International Conference on Control, Engineering & Information Technology (CEIT'13)
Proceedings Engineering & Technology - Vol.3, pp. 20-26, 2013

NEW OPTIMIZATION FOR RECONFIGURABLE NETWORKED EMBEDDED CONTROL SYSTEMS

Amen Ben Hadj Ali[#1], Mohamed khalgui[*2], Samir Ben Ahmed[#3]
[#] *Tunis El Manar University*
El Manar 1, Tunisia
[1] amen.benhadjali@gmail.com
[3] samir.benahmed@fst.rnu.tn
[*] *University of Carthage*
Tunis, Tunisia
[2] mohamed.khalgui@gmail.com

Abstract— **This research paper deals with Distributed Reconfigurable Embedded Control Systems (RECS) which can dynamically follow different behaviors at run-time according to user requirements or any possible evolution in its environment. We optimize a multi-agent architecture for the system in which a Reconfiguration Agent is affected to each device to apply local reconfigurations, and a Coordination Agent is proposed for the coordination between devices in order to guarantee safe, coherent and adequate distributed reconfigurations. A Communication Protocol is proposed to handle this coordination between agents by using well-defined Coordination Matrices.**

Keywords—**Distributed Embedded Control System, Reconfiguration, Software Architecture, Multi-Agent Architecture, Reconfiguration Protocol, Coordination.**

I. INTRODUCTION

The constant growth of complexity of embedded control systems makes reconfiguration increasingly important. In this context, reconfiguration refers to the ability of a system to change its functionality at run-time, performing different functions at different instances in time. This ability to reconfigure a system in real-time allows available resources to be shared between multiple functions and configurations. The challenges, in reconfiguration, are as much about the design model as the level of the environment that supports execution. We distinguish two kinds of reconfigurations: static [1] and dynamic reconfigurations [2]. Static reconfigurations are applied off-line to apply changes before the system could start, whereas dynamic reconfigurations are applied dynamically at run-time. In the last case, two sub-classes exist: manual reconfigurations to be executed by users [3] and automatic reconfigurations to be assured by intelligent agents [2], [4], [5]. The reconfiguration of control systems is currently a very active research area where considerable progress has been made [1], [5], [9], [10].

To deal with the dynamic reconfiguration of Distributed Embedded Control Systems (DECS), we propose, in this work a new Multi-Agent distributed architecture. We define two kinds of agents: software Reconfiguration Agents (RA) which are responsible for controlling the devices and a software Coordination Agent (CA) which handles the coherence of distributed concurrent reconfigurations of different devices. The coordination between devices after any distributed reconfiguration scenario is mandatory in order to avoid any risk of incoherence. We define also the concept of “coordination matrix” to specify for each reconfiguration scenario the behavior of all concerned agents that should react simultaneously. We define a reconfiguration protocol to manage the coordination between the networked devices. When a RA wants to apply a new reconfiguration, it sends a request to CA. A request represents a need to improve the system’s performance, or also to recover and prevent hardware/software errors, or also to adapt the system’s behavior to new requirements according to the environment’s evolution. Once the request is received by the CA, it informs all other concerned agents which should react with such RA which wants to trigger the new behavior. The execution of the reconfiguration scenario depends effectively on the answers of these reconfiguration agents which should decide if the new behavior can be executed or not. This protocol allows us to win an important number of exchanged messages on the network of distributed devices.

This paper gives new extensions of our previous works [4], [5], [6] with the purpose to allow high reconfigurability and also functional safety of DECS. The work presented in [4] deals with distributed multi-agent reconfigurable embedded-control systems following the component-based International Industrial Standard IEC61499 [11]. The authors define an architecture of reconfigurable multi-agent systems and propose a coordination agent that coordinates between devices by using a communication protocol. The reconfiguration requests are managed by the coordinator according to their priority. The role of the coordinator is to accept or to reject a reconfiguration request. The major contribution of the current paper is to provide new optimizations for the proposed communication protocol [4] in several directions. Firstly, we assume that the CA handles for each request an historic by giving for each RA the possibility to recall the execution of a given request at different times. The management of the reconfigurations historic allows saving knowledge on the requests frequency and eventual interactions between them (e.g. conflict or redundancy). Such knowledge will optimize the reconfigurability of the system and the CA behaviour for future reconfigurations. Furthermore, each RA can exhibit the behavior of the CA when this later decides to delegate the execution of a secondary request to its sender in the case that this request is sent at the same time than the one having the highest priority. Thus, we add a new functionality to the CA which is the delegation of reconfigurations management to RAs. The delegation functionality presents two major advantages. Firstly, it aims to improve the performance of the CA by reducing the number of requests it handles. Secondly, we optimize the functional safety when the coordinator is broken. In [5] and [6], the authors present a UML-based design approach for agent-based reconfigurable ECS having a centralised architecture. In the current paper, our aim is to extend this previous work by considering distributed architectures. Therefore, we assume that DECS are described as a network of interconnected controller components that can have different configurations. A configuration is defined by a set of components and connections between them. The execution of a reconfiguration request must bring the system from a valid configuration to another one while respecting the reconfiguration constraints.

