UKACC International Conference on Control 2012
Cardiff, UK, 3-5 September 2012

Graph Theory : Application to System Recovery

Ahmed Mekki
Univ Lille Nord de France F-59000 Lille, France,
EC LILLE, LAGIS, F-59651, Villeneuve d'Ascq, France
Email: ahmed.mekki@ec-lille.fr

Simon Collart-Dutilleul
Univ Lille Nord de France F-59000 Lille, France,
EC LILLE, LAGIS, F-59651, Villeneuve d'Ascq, France
Email: simon.collart_dutilleul@ec-lille.fr

Abstract—**The aim of the work presented in this paper is to introduce a method for assisting the recovery of a given system in case of failure detection. The proposed method rely on a graph-based algorithm that allows the identification of the alternative system configuration. In this way, the method guarantees the system under study functionalities/missions even in case of fault. The method is detailed in the sequel. Furthermore, in order to provide user with automated means which are at the same time simple, intuitive and rigorous, the whole of the developed mechanisms have been implemented in a prototype tool with an intuitive graphical interface that offers interesting facilities in terms of system recovery. The method is illustrated using an intelligent and autonomous vehicle case study.**

I. Introduction

Life-critical systems are systems that its dysfunction could cause human-life death as well as an important equipment damage or loss. Thereby, this kind of system (e.g. transportation systems, intelligent and autonomous vehicle, nuclear plants, manufacturing systems, medical devices) must achieve a high level of robustness, availability, reliability and safety. Usually, system life-cycle could be divided into two main phases : before implementation and after implementation. The former phase rely on some research topics such as specification, modelling design and V&V (validation and verification). For the later phase, one can find maintenance and system recovery in case of failure detection. In our case, we use system recovery to design a re-configuration or a new arrangement of the system functional units in case of failure. Various are the causes of failure: performance problems like access to shared resource (resource contention), hardware faults, software bugs, system operators misconfiguration, . . . [1]. Nevertheless, all failure causes are often due to a material, soft dysfunction or operator manipulation.

Given the human/material impact of critical system dysfunction, such systems must guarantee the availability of its services/missions. Availability express the quality of being at hand when needed. This including the case when failure occur. Therefore, a recovery technique is strangely recommended. The aim of the work presented here is to guide the user during the system recovery and control phase.

Intelligent and autonomous vehicle (IAV) is a vehicle that is expected to achieve different tasks without the intervention of a human operator. Several projects rely on IAV but the most famous ones are the NASA's rovers, for instance the Mars Exploration Rover Mission (MER) [3]. MER is an ongoing robotic space mission involving two rovers, Spirit and Opportunity, exploring the planet Mars. It began in 2003 and its cost raises to more than US$ 900 million. Actually, to guarantee expected tasks, IAV rely on set of services provided by several hardware components (sensors, actuators, . . .) as well as software components. Nevertheless, due to failures, it is possible that one or more services are no longer available and thereby the achievement of some tasks becomes no longer guaranteed. IAV design rely on fault tolerant control procedures in order to define strategies allowing the system to continue its operations with the required performances despite component faults [4].

As mentioned previously in this paper, we focus on the system recovery step in case of failure detection. In fact, the idea is to propose assisting means that are easy to manipulate and, at the same time, accurate. However, the more accurate and rigorous a notation is, the more abstract and difficult to handle and understand it becomes. Therefore, one of the challenges that we faced while dealing with this work was to look at both intuition/simplicity and rigour/accuracy. To deal with this, we propose a graph-based algorithm able to cover system configuration and able to identify the alternative configuration in case of system-down. However, in order to hide all the formal aspects -met when dealing with system recovery- from user, a GUI tool have been implemented to automate this step.

The paper is organized as follows: in Section II, an overview of the context and related works is given. In Section III, we present the algorithm that we have proposed. The developed tool is introduced in Section IV. The method is illustrated using an intelligent and autonomous vehicle case study in section V before concluding and suggesting some future works in Section VI.

