Agents for Fault Detection and Fault Tolerance in Component-based Systems

Meriem Zaiter
Larbi Ben M'hidi University of Oum El-Bouaghi,
LIRE Laboratory at the University of Constantine 2,
Constantine, Algeria
meriem.zaiter@gmail.com

Salima Hacini and Zizette Boufaida
LIRE Laboratory at the University of Constantine 2,
Constantine, Algeria
{salimahacini, zboufaida}@gmail.com

***Abstract*— In recent years, the use of component-based systems has become increasingly large in the daily life such as domestic applications. In addition, the diversity and the dynamic components that can build them make this type of system very awareness. For this reason, the assurance of dependability and safety execution are required in order to propose a better system performance and the best user satisfaction by providing services continuity which consequently leads to reliability. That is really a challenge problem. Our goal in this paper is to propose an adaptation mechanism, based on the mirror services, to make such a system more efficient, and thus, more and more operational, even the existence of faults in it. To this end, the fault tolerance is a good solution. So, the contribution in this paper is based on a set of algorithms that will be employed by a set of local agents controllers and one global agent controller.**

Keywords-dependability; agent; fault detection; fault tolerance.

I. INTRODUCTION

Multi component system is composed of many different components, each of which is an individual system. The complete system has a set of fixed functionalities. Every component may have a varied composition and implementation. As an example, we find the domestic applications, which use a computer component in the home environments. In this space, the wireless communication between the components and the sensors devices are generally used. So, that system is very awareness and dysfunction error localization is a delicate task. Indeed, an abnormal execution in this kind of system can be caused by the failure of any component, which can imperatively causes a dysfunction of the overall system. So, it is indispensable to detect such situation before the crash of the overall system. One way to insure dependability of systems is to allow continuity of execution in the case of fault occurrence, which is the aim of this paper. A promising technique to do this; is the fault tolerance, which is defined as the ability of system to continue normal operation despite the presence of faults.

Our fault detection mechanism [1] focuses on the use of a global controller and a set of local controllers, which aims first to detect the fault whatever its nature then applies the algorithms to support it automatically, and adapt the execution to the suitable context. In this paper, our goal is to enhance the mechanism with the use of both the agents and the replication advantages. The replication is considered as one of the basic tools in a fault tolerance technique [2].

Moreover, the agent technology has been largely used and gives an interesting proposition to various problems such as e-commerce, distributed computing telecommunication networks services, monitoring and notification [3], etc. They provide several advantages, in the dependability area, the fact that an agent [4]:

- Has the ability to communicate
- Can migrate from a defective component to another in order to continue its execution, by the weak or the strong mobility characteristics.
- Can keep track of the execution follow.
- Can be duplicated and cloned as needed, or killed for example in a case of its failure
- Uses of low-cost and a low-power requirements when it is executed on an equipment
- Etc.

Also we find that the interaction agent-agent is exclusively via message-passing communication and the asynchronous message-passing have good scalability characteristics.

The remainder of the paper is devoted to the details of our agents based fault detection mechanism. Section 2 gives a state of the art of the fault detection techniques. Section3 presents an overview of faults' types that may affect the normal function. Section 4 details all the algorithms that handle the detected faults. Finally, a conclusion achieves this paper.

II. RELATED WORK

Fault tolerance is an indispensable characteristic required by different types of computer systems and specifically distributed systems. The latter can fail due to the failure of its components; why researchers are still trying to find a way to ensure dependability by fault tolerance. In some studies, [5, 6] the authors propose a service migration from one component to another to ensure a permanent presence of service despite it is being required; this can be insured through three mechanisms, the first is used to manage the context of interaction among the system components; the second is employed to specify the rules according to the current context and the changes that may occur, and the third one identifies the migrated service. This approach has some disadvantages:

- It is only applied in a context where all components of the system have the same architecture on which mobile service can migrate.

 ISBN: 978-1-61208-301-8

- Regardless to the state of the component the service should automatically migrate (even if the component was in a good state). So, unnecessary transmissions can be realized. In addition this technique is ineffective in case of a sudden fault (there is no fault detection) [5].

There are also some mathematical methods for fault detection. For example, in [7,8] the approach is rule-based. It requires first a definition of a fault model that categorizes the occurred fault as short, constant or noise, then based on the standard deviation between the later model and the current system state a fault can be detected. These methods [7][8] require a knowledge on the domain to identify the faults' type and a careful specification of the standard deviation threshold.

