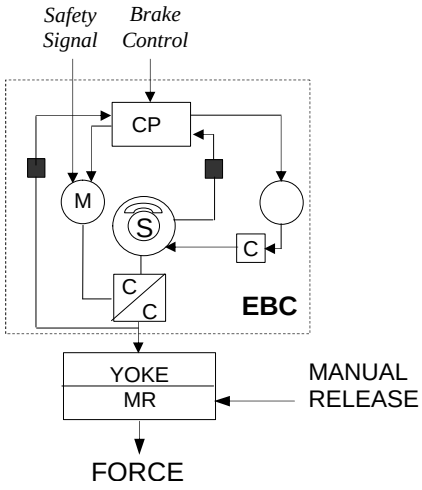

Figure 1.2: EBC Brake Actuator

1.4 Scope and objectives

The control of a train set dynamics may be described as a hierarchy of control levels (Figure 1.3). In this paper we are mainly concerned with a computer based brake *control* system. Our conceptual brake control in mind is intended for by-wire control applications. By this we mean that we do not anticipate any non-electronic information carrying components. We further assume very high safety and reliability requirements and we strongly emphasis the economy (production and maintenance) aspects. We do not treat physical actuators in this context. We do assume that a vehicle is equipped with more than one physical actuator. In fact, the key point is to design a brake control, built mostly around standard components (COTS) and still provide adequate dependability.

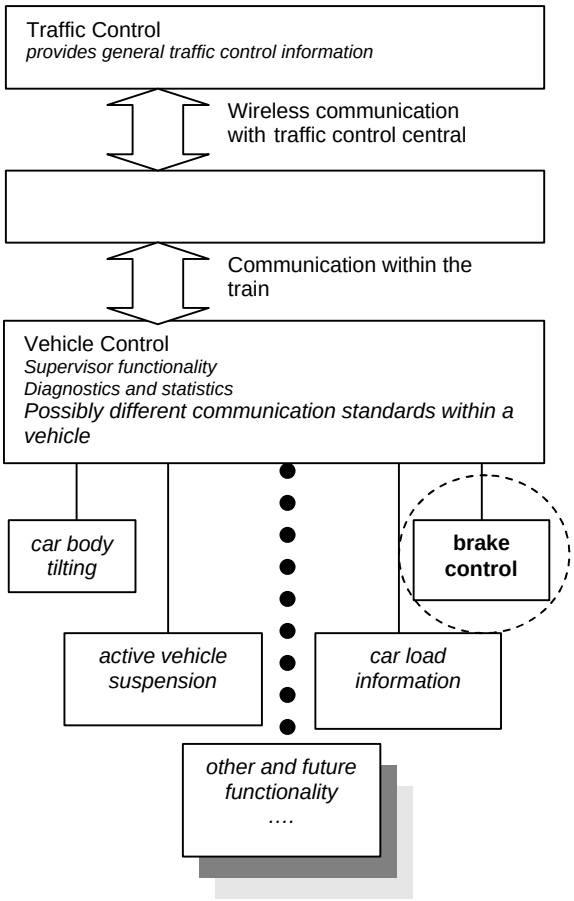

Figure 1.3: Control levels

1.5 Requirements on the redesigned system

A brake control system is a safety critical application. System failures may introduce hazards with accidents and severe damage and human injuries as consequences. Above all, dependability requirements apply. See for example [1,2] for detailed discussions of general requirements on computer based safety critical control applications.

A train brake control system should provide several functions. See [3,4] for requirements on functionality.

The use of electronic devices for safety critical train applications are subject to standardisation, see for example [5,6,7].

Now, turning to a potential manufacturer's point of view we might identify following requirements:

Weight and volume; units should be easily adaptable to different types of bogies. This normally means that they should be small, compact and low weight.

Maintainability; units should be easy to install and configure (deliverance). They should be easy to repair during rest of their life cycle.

Environmental; unit should be design to have lowest possible negative environmental impacts.

Economy; units are highly cost sensitive. Unlike, for example, automotive applications they do not manufacture in high volume. On the other hand they are likely to exhibit a longer life cycle.

The requirement analysis will expose some key design issues:

- Economy, the redesigned system should be less expensive to manufacture (production and test), deliver (install and configure). Product diversion (variants) should be achieved by the diversity of software modules. As a trade off, it becomes attractive to design the system with standard components as far as possible. This will cut both development and production costs.
- Maintainability, the redesigned system shall provide extensive support for diagnose facilities. These facilities shall be designed in a way that they ease and simplify diagnose, optimises maintenance intervals and provide detailed information on parts needing to be replaced or repaired. As a trade off, the system relies heavily on well designed software and adequate sensors to establish the units overall condition as well as individual key parts conditions.

1.6 Related work

The subject of distributed control has been studied for many years. Several problem areas has been identified and substantial efforts has been undertaken in studies within some of the areas. In particular, system dependability and safety assessment has gained focus [8,9,10,11].

Fault-tolerant computing addressing train transportation at high velocities such as the Japanese *Shinkansen* is treated in [12]. A discussion of French high speed train *SACEM* can be found in [13].

Brake-by-wire systems for railway vehicles are predicted in [14]. Deceleration can be accomplished without any blocks or discs. A recent study [15] shows promising within the application of pure electric braking systems. Economy and maintainability in computer controlled brake systems are addressed in [16] and [17].

Brake functionality is distributed, i.e. a single railway car is equipped with several devices implementing the physical brake function Contemporary proposed computer architectures for avionics and automotive electronics control by-wire systems are distributed [18,19,20,21,22,23].