[1]This research has been supported by CISIT project.

978-1-4673-1558-6/12/$31.00 ©2012 IEEE

II. Context and Related Work

Availability of a system is its capacity to achieve its missions/services in the occurrence of the failure of (or one or more faults within) some of its components [5]. One of the main properties that characterize availability is fault-tolerance [6]. Here, we focus on life-critical systems and thereby, given their failure cost, fault-tolerance is particularly sought-after such systems. In practice, some fault can cause a system failure by propagating the fault to the rest of the system. Therefore, fault tolerant systems (FTS) must deal with multiple failure types and thereby, must achieve fault isolation capacity and reversion modes availability (redundancy). Actually, FTS are typically based on three main concepts namely replication, redundancy and diversity [6]. These concepts could been defined as follows:

- Replication means providing multiple instances of the same system. Then, tasks/jobs are directed to all system instances in parallel. The correct result is determined by a quorum;
- Redundancy means providing multiple instances of the same system/service and switching to one of the remaining non-faulty instances, in case of a failure-detection;
- Diversity means providing various system implementations in order to deal with with errors in some specific implementation.

It should noticed that before recovery can be carried, one must first detect and diagnose the failure. Indeed, failure detection is to determine the instances while a system is facing dysfunctions. Afterwards comes the failure diagnosis which allow to locate the source of detected failure. Failure detection as well as failure diagnosis are not in the scope of this paper.

Severals studies dealing with system recovery have been proposed and published [6], [7], [8], [9], [10], [11]. Unlike this meantionned studies, our work propose a graph-based approach to tackle the system recovery issue. Furthermore, a GUI tool have been implemented to automate this step.

III. Graph-based Method for System Recovery

A. Idea

Let us here recall the aim of our study: we want to provide the user with simple means for assisting the system-recovery in case of failure detection. Our idea is to elaborate a supporting approach which hides the formal foundation to the user. Concretely, we will develop a GUI-tool that automate all the steps of system-recovery procedure.

The idea is to express the service-architecture of a system with a (directed) graph. In this graph the root element represents the system while the leaf represent the elementary services. We assume that a system is defined as a (non-empty) set of exploitation modes. At a given instance, only one exploitation mode is active. Each exploitation mode is composed of a (non-empty) set of missions. All missions of active exploitation mode should be active. Multiple versions are defined for each mission and thereby, at a given instant, only one version is active. A mission version is composed of (non-empty) set of service that, all of them, should be active. Various version of each service could be defined but only one service version is active at a given instance. A version service is (non-empty) set of elementary services that should be active, all of them. This description could be expressed as follow:

system	= OR({Exploitation-Mode})
Exploitation-Mode	= AND({Mission})
Missions	= OR({Version-Mission})
Version-Mission	= AND({Service})
Service	= OR({Version-Service})
Version-Service	= AND({Elementary-Service})

Where

- $E = AND(s)$ means that an element E is active if all the elements composing it are active and vice-versa;
- $E = OR(s)$ means that an element E is active if one and only one of elements composing it is active and vice-versa.

B. Foundations and Problem's Formalization

A graph $\mathcal{G}$ [12] is a pair $\mathcal{G} = (\mathcal{V},\mathcal{E})$ where

- $\mathcal{V}$ is a finite set of vertices (nodes), and
- $\mathcal{E}$ is the set of edges, formed by pairs of vertices, $\mathcal{E}$ is a subset of $\mathcal{P}_2(V)$ ($\mathcal{E} \subseteq (\mathcal{V} \times \mathcal{V})$).

Graphically, a graph is pictured by drawing a four-square (or point) for each vertex and representing each edge by a curve joining its endpoints.

Usually, *a simple graph* denotes a graph having no loops or multiple edges where each edge $e \in \mathcal{E}$ can be specified by its endpoints $(a, b) \in \mathcal{V}$ and is denoted $e = ab$. In this case, we say a and b are *adjacent*. Thereby, a path is defined as a set of consecutive and ordered nodes so that two nodes are adjacent if and only if they are consecutive in the ordering.