The method used in [9] also requires the definition of the types of faults that can occur in order to detect them on run-time. Some other software techniques [10][11][12] add a set of instructions to control the flow of execution, and thus detect the existence of fault by comparing for example a duplicated variables values with the variables themselves [12]. Generally, these techniques rely on external equipment to handle the fault that will certainly cause a significant perturbation treatment.

Some techniques incorporate exceptional behaviors during the entire development of fault tolerant distributed systems implemented within component [13]. Other model introduced in [14][15], specify the normal and exceptional behaviors of system components; so while exceptional responses, errors are detected.

In spite of the number of solutions to insure fault tolerance, the fault problems' detection and support persist and not treated definitely. On the other side, the requirement of safety running of systems and the availability of delivered services is very required.

III. THE FAULTS'S CATEGORIES

An error is the manifestation of a fault in the system, and a failure is the manifestation of an error on the provided service by the system [16]. The fault type plays a very important role if we want to get a fault tolerance. Moreover, faults can be categorized according to several criteria, like the degree of severity, degree of permanence and their nature.

The based component systems are considered as context-aware systems because a communication context varies from one moment to another. So, in order to classify faults, we exploit, in this section, the following definition of context [1] "*Context is any internal or external information, related to an entity, could be used and have an impact on the future state of the application. This information can be linked to one or more entities. The latter, regardless of their nature (hardware / software / human), can trigger events that affect the global state system. To this end, the occurrence of a fault causes certainly an immediate dysfunction to the global system*".

So, we suggest classify the faults on the base of the faults's sources, their manifestations and their persistence. One component in a component-based system may announce a dysfunction on behalf of another component, if the latter did not satisfy the needed request.

Also, failures can be a result of an improper use of the system by the user or due to:

- Software errors: that can be an arbitrary deviations related to the code,
- Materials errors: that can be the shutdown of a component or its internal constituents, or
- Transmission errors: such as the omission of sending or receiving messages or even to malicious attack (citing as an example an injection of a code into the system, by a malicious user, can cause a deviation of the normal execution flow).

In Table 1, the source of the fault is related to the element of context. The persistence of a fault means its duration; it may be transient or permanent. A permanent fault is a fault which requires a software maintenance or human intervention.

TABLE I. THE FAULTS' CATEGORIES

Elements of context		**Categories of faults**	
		external fault	***internal Fault***
Entity	Hardware / user	An error of interaction (such as error identification, or an input mistake ...)	/
		(transient / permanent) fault	
	Hardware	/	–Error referencing of an internal component (processor, memory…) –internal hardware failure
			permanent fault
	Software/ user	An entry outside the domain specification of the application	/
		(transient/ permanent) fault	
	Software	/	Design fault in the application itself
			permanent fault
Temporal aspects (date, time)		Fault in scheduling and synchronization of messages among system components	– the local Clock is not synchronized – physical error due to a transmission problem
		transient fault	(transient/ permanent) fault
Location		Localization problem of neighboring entities (effect of fog in the environment)	Fault in the physical controllers of component
		transient fault	permanent fault

After the presentation of the proposed faults' categories, the next section details our proposition that aims to describe how the presented kind of faults are detected and supported.

IV. THE DETAILS OF OUR PROPOSITION

Dependability of our system is associated with the dependability of its components. It can be provided by insuring the availability of the exchanged events and services among the system components (each component will be called entity "ei") (Figure 1).

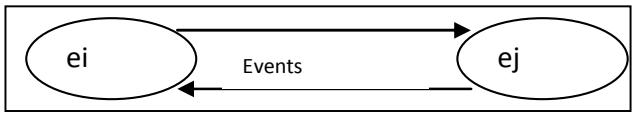

Figure 1. the dynamic exchange of events in the system.