2. CONCEPTUAL ARCHITECTURES

We now turn to a discussion of a suitable architecture for the revised implementation of a computer based brake control. The architecture should address all initial requirements on functionality and dependability but now we also emphasis *economy*. Our architecture of choice should further support a true brake by-wire system, i.e., a final system that can be implemented without use of any mechanical information system or expensive mechanical backup-systems. In the final system mechanical or pneumatic devices should be used only to distribute the forces required to apply or release pressure at the brake discs.

We will recapture three conceptually different architectures for potential use in safety critical applications. We start by describing the architectures and discuss them from our new point of view. As we find that no one of the discussed architectures meets our requirements on dependability and economy at the same time, we finally propose a new conceptual architecture we believe will comply.

2.1 Prerequisites and assumptions

In our model, the brake control system interfaces to:

- The supervisor, a command system that supplies the brake control system with information such as:
  - Configure,
  - Diagnose, demand for various system status such as actuators condition, operating conditions and the need for maintenance.
  - Required speed, demand for maximum vehicle speed.
  - Environmental changes, for example, the vehicle is about to enter an environment where appropriate actions undertaken upon failure should change.
- Physical actuators; in our model we assume reliable actuators that takes as input a digital value and apply disc pressure proportional to this value.

In other words, the brake control system should transform trusted instructions from the supervisor to trusted values for the actuators and safely deliver these values to the actuators.

2.2 N-modular redundant

In centralised control solution computer architectures all processing units, memory and IO-peripherals are housed in a few cabinets located near by each other. Several buses connect the units and fault tolerance is achieved by means of hardware and/or software redundancy. Sensors and actuators are connected directly to IO-peripherals in the system. Sensors are normally duplicated for reliability and actuators with fail-safe properties must be used to prevent hazards in the case of a fault in the actuator.

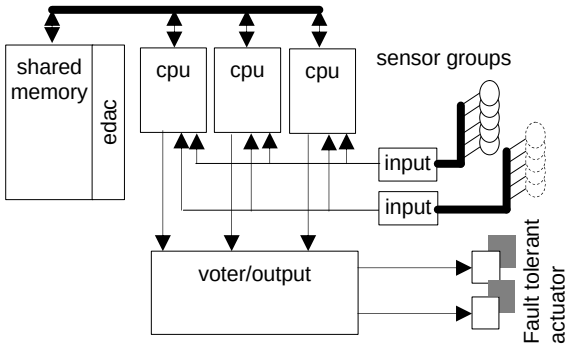

Figure 2.1: TMR control system architecture

Figure 2.1 illustrates a TMR (*Triple Modular Redundancy*) system with redundant input and sensor group as well as a redundant fail-safe actuator (fault tolerant). A single voter is used to determine a majority vote for the value that should be sent to the actuator. Each processor computes the same output, possibly by use of diverse software. The processors may use a shared memory with additional EDAC (*Error Detection and Correction*) logic for processor/memory communication.

The centralised solution has been used for space and avionic application for many years and has been proven highly reliable in a number of missions. A recent design is the primary flight control computer of the Boeing 777 airplanes fly-by-wire system [24]. Some of the requirements for the flight control computer include; No single fault of any kind should cause degradation below the minimum configuration to meet requirements; no single fault should result in the transmission of erroneous outputs without a failure indication; fully automatic redundancy management; Mean-Time-Between-Maintenance-Action of 25,000 hours assuming 13,6 operating hours per day.

The major drawbacks with NMR-architectures are their large complexity and the huge costs for developing these kinds of systems. Massive redundancy also contributes to high unit costs. Therefore they are not realistic as solutions in a brake-by wire system for rail vehicles.

2.3 Master Slave Solution

A *Master/Slave*, or *duplex* system has two computers providing the same functionality. They might have an error detection capability which creates output to an executive unit which chooses the computational result from either the Master or the Slave. As an alternative, an operator (human) can act as executive and switch from Master to Slave upon detection of a computer failure.

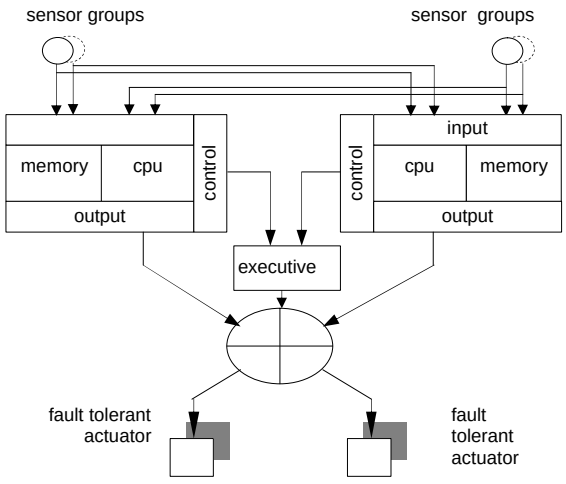

Figure 2.2: Master/Slave control system architecture

The European aircraft Airbus 340 flight control system is comprised of five computers: three primary computers and two secondary computers. The primary computers employ Intel 80386 processors while the secondary computers employ Intel 80186 processors. The computers use diverse software and hardware implementations. One of the primary computers is active, another standing by prepared to take over and the third computer is spare. Thus, in effect there are four dissimilar computers, with four software packages used to perform the flight control function when one would functionally suffice [25].

A Master/Slave-solution was also designed into the European space shuttle Hermes [26] however this vehicle was never taken into service.

2.4 Distributed Solution

By adding a general communication interface/bus the control computer processing may be distributed among several, more or less, independent computers (nodes). The nodes may be identical, or there may be nodes of varying types according to performance, size, power consumption, cost etc. A system may be distributed for a number of reasons.