A graph variant is called directed graphs $\mathcal{D} = (\mathcal{V}, \mathcal{E})$, where the edges have a direction. In other words, edges are ordered and $ab \neq ba$. Graphically, the edges are drawn as arrows.

C. System description

In our case, a system is defined as directed graph $\mathcal{DS}$=($\mathcal{V}$, $\mathcal{E}$) where

- $\mathcal{V}$ is a finite set of nodes, $\mathcal{V} = Sys \cup EM \cup M \cup VM \cup S \cup VS \cup ES$:
    1) Sys is the root element of the system under study,
    2) EM is the set of exploitation modes of the system under study,
    3) M is the set of missions defined by EM,

4) VM is the set of mission version (for each mission, one can define at least one mission version),
5) S is the set of services composing missions,
6) VS is the set of service version (for each service, one can define at least one service version),
7) ES is the set of all elementary services proposed by the system.

- $\mathcal{E}$ is a finite set of edges, $\mathcal{E} \subseteq (Sys \times EM) \cup (EM \times M) \cup (M \times VM) \cup (VM \times S) \cup (S \times VS) \cup (VS \times ES)$.

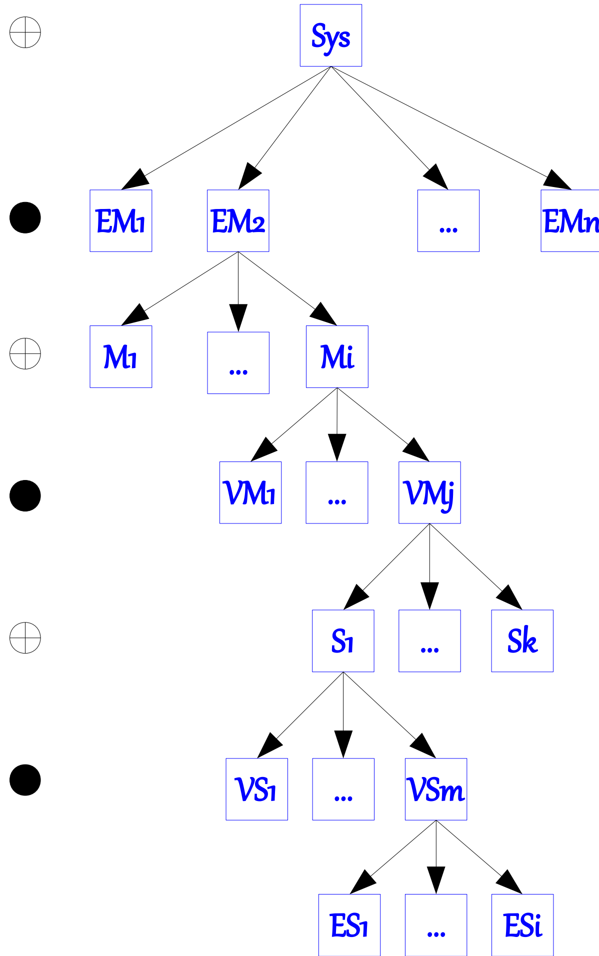

Fig. 1. System description as a directed graph

Figure 1 depicts graphically the system definition given in this section. To this definition, we add a set of binary relations (using AND and OR binary operators) between elements. Indeed, these relations define the type of composition between nodes, given above in section III-A. Actually, for each element type (node), a symbol is linked. Two kind of symbols are used: ⊕ and ●. Node linked to ⊕ is defined as OR-composition of its children. In the same way, node linked to ● is defined as AND-composition of its children.

D. Algorithm

Failure detection is the task of identifying system dysfunction. Different source of failure could be identified in a given system. A recovery step is switching to an exploitation mode where all (or a part) of system capabilities/facilities are guaranteed in case of failure. Thereby, the exact cause of error is often not required for recovery to take place. Nevertheless, an automated approach for the recovery step is necessary in order to improve system availability. In this section, we propose an algorithm that automate this step. Given a graph model of the system under study and given the detected failure, the algorithm determine and update the list of the alternative exploitation modes. Actually, based on the system graph, the algorithm determine a sub-graph (recovery tree) that represent the non-faulty nodes.

