Available Fail-Safe Systems*

D. Essamé, J. Arlat, D. Powell

LAAS-CNRS
7, avenue du Colonel Roche
31077 Toulouse Cedex 4, France

ABSTRACT

Continuity of service and cost-effectiveness are adding new challenges to life critical systems over and above the underlying safety concerns. The introduction of redundant components is a necessary condition for increasing the overall system availability with respect to physical component failures. Here we consider redundancy by means of replicating fail-safe components in a distributed real-time system for railway applications. In such a system, some functions cannot tolerate even a brief service interruption. These functions have to be replicated using active redundancy, and their outputs must be consolidated with the goal that the failure of one component has no effect on the delivered service. We formally investigate conditions for preserving safety properties of fail-safe components when replicating them using active redundancy. We focus our analysis on duplex computers with two fail-safe units. Given some safety constraints, we show that inconsistency of replicated units can lead to safety degradation even if each replicated component (taken individually) satisfies the given safety constraints. Two solutions are studied: masking and detection of state or context inconsistency. The former leads to requirements on the output consolidation function and the latter to requirements on the redundancy management mechanisms.

Keywords: *fault tolerance, redundancy, safety, fail-safe systems, safety property preservation, output consolidation, real-time systems, railway applications.*

1. Introduction

During the last decade, many applications of computer systems have emerged in the field of railway systems: automatic speed control (SACEM [11], Shinkansen [2]), interlocking systems (SMILE [1], CBI [13], ELEKTRA [12]), train route and traffic control (COMTRAC [10]), fully automated train control (MAGGALY: *Metro A Grand Gabarit de l'Agglomération LYonnaise*) and others. Fault-tolerant computing systems are increasingly used to meet the stringent dependability requirements that, besides safety, extend to availability and to maintainability. Indeed, improvement of quality of service, continuity of service and cost-effective exploitation are adding new challenges to railway system designers beyond the underlying safety concerns [3].

Different architectural solutions have been used to ensure safety. These range from simplex and duplex to dual duplex and TMR architectures. Similarly, several approaches have been taken to satisfy the availability requirement. In particular, two different approaches can be identified:

- Ensure availability first and then address the safety issue globally. A typical example of such an approach is the electronic interlocking system ELEKTRA [12]. This system has two TMR channels (VOTRICS node [15]) in a Control/Monitor configuration implementing the "safety bag technique" [14]. The control channel, or Interlocking Computer performs interlocking control functions whereas the monitor channel, or Safety Bag Computer ensures that safety conditions are respected. Thus, reliability and availability are achieved by using actively triplicated hardware in each channel while safety is achieved by inter-channel checking.
- Replicate safe building blocks to ensure availability; on the contrary to the previous approach, each basic block can ensure safety. Typical examples of such an approach are systems based on the "coded processor" [8]. This is an informational redundancy technique associating arithmetic coding and signature checking. Since encoded data processing is used to meet the safety requirements, replication is only necessary for availability.

In this paper, we consider the second approach above. The main problem we address is how to increase the availability of a system without degrading its safety. We have based our study on the fully automated train control system METEOR (*METro Est Ouest Rapide*), designed to control the new east-west subway line in Paris.

This system consists of duplex computers interconnected by a network. Both units of the duplex computer are fail-safe and are based on the coded processor approach. In such a system, some functions cannot tolerate even a brief service interruption. An example of such a function is the high voltage control where any interruption of service leads to a system shutdown. Since a system shutdown has a very negative effect on system availability (restart can take a long time) such functions have to be

* This work was partially supported by **Matra Transport Internationnal**.

replicated using active redundancy. For this target system we define some safety constraints that are assumed to be satisfied by each redundant component. We consider output consolidation techniques that allow continuous service to be achieved. However, we show that state or *context* inconsistency of replicated components can lead to safety degradation even if each replicated component (taken individually) satisfies the given safety constraints. We formally investigate the conditions for preserving safety properties of fail-safe components when replicating them using active redundancy within a safety constraints analysis framework. Two approaches are considered: masking and detection of inconsistency.

The rest of this paper is structured as follows. After defining the notion of a safety constraint, we show how context inconsistency of actively replicated components can lead to safety degradation even when each component is fail-safe. Afterwards, we formally investigate the conditions under which the safety properties are preserved. Both output consolidation and redundancy management mechanisms are studied. We then describe a protocol that ensures that the safety properties hold.

2. Definition of safety constraints

Given a fail-safe component, let X be the set of actions that this fail-safe component can undertake on its environment. These can be safety-related actions (e.g., high voltage control) or functional actions (e.g., sound alarm control).