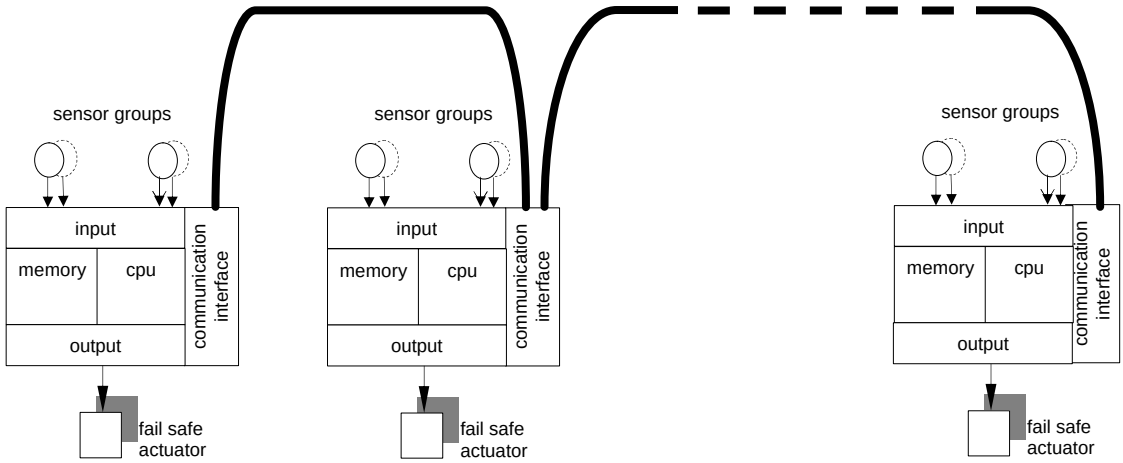

Figure 2.3: Distributed control system architecture

Distributed solutions are attractive in systems where sensors and actuators are distributed because they reduce cabling. A node can be implemented with a single microcontroller which makes efficiency in size, power consumption and price. As new nodes are added to the system the computational capacity will be linearly increased, the added capacity can be used to increase system throughput or as redundant (spare) capacity. The latter thus makes provision for fault-tolerance. However a distributed solution can be vulnerable since it might expose single point of failures; a malfunctioning communication unit or microprocessor may cause a node failure. A crucial bottleneck is the communication bus and the attached controllers. Any communication system introduce delays which might be susceptible to violate real-time constraints. There are only a few protocols capable of handling hard real-time requirements [27] and thus, despite the potential, distributed control systems are rare in safety-critical applications.

2.5 Trade off

We believe that both *NMR* and *Master/Slave* solutions are less suitable for a new brake control design. If anyone of them were chosen as a centralised brake control unit then the result would be a system that differs only slightly from the current system and it would probably not be less expensive than the current system, we cannot gain from such solutions. At the same time, it is not very attractive to equip each actuator with a separate *NMR* or *Master/Slave* design. This would probably be far too expensive.

In a distributed brake control unit we find several attractive features. The hardware in each node can be made identical. Each node, in the distributed control, could easily be integrated with a single actuator and there is no need for a separate *Brake Control Unit* with accompanying *Distribution Units* as in the SWEBC10. The resulting system should be less expensive to manufacture and easier to maintain. Thus, a distributed solution is well suited to meet economical requirements, however we has to make the final design fault tolerant to fulfil reliability and safety requirements.

A vehicle is always equipped with several actuators. Thus, by choosing a hardware distributed solution we already have the basic redundant configuration required to build a fault tolerant system. The question now becomes how to take full advantage of the natural redundancy in the distributed system at minimal costs? We now arrive at our proposal of the fault tolerant distributed computer architecture (See figure 2.4).

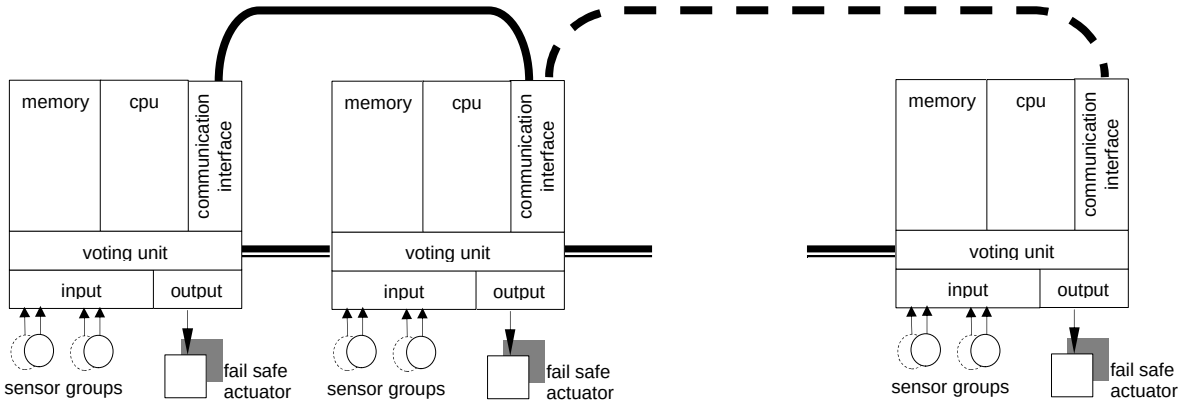

Figure 2.4: Distributed control system architecture with Fail Operational actuator control and sensor data acquisition

The main idea is to add a small "voting unit" between the microprocessor and the physical actuator(s) and sensors in each actuator device. The new units communicate with each other distributing computed values for actuators as well as sensor values for the physical actuators. This unit also performs voting before writing a final value to the physical actuator.

The resulting architecture could provide *N*-modular redundancy, where *N* is the number of actuators mounted in a single vehicle.

At this point we should realize some basic assumptions about the configuration.

The microprocessor/memory and communication unit(s) is considered far more complex than the voting unit.

A highly complex electronic device is much more likely to fail than a simpler one, due to its higher fault intensity.