In practice, for each node, we add two attributes: *statut* and *activity*. *Statut* denotes the statement of a node: *functional* or *not-functional*, where *activity* denotes the activity statement of a node: *active* or *not-active*. Furthermore, the approach that we propose require that the failure lists all the element that are no longer available. In other word, the failure should be defined as the set of nodes that are no longer available in the current system description model.

The recovery algorithm rely on two parameters, namely the system under study graph and the failure (a list of faulty elements), and is composed of three main functions and is conducted as follow:

1) the *statut* attribute of all faulty element is set to *not-functional* within the graph model of the system under study,
2) For each node n in the list of faulty elements (the detected failure), the failure spread function, *FS(n)*, is carried,
3) Determine alternative exploitation mode

Failure spread function of a node n: *FS(n)*

Step 1: Set the *statut* attribute all children of n to *non-functional*,
Step 2: Set the *statut* attribute of n to *non-functional*,
Step 3: If parent of n is an AND-composition type element, then

1) n ← Parent_of(n),
2) go to **Step 2**.

Step 4: If parent of n is an OR-composition type element and if all sibling of node "n" are *non-fonctionnel*,then

1) n ← Parent_of(n),
2) go to **Step 2**.

Determine alternative exploitation mode

Once the failure is spread, the next step is to determine the alternative exploitation modes that could be activated. Actually, the aim of this task is to determine

the sub-graph where all nodes have a *statut* attribute value as *functional*. In other words, the alternative graph is composed only by nodes that statut is *functional*.

Let $\mathcal{G} = (\mathcal{V},\mathcal{E})$ be the graph of system under study and let $\mathcal{G}' = (\mathcal{V}',\mathcal{E}')$ be the sub-graph of $\mathcal{G}$ where all nodes are *functional* and is determined as follow:

Step 0: $\mathcal{V}' = \emptyset \wedge \mathcal{E}' = \emptyset$,
Step 1: if root of $\mathcal{G}$ is faulty, exit algorithm
Step 2: the root of $\mathcal{G}$ is chosen and is added to $\mathcal{G}'$,
Step 3: a walk[1] ω that starts from the chosen node is constructed by adding only *functional* nodes.
Step 4: if all nodes of $\mathcal{G}$ are tested, then the walk ω represents the graph $\mathcal{G}'$ (break algorithm) else go to step 5,
Step 5: Select the parent of the last node of ω, go to **Step 6**
Step 6: A walk ω' that starts from the chosen node is constructed by adding only *functional* nodes.
Step 7: $\omega \leftarrow$ concatenate(ω,ω'). Go to **Step 4**

It is worthy to notice that the obtained walk is a connected graph $\mathcal{G}'$ since the start node of this function is the root node. This connected graph is equivalent to the *spanning tree* for the undirected graph representation of graph $\mathcal{G}$. Once the *spanning tree* is returned, a product of all possible exploitation mode configuration is computed based on it. Thereby, from the list of all possible configuration, the user (operator) could choose one (and only one) to activate.

IV. Recovery Tool

The various mechanisms we have developed have been implemented within a software tool. This tool guides the user during the recovery step, it offers a quite intuitive graphical user interface. Actually, the tool rely on two main elements:

- Recovery algorithm: first, the developed tool displays the system description and lists its various parameters. Indeed, as shown above in the paper, the system description is based on a graph theory. A graphic representation of this graph is done ones its description (graph description of the system to check) is loaded. This representation is made on a tab within the recovery tool. Then, based on the algorithm discussed in section III-D and given a failure, the tool can automatically determine all possible alternative exploitation modes. Finally, the switch to the selected exploitation mode is carried.
- Data structure: since we aim that the developed tool can be used in combination with other tools within a global system supervision and control approach, we have chosen XML (Extensible Markup Language) [13] standard format for the input/output files. The main advantage of using the standard XML is that format of the encoding documents is both human-readable and machine-readable. Actualy, our tool rely on three different files: a system description which is an input/output file, the initial configuration file and the failure file, both are input files. The corresponding structure of each file has been defined by an XML schema. Listings 1, 2 and 3 define respectively the XML schema for the system description, the initial configuration and the failure.