Definition 1—Dependency relation

Given two actions $x, y \in X$, we say that action x depends on action y if and only if action y is a necessary precondition for action x. We note this relation $y \prec x$.

Two actions x and y are **dependent** if they are linked by the dependency relation; if not, they are **independent**. The dependency relation defines a partial order on the set of actions X. The dependency relation links actions that have to be undertaken in a given order to meet either safety or functional requirements. Undertaking such actions in a different order can lead to system failures ranging from benign to catastrophic.

Definition 2—Safety constraint

*Given two dependent actions $x, y \in X$ such that $y \prec x$. There is a **safety constraint** between x and y if transgression of the dependency relation can lead to a catastrophic failure.*

It is important to note that two actions can be dependent without there being a safety constraint.

Let p be the following predicate:

$$p : X \mapsto \{ false \ ; \ true \}$$

with $p(x)=true$ if the action x is undertaken and $p(x)=false$ if not, $\forall x \in X$.

Given this predicate, the respect of the dependency relation for two actions x and y such that $y \prec x$ requires that $p(y)$ be a valid consequence of $p(x)$. More formally, the formula $p(x) \rightarrow p(y)$ must always be true. In other words, the interpretation where $p(x)=true$ and $p(y)=false$, must be avoided. Therefore, the safety constraint between actions x and y can be stated formally by:

$$\left(p(x) \rightarrow p(y)\right) \equiv true$$

or, equivalently:

$$\overline{\left(p(x).\overline{p(y)}\right)} \equiv true$$

3. System Model

The system consists of networked duplex controllers made up of two fail-safe units based on the coded processor.

The units can communicate with each other and with remote units by messages sent over a network. The network is very reliable, but not enough for human lives to depend on it. Thus, from a safety viewpoint, we must assume that the network can loose or delay messages. However, message integrity is ensured by error-detecting codes. In the terminology of [4], messages sent over the network have omission/performance failure semantics. The local clocks of units are not synchronized. However, every unit checks the rate of drift of its local clock with respect to real-time and switches to the safe mode if the drift exceeds a predefined bound. Consequently, the local clock of an operational unit has a bounded rate of drift from real-time. Thus, the system satisfies the timed asynchronous model [6].

Since delayed messages can impair safety, a fail-aware datagram service similar to the one described in [7] is available for message delivery. Thus delayed messages can be thrown away.

3.1 Replication

Both availability and safety are key issues in railway systems. To achieve the availability requirements, some functions are replicated on different components of the system. Three modes of replication can be considered: *active replication* (all the replicas receive inputs, update their internal state or and apply outputs to the environment), *semi-active replication* (all the replicas receive inputs, update their internal state, but only one replica—the primary replica—applies outputs to the environment) and *passive replication* (only one replica—the primary replica—receives inputs and applies outputs to the environment). Active replication is necessary when the function cannot tolerate any service interruption, but requires a specific mechanism to combine redundant outputs into a single effect. Semi-active replication can be used when a short service interruption (due to switchover delay) can be tolerated. Passive replication is particularly well adapted to functions with no internal state since, in this case, there is no need for checkpointing.

In the particular case where the primary (resp. secondary) replicas of all functions are allocated exclusively to the same unit, we call this unit the *Primary* (resp. *Secondary*) unit. Here, we consider the case where two fail-safe unit are used in a primary/secondary configuration to meet the availability requirement.

3.2 Output consolidation

We call output consolidation the process of combining redundant outputs from active replicas into a single effect (Fig. 1).

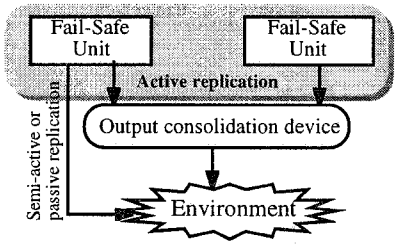

Fig. 1 : Output consolidation device

To fulfil the continuity of service requirement, the replica outputs must be consolidated with the goal that the failure of one replica has no effect on the expected service. Let us consider the control of the High Voltage (HV) power to the train track. We consider a duplex computer with two fail-safe units **A** and **B**. Since the loss of HV has a very negative effect on the system availability[1], this function has to be replicated using active replication. Each unit has an HV control output (*AutHVA* for **A** and *AutHVB* for **B**) which is *true* if the unit allows HV and *false* if not; with the safe position being when this control output is false. Since each unit is fail-safe, in the case of a detected failure, it always switches to the safe position by setting the given output to *false*. For this example the OR logic function can be used to allow continuous service. Indeed we have:

$$AutHV = AutHVA + AutHVB$$

where *AutHV* is the final control output to be applied to the environment. If the unit **B** fails, it sets *AutHVB* to *false* so one has : $AutHV = AutHVA + false = AutHVA$.

