AGENT-BASED FAULT MANAGEMENT OF EMBEDDED CONTROL SYSTEMS

Atef Gharbi[1], Mohamed Khalgui[2], Jiafeng Zhang[2] and Samir Ben Ahmed[1]

[1]*INSAT, Tunis, Tunisia*

[2]*Xidian University, Xi'an, China*

Keywords: Software Control Component, Intelligent Agent, Functional Safety, Queueing System.

Abstract: The paper deals with reconfigurable component-based embedded control systems to be safe when hardware or software faults occur at run-time. We define an agent-based architecture to handle automatic reconfigurations under well-defined conditions when run-time faults occur. We propose an implementation for the agent which maintains many queues to save run-time faults. This implementation aims to minimize the global waiting time of faults in queues. Multiple simulations are applied in the paper to find the best policy allowing an optimal reactivity of the system. We develop the tool "SimulatorAgent" to encode this approach that we apply to a Benchmark Production System.

1 INTRODUCTION

The[1] new generation of industrial control systems is addressing today new criteria as flexibility and agility (G. Pratl and Penzhorn, 2007). We distinguish two reconfiguration policies: *static* and *dynamic* policies such that static reconfigurations are applied off-line to apply changes before any system cold start (Angelov et al., 2005), whereas dynamic reconfigurations are dynamically applied at run-time (Al-Safi and Vyatkin, 2007). We are interested in automatic reconfigurations of an agent-based embedded control system when hardware or software faults occur at run-time. The system is implemented by different complex networks of Control Components (event-triggered software units) such that only one is executed at a given time when a corresponding reconfiguration scenario is automatically applied by the agent under well-defined conditions. We propose an agent-based architecture to handle automatic reconfigurations by creating, deleting or updating components to bring the whole system into safe and optimal behaviors when faults occur.

We aim in this paper to find the best solution for the optimal management of run-time faults in order to guarantee an optimal reactivity of the whole system. We assume three types of faults: the first type affects sensors of the plant, the second affects actuators and the last affects control components. The agent maintains many queues to save run-time faults. To decide what is the fault queue that the agent should choose first, we propose to evaluate the performance by applying four approaches (Priority/FIFO, Priority/Round Robin, Priority/Priority and Priority/Random). The measure of performance is based on the waiting time of a fault in a queue. A comparative study shows that Priority/Round Robin is considered as the best approach whereas Priority/Priority as the worst one. The simulation is ensured through the tool "SimulatorAgent" which enables to check the agent-based embedded control system when hardware or software faults occur at run-time.

We describe in the next Section the agent's algorithm ensuring optimal management of run-time faults. We present the experimentation in Section 3 and finally conclude the paper in Section 4.

2 AGENT'S ALGORITHM

By considering that a fault can affect a sensor, an actuator or a Control Component, we define a list of faults

[1]This work was supported in part by the Natural Science Foundation of China under Grant 60773001, the Fundamental Research Funds for the Central Universities under Grant No. 72103326, the National Research Foundation for the Doctoral Program of Higher Education, the Ministry of Education, P. R. China, under Grant No. 20090203110009, "863" High-tech Research and Development Program of China under Grant No 2008AA04Z109, the Research Fellowship for International Young Scientists, National Natural Science Foundation of China, and Alexander von Humboldt Foundation.

for each one of them. For each kind of faults, we associate a queue to save the occurrence of each fault, in particular the fault type, the occurrence time and the treatment time. Therefore, we have three kinds of queues: to save the faults affecting sensors, we use the sensor fault queue denoted by $Queue_j^S$ ($1 \leq j \leq N^S$, where N^S represents the number of all fault queues associated to sensor), to save the faults affecting actuators, we use the actuator fault queue denoted by $Queue_j^A$ ($1 \leq j \leq N^A$, where N^A represents the number of all fault queues associated to actuator) and for faults affecting Control Components, we define component fault queue denoted by $Queue_j^{CC}$ ($1 \leq j \leq N^{CC}$, where N^{CC} represents the number of all fault queues associated to control component).

The agent manages the system's reactivity when faults stored in queues should be treated. The general algorithm is based on well-known scheduling policies. Our goal is to have an optimal behavior of the agent for a safety system.

Formalization
We introduce the notations used in the algorithm:

- N^S (resp. N^A, N^{CC}) represents the number of the whole fault queues related to a sensor (resp. an actuator, a control component)
- $Queue_j^S$ (resp. $Queue_j^A$, $Queue_j^{CC}$) represents a queue associated to a defined kind of a fault concerning a sensor (resp. an actuator, a control component). This queue saves different occurrences of faults and their characteristics especially time occurrence and time treatment
- GWT^S (resp. GWT^A, GWT^{CC}) represents the global waiting time for the different faults in a queue related to a sensor (resp. an actuator, a control component)
- $MGWT^S$ (resp. $MGWT^A$, $MGWT^{CC}$) represents the mean global waiting time for the different faults in a queue related to a sensor (resp. an actuator, a control component)

For the sake of simplicity, we present here only the main steps of the algorithm applied to the different fault queues related to a sensor and these steps are the same for the others (i.e. fault queues related to an actuator or a control component). Let N^S be the number of faults queues that handle faults occuring at run-time related to a sensor. Let $Fault_i$ be an occurrence of a fault related to the queue $Queue_j^S$ such that $1 \leq i \leq Queue_j^S.length()$. We assume that the agent computes the waiting time of each fault $Fault_i$ denoted by $WT_{i,j}$. The waiting time is a measure of the total time that a fault waits in a queue. It corresponds to the duration between the occurrence time of the fault (denoted by $time_{i,j}$) and the end of its treatment time by the agent (denoted by $treatmentT_{i,j}$).

$$WT_{i,j} = treatmentT_{i,j} \text{ - } time_{i,j}$$

We denote in addition by GWT_j^S the global waiting times of all the faults belonging to the same queue $Queue_j^S$. It is equal to the sum of the different waiting times of different faults $Fault_i$ divided by their number where $1 \leq i \leq Queue_j^S.length()$.

$$GWT_j^S = \frac{\sum_{i=1}^{Queue_j^S.length()} WT_{i,j}}{Queue_j^S.length()}$$

We denote also by Mean Global Waiting Time (denoted here $MGWT^S$) the sum of the global waiting times of all faults in queues related to a sensor divided by their number N^S.

$$MGWT^S = \frac{\sum_{j=1}^{N} GWT_j^S}{N^S}$$

We distinguish in the agent's algorithm two periodic actions : fault occurrence and fault management. When a fault occurs, the agent searches the kind of this fault and puts it in the associated queue. The fault management is a periodic task where the agent treats a fault, calculates the waiting time associated to this fault and then deletes it from the corresponding queue. Finally, the agent calculates the mean global waiting time for faults related to sensors, actuators or control components. We present in the following the detailed algorithm of the agent handling different faults. This algorithm tries in particular to minimize the global waiting time of faults in a queue.

Detailed Agent's Algorithm

```
    (0) Initialization
$\forall j \in [1..N^S]$ $Queue_j^S.clear()$;
$\forall j \in [1..N^A]$ $Queue_j^A.clear()$;
$\forall j \in [1..N^{CC}]$ $Queue_j^{CC}.clear()$;

    (1) Fault occurrence
For each period $\Delta$
  If occurrence(fault) then
    fault.time $\leftarrow$ currentTime();
    Switch type(fault)
      case $fault^S$ :
        $\exists j \in [1..N^S] / fault.kind = j$
        $Queue_j^S.push(fault)$;

      case $fault^A$ :
        $\exists j \in [1..N^A] / fault.kind = j$
        $Queue_j^A.push(fault)$;

      case $fault^{CC}$ :
        $\exists j \in [1..N^{CC}] / fault.kind = j$
        $Queue_j^{CC}.push(fault)$;

    (2) Fault management
    For each period $\Delta'$
  If treat(fault) then

      Switch type(fault)
      case $fault^S$ :
        $\exists j \in [1..N^S]/fault.kind = j$
        & i = the most priority fault in the $Queue^S_j$
        $WT_{i,j} \leftarrow currentTime() - Queue^S_j.get(i).time$
        $Queue^S_j.pop(i)$;
        $GWT^S_j \leftarrow GWT^S_j + WT_{i,j}$;

      case $fault^A$ :
        $\exists j \in [1..N^A]/fault.kind = j$
        & i = the most priority fault in the $Queue^A_j$
        $WT_{i,j} \leftarrow currentTime() - Queue^A_j.get(i).time$
        $Queue^A_j.pop(i)$;
        $GWT^A_j \leftarrow GWT^A_j + WT_{i,j}$;

      case $fault^{CC}$ :
        $\exists j \in [1..N^{CC}]/fault.kind = j$
        & i = the most priority fault in the $Queue^{CC}_j$
        $WT_{i,j} \leftarrow currentTime() - Queue^{CC}_j.get(i).time$
        $Queue^{CC}_j.pop(i)$;
        $GWT^{CC}_j \leftarrow GWT^{CC}_j + WT_{i,j}$;

    (3)     Measure of Mean Global Waiting
Time

    $MGWT^S \leftarrow 0$;
$MGWT^A \leftarrow 0$;
$MGWT^{CC} \leftarrow 0$;

    For j:= 1 to $N^S$ do
  $MGWT^S \leftarrow MGWT^S + GWT^S_j/Queue^S_j.length()$
$MGWT^S = MGWT^S/N^S$

    Print("MGWT for sensor faults: ",$MGWT^S$ )

    For j:= 1 to $N^A$ do
  $MGWT^A \leftarrow MGWT^A + GWT^A_j/Queue^A_j.length()$
$MGWT^A = MGWT^A/N^A$

    Print("MGWT for actuator faults: ",$MGWT^A$ )

    For j:= 1 to $N^{CC}$ do
  $MGWT^{CC} \leftarrow MGWT^{CC} + GWT^{CC}_j/Queue^{CC}_j.length()$
$MGWT^{CC} = MGWT^{CC}/N^{CC}$

    Print("MGWT for Control Component faults: ",$MGWT^{CC}$ )
```

We note finally that the approach complexity is O(n) where n is the greatest number among N^S, N^A and N^{CC}.

3 EXPERIMENTATION

The goal of this research paper is to define an optimal agent's policy for feasible management of software and hardware errors at run-time. We present a comparative study based on the global waiting time of faults in queues according to well-known scheduling policies (Priority/FIFO, Priority/Round Robin, Priority/Priority and Priority/Random). We propose to evaluate the performance by applying four approaches so that we determine the best approach that the agent should take:

Priority/FIFO Approach: for faults from the same queue, we use the priority criteria; for faults related to different queues, we use the First In/First Out criteria;

Priority/Round Robin Approach: for faults from the same queue, we use the priority criteria; for faults related to different queues, we use the Round Robin criteria which means for the first time, we take a fault from the first queue; for the second time, we take a fault from the second queue, and so on;

Priority/Priority Approach: for faults from the same queue, we use the priority criteria; for faults related to different queues, we use the priority criteria between different queues;

Priority/Random Approach: for faults from the same queue, we use the priority criteria; for faults related to different queues, we use a random choice.

To have a correct result, all the tests are based on the same characteristics of faults which enable to generate the following results (Table 1, Table 2, Table 3, Table 4).

Table 1: Waiting time according to Priority/FIFO approach.

Time unit	Sensor	Actuator	Component
1	38	22	6
2	40	34	29
3	54	64	51
4	61	53	41
5	67	64	55
6	57	67	79
MGWT	52,83	50,67	43,50

Interpretation:

The Figure 1 presents the Mean Global Waiting Time ($MGWT$) for each approach. As seen from the curves in Figure 1, we conclude that the best solution to be applied by the agent is the Priority/Round Robin approach. This result may be expected because

Table 2: Waiting time according to Priority/Round Robin approach.

Time unit	Sensor	Actuator	Component
1	38	19	25
2	17	25	33
3	51	21	38
4	27	47	51
5	55	67	49
6	50	38	58
MGWT	39,67	36,17	42,33

Table 3: Waiting time according to Priority/Priority approach

Time unit	Sensor	Actuator	Component
1	100	120	11
2	130	98	16
3	120	110	21
4	150	100	21
5	130	125	19
6	135	130	35
MGWT	127,5	113,83	20,5

the Priority/Round Robin approach ensures equality between all the fault queues of the different categories which leads to treat the diffrent faults for every queue without waiting a long time. We consider also that the Priority/Random approach provides interesting results as $MGWT$ values are not important. Nevertheless, the Priority/FIFO approach generates a medium values of $MGWT$ so it can not be considered as the best neither the worst approach. The Priority/Priority approach is the worst one. This degradation of $MGWT$ is due to that the agent gives priority to only one queue whereas the other queues are neglected which leads to heavy $MGWT$. By considering all these interpretations, we recommend to apply the Round Robin policy for the optimal implementation of the agent.

4 CONCLUSIONS

To guarantee a safe behavior of the whole system, we define an agent-based architecture where the agent controls the plant and treats fault whenever it occurs. To do so, we classify the faults (faults related to sensor, actuator or control component); for each category, we define many kinds of faults. In order to know what is the fault queue that the agent should choose first, we propose to evaluate the performance by applying four approaches (Priority/FIFO, Priority/Round Robin, Priority/Priority and Priority/Random). The results obtained permit to calculate the Mean Global Waiting Time ($MGWT$) which leads to consider the Priority/Round Robin approach as the best solution and the Priority/Priority approach as the worst one.

Table 4: Waiting time according to Priority/Random approach.

Time unit	Sensor	Actuator	Component
1	18	0	18
2	18	29	84
3	52	14	79
4	62	50	81
5	36	75	48
6	26	52	37
MGWT	35,33	36,67	57,83

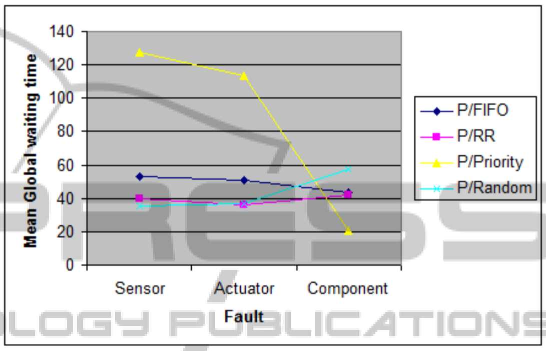

Figure 1: Comparative study.

REFERENCES

Al-Safi, Y. and Vyatkin, V. (2007). An ontology-based reconfiguration agent for intelligent mechatronic systems. In *Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems*. Springer-Verlag.

Angelov, C., Sierszecki, K., and Marian, N. (2005). Design models for reusable and reconfigurable state machines. In *L.T. Yang and All (Eds): EUC 2005, LNCS 3824, pp:152-163. International Federation for Information Processing.*

G. Pratl, D. Dietrich, G. H. and Penzhorn, W. (2007). A new model for autonomous, networked control systems. *IEEE Transactions on Industrial Informatics*, 3(1).