Listing 1. System XML schema

```
<!ELEMENT Systeme (mode+)>
<!ELEMENT mode (nom?, Mission+)>
<!ELEMENT Mission (nom?, VMission+)>
<!ELEMENT VMission (nom?, service+)>
<!ELEMENT service (nom?, vservice+)>
<!ELEMENT vservice (nom?, eservice+)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT eservice (#PCDATA)>
<!ATTLIST mode
        activity ( Active | notActive ) #REQUIRED
        statut ( nonFunctional | Functional ) #REQUIRED>
<!ATTLIST Mission
        activity ( Active | notActive ) #REQUIRED
        statut ( nonFunctional | Functional ) #REQUIRED>
<!ATTLIST VMission
        activity ( Active | notActive ) #REQUIRED
        statut ( nonFunctional | Functional ) #REQUIRED>
<!ATTLIST service
        activity ( Active | notActive ) #REQUIRED
        statut ( nonFunctional | Functional ) #REQUIRED>
<!ATTLIST vservice
        activity ( Active | notActive ) #REQUIRED
        statut ( nonFunctional | Functional ) #REQUIRED>
<!ATTLIST eservice
        activity ( Active | notActive ) #REQUIRED
        statut ( nonFunctional | Functional ) #REQUIRED>
```

Listing 2. Initialization XML schema

```
<!ELEMENT Init (mode?)>
<!ELEMENT mode (nom?, eservice+)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT eservice (#PCDATA)>
```

Listing 3. Failure XML schema

```
<!ELEMENT Erreur
        (mode*,
        Mission*,
        VMission*,
        service*,
        vservice*,
        eservice*)>
```

In the sequel we will discuss the steps of a recovery process and we show how the tool facilities are helpful and useful when carrying this process (figure 2). From the menu bar (green box),

1) First, the user should start by loading the system specification. The uploaded description will be then drawn in the shape of tree (blue box),
2) Once the system description is uploaded, an initial configuration should be selected. In the same way, the initial configuration is then represented in the shape of tree (black box),

[1]A walk is a list $v_0, e_1, v_1 \ldots e_n, v_n$ of nodes and edges such that for $1 \leqslant i \leqslant n$, the edge e_i has endpoints v_{i-1} and v_i.

3) Then, relying on the uploaded failure file, the recovery algorithm is carried automatically. Afterwards, a box showing (red box) all possible exploitation mode is shown and from which a system mode configuration could be selected (to update the current configuration.

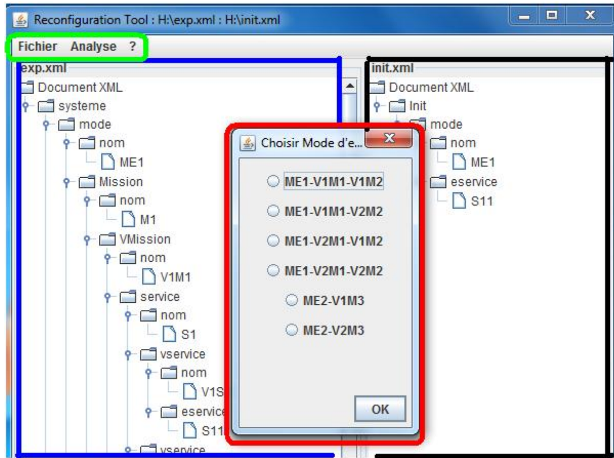

Fig. 2. Main GUI of the developed tool

In order to illustrate all this steps an IAV case study is presented hereafter.