To ensure a high dependability of the system and its entities, we focus, as it is explained in the introduction, on the advantages of agents as well as the replication. To this end, some tools (local and the global agents controllers (see the next sub section)) and the next constraints must be set:

- Each entity (ei) proposes a set of services OS (ei) that can be provided in the form of quadruplet:
  OS (ei)={(s0, child(s0), c0 , a0),(s1, child (s1), c1,a1),(s2, child (s2), c2,a2)…} where child (si) are all sub-services managed by the service "si" ," ci" and "ai" denote respectively the cost and the availability of the service (the latter takes the value available or busy). These quadruplets are sent to the global agent controller by the entity.
- For each connection of an entity (ei) to the system, two tasks will be performed:
  - An entity sends its request.
  - Upon receiving the entire answers, the entity defines all the functional dependencies D (ei) and sent them to the global agent controller. The Functional dependencies D (ei) are defined by a set of pairs, as an example (D (ei) = {(ej, sk) ...} where sk is the delivered service by ej to ei.
- Each exchanged message must be double signed (through an hash function) by both the entity and its agent controller that helps in the control task of the local agent controller operation itself. So, if an agent controller does not sign its message a "Raise not-signed" event will be reported.
- A duplicated global agent controller is set and updated periodically in order to take the control task if the principal global agent controller fails.
- Each agent is supported by an entity and it (the agent) will be killed if this entity fails or disconnects; except the global agent controller and its cloned agent (duplicated agent) which will migrate, if the entities where the agents are running fail.

A. *The global agent controller*

The global controller is seen as an agent. It provides a set of functionalities:

- Manages the faults' detection and,
- Takes charge the occurrence of faults.

So, switch the kind of the received event the agent performs the suitable action. To do this, the global agent controller (Gac) must has a set of information in database knowledge. That contains an entry for each entity composed of (see, table 2):

- the identity of the entity noted ei,
- its state as a set of pairs (ss, state) where each pair represents a name "ss" and the state (that can has the value "good" or "bad") of every its provided service.
- a description of the list of the offered services of the current entity and all its functional dependencies
- The execution state "ESi" which represents the status of the executing operation, on the entity, which is periodically updated.

TABLE II. THE REPRESENTATION OF THE INFORMATION THAT CHARACTERIZES AN ENTITY AT GAC.

The entity	state	offered Service	Dependence	Execution state
e1	(s1, good) (s2, bad)…	(s1, child (s1), c1, a1) …	(e2, s4) ……	ES1

1) *The global agent Controller as a manager of a fault detection*

The operations of the global agent controller are detailed through a set of algorithms (see Figure 2), some of them used to manage events flowing through the system, as the indication of a fault of an entity ei:

- By its local agent controller (see Figure 2, instruction 10) or,
- By another local agent controller (other than its own agent controller) (see Figure 2, instruction 15).

Upon receipt of the entire functional dependencies of a given entity (ei) the global agent controller exploits them to update the availability of services (instruction 9 in Figure.2) from "available" to "busy".

The other types of events are detailing in the next algorithm:

```
1 Input: event
2 Begin
3 Repeat
4       case (event) of:
5       nw_elt:
6       Creat and send the local agent controller Laci
7       Load _DB (ei)
8       D(ei):
9       Update availability (D (ei))
10      alert (Laci , ei,ss) :
11      P← takecharge(ei,ss)
12      Send fault (ei, Gac,ss) to every element of P
13      For every controller element of (P) do: Research (E,S)
14      For every controller element of (P) do: Send Mirror
        (Gac, em, em)
15      alert (Laci , ej,ss) or good_state(Lacj ,ej,ss) :
16      Check_state (ej,ss),
17      Raise not-signed (i): preparing and send a new Laci
18Until (false)
19End
```

Figure 2. the global agent Controller algorithm.

– **Terminology**

nw_elt: means that a new entity is connected to the overall system.

Research (E, S): allow proposing a set of mirror services (ms) to the entity affected by the fault.

Load _*DB (ei)*_: adds to the database an entry contains the information concerning the entity (ei) (state (ei), D (ei) ...) (see Table 2).

Extract_state (ej, ss): This function enables the recuperation of the service state ss (ej) from the database (see Table 2).

Raise not-signed (i): this event means that there is a dysfunction on the Laci, so a substitution of the defective Lac must be done. Therefore, the new agent takes the needed information like (the most recent value of the execution state ESi) and continues the control of its entity.

Check_state (ej,ss) is a verification function (see Figure 3) for testing the entity state.