4. Problem statement

Redundancy introduces potential safety problems that would not exist in an unreplicated system. We state the problem by means of an example. In fully automated train systems, one of the critical procedures to be handled is the emergency evacuation of passengers. This procedure needs many actions to be undertaken on the environment. Two of these actions are: cut power to the rails (*HV_Cut:* setting *AutHV* to *false*) and open doors of the train (*Doors_Open*). Since opening doors without cutting the High Voltage can lead to a catastrophic failure, according to Definition 2, there is a safety constraint between *HV_Cut* and *Doors_Open*. Therefore, according to the previous predicate *p*, one has :

$$(p(Doors\_Open) \rightarrow p(HV\_Cut)) \equiv true$$

Let us consider two fail-safe units (**A** and **B**) of a duplex computer (**D**) in a primary/secondary configuration with unit **A** as Primary. We assume that the High Voltage control function is replicated using active redundancy with OR-logic for output consolidation. However, since the door opening function can tolerate service interruption, this function can be replicated using semi-active or passive replication. Thus, we assume that only the primary applies the corresponding output to the system. Let p_A (*resp.* p_B, p_D) stand for the predicate *p* with respect to unit **A** (*resp.* unit **B**, duplex computer **D**).

We consider a scenario in which, due to some transmission error, the states or *contexts* of unit **A** and **B** have become inconsistent, such that:

- **A** has detected an emergency evacuation situation,
- **B** has not (yet) detected the emergency.

This leads to the following situation:

For unit **A** which has detected the emergency evacuation:

- Set the *AutHVA* to *false* to shutdown the HV: $p_A(HV\_Cut) = true$;
- Open train doors: $p_A(Doors\_Open) = true$;
  Thus: $(p_A(Doors\_Open) \rightarrow p_A(HV\_Cut)) = true$.

For unit **B** which has not detected the emergency evacuation:

- No HV shutdown, *AutHVB=true*: $p_B(HV\_Cut) = false$;
- No train door opening: $p_B(Doors\_Open) = false$;
  Thus: $(p_B(Doors\_Open) \rightarrow p_B(HV\_Cut)) = true$.

For the duplex computer **D**:

- Since the final output for HV is obtained by the OR logic function: $AutHV = AutHVA + AutHVB$, one has no HV shutdown: $p_D(HV\_Cut) = false$;
- Since unit **A** is the primary unit and the door opening function is replicated in semi-active replication mode, one has $p_D(Doors\_Open) = true$;
  Thus: $(p_D(Doors\_Open) \rightarrow p_D(HV\_Cut)) = false$.

Both units **A** and **B** individually satisfy the safety constraint, however the duplex computer **D** does not. This example shows how context inconsistency of redundant units can lead to safety degradation even when each unit is fail-safe. It can be shown that such situations arise if some or all actions result from actively-replicated functions. For simplicity, we choose here to consider only the case where the other action results from a semi-actively replicated function.

One solution to this problem would be to implement an atomic broadcast service to ensure that replicas agree on a consistent context. Given a team of *n* units and dynamically-formed groups, Cristian identifies three different specifications for replica consistency in timed asynchronous systems [5]:

- *Group agreement* which ensures that all members joined to the same group agree on a history of updates.
- *Majority agreement* which ensures that all members joined to a majority group (a group with more than half of the team's members) agree on a history of updates.
- *Strict agreement* which ensures that all team members agree at any time on a unique history of updates.

Unfortunately, none of these forms of agreement solves the inconsistency problem for a duplex controller (a team with two members). With the Group agreement specification, in case of communication failure, the team will be divided into two groups allowing replica updates to

[1] A power cut brings all trains on the line to a halt. Recovery from this situation requires a lengthy verification procedure including, for example, checking that no passengers have strayed from the immobile trains.

occur in parallel with the possibility of context inconsistency. Majority and Strict agreement are also unsuitable since in case of communication failure, it is impossible to obtain a majority group. It has in fact been shown that it is impossible for two units to agree if messages between them can be lost (see the "two-generals problem" in [9]).