This paper is structured as follows. In section 2 we present the optimizations of the multi-agent architecture with the specification of the RAs and the CA behaviors. Section 3 deals with the optimizations of the communication protocol. Section 4 presents experimental results. Finally, the major contributions of this work and future work are emphasized in the conclusion.

II. OPTIMIZATION IN THE MULTI-AGENT ARCHITECTURE

In this section, we present an optimization in the multi-agent architecture for reconfigurable DECS [4]. A system Sys is composed of n networked devices $\{dev_1, \dots dev_n\}$. Within the proposed architecture we distinguish two kinds of agents: Coordination Agent (CA) and Reconfiguration Agents (RA) (see Fig. 1). Both kinds of agents are represented by software components that act on the software control architecture in order to execute a particular task. The role of any RA_i affected to a particular device dev_i (i=1..n) is to apply automatic reconfigurations on the system's architecture at different granularity levels. The execution of reconfigurations must bring the whole system from a valid configuration to another one while respecting the reconfiguration constraints. Because we assume a distributed system, each RA acts on a sub part of the system's architecture but cannot act in his one: it receives reconfiguration requests from different sources and executes them in collaboration with the other RAs under certain conditions in order to bring the whole system to a safe state. Therefore, before execution, each reconfiguration request must be approved by an entity of the multi-agent architecture that manages the collaboration and the communication between distributed RAs. Consequently, we define the concept of Coordination Agent that handles the coherence of distributed reconfigurations between RA. When a RA wants to apply a new reconfiguration, it sends a request to the CA in order to have its approbation. The coordination in the context of DECS is very important because any uncontrolled dynamic reconfiguration can lead to critical problems when it brings the system to an incoherent and unsafe behaviour. In order to manage the coordination between RAs, we also define the concept of Coordination Matrix (CM) which contains safe reconfiguration scenarios that can be applied simultaneously by the different RAs. The Coordination Agent is therefore the entity that handles the set of CMs corresponding to the different reconfiguration scenarios. In addition, we propose, a communication protocol between distributed RAs to manage distributed reconfiguration scenarios. In this protocol we distinguish, three kinds of communication primitives between distributed agents: a RA can send a request to the CA in order to have its authorization for the execution of a reconfiguration scenario. As response to the request, the CA can accept, reject (definitively or provisory) or delegate the execution of the concerned scenario. These responses of the CA correspond respectively to three primitives: Acceptance primitive Rejection/Recall Primitive and Delegation primitive. The new extensions of the communication protocol (as presented in [4]) concern mainly the addition of two functionalities: delegation and recall. The purpose of these extensions is to have high reconfigurability and functional safety especially when the CA is broken.

A. Specification of the Reconfiguration Agent behavior

As previously presented in [5], the behaviour of a RA is formalized by using nested state machines. Indeed, we define three levels of reconfiguration: the first deals with the system architecture, the second deals with the internal structure of devices or with their connections, finally the third deals with reconfigurations of data. Therefore, in order to apply a reconfiguration scenario $\boldsymbol{R_{i,j,k,h}}$, the reconfiguration agent executes three steps as follows (i) the architectural configuration $\boldsymbol{AC_i}$ is loaded in the memory ($\boldsymbol{AC_i}$ denotes a particular architectural configuration), (ii) then the structural configuration $\boldsymbol{SC_{i,j}}$ is chosen between different structural configurations corresponding to AC_i (iii) finally, the data configuration $\boldsymbol{DC_{i,j,k,h}}$ is applied. $\boldsymbol{DC_{i,j,k}}$ correspond to a particular state machine relative to $\boldsymbol{SC_{i,j}}$ and $\boldsymbol{DC_{i,j,k,h}}$ denotes a state in $\boldsymbol{DC_{i,j,k}}$ which correspond to one of the following cases: (i) one or more states of a SC state machine,(ii) more than one SC state machine, (iii) all the AC state machines.

B. Specification of the Coordination Agent behavior

Coordination between RAs appears to be essential in the automatic reconfiguration of DECS. Indeed, uncontrolled reconfigurations can lead to serious disturbances or critical

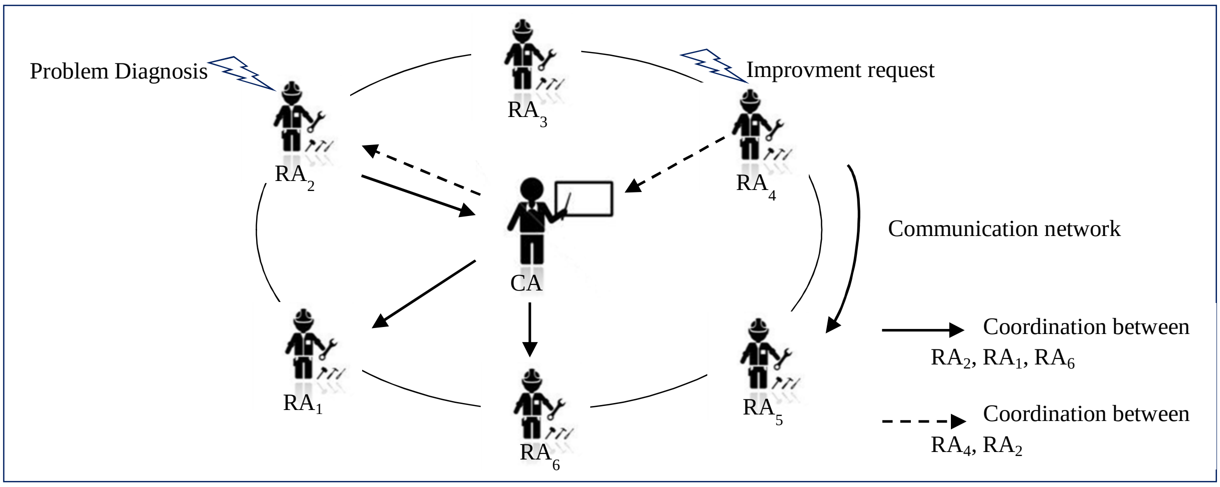

Fig. 1 Multi-agent architecture of reconfigurable DECS

problems in the system behavior because distributed RAs can execute incoherent and contradictory reconfiguration scenarios if they don’t communicate correctly with respect to system and time constraints. To deal with these difficulties, we define in this section the concept of Coordination matrix with the purpose of handling coherent reconfiguration scenarios in distributed ECS and we propose, thereafter, a multi-agent architecture for distributed reconfigurable systems, where a communication protocol between agents is defined to guarantee safe behaviors.

Coordination Matrix

Let **Sys** be a distributed reconfigurable system of n devices, and let Ag_1,..., Ag_n be n agents to handle automatic distributed reconfigurations of these devices. We denote in the following by ***Reconfiguration***$_{ia,ja,ka,ha}$ a reconfiguration scenario applied by the RA Ag_a ($a \in [1, n]$) as follows: (i) the corresponding AC state machine is in the state AC_{ia} . Let $cond^a_{ia}$ be the set of conditions to reach this state; (ii) the SC state machine is in the state $SC_{ia,ja}$. Let $cond^a_{ja}$ be the set of conditions to reach this state; (iii) the DC state machine is in the state $DC_{ka,ha}$. Let $cond^a_{ka,ha}$ be the set of conditions to reach this state. To handle coherent distributed reconfigurations that guarantee safe behaviors of the whole system ***Sys***, we define the concept of coordination matrix (CM) of size (n,4) that defines coherent scenarios to be simultaneously applied by different RAs (see Fig. 1). A CM is characterized as follows: each line a ($a \in [1, n]$) corresponds to a reconfiguration scenario ***Reconfiguration***$_{ia,ja,ka,ha}$ to be applied by $\mathbf{Ag_a}$ as follows (see Fig. 2):

CM[a, 1] = ia ; CM[a, 2] = ja ; CM[a, 3] = ka ;CM[a, 4] = ha

According to this definition: If an agent Ag_a applies the reconfiguration scenario ***Reconfiguration***$_{ia,ja,ka,ha}$, therefore it is equivalent to say that it applies the **Reconfiguration CM[a,1],CM[a,2],CM[a,3],CM[a,4]**. Each other RA $\mathbf{Ag_b}$ ($b \in [1, n]\backslash\{a\}$) has to apply the scenario **Reconfiguration CM[b,1],CM[b,2],CM[b,3],CM[b,4].** We denote in the following by *idle agent*, each agent $\mathbf{Ag_b}$ ($b \in [1, n]$), which is not required to apply any reconfiguration when others perform scenarios defined in CM. In this case:

$$CM[b, 1] = CM[b, 2] = CM[b, 3] = CM[b, 4] = 0$$

$$cond^b_{CM[b,1]} = cond^b_{CM[b,2]} = cond^b_{CM[b,3],CM[b,4]} = \text{True}.$$

$$\begin{array}{c|cccc|l}
 & 1 & 2 & 3 & 4 & \text{Applicable reconfigurations} \\
1 & \vdots & \vdots & \vdots & \vdots & \\
\vdots & 0 & 0 & 0 & 0 & \leftarrow \text{Idle agent} \\
Ag_a & ia & ja & ka & ha & \leftarrow \text{Reconfiguration to be applied by the RA } Ag_a \\
\vdots & \vdots & \vdots & \vdots & \vdots & \\
Ag_b & ib & jb & kb & hb & \leftarrow \text{Reconfiguration to be applied by the RA } Ag_b \\
\vdots & \vdots & \vdots & \vdots & \vdots & \\
Ag_n & in & jn & kn & hn & \leftarrow \text{Reconfiguration to be applied by the RA } Ag_n
\end{array}$$

Fig. 2 The Coordination Matrix

We denote in addition by **ξ(Sys)** the set of coordination matrices to be considered for the reconfiguration of the distributed embedded system **Sys**. Each coordination matrix CM is applied at run-time if for each agent $\mathbf{Ag_a}$ ($a \in [1, n]$) the following conditions are satisfied:

$$cond^a_{CM[a,1]} = cond^a_{CM[a,2]} = cond^a_{CM[a,3],CM[a,4]} = \text{True}.$$

On the other hand, we define *concurrent* coordination matrices, CM_1 and CM_2 two matrices of ξ(Sys) that allow different reconfigurations of a same RA Ag_b ($b \in [1, n]$) as follows:

- CM_j [b, i] ≠ 0 ∀ j ∈{1, 2} and i ∈[1, 4]; in this case Ag_b should react when CM_1 or CM_2 is loaded.
- CM_1 [b, i] ≠ CM_2 [b, i] ∀i ∈ [1, 4]; in this case, the agent Ag_b has to apply different reconfiguration scenarios at the same time.

To guarantee a deterministic behavior when concurrent coordination matrices are required to be simultaneously applied, we define priority levels for them such that only the matrix with the highest priority level should be applied. We denote in the following by:

- *Concur(CM)* is the set of concurrent matrices of CM ∈ ξ(Sys);
- *level(CM)* is the priority level of the matrix CM in the set Concur(CM) ∪{CM}.

III. OPTIMIZATION IN RECONFIGURATION PROTOCOL

In this section we present an optimization in the reconfiguration protocol [4] which describes the behaviour of distributed RAs orchestrated by a CA to dynamically reconfigure DECS. The software architecture of such systems is a network of control components where each one controls a sub-part of the system. We assume in addition that software architecture of DECS is designed using a UML-compliant standard. In order to guarantee safe and coherent reconfigurations, we define a Coordination Agent denoted by CA that handles a set of Coordination Matrices $\mathcal{E}$ (Sys) to control the set of Reconfiguration Agents (Ag_i , i ∈ [1, n]) as follows:

- When a particular agent Ag_a (a ∈[1, n]) should apply a ***Reconfiguration***$_{ia, ja, ka, ha}$ it sends the following request to CA ($\mathcal{E}$ (Sys)) to obtain its authorization:

  request (Ag$_a$***, CA, Reconfiguration***$_{ia, ja, ka, ha}$***).***

- When the CA receives ***r*** requests (r≥1) from different RAs at the same time then, it supports the highest priority request according to its $\mathcal{E}$ (Sys).
- When CA ($\mathcal{E}$ (Sys)) supports this request that corresponds to a particular coordination matrix CM ∈ $\mathcal{E}$ (Sys) and if CM has the highest priority between all matrices of Concur(CM) ∪ {CM}, then CA($\mathcal{E}$ (Sys)) informs the agents that have simultaneously to react with Ag_a as defined in CM. The following information is sent from CA ($\mathcal{E}$ (Sys)) for each Ag_b, b ∈ [1, n]\ {a} and CM[b,i]≠0, ∀ i ∈ [1, 4]:

  Reconfiguration (CA, Ag$_b$***, Reconfiguration***$_{CM[b, 1], CM[b, 2], CM[b, 3], CM[b, 4]}$***).***

- According to well-defined conditions in the control component of each Ag_b, the CA ($\mathcal{E}$ (Sys) request can be accepted, delegated or refused. In the following we present the reconfiguration algorithm and its procedures relative to the three different identified cases corresponding respectively to acceptance, delegation and rejection/recall primitives:

A. *Acceptance primitive*

In this case (see Fig. 3, a reconfiguration agent RA Ag_a (a=1..n) sends a request to the CA to have its authorization for applying a reconfiguration scenario. The CA must verify the applicability of the requested scenario by transferring the request to the other reconfiguration agents RA Ag_b (b=1..n, b≠a). Then, the requested scenario is applicable only if all the RA Ag_b send a positive response to the CA. Thereafter, the CA will authorize to the requester the execution of the requested scenario and the other RAs must follow by applying appropriate reconfigurations in order to bring the whole distributed system into a safe state.

```
BEGIN
If(priority = MAX)
/*the reconfiguration request that has the highest
priority is sent by $Ag_a$
nRec$_a$=0
/*initialization of the number of recalls for the RA $Ag_a$
nRec$_c$=0
/*initialization of the number of recalls for the RA $Ag_c$
reply = True
/*a Boolean that represents the reply of CA to the
reconfiguration request sent by $Ag_a$
While (b≤n)
/*for each $Ag_b$, b ∈[1, n]\ {a} and CM[b,i]≠0, (1≤i≤4 )*/
   If (cond$^b_{ib}$ = cond$^b_{jb}$ = cond$^b_{kb,hb}$ = True)
   Then
/* $Ag_b$ b ∈[1, n] and CM [b, i] ≠ 0, ∀i ∈[1, 4] */
Accept ($Ag_b$, CA, Reconfiguration$^b_{CM[b, 1], CM[b, 2], CM[b, 3], CM[b, 4]}$))
/*Acceptance reply sent from $Ag_b$ to CA  */
  Else
  Reject ($Ag_b$, CA), Reconfiguration$_{CM[b, 1], CM[b, 2], CM[b, 3], CM[b, 4]}$
, 0)
/* Rejection reply sent from $Ag_b$ to CA  */
reply= False
  End If
End
If (reply= True)
/*If  CA  receives positive answers from all $Ag_b$ then it
authorizes reconfigurations in the concerned devices*/
Then
For each $Ag_b$ /* $Ag_b$ b ∈[1, n] and CM [b, i] ≠ 0, ∀i ∈[1, 4]
Apply (Reconfiguration$_{CM[b, 1], CM[b, 2], CM[b, 3], CM[b, 4]}$)
/*Execution of the reconfiguration scenario in the device
```

```
    supervised by $Ag_b$*/
  Else
    Call Rejection/Recall primitive
  End If
Else
  /*the case of a reconfiguration request r that haven't the highest priority sent by a RA $Ag_c$*/
  Call Delegation primitive
End If
```

Fig. 3 Acceptance Primitive.

B. Optimization: Rejection/Recall primitive

In the case of acceptance, all the RA Ag_b (b=1..n, b≠a) must send a positive response to the CA before applying a reconfiguration scenario. In the rejection case (see Fig. 4), if there is only one RA Ag_b that sends a negative response then, the reconfiguration request is rejected. In addition, we assume that the CA can manage a history which is relative to each reconfiguration request. Indeed, the CA gives the possibility to the RAs to make several attempts to execute a given scenario before a definitive final rejection. The maximum number of attempts relative to a reconfiguration scenario is fixed by the CA. When a request is rejected, it is placed in a waiting queue which is managed by the CA, while the RA has not reached the allowed maximum number of attempts. The RA, can subsequently recall the CA of its request. Otherwise, if the maximum number is reached, then the request is definitively rejected and will not be stored in the waiting queue. Therefore, in addition to Rejection primitives, the CA has the ability to manage particular rejection cases (not definitive) by Recall primitives under known conditions. The purpose of the recall process is to allow a high reconfigurability of the whole system.

```
/*CA receives a negative answer from a particular agent $Ag_b$
If (nRec<maxRec)
  /*maxRec is a constant predefined by the CA and it represents the maximal number of recalls authorized by the CA for a reconfiguration request*/
  nRec=nRec+1
  Reject(CA)), $Ag_a$, Reconfiguration $_{ia, ja, ka, ha}$, $nRec_a$)
  /*Provisory Rejection reply sent from CA to $Ag_a$ */
Else /* if nRec= maxRec
  Reject(CA), $Ag_a$, Reconfiguration $_{ia, ja, ka, ha}$, maxRec)
  /*Definitive Rejection reply sent from CA to $Ag_a$*/
End If
```

Fig. 4 Rejection/Recall Primitive.

C. Optimization: Delegation primitive

In the common case, the CA defines by considering several constraints, a priority order to handle the reconfiguration requests coming from different and distributed RAs. In the case that the CA receives two requests at the same time, then it will deal with the request having the highest priority (sent by a RA Ag_a (a=1..n)). Then the execution of a second request can be reported to an ulterior time. Nevertheless, to give more flexibility and optimality to our multi-agent architecture, the CA can delegate to a RA Ag_c (c=1..n, c≠a), the application of the second reconfiguration request (see Fig. 5) when it is not conflicting with the first scenario (with the highest priority) i.e. it doesn't bring the system to an unsafe state.

```
i=1 /*initialization of line's index
j=1 /*initialization of column's index
k=1 /*initialization of a counter for the list Conc
h=1 /*initialization of a counter for the list notConc
For each line in CM[i,j]
  /*research in CM for the list of RAs that are not idle and must apply the same reconfiguration scenario than $Ag_a$*/
  while(CM [i,j]= CM [a,j] and j≤4)
    /*CM corresponds to the reconfiguration request that has the highest priority*/
    j=j+1
  End
  If(j=4)
  Then
    Conc[k]=i
    /*Conc is the list of RAs (different of $AG_a$) concerned by the highest priority request in CM and that must apply the same reconfiguration scenario than $Ag_a$*/
    k=k+1
  Else
    notCon[h]=i
    /*notConc is the list of RAs (different of $AG_a$) concerned by the highest priority request in CM and that not apply the same reconfiguration scenario than $Ag_a$*/
  i=i+1
  End If
END
i=1 /*initialization of line's index
j=1 /*initialization of column's index
while(i≤k) //for each element in Conc
  j=1 /*Initialization of column's index
  while($CM_r$[Conc[i],j]= 0 and and j ≤4)
    /*$CM_r$ represents a reconfiguration reqeststhat haven't the highest priority sent by a RA $Ag_c$*/
    j=j+1
  End
  If (j=4)
    /* the RA having the same line index than Conc[i] in $CM_r$ is idle*/
  Then
    i=i+1
  Else
    If($nRec_c$≤maxRec)
      /*max recall number of the RA $Ag_c$ is not reached*/
      Reject(CA), $Ag_c$, Reconfiguration $_{ic, jc, kc, hc}$, $nRec_c$)
```

```
$nRec_{c=} nRec_c+1$
/*The reconfiguration request is rejected because there exists a not idle Ra having the same line index than the RA Conc[i] in $CM_r$ */
Else
Reject(CA), $Ag_{c,}$ Reconfiguration $_{ic, jc, kc, hc}$, maxRec)
End If
End
i=1 /*initialization of line's index
j=1 /*initialization of column's index
while(i≤h)  /*for each element in notConc
j=1 /*Initialization of column's index
while($CM_r$[notConc[i],j]= CM[notConc[i],j] and j ≤4)
/*$CM_r$ and CM have exactly the same line then the reconfiguration request r is the same than the highest priority one and it will be definitively rejected*/
Reject(CA, $Ag_c$, Reconfiguration $_{ic, jc, kc, hc}$, maxRec)
End
If (j=4)
/*the RA having the same line index than Conc[i] in $CM_r$ is idle*/
Then
i=i+1
Else
/*The reconfiguration request is rejected because there exists a not idle RA having the same line index than the RA Conc[i] in $CM_r$*/
If(nRec<maxRec)
  Reject(CA, $Ag_{c,}$ Reconfiguration $_{ic, jc, kc, hc}$ $nRec_c$)
  $nRec_{c=} nRec_c+1$
Else
  Reject(CA, $Ag_{c,}$ Reconfiguration $_{ic, jc, kc, hc}$, maxRec)
End If
End If
End
/*The reconfiguration request r is delegated when all the RAs of Conc are idle in $CM_r$ and all RAs in notCon have to execute a different request than the highest priority one*/
Delegate(CA), $Ag_c$, Reconfiguration $_{ic, jc, kc, hc}$)
End If
END
```

Fig. 5 Delegation Primitive.

IV. EXPERIMENTAL RESULTS

In this section, we give an evaluation of the proposed communication protocol for intelligent reconfigurations of DECS by varying the number of reconfiguration messages exchanged within the network of distributed agents. We assume that n RAs send n reconfiguration requests at the same time. We denote by msg_c the number of exchanged messages by distributed agents when we use a CA in the network. In the case of absence of coordination, we denote by msg the number of exchanged messages. A message in both cases can represent a request, an acceptance, a rejection (provisory or definitive), a delegation or an execution order (apply) from the coordinator. The gain (denoted by G) obtained by the proposed protocol is msgc / msg and it represents the decrease of the exchanged messages between distributed devices when we use a CA. In the following, we will detail different cases of execution:

- If one message is accepted (among n requests sent by n RAs) and all others are refused (only the highest-priority message is accepted). Then, the number of exchanged messages with coordination is msg_c = 5*n-3. In the case of absence of coordination, we will have msg = $2*n^2$-n-1. The gain with the use of a coordinator is G =msg_c/msg = 5*n-3 / $2*n^2$-n-1.
- If the delegation primitive is applied (in presence of a CA), for example we assume that among n messages, only one is accepted by the CA, (n-1)*0,5 are rejected (i.e. 50% of the rest of requests) and (n-1)*0,5 are delegated to different RAs. Thus, msg_c = $3*n^2$/2+2*n-3/2, msg = $5*n^2$/2-2*n-1/2 and G = $3*n^2$/2+2*n-3/2 / $5*n^2$/2-2*n-1/2.

  As application, we consider a network of 100 distributed RAs transporting 60 messages per minute. We assume in addition that probably 20 reconfigurations are requested per minute. Therefore, the gain in the first case (coordination without delegation as published in [4]) is G=0,12 and with delegation G= 0,66.

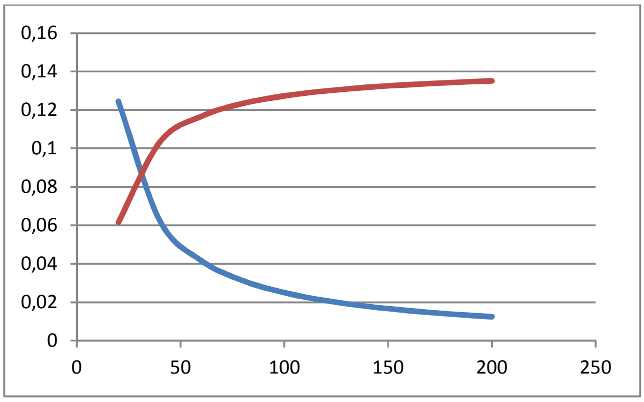

Fig. 6 Evolution of the gain in number of exchanged messages.

The graph of Fig. 6 shows two curves corresponding to the evolution of the gain in number of exchanged messages on the network of N RAs (100≤N≤1000). The values of the abscises axis correspond to the number of reconfiguration requests per minute. The curve in bleu corresponds to the gain when we apply a simple acceptance primitive (i.e. acceptance of the highest-priority message by the CA). The curve in red corresponds to the gain when we apply in addition to the coordination, the delegation primitive. In particular, it represents the evolution of gain when only 10% of messages are delegated. It is important to note that the gain increases proportionally to the percentage of delegated messages. In conclusion, the presence of a CA on the network of distributed RAs allows obtaining a gain which decreases when the number of RAs increases. However, this gain can be clearly optimized when we apply the proposed extensions. In particular, the addition