All the other instructions will be carefully explained in their appropriate context.

```
1 Input : entity : ej, service ss
2 Output: state of ej
3 Begin
4          State ( ej,ss) ←Extract_state (ej,ss);
5   if (state (ej,ss) =bad) then
6          send fault (ej,Gac,ss) to Laci
7   else
8          send Rv (Gac,ej,ss) to Lacj
9          if (Rep (ej,ss)) then
10         send good_state (Gac, ej,ss) to  Laci
11         else
12         P← takecharge(ei,ss)
13         send fault (ej, Gac,ss) to every element of P
14         For every element of (P) do research (E,S)
15         For every element of (P) do Send Mirror(Gac, em, em)
16         End if
17  End if
18End
```

Figure 3. Check_state function.

Rv (Gac, ej, ss) is a verification request sent by the global agent controller to (ej) in order to test its state.

2) The agent global Controller as a responsible of the fault tolerance

Upon the confirmation of the detection of a fault the global agent controller performs the two following tasks to support the fault:

- Declares the entity (ei) as partially defective, in service s, by following the algorithm in Figure 4, which may indicate the fault of the service provided by an entity to all its dependencies (see Figure 2 instruction 11and 12; Figure 3 instruction 12 and 13):

```
1 Input : (ei, s)
2 Output : list of pair P of (entity e, service s)
3 Begin
4     For j=1 to NE do
5         repeat
6          Dt(ej)← D(ej)
7         (en,sn) ←  extract an element from  Dt(ej)
8         if (en = ei) and sn belong to {s} U {child (s)} then
9                 insert P(ej,sn)
10                update_state (ei,s)
11        End if
12        until Dt(ej)= Φ
13    End For
14        return (p)
15 End
```

Figure 4. Algorithm of the takecharge function.

NE: indicates the number of entities in the system

The procedure update_state aims to update the operational state, by " bad", of the defective service "s" and its child(s).

- The second sub-task of the global controller is to make sure the continuity of system and the service deliverance by following the algorithm (in Figure 5) that aims to research the similar services (see Figure 2, Instruction 13; Figure 3, Instruction 14)

(es, ec, ae) : corresponds to the elected service

```
1 Input : (ei, s)
2 Output : list of pair P of (entity e, service s)
3 Begin
4         ae ←  busy// the availability of service
5         State e←  bad // the operational state of service
6         es←s
7         ss←es
8         ec←max (c)
9          elect←i
10  For j=1 to NE do
11         if ((ss = es) and (ae =available)
           and (sc <= ec) and ( state s = good))  then
12                (es, ec, ae) ←  (ss, sc, ae)
13                elect← j
14         End if
15  (ss, sc, ae) ←  extract an offered services from the table
16   State e ←  state (ss)
17  End For
18   if (State e = bad) then
19         return (Φ , Φ )
20   else
21         return (e elect , es)
22   End if
23 End
```

Figure 5. The function Research

B. The local agent controller

To insure our control a set of a local controllers have been used; these controllers are seen as a local agents noted (Lac), one for each entity. Our suggestion of implementing an agent controller is based on the idea of self-testing, which allows individual control of each entity. This local controller executes a set of tasks that allows it to control the operational state of the entity. Faults are declared if an entity deviates from this normal operation:

- the entity sends and saves a simple request (RS (ei, ej, ss), (save (RS (ei, ej, ss))), waiting for answers (waiting (RP (ej, ss))) (see Figure 6 instructions 7 thru 9), and a possible definition of functional dependencies (see Figure 6 instructions 7 thru 9),

So, the abnormal functional of an entity is represented as events sent by the agent. Such as sending an alert (alert (Laci, ej, ss)) if an entity ej has promised the entity ei to ensure the service ss and it has not responded, or if it returned a wrong result. Indeed, each controller provides the following tasks:

1) Keeping track of execution in order to capture the execution state, this will be used in a fault recovery (see Figure 6, instruction 42).

2) Receiving the external events coming into the entity ei. In this case an external event can be:

- a simple request from an entity ej (Figure 6, instruction 5);
- a negative feedback from the Lacj, the controller of an entity ej, that is resulting from an eventual previous interaction with the entity ei;
- An inquiry concerning the entity ei, or an information failure of an entity ej if ei depends functionally from the defective entity ej (this event is raised by the global agent controller) (Figure 6, instruction 10…);

3) Informing the failure of its entity ei (see Figure 6, instruction 18, 22,…); in the case of a no-response to the periodical test of inspection performed by the agent controller Laci itself,

The clarification of the terminology used in the next algorithms (in Figure 6 and Figure 7) is explained bellow:

RS (ei, ej,ss): it is a simple request send by ei to ej requesting the service ss.