Given that context inconsistency cannot be avoided in duplex controllers interconnected according to the timed asynchronous model, we now focus our study on how to preserve safety constraints by *tolerating* context inconsistency. We address the problem by means of context inconsistency masking and context inconsistency detection.

5. Safety constraint preservation

To enhance the fail-safe property of a unit **U** such that $\mathbf{U} \in \{\mathbf{A},\mathbf{B}\}$, we introduce a boolean variable u (i.e., a for unit **A** and b for unit **B**) which is equal to *true* if it is providing its nominal service and to *false* if the unit switches to the safe mode. When a unit is in the safe mode, all its safety outputs are in their safe position and no control outputs are sent to the environment. When a failure occurs, the fail-safe property guarantees that the unit always switches to the safe mode.

Given $x, y \in X$, two dependent actions such that $y \prec x$ with a safety constraint, we can state the following axiom for a fail safe unit **U**, with $\mathbf{U} \in \{\mathbf{A},\mathbf{B}\}$:

Axiom 1

Any safety constraint between actions x and y is always satisfied by a fail-safe unit **U**, *i.e.* $\overline{p_U(x).\overline{p_U(y)}} \equiv true$

where p_U *stands for the previous predicate p with respect to unit* **U**.

Both **A** and **B** are fail-safe, so we assume that all outputs towards the environment are put into a safe state if both units should fail. Thus, we can henceforth neglect the case a=b=*false*.

5.1 Context inconsistency masking

Let us suppose that action x is the result of a function replicated in semi-active mode while the action y is the result of a function replicated in active mode. Each output consolidation is defined in terms of the properties that it ensures. Here we give such properties for semi-active and active replication modes. We assume that unit **A** is the current Primary.

When *semi-active replication* is used (action x), one expects that the following two exclusive and complete statements be ensured:

1_{sa} As long as the Primary unit **A** provides its nominal service (a=*true*) then it ensures the action x:
$a \rightarrow p_D(y) = p_A(y)$

2_{sa} If the Primary switches to the safe mode (a=*false*) then the action x will be ensured in a bounded delay[2] by the Secondary unit (**B**) if it does not fail (b=*true*):
$\bar{a}b \rightarrow p_D(y) = p_B(y)$

Both statements can be summarized formally by the following property:

OCsa : $p_D(x) = a.p_A(x) + \bar{a}.b.p_B(x)$

For *active replication* (action y), the first requirement is continuity of service:

1_a If one unit switches to the safe mode (a=*false* or b=*false*) then responsibility for action y will be transferred immediately to the other unit:
$a\bar{b} \rightarrow p_D(y) = p_A(y)$
$\bar{a}b \rightarrow p_D(y) = p_B(y)$

While both units are providing their nominal service, we successively consider the following exclusive statements:

2_ae When both units are providing their nominal service, the action y is undertaken by the duplex computer **D** if <u>*either*</u> unit decides:
$ab \rightarrow p_D(y) = p_A(y) + p_B(y)$

2_ab When both units are providing their nominal service, the action y is undertaken by the duplex computer **D** if <u>*both*</u> units decide:
$ab \rightarrow p_D(y) = p_A(y).p_B(y)$

As for semi-active replication, these statements can be summarized formally by the following two properties, when combined with the requirement **1_a**:

OCe : $p_D(y) = a.\bar{b}.p_A(y) + \bar{a}.b.p_B(y) + a.b.(p_A(y) + p_B(y))$
$= a.p_A(y) + b.p_B(y)$

OCb: $p_D(y) = a.\bar{b}.p_A(y) + \bar{a}.b.p_B(y) + a.b.p_A(y).p_B(y)$
$= (\bar{a}.b + a.p_A(y))(a.\bar{b} + b.p_B(y))$

We now consider the consequences of these two approaches (***OCe*** and ***OCb***) on the safety constraints of the duplex computer **D**.

1st case : $p_D(y) = a.p_A(y) + b.p_B(y)$

we have:

$$\overline{p_D(x).\overline{p_D(y)}} = \overline{(a.p_A(x) + \bar{a}.b.p_B(x))(\overline{a.p_A(y) + b.p_B(y)})}$$

which gives by developing and using axiom 1:

$$\overline{p_D(x).\overline{p_D(y)}} = \overline{(\bar{b} + p_B(y)).a.\underbrace{p_A(x).\overline{p_A(y)}}_{=false} + \bar{a}.b.\underbrace{p_B(x).\overline{p_B(y)}}_{=false}}$$

from which one concludes that: $\boxed{\overline{p_D(x).\overline{p_D(y)}} \equiv true}$