V. Application to IAV system

A. System description

In real environment, an IAV could be repaired after the fault detection and isolation is carried. Nevertheless, within some critical environments, where the intervention (the necessary repairs) of a human operator is impossible, fault tolerance property must be provided to the IAV. In this case, IAV must be kept continuously operating even with a fault, such as when IAV operating in distant and/or dangerous areas. Consequently, IAV systems rely on service (component) redundancy in order to guarantee fault tolerance property. For illustration, let us consider an IAV described (Figure 3[2]) as follows :

- RoBuCar can carry two different exploitation modes : normal mode and degraded mode.
- Unlike normal mode where the four wheels are used, degraded mode rely on three or two wheels. Two main differences that distinguish the two exploitation modes: the maximum authorized speed and the weight of the transported goods for each mode.
- RoBuCar is expected to provide several missions : moving on, moving back, curbing, ...
- Each wheel is equipped with two electrical motors : only one is used to move the wheel. The second motor is used in case of dysfunction of the first motor.
- To move a wheel, four services are required : (1) the generation of an electrical power, (2) the conversion of the electrical power to a mechanical one, (3) the transmission of the mechanical energy to the tire and (4) a measurement service required to control the motor in closed loop.
  1) The generation of an electrical power rely on three elementary services namely: (a) supply the motor, (b) induce an electrical power and (c) limit the electrical current.
  2) The conversion of the electrical power to a mechanical one is composed by one elementary service.
  3) The transmission of the mechanical energy to the tire is composed of two elementary services namely: (a) keep all the pieces in rotation and (b) load service.
  4) The measurement service is composed of three elementary services namely: (a) measure the electrical current, (b) measure the velocity and (c) measure distances.

[2]RoBuCar : is an IAV at LAGIS laboratory built by the Robosoft Company http://lagis.ec-lille.fr/

Let us recall that, in order to keep system operating even in case of fault, service (component) redundancy is essential. Therefore, different versions of the same service as well as of the same mission are available.

Fig. 3. RoBuCar

B. Recovery procedure

The service graph associated to the RoBuCar system described above is depicted graphically in figure 4 (for simplicity reason, some details are omitted). The XML file describing this graph is used to carry on the recovery tool. Afterwards, in case of a failure detection, the recovery tool is carried.

For example, suppose that one of the four wheels is no longer available. In other words one of the two

electrical motors that equippe the wheel is no longer available. The XML file describing this failure is defined according to the XML schema given above. The failure XML file is needed by the recovery tool. Once the failure file is loaded, the tool, automatically, finds alternative exploitation modes to keep the vehicle working despite the failure. In our case, two alternative exploitation modes are retuned :

1) Keep the "normal mode" by proposing to activate the second electrical motor of the wheel.
2) Skip to the "degraded mode" where only three wheel are used.

The main advantage is that the procedure is automatic and rely on standard format (XML) for systems exchange (the system description, the failure description, ...). Therefore, its integration in operating system supervision approach can easily be done.