RP(ei,ss/Gac): it is an answer for a request sent by the entity (ei) (or Gac).

Fault (ej, Gac,ss): it indicates a failure of an entity ej, at the service ss, reported by the global agent controller.

Rep (ei,ss): this is a Boolean function. It represents the answer or not of ei to the local test relating to the service ss, triggered by the local agent controller Laci .

Check_ local _state (ei,ss, t): it is a function that represents the local test triggered after a time t. This function has a value 0 if the service ss of the entity ei did not answer, and 1 otherwise (see Figure *9)*.

Time: it represents the duration between two periodical tests.

Alert (Laci , ei,ss): It denotes a failure of a service ss of the entity ei reported by the agent controller Laci .

good_state (Laci , ei,ss): it is an event emanated from the local agent controller Laci. It shows that the service ss of it entity ei is in a good state.

good_state (Gac, ei,ss) it is an event emanated from the global controller. It shows that the entity ei is in correct state.

fd (ej,ss): denotes a promise from the entity ej to perform the service ss (functional dependency in the service ss).

Verify_Rq (ei, ej,ss): the role of this function is to check if a request emitted by an entity ei is being processed by an entity ej or not.

Save (RS(ei ,ej,ss)) : its role is to save the request (sends by the entity (ei)) that will be processed by the entity (ej)

Wait (RP(ej,ss)): it aims to start the control of the duration of the response of an entity ej to the ei request's.

Mirror (Gac, em, sm): indicating the elected service mirror " sm" and the identity of the entity that provides them. In order to ensure the continuity of operation of the overall system.

```
1 Input: event;
2 Begin
3 Repeat
4  case (event) of:
5   RS(ej ,ei,ss) :
6          save (RS(ei,ej,ss)) ;
7          Send fd((ei,ss) to Lacj
8          Treat  (RS(ei,ej,ss)) ;
9          send (RP(ei,ss)) ;
10 fault (ej ,Gac) OR alert(Lacj, ej,ss) :
11     If (Verify_Rq (ei, ej,ss)) then
12                 Cancel  (RS(ei,ej,ss))
13    else
14                 state (ej)← bad
15    End if
16 alert (Lacj, ei,ss):
17    if (state (ei,ss)=bad) then
18         send  alert (Laci ,ei,ss) to Gac
19         send  alert(Laci ,ei,ss) to Lacj
20    else
21    if ((check_ local _ state (ei,ss,0))=0) then
22         send  alert(Laci, ei , ss) to Gac
23         send  alert(Laci ,ei,ss) to Lacj
24    else
25         send good_state(Laci ,ei,ss) to Lacj
26         send good_state(Laci ,ei,ss) to Gac
27   End if
28   End if
29  fd (ej,ss):
30         update dependence D(ei)
31         Send D(ei) to Gac
32 good_state(Lacj , ej,ss) and not RP(ej,ss):
33         send RS(ei,ej,ss) to Lacj
34 good_state(Gac, ej,ss) and not RP(ej,ss):
35         send RS(ei,ej,ss) to Lacj
36         save (RS(ei ,ej,ss)) ;
37         wait (RP(ej,ss));
38 Mirror (Gac, em, em)
39          send RS(ei,em,sm) to Cm
40          save (RS(ei ,em,ss)) ;
41          wait (RP(mj,ss));
42         continue the execution from the current state
43 good_state(Lacj,ej,ss) and not RP(Gac):
44         send RS(ei ,ej,ss) to Lacj
45         raise degraded mode
46         activate the duplicated Gac
47 until (false)
48 End;
```

Figure 6. The agent local controller algorithm.

The periodical test is performed by the execution of the "check_ local _ state" function (see Figure 7) that will be performed by the local agent controller, every a given time t or in any other necessary moment like in the case of the instruction 21 in Figure 6.

```
1 Input: entity ei,ss, time t;
2 Begin
3       if (not (Rep (ei))) then
4               state (ei,ss)← bad
5               send alert(Ci ,ei,ss) to Gac
6               t←time
7               Return (0)
8       else
9               state (ei)← good
10              t←time
11              Return (1)
12      End if
13End
```

Figure 7. The function check_ local _state.

This last function (Figure 7) is used by the agent Lac(i) to control the operational state of entity ei and all its offered services.