2nd case : $p_D(y) = (\bar{a}.b + a.p_A(y))(a.\bar{b} + b.p_B(y))$

we have:

$$\overline{p_D(x).\overline{p_D(y)}} = \overline{(a.p_A(x) + \bar{a}.b.p_B(x))\left((\bar{a}b + ap_A(y))(\overline{a\bar{b} + bp_B(y)})\right)}$$

which gives by developing and using axiom 1:

$$\overline{p_D(x).\overline{p_D(y)}} =$$

$$\overline{(a + \bar{b}).a.\underbrace{p_A(x).\overline{p_A(y)}}_{=false} + a.b.p_A(x).\overline{p_B(y)} + (\bar{a} + b).b.\underbrace{p_B(x).\overline{p_B(y)}}_{=false}}$$

and finally: $\boxed{\overline{p_D(x).\overline{p_D(y)}} = \bar{a} + \bar{b} + p_A(x).\overline{p_B(y)}}$ (1)

In the first case, the safety constraint always holds, while in the second, we must either have one unit switched

[2] For simplicity, we do not formalize here the temporal aspects of the problem.

to the safe position (a=*false* or b=*false*) or $p_B(y) = p_A(y)$ (no context inconsistency: both units must reach the same decision) for the safety constraint to hold.

In conclusion, an output consolidation function ensuring property ***OCb*** is not sufficient to tolerate context inconsistency. This was the case of the OR-logic function described in the previous example with:

$$AutHV = a.AutHVA + b.AutHVB$$

For the action y = *HV_Cut*, this gives:

$$\overline{p_D(y)} = a.\overline{p_A(y)} + b.\overline{p_B(y)}$$

which can be shown to satisfy ***OCb***, but not ***OCe***.

When such an output consolidation technique is used, it is necessary to avoid context inconsistency to preserve the safety constraint. However, with an output consolidation function ensuring property ***OCe***, there is no need to give an absolute guarantee on context consistency, to preserve the safety constraint, since such an output consolidation function effectively tolerates inconsistencies. It can be shown that property ***OCe*** is satisfied by the following function:

$AutHV = a.\bar{b}.AutHVA + \bar{a}.b.AutHVB + a.b.AutHVA.AutHVB$

which achieves both continuity of service (when one unit fails) and safe operation in case of inconsistency.

5.2 Context inconsistency detection

We have shown in the previous section that an output consolidation technique fulfilling ***OCe*** can tolerate context inconsistency. Unfortunately, a fail-safe implementation of such a function would be quite expensive in practice. Here we present an alternative solution that allows the safety constraints to hold even if the output consolidation technique does not fulfil ***OCe***. The key idea of our approach is to detect context inconsistency and switch the duplex controller to a configuration which allows the safety constraint to hold.

We define four states for each fail-safe unit: *primary*, *standby*, *quarantine* and *failed*. When the unit is in the quarantine or failed states, it is said to be in the *non-operational* or safe mode, in which it cannot deliver outputs to the environment. A unit in the primary or standby states is in the *operational mode* (Fig. 2).

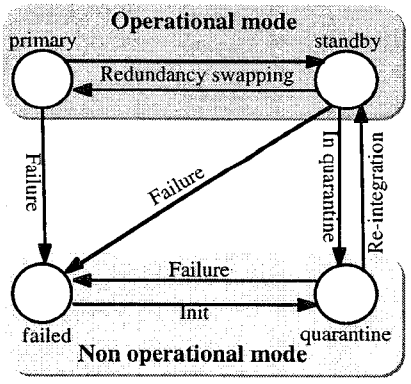

Fig. 2 : State graph of a fail-safe unit

A unit is in the *primary* state when it is the current Primary. The current Secondary unit can be in the *standby* state, if its context is consistent with the current Primary's context. Otherwise, it is in the *quarantine* state. The quarantine state is an intermediate state that is introduced for safety purposes: the Secondary unit is put in quarantine when its context is inconsistent with the Primary's context. When a unit is in quarantine, it switches to the safe mode. Toward this goal, we can state the following objective for safe operation:*The protocol that manages the redundant pair of units must either ensure that their contexts are kept consistent or else force the Secondary unit into the quarantine state.*

Let $\mathbf{C_U}$ be the context of unit **U**, with $\mathbf{U} \in \{\mathbf{A},\mathbf{B}\}$ and E_U its state, with $E_U \in \{primary, standby, quarantine, failed\}$