If these assumptions hold, we should realize the 'voting unit' in hardware and add fault tolerance to the unit. Otherwise we would prefer to realize the unit in software which would be a much less expensive solution.

3. REFINING A DISTRIBUTED ARCHITECTURE

In this chapter we will apply a slightly modified distributed architecture by adding a small unit we call *Fault Tolerant Control and Communication unit* (FTCC). The primary purpose with this unit is to make it possible to take full advantage of application inherent redundancy and to provide adaptive fail safe states from the rail vehicle brake system.

In the railway car model each bogie has two stiff axles with a brake actuator attached to each axle. A fault tolerant actuator configuration i.e. a vehicle brake system should have at least four physical actuators, which are able to independently apply pressures to the brake disks. One or more brake systems are connected to a vehicle control computer also referred to as "supervisor". A vehicle control computer operates the brake system via the *vehicle bus*.

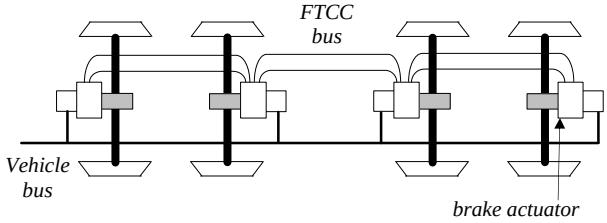

Figure 3.1: Railway car model

The brake system is organized in four physical brake actuator units where each unit is connected to all other units via the FTCC bus forming a cluster of brake actuators. Furthermore each unit communicates directly with the supervisory vehicle control computer via the vehicle bus.

3.1 Physical actuator organisation

An actuator is organized in three significant modules; *Brake Intelligence Module* (BIM), *Actuator/Sensor Control Unit* (ASCU) and *Fault Tolerant Control and Communication unit* (FTCC).

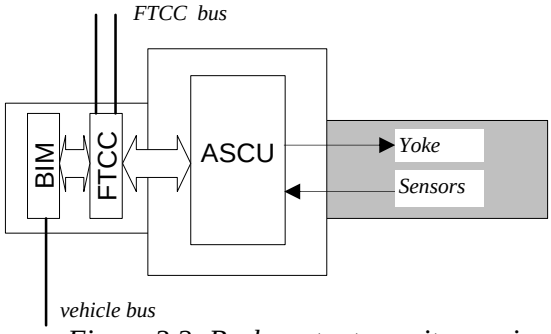

Figure 3.2: Brake actuator unit overview

The *Brake Intelligence Module* (BIM) can be thought of as a conventional embedded system (microprocessor, memory, communication units and ports). The software is designed to achieve optimal brake functionality, for example, compensate for worn discs, implement wheel slide protection etc. BIM should also be able to supply diagnostic information about the physical brake system to the vehicle control computer. Such diagnostics could be used to optimize the maintenance of the brakes. BIM thus handles all communication with the supervising vehicle computer and carries out all data processing needed to transform commands from the vehicle computer into appropriate instructions (apply/release force etc.). The calculated result, typically a binary value within the range of 12-16 bits, is then supplied to the FTCC by means of a register set: *Write*, where values may be written into FTCC state memory, *Read* where values from FTCC state memory may be read, and *Control*, where the watchdog function and fail safe states are initiated. The BIM is assumed to be far more complex than any of the other modules in the physical actuator. It is expected to have at least a magnitude of ten lower reliability. The module is not generally required to be fault tolerant to accomplish a safe brake system. There are however common mode software failures that could be fatal. Software fault-tolerance is not within the scope of this paper and will not be considered.

The *Actuator/Sensor Control Unit* (ASCU) is used to transform a digital value (brake pressure information) into mechanical force and apply pressure to the disc, thus accomplishing the actual brake function. The ASCU also provides sensor values for real applied pressure as well as real axle rotational speed. Obviously this is one of the most critical parts in the entire system and it must be extremely safe. However a detailed discussion of the ASCU's implementation is not within the scope of this work, we therefore assume a perfect ASCU and leave these discussions as further work.

The FTCC, is used to distribute critical data among all physical actuators in the cluster. This module also supplies final values to the ASCU's in each actuator. The modules organization and function will be thoroughly detailed below.

3.2 State memory interface

A BIM interfaces to FTCC through two sets of registers. One register set is used to verify functionality and establish FTCC mode of operation (Control Registers). Another set is used to communicate data via FTCC *state memory* to other nodes in the network (Data Registers). The actual number of data registers is configuration (application) dependent. The general idea is that each actuator and each sensor is allocated a matrix in FTCC state memory. The matrix is of size *N***N*, where *N* is the number of nodes in the network. BIM continuously computes values for *all physical actuators* in the system (regardless of which node the actuator is attached to) writes the values to FTCC state memory via the data registers. Then, the FTCC is responsible for distributing all values in its state memory to all other nodes in the network. Each entry (value) emerging from remote nodes in

a matrix has a corresponding *update bit*, which is set and cleared by FTCC according to message transmission status.

Actuator A	Node 1	Node 2	.	Node *i*	.	Node *N*
Node 1	not used			Node 1's computed value for actuator A in Node *i.*		Node 1's computed value for actuator A in Node *N.*
Node 2	Node 2's computed value for actuator A in Node 1.	not used		Node 2's computed value for actuator A in Node ***i.***		Node 2's computed value for actuator A in Node ***N.***
…..						
Node *i*	Node *i*'s computed value for actuator A in Node 1.			not used		Node *i*'s computed value for actuator A in Node *N*.
…..						
Node *N*	Node *N*'s computed value for actuator A in Node 1.			Node *N*'s computed value for actuator A in Node ***i.***		not used

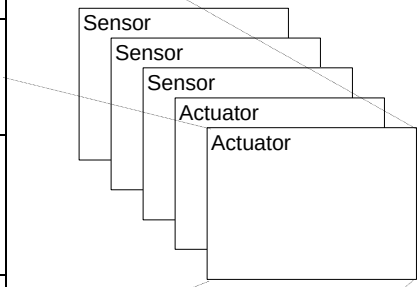

Figure 3.3: State memory representation of sensor and actuator values

Figure 3.3 illustrates how a row represents all actuator A's values computed in a node while a column represents actuator A values as computed by all nodes. Each location in all matrixes, except from those marked 'not used' is accessible from BIM via the FTCC data registers. The FTCC control registers are used for three purposes.