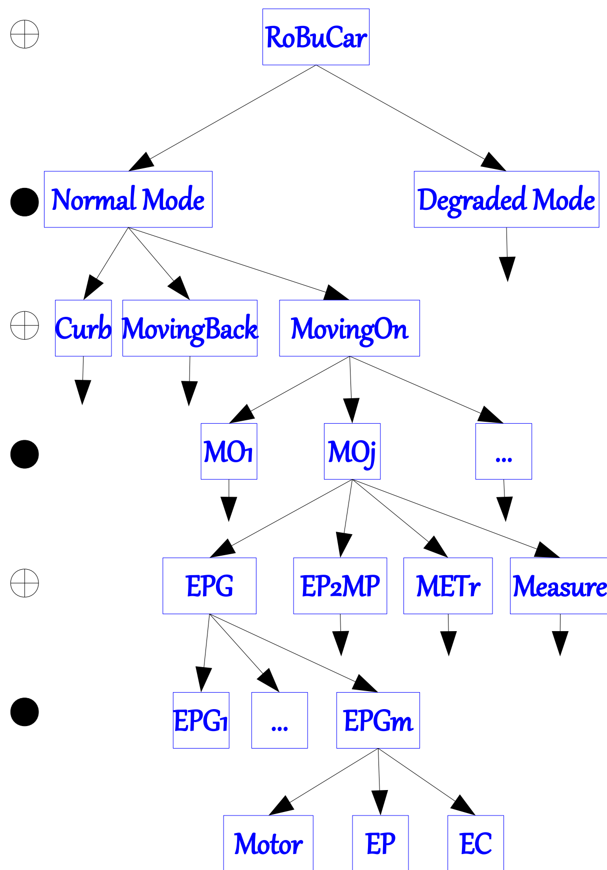

Fig. 4. Simplified version of the graph service of the RoBuCar

Therefore, the method allows to keep IAV continuously operating even with failures. In practice, this is very interesting specially for IAV operating in distant or/and dangerous areas where human cannot operate.

VI. Conclusions

In this paper we aim at introducing a means to assist and guide the user while carrying the system recovery in case of failure detection. The proposed approach rely on the graph theory. The main advantage of this notation that is formal and thereby, automatic tool using this notation could implemented. Actually, we started by proposing a graph-based description of the study system. Then, we defined an algorithm for the recovery system in case of failure detection. All proposed mechanisms we have developed have been implemented within a software tool. In order to illustrate the developed method an intelligent and automatic vehicle (IAV) case study was presented. First, a textual description of system was given. Then, the service graph is depicted on which the recovery tool is carried. Although, within some critical environments, where the intervention of a human operator is not possible, IAV must be kept continuously operating even with a fault, such as operating in distant and/or dangerous areas. Here we showed how the developped approach could, automatically, be used to detremine alternative exploitation modes.

References

[1] S. Pertet and P. Narasimhan, *Causes of failure in web applications*, Carnegie Mellon University Parallel Data Lab, Tech Rep CMU-PDL- 05-109, Dec 2005.

[2] http://esj.com/blogs/enterprise-insights/2011/06/ it-systems-failure-costs-quantified.aspx

[3] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz, *Global Planning on the Mars Exploration Rovers: Software Integration and Surface Testing*, Journal of Field Robotics, 26(4), April 2009, 337-357.

[4] N. Chatti, *Online supervision of intelligent vehicle using functional and behavioral models*, Intelligent Vehicles Symposium (IV), 2011 IEEE, p. 827-832, 2011.

[5] DP. Siewiorek and RS. Swarz, *The theory and practice of reliable system design*, Digital Press, Bedford, Massachusetts 1982.

[6] J. Bowen and V. Stavridou, *Safety-critical systems, formal methods and standards*, Software Engineering Journal, 8(4), p. 189-209, IET.

[7] N.G. Leveson, *Software safety: Why, what and how*, ACM Computing Surveys, 18, p. 125–163, 1986.

[8] E. Hammami, *Déploiement sensible au contexte et reconfiguration des applications dans les sessions collaboratives*, Thèse à l'Université de Toulouse, 2007.

[9] R. Sirdey, *Modèles et algorithmes pour la reconfiguration de systèmes répartis utilisés en téléphonie cellulaire*, Thèse à l'Université de Technologie de Compiègne, 2007.

[10] M. Staroswiecki and A-L. Gehin, *From control to supervision*, Annual Reviews in Control, vol. 25, p. 1-11, 2001.

[11] G. Bajpai, H.G. Kwatny and B.C. Chang, *Control systems perspective on safety critical systems*, 8th Asian Control Conference (ASCC) p. 413 -417, 2011.

[12] R. Diestel, *Graph Theory*, Springer-Verlag, Graduate Texts in Mathematics, Third edition, 173 pages, 2005.

[13] W3C, *XML specification 1.0*, http://www.w3.org/TR/xml/