Here we define two safety properties for the redundancy management protocol and show that these properties are sufficient to guarantee fail-safe behaviour of a redundant pair. For $i, j \in \{A,B\}$, $i \neq j$

- *Unique Primary property (**UP**)*:
  $(E_i = primary) \rightarrow (E_j \neq primary)$
- *Quarantine property (**Q**)*:
  $(E_i = primary) \wedge (\mathbf{S}_i \neq \mathbf{S}_j) \rightarrow (E_j \neq standby)$

With the ***UP*** requirement we prohibit the possibility of having two Primary units. This is for safety, since we need only one leader at any given instant. The ***Q*** requirement states that the Secondary unit must not become or remain in the standby state if its context is inconsistent with the current Primary. We now show formally how these requirements can circumvent the context inconsistency problem identified before. For the proof, we need the following axiom which emphasizes the deterministic behaviour of each fail-safe unit.

Axiom 2

There exists a time constant t_d such that, if a unit detects at time t an event which should cause it to undertake an action, then if the unit does not fail, this action will be undertaken by time $t + t_d$.

Theorem 1

Consider a duplex computer **D** *with two fail-safe units* **A** *and* **B**. *If there exists in* **D** *a mechanism that guarantees* ***UP and Q*** *then every safety constraint that holds on* **A** *and* **B** *also holds on* **D** *even if the output consolidation function does not achieve* ***OCe***.

Proof

Let us consider $x, y \in X$, two dependent actions such that $y \prec x$ with a safety constraint. Let us also assume that the action x is the result of a function replicated in semi-active mode while the action y is the result of a function replicated in active mode. Assuming that unit **A** is the Primary and that the output consolidation functions for x and y satisfy respectively ***OCsa*** and ***OCb***, we have shown that the safety constraint for **D** is given by (1):

$$\overline{p_D(x)p_D(y)} = \bar{a} + \bar{b} + \overline{p_A(x)p_B(y)}$$

Let's $\mathbf{C_A}$ (*resp.* $\mathbf{C_B}$) be the context of unit **A** (*resp.* ***B***).

Unit **A** is the Primary so $E_A = primary$, property ***UP*** gives by the modus: $E_B \neq primary$ or, equivalently:

$(E_B = standby) \vee (E_B = quarantine) \vee (E_B = failed)$

If b=*true* (unit **B** in operational mode) then the quarantine and failed states can be excluded since they belong to the non operational mode. Therefore: $E_B = standby$. Then, by applying modus tollens on property Q we obtain: $\mathbf{C_B} = \mathbf{C_A}$. Moreover, **Axiom 2** guarantees that if $\mathbf{C_B} = \mathbf{C_A}$ at time t then at most at time $t + t_d$ one has $p_B(y) = p_A(y)$ if unit **B** does not fail. Therefore, since:

$$\overline{p_D(x)\overline{p_D(y)}} = \bar{a} + \bar{b} + \overline{p_A(x)\overline{p_B(y)}}$$

then, at time $t + t_d$:

-if unit **B** does not fail (b=*true*):

$\overline{p_D(x)\overline{p_D(y)}} = \bar{a} + \overline{p_A(x)\overline{p_A(y)}} = true$ (from Axiom 1)

-or if unit **B** fails (b=*false*):

$$\overline{p_D(x)\overline{p_D(y)}} = true + \bar{a} + \overline{p_A(x)\overline{p_B(y)}} = true$$

If $\mathbf{C_B} \neq \mathbf{C_A}$ at time t (*context inconsistency*) then under the assumption that unit **A** is the Primary ($E_A = primary$), property Q gives by modus ponens: $E_B \neq standby$. Since the property ***UP*** gives by modus ponens: $E_B \neq primary$, we conclude that the Secondary unit **B** will be in the non operational mode, $(E_B = quarantine) \vee (E_B = failed)$, and thus switched to safe mode, implying b=*false*. Therefore, since:

$$\overline{p_D(x)\overline{p_D(y)}} = \bar{a} + \bar{b} + \overline{p_A(x)\overline{p_B(y)}}$$

we have:

$$\overline{p_D(x)\overline{p_D(y)}} = true \quad \blacksquare$$

6. A duplex fail-safe controller

In this section, we describe a redundancy management protocol for networked fault-tolerant duplex controllers that provide high availability while ensuring the safety properties Q and ***UP*** of the previous section.

We consider controllers made up of two fail-safe units. Each unit provides a failure-status output indicating whether it is in the *operational* or *safe* mode. The ***UP*** requirement is ensured by hardware using a bistable *safety relay* controlled by the failure-status outputs of each unit (Fig. 3).