1) To configure FTCC matrixes when powering on the device.
2) Implement a watchdog function to tell FTCC that BIM is working properly.
3) Define the current *Fail Safe Mode* (further discussed below). It is desirable to allow different actions to be taken by the FTCC upon fatal failures. Application software is responsible for setting the appropriate Fail Safe Mode depending on system state and environmental conditions.

The use of data and control registers will be further discussed below.

3.3 FTCC operation

In this paragraph we begin with describing FTCC operation during normal conditions. We then describe the alternate modes, which are used to handle exceptional (error) conditions.

FTCC is a small unit located between the BIM and the actuator in each node. The primary purposes with FTCC are:

- Supply value(s) to the ASCU.
- Gather sensor(s) data from the ASCU.
- Communicate results from the node to all other nodes in the brake actuator cluster.

There are three essential parts in the FTCC:

- Communication unit, a time triggered communication subsystem used to interchange data among all FTCC's in the system. Since the FTCC should be implemented fault-tolerant, the communication subsystem utilizes a two separate buses.
- The voter, performs majority voting before a value is distributed to the physical actuator.

- The State Memory is used to hold data from all sensors and for all actuators from all nodes in the system.

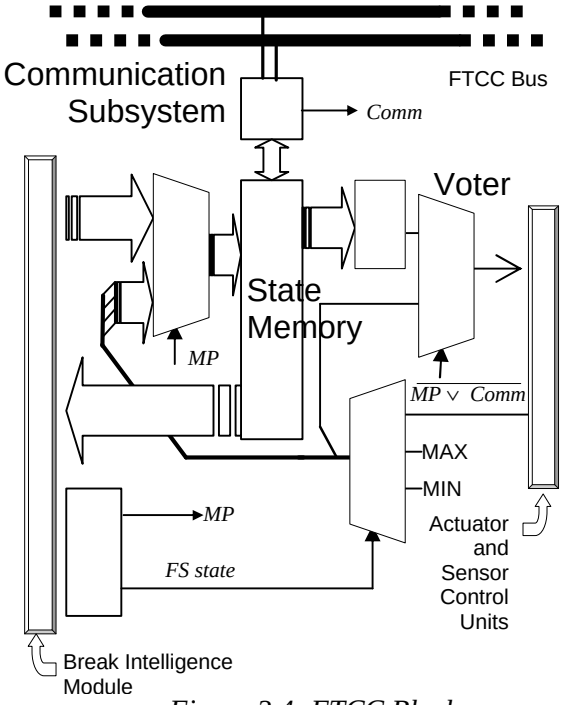

Figure 3.4: FTCC Blocks

The communication subsystem uses a TDMA (*Time Division Multiple Access*) protocol to distribute data among the FTCC's. During a TDMA-round (communication cycle) each node transmits it's sensor and actuator values. Thus, upon the completion of a communication cycle, all sensor/actuator values has been updated to reflect the latest samples.

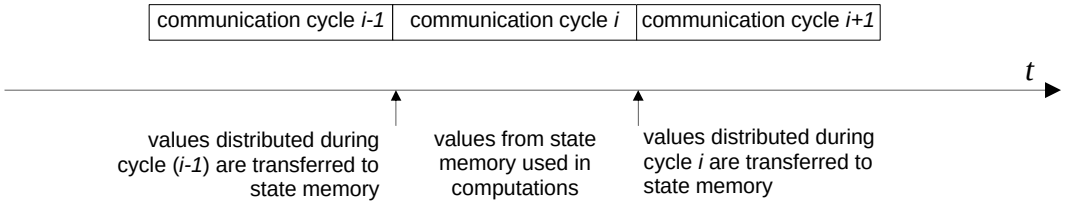

Figure 3.5: Static Bus Communication Schedule in FTCC

Three signals are used to determine the FTCC mode of operation:

- *Comm*, (Communication). If at least one nodes messages were received correctly this signal is set to TRUE upon the completion of a communication cycle. The signal is set to FALSE if the unit entirely failed to execute the communication protocol. If this signal is FALSE then the FTCC assumes a faulty communication subsystem and will take appropriate actions. Since this signal is checked every communication cycle reintegration will be attempted immediately and, in the simplest case, a transient fault with short duration, reintegration will take place already in the next cycle.
- *MP*, (Micro Processor). The signal is set to TRUE and a watchdog is initiated each time the BIM executes a *Watchdog Refresh* protocol through the control registers. If the BIM fails to comply with the protocol then the FTCC assumes a faulty BIM and will take appropriate actions. This signal is checked at the end of each cycle so the BIM is allowed to try reintegrating immediately upon a failure.

- *FS State*, this signal is accomplished by the BIM executing a *Set Fail Safe State* protocol. Fail safe state may be set to one of:
    - MAX, use maximum value according to representation.
    - MIN, use minimum value according to representation
    - SENS, use value(s) obtained from sensor(s).

The FTCC communication cycle is divide into thee phases (Figure 3.6). Applications synchronise with these phases by checking a status register in the FTCC.