V. CONCLUSION

This paper contains an effective contribution in fault tolerance area applied to component-based systems. First, some faults' categories have been established by giving an overview of the errors' types, then this paper describes our reflection to insure a high dependability by a fault tolerance, which is based on a global agent controller and a set of local agents controllers. Diverse situations (theoretical scenario) have been treated: (1) even a fault, insuring the continuity of delivering services, in a right way, by exploiting the agent ability of keeping tack to capture the recent execution context. (2) Insuring continuity of control even a dysfunction at the global agent controller itself or at one or more local agent controllers, through the use of the following features: the replication and the migration ability, etc. In order to validate the proposed mechanism, a simulation of a domestic application is on the way with the purpose of giving some statistics; and improving our theoretical. We have chosen an application for monitoring a patient at home, on which we have selected a set of adequate components, some components are strongly coupled and other are not, to inject faults and test how the system react, etc.

REFERENCES

[1] M. Zaiter, S. Hacini, and Z. Boufaida "Towards a Functional Control of a Context-Aware Systems", Information Studies: Online ISSN: 1911-8414, vol. 4, no.1 January 2012, pp, 9-17.

[2] C. Leangsuksun, T. Liu, L. Shen, and S. L. Scott. "Building high availability and performance clusters with ha-oscar toolkits". Proc. the High Availability and Performance Workshop, Santa Fe, NM, October, 2003.

[3] D. B. Lange and M. Oshima, "Seven Good Reasons for Mobile Agents". Communications of the ACM, vol. 42, no.3, March. 1999.

[4] H. Paulino "An Overview of Mobile Agent Systems'", Technical Report Series: DCC-02-1. February. 2002.

[5] O. Riva, J. Nzouonta, and C. Borcea. "Context-Aware Fault Tolerance in Migratory Services", MobiQuitous 2008, Dublin, Ireland, July 21- 25, 2008.

[6] R. Handorean, R. Sen, G. Hackmann, and G.-C. Roman. "Context Aware Session Management for Services in Ad Hoc Networks". Proc. IEEE International Conference on Services Computing (SCC'05), July, 2005, pp, 113–120.

[7] Ramanathan et al. "The Final Frontier: Embedding Networked Sensors in the Soil", Tech. Rep. 68, CENS. November. 2006.

[8] A. B. Sharma, L. Golubchik, and R. Govindan "Sensor faults: Detection methods and prevalence in real-world datasets" Journal ACM Trans. on Sensor Networks TOSN Vol. 6, Issue. 3, A23, ACM New York, NY, USA, June. 2010.

[9] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom,. "Declarative Support for Sensor Data Cleaning". Proc. 4th International Conference. Dublin, Ireland,, pervasive 2006.May 7-10. 2006, pp, 83-100.

[10] N. Oh, P.P. Shirvani, and E.J. McCluskey, "Control Flow Checking by Software Signature" IEEE Trans. on Reliability, vol. 51, no. 1, 2002, pp, 111–122.

[11] Cheynet, et al. "Experimentally Evaluating an Automatic Approach for Generating Safety-Critical Software with Respect to Transient Errors," IEEE Trans. on Nuclear Science, vol. 47, no. 6, 2000, pp, 2231–2236.

[12] M. Rebaudengo, M. Sonza reorda, and M. Violante "A New Approach to Software-Implemented Fault Tolerance". journal of electronic testing: Theory and Applications 20, Kluwer Academic Publishers. United States. 2004, pp, 433–437.

[13] C. M. F. Rubira, R. de Lemos, G. R. M. Ferreira, and F. C. Filho. " Exception handling in the development of dependable component-based systems". Softw. Pract. Exper., 35(3): , 2005,pp,195–236.

[14] P. Lee and T. Anderson. "Fault Tolerance: Principles and Practice", Second Edition. Prentice-Hall, 1990.

[15] A. Bucchiarone "Architecting Fault-tolerant Component-based Systems: from requirements to testing"; Electronic Notes in Theoretical Computer Science 168 Elsevie, 2007, pp, 77–90.

[16] J.P. Blanquart "Sûreté de Fonctionnement des systèmes embarqués critiques Les enjeux industriels (Domaine Spatial) (Astrium Satellites)" ETR'09 Ecole d'Eté Temps Réel, TELECOM ParisTech, Aug 31-Sept 4, 2009 (in French).

 ISBN: 978-1-61208-301-8