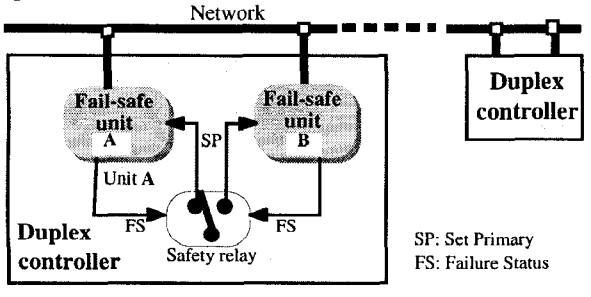

Fig. 3 Duplex controller

6.1 Protocol overview

The purpose of the protocol is to ensure, if possible, that messages received over the network are delivered to both units. If a transmission error should occur that prevents the message from being delivered to both units, then a context inconsistency can occur. In this case, the protocol must ensure that property Q holds by forcing the Secondary unit to switch to the quarantine state. While a unit is in the quarantine state, it cannot deliver outputs to the controlled process. Moreover, it is unable to replace the other unit should the latter fail. Consequently, to provide availability, the protocol must attempt to bring the unit in the quarantine state back to the secondary state.

To ensure availability, two progress properties must therefore be respected, but only in the absence of failures :

- *Agreement (**A**).* Every message accepted by one unit at time t must have been or be accepted by the other unit within the interval $[t - \tau, t + \tau]$
- *Limited quarantine (**LQ**).* A unit in the quarantine state must eventually switch back to the standby state (subject to the safety property Q)

Property ***LQ*** prevents the trivial solution in which one unit always remains in the quarantine state. Property ***A*** prevents useless solutions in which the unit in the standby state immediately switches to the quarantine state.

For safety, there is no obligation for the protocol to achieve consistency or to maintain both units in the operational mode since those are availability needs. However there is an obligation to put and keep the Secondary unit in quarantine while its context is inconsistent with the Primary's state.

6.2 Protocol description

We successively describe the cases where i) both units are in operational mode, and ii) the Secondary unit is in quarantine.

Both units are in operational mode

When both units are in operational mode, safety is the key issue. The main idea is to attempt to ensure context consistency through broadcasting inputs to both units atomically. If atomicity cannot be ensured, the Secondary unit is put into quarantine, to ensure safety property Q. The principle used is the following:

a) Primary

- Send, periodically, a message to the Secondary "***Don't switch to quarantine***".
- Each time an input message is received from a remote controller, forward this message to the Secondary, set a time-out and wait for an acknowledgement :
  - if the acknowledgement is received before the time-out expires, accept the message;
  - if the time-out expires then:
    - stop sending "***Don't switch to quarantine***" messages,
    - stop forwarding input messages to the Secondary,
    - accept the message.
- If the Primary fails, the safety relay will switch the current Secondary to the primary state.

b) Secondary

- Wait for the periodic "***Don't switch to quarantine***" message. If there is no such message within a given time interval, then switch to the *quarantine* state.
- Each time an input message is received directly from a remote controller, forward this message to the Primary

(when the Primary receives this message, it behaves as previously).

- Each time an input message is received from the Primary, send an acknowledgement to the Primary and accept the message. (Note that the message can be accepted immediately by the Secondary since the Primary has seen the same message, so the latter will either accept the message in a bounded time or cause the Secondary to switch to the *quarantine* state.)
- The failure of the Secondary has no immediate effect. The Primary will be informed of the failure when it next attempts to forward a message since it will not receive an acknowledgement.

The Secondary unit is in quarantine

Here the key issue is availability since, while it is in quarantine, the Secondary is not in a position to replace the Primary should the latter fail. For availability, the context of the Secondary has to be made consistent with that of the Primary so that it can revert to its backup role. This is done by executing a protocol that transfers the context of the Primary to the Secondary. During context transfer, a specific mechanism is used to detect and propagate concurrent context modifications. The last context transfer message is identified as such by the Primary. The Secondary must remain in quarantine until context transfer has been successfully completed.

The protocol can be summarized as follows:

a) Primary

- Transfer context to Secondary. When the last context transfer message has been acknowledged by the Secondary:
  - resume sending "***Don't switch to quarantine***" messages,
  - resume forwarding input messages.

b) Secondary

- Wait for last context transfer message and switch to the *standby* state.

For improved availability, messages and message acknowledgements can be repeated.