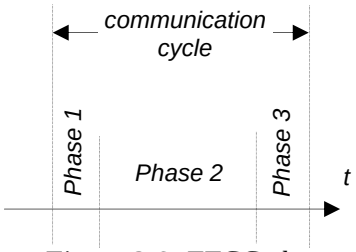

Figure 3.6: FTCC phases

***Phase 1*:**

FTCC:Synchronising bus. Actuator values as determined during previous cycle is transferred to, and sensor values are sampled from, the nodes ASCU. The new sensor values will be propagated into state memory during phase 3. State machines are checked. If the *Watchdog Refresh* protocol has been successfully executed MP is set TRUE, otherwise FALSE. State memory is accessible (readable) via data registers.

Application:

Read distributed sensor and actuator values from state memory, i.e. data from previous cycle is made available for calculations.

***Phase 2*:**

FTCC: Bus communication, message exchange with all nodes in the cluster takes place during phase 2. For each successfully received message an update bit is set and the message is transferred to it's propriate position in state memory. FTCC data registers should not be accessed during this phase, any write will be ignored and any read may return unconsistent data since the state memory is updated during this phase. FTCC control registers are writeable.

Update bits for values from remote nodes are cleared.

Application:

During this phase application should do control loop calculations, error checks and so on based on values from the previous cycle.

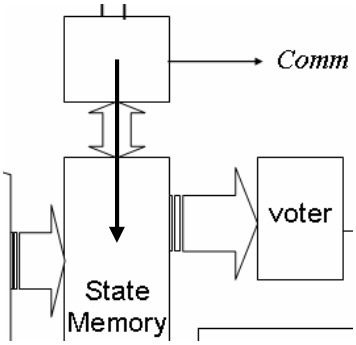

Figure 3.7: Data transfer to state memory during phase 2

***Phase 3*:**

For each successfully received message the corresponding values update bits now has been set. If no message was recieived correctly, *Comm* is set to FALSE otherwise it is set to TRUE. Fail Safe state is latched from the last executed *Set Fail Safe State* protocol. If the BIM has complied to update the watchdog (*MP* is TRUE) then this value is fetched from BIM interface registers and transferred into state memory for distribution during the next cycle.

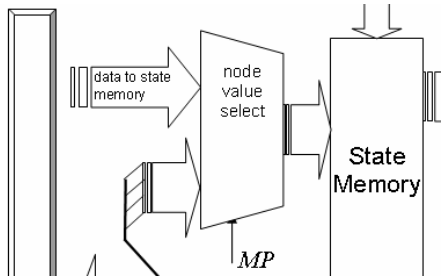

Figure 3.8: Data transfer from BIM to state memory

In situations where either or both *MP* and *Comm* are false alternate data flows will occur. This will be described in the next paragraphs.

Finally, during phase 3, voting takes place. A weitghted majority decision is made based upon values in the state memory, e.g. a value who's update bit is zero has lower weight than a recently refreshed value. This value is passed to a multiplexer *node value select*. This selector disqualifies the value if (and only if) *both* the *MP* and *Comm* signals are FALSE. In this case, the new value is chosen based on the current *fail safe state* selection, i.e. apply minimum value, apply maximum value or apply current value as obtained from a sensor in the brake unit.

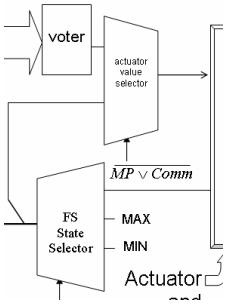

Figure 3.9: If either MP or Comm is TRUE, value from voter is used

In situations where either or both *MP* and *Comm* are false alternate data flows will occur, or more formally, the FTCC will change mode of operation. Figure 3.10 shows the possible nodes and the next paragraphs are devoted to a more detailed description of the different modes.

MP	Comm	Mode
TRUE	TRUE	Normal
TRUE	FALSE	Local
FALSE	TRUE	Neighbour
FALSE	FALSE	Stand Alone

Figure 3.10: FTCC operating modes

3.3.1 Normal Mode

A *Normal Mode cycle* is started when the status signals *MP* and *Comm* generated by the previous cycle both were set to TRUE. Data from the nodes BIM as well as data from all other nodes in the cluster which was stored in State Memory during the previous cycle is passed to the majority voter and then via the actuator value selector to the actuator.

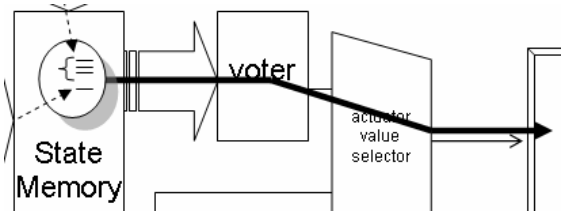

Figure 3.11: Data flow in Normal Mode

Since no errors were detected during the last communication cycle the system now acts as an NMR-configuration, data computed in all actuators are used in a majority decision of each actuator value. This implies that software should be designed so as to compute values for all actuators making up the cluster.

3.3.2 Locally controlled

The *locally controlled mode* is entered when bus communication during the previous cycle failed to communicate data via the FTCC bus (*Comm* is FALSE) but the local microprocessor is working properly (*MP* is TRUE). Data from BIM is now copied to all placeholders for the actual cycle in State Memory before they are passed to the voter.

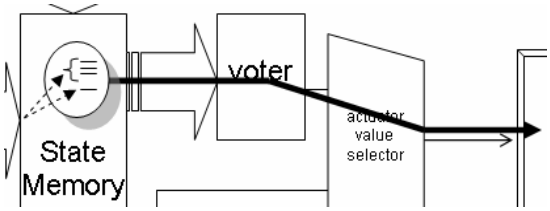

Figure 3.12: Data flow when node is locally controlled

This is equivalent to a simplex system; however we know that BIM ha exercised a watchdog refresh protocol and thus we may rely on the computed values.

3.3.3 Neighbor controlled

When the local microprocessor fails (*MP* is FALSE) but the communication unit works properly, placeholder in State Memory for BIM value is left unchanged. Other placeholders are updated from the communication unit.

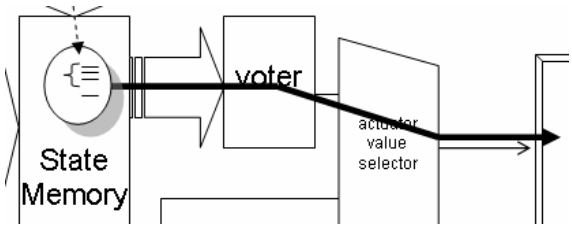

Figure 3.13: Data flow when node's microprocessor fails

The actuators microprocessor has failed. Since there is at least four actuators in the cluster we still have three working properly (assuming a single fault). The majority voter will discriminate any deviating value from the local BIM and send the working nodes values to the physical actuator. During the next communication cycle the FTCC will tag the failing nodes value as "not updated", i.e.

since state memory was not updated the node sends a copy of the previous cycle value but with this special notification. This allows the voters to exclude the old value from voting during the next cycle and thus there is provision for tolerating even a second fault.

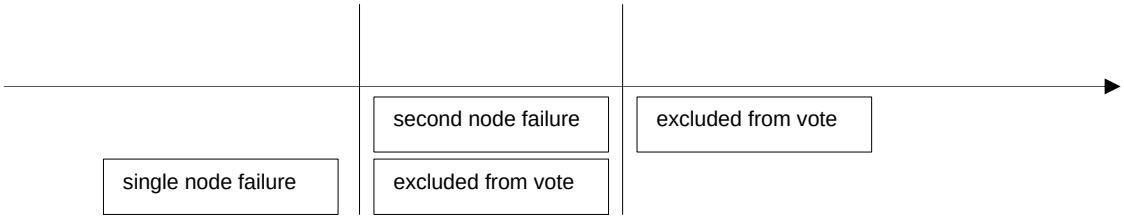

Figure 3.14: Multiple node failure

As a special case, when all values in the *State Memory* are tagged "old" the FTCC shifts to a final mode (*Stand Alone*) described below.

3.3.4 Stand alone

Stand alone mode is introduced to handle the severe case where neither the local BIM nor the communication unit works properly. Actuator value is now taken from the FS State Selector, which in turn holds one of:

MIN, application defined value where the semantic of MIN stands for minimal actuating. This could for example be to release pressure from the brake discs.

MAX, application defined value where the semantic of MAX is maximum actuating. This could for example be to apply maximum pressure to the brake discs.

A feedback value as obtained from a corresponding sensor, which causes actuating to maintain current dynamic behavior.

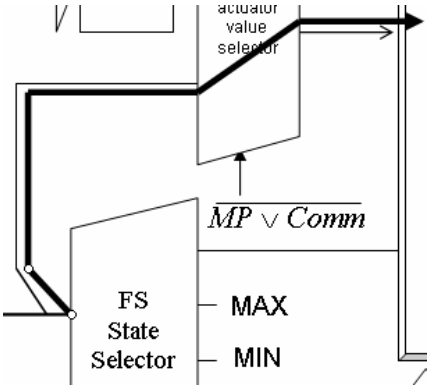

Figure 3.15: Data flow when node's microprocessor and the communication unit fails

This is an abnormal case with low probability. In the case of transient or intermittent occurrence the system will degrade during the persistence of errors influence. In absence of disturbances it will immediately recover during the next communication cycle and enter a controlled mode (described above). If the mode stays permanent this indicates a more or less total blackout, all microprocessors in the cluster are malfunctioning or the FTCC communication system is down and the local microprocessor is faulty. The FTCC will still assure a decent behavior from the physical actuator since it is continuously updated with a reasonable value.

4. THE REDESIGNED BRAKE SYSTEM

In our redesign the original Brake Control Unit has been replaced by four Break Intelligence modules which are integrated into the physical actuators. Thus the cost for the microcomputer (cpu, memory communication units and IO-units) has been roughly increased by a magnitude of four. We have on the other hand removed the need for a separate housing for the BCU as well as the need for separate distribution units. This savings compensate for substantial more than the cost increase for duplicated microcomputers. At the same time we use the redundant microcomputer configuration for implementation of fault tolerance via the FTCC module.

The introduced FTCC-module improves on both system functionality and dependability. We have added programmable *Fail Safe* behaviour as an option. Because the FTCC is time-triggered control jitter is reduced to a minimum.

Despite the relatively low complexity of the FTCC it can add substantially to brake system reliability and safety. At the same time it will increase availability. Consider the original system (Figure 1.1), assuming the *safety line* works properly, the system will enter a fail-safe state (FS) which is defined so that all brake actuators will be activated and brake the vehicle. The use of FTCC would only slightly degrade the brake system using available redundancy upon a single failure in any of the nodes and brake performance would be maintained.

We have successfully implemented the first specification of FTCC using standard VHDL. It has been synthesized in “Xilinx Spartan 2” (FPGA, 100 000 gates). Functionality was verified in a simple test bench where four nodes were used. We believe that valuable experiences can be gained from such implementations even though an FPGA solution, due to the expected fault intensity, would hardly be used in an implementation for real use.

Currently, the FTCC is being redesigned and a specification for the next version (2) is prepared. In version 2 we will put efforts in a modular design with generic specification of the parts that builds the FTCC. This will form the basis of a full-scale fault tolerant design which can then be implemented and assessed.

ACKNOWLEDGMENTS

Thanks to Mr Jörgen Andersson of SAB WABCO for providing source material for this research as well as comments and suggestions regarding this paper. I am also grateful to Prof. Jan Torin, Prof. Bertil Svensson, Mr. Håkan Edler and Mr. Jan Jacobsson for valuable comments on earlier versions of this report.

REFERENCES

[1] Johansson R., *Dependability characteristics and safety criteria for an embedded distributed brake control system in railway freight trains*, Chamers Lindholmen University College, August 2001.

[2] prEN 50126:1995, Railway applications, The specification and demonstration of dependability, reliability, availability, maintainability and safety (RAMS).

[3] prEN 13452-1:1999E, Railway applications – Braking – Mass transit brake system, Part 1: Performance requirements

[4] prEN 13452-2:1999E, Railway applications – Braking – Mass transit brake system, Part 2: Methods of test

[5] EN 50155:1995, Railway applications, Electronic equipment used on rolling stock

[6] prEN 50125-1:1998, Railway applications, Environmental conditions for equipment, Part 1

[7] prEN 50125-2:1998, Railway applications, Environmental conditions for equipment, Part 2

[8] Freedman, P.; Das, A., *Formal modelling and analysis of computerized control in rail transport: a case study*, Communications, Computers, and Signal Processing, 1995. Proceedings. IEEE Pacific Rim Conference on , 17-19 May 1995, pp 610-613.

[9] Harley, W.L.; Riley, C.E., *Performance based safety critical contracts*, Systems Engineering on Large Railway Projects (Digest No: 1997/140), IEE Colloquium on , 7 May 1997, pp 5/1 -5/6

[10] B. McDonnell. P.Stefow, *Safety Certification of Toronto's New Sheppard Subway Line*, Proceedings of the 21'st System Safety Conference, System Safety Society, 4-8 August 2003, pp. 329 –336.

[11] N. Leveson, *Integrating Safety Information into the System Engineering Process*, Proceedings of the 21'st System Safety Conference, System Safety Society, 4-8 August 2003, pp. 1009 –1028.

[12] Hachiga. A., Akita, K. Hasegawa, Y., A., *The design concepts and operational results of fault-tolerant computer systems for the Shinkansen train control*, Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on , 22-24 June 1993, pp 78 -87.

[13] Hennebert, C.; Guiho, G., *SACEM: A fault tolerant system for train speed control*, Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on , 22-24 June 1993, pp 624 -628

[14] Holding, K., *Brake by Wire [computer controlled braking]*, Chassis Electronics, IEE Colloquium on , 23 Mar 1990, pp 5/1 -5/2

[15] Ashiya, M.; Sone, S.; Sato, Y.; Kaga, A., *Application of pure electric braking system to electric railcars*, Advanced Motion Control, 2000. Proceedings. 6th International Workshop on , 30 March-1 April 2000, pp 163 -168

[16] Khmelnitsky, E., *On an optimal control problem of train operation*, Automatic Control, IEEE Transactions on , Volume: 45 Issue: 7 , July 2000, pp 1257 -1266

[17] En-Dean Chen; Tse, Y.H.; Myers, L.F., *Economic considerations of operating a train with electronically controlled pneumatic (ECP) brakes*, Railroad Conference, 1998. Proceedings of the 1998 ASME/IEEE Joint , 15-16 April 1998, pp 9 -19

[18] Roger Johansson, Per Johannessen, Kristina Forsberg, Håkan Sivencrona, Jan Torin. *On Communication Requirements for Control-by-Wire Applications*, In Conference Proceedings of the 21st International System Safety Conference 2003 (ISSC21) August 4-8 2003, Ottawa, Canada

[19] Forsberg, K. *Design Principles of Fly-By-Wire Architectures*, PhD. Thesis, Dept. of Computer Eng., Chalmers University of Technology, Goteborg, Sweden, 2003.

[20] Bridal, O., L.-Å. Johansson, J. Ohlsson, M. Rimén, B. Rostamzadeh, J. Torin and R. Snedsböl, DACAPO: *A Dependable distributed computer architecture for Control Applications with Periodic Operation*, Technical Report 165R, Dept. of Computer Eng., Chalmers University of Technology, Goteborg, Sweden, 1993.

[21] Hansson, H., H. Lawson, O. Bridal, C. Eriksson, S. Larsson, H. Lönn, M. Strömberg, *BASEMENT: an Architecture and Methodology for Distributed Automotive Real-Time Systems*, IEEE Transactions on Computers, vol.46, no9, pp 1016-27, 1997.

[22] Kopetz, H., A. Damm, C. Koza, M. Mulazzi, W. Schwabl, C. Senft, R. Zainlinger, *Distributed Fault-Tolerant Real-Time Systems: The MARS approach*, IEEE Micro, vol 9, no 1, pp 25-40, 1989.

[23] Powell, D., J. Arlat, L. Beus-Dukic, A. Bondavalli, P. Coppola, A.Fantechi, E.Jenn, C. Rabéjac, A. Wellings, *GUARDS: A Generic Upgradable Architecture for Real-Time Dependable Systems*, IEEE Transactions on Parallel and Distributed Systems, vol 10, no 6, 1999.

[24] Y.C. Yeh, *Triple-Triple Redundant 777 Primary Flight Computer*, Proceedings of the 1996 IEEE Aerospace Applications Conference, Vol. 1, 1996, pp. 293 –307.

[25] A.K. Naidu. *Case Study: Airbus A340 Flight Control System*, Dept of computer science, University of Virginia.

[26] Philippe David and Claude Guidal, *Development of a Fault Tolerant Computer System for the HERMES Space Shuttle*, Digest of Papers: The Twenty-Third International Symposium on Fault-Tolerant Computing (FTCS 23), Toulouse, France, June 22 – 24, 1993, pp. 641 – 646.

[27] Rushby, J. *A Comparison of Bus Architectures for Safety-Critical Embedded Systems*, CSL Techical Report, SRI International, September 2001.