This protocol has been described and modelled with Petri nets. Some properties have been proved. Specifically, we have shown that this protocol ensures the agreement property ***A*** in the absence of failures. If both units accept the same inputs from the same initial state within a given time interval then they will carry out identical context changes within this time interval, or fail safely. The principle which consists of sending the "***Don't switch to quarantine***" messages ensures that if the agreement may not be reached (because of failure(s)), the Secondary will switch to the safe mode in a bounded time interval. This ensures the safety property ***Q***.

7. Conclusion

In order to tolerate context inconsistency in a duplex fail-safe controller, two approaches have been studied in this paper: masking and detection of context inconsistency. For the latter a protocol is given. The key idea of the protocol is to try to keep both units consistent by attempting to agree on input messages; however if this agreement fails, it switches the duplex controller to a mode ensuring safe operation.

References

[1] H. K. Akita, T. Watanabe and I. Okumura, "Computerized Interlocking System for Railway Signaling Control : Smile", *IEEE Transactions on Industry Applications,* 1A-32 (4), pp.826-834, 1985.

[2] K. Akita and H. Nakamura, "Safety and fault-tolerance in computer-controlled railway signalling systems", in *Dependable Computing for Critical Applications,* (C. E. Landwhehr, B. Randell and L. Simoncini, Eds.), 3, pp.107-131, Springer-Verlag, New-York, 1993.

[3] J. Arlat, N. Kanekawa, A. M. Amendola, J.-L. Dufour, Y. Hirao and J. A. Profeta, "Dependability of Railway Control Systems", in *Proc. 26th Int. Conf. on Fault-Tolerant Computing (FTCS-26),* (Sendai, Japan), pp.150-155, IEEE CS Press, 1996.

[4] F. Cristian, "Understanding Fault-Tolerant Distributed Systems", *Comm. ACM,* 34 (2), pp.56-78, 1991.

[5] F. Cristian, "Group, Majority, and Strict Agreement in Timed Asynchronous Distributed Systems", in *Proc. 26th Int. Conf. on Fault-Tolerant Computing (FTCS-26),* (Sendai, Japan), pp.178-187, IEEE CS Press, 1996.

[6] F. Cristian, "Syncronous and Asynchronous Group Communication", *Comm. ACM,* 39 (4), pp.88-97, 1996.

[7] C. Fetzer and F. Cristian, "Fail-Awareness in Timed Asynchronous Systems", in *Proc. 15th ACM Symp. on Principles of Distributed Computing,* (Philadelphia, USA), pp.314-321, May 1996.

[8] P. Forin, "Vital Coded Microprocessor : Principles and Application for Various Transit Systems", in *Proc. IFAC-GCCT,* (Paris, France), pp.79-84, September 1989.

[9] J. Gray, "Notes on Database Operating Systems", in *Operating Systems: An Advanced Course,* (R. Bayer, R. M. Graham and G. Seegmuller, Eds.), Lecture Notes in Computer Science, 60, Springer-Verlag, Berlin, 1978.

[10] A. Hachiga, K. Akita and Y. Hasegawa, "The Design Concepts and Operational Results of Fault-Tolerant Computer Systems for the Shinkansen Train Control", in *Proc. 23rd Int. Conf. on Fault-Tolerant Computing (FTCS-23),* (Toulouse, France), pp.78-87, IEEE CS Press, 1993.

[11] C. Hennebert and G. Guiho, "SACEM: A Fault-Tolerant System for Train Speed Control", in *Proc. 23rd Int. Conf. on Fault-Tolerant Computing (FTCS-23),* (Toulouse, France), pp.624-628, IEEE CS Press, 1993.

[12] H. Kantz and C. Koza, "The ELEKTRA Railway Signaling-System : Field Experience with an Actively Replicated System with Diversity", in *Proc. 25th Int. Conf. on Fault-Tolerant Computing (FTCS-25),* (Passadena, California), pp.453-158, IEEE CS Press, June 1995.

[13] G. Mongardi, "Dependable Computing for Railway Control Systems", in *Dependable Computing for Critical Applications,* (C. E. Landwhehr, B. Randell and L. Simoncini, Eds.), 3, pp.255-277, Springer-Verlag, New York, 1993.

[14] M. Mulazzani, "Reliability and safety in electronic interlocking", in *Proc. IFAC Control in Transportation Systems,* (Vienna, Autria), pp.321-328, 1986.

[15] G. Wirthumer, "VOTRICS—Fault Tolerance Realized in Software", in *Proc. IFAC SafeCom,* (Vienna, Austria), pp.135-140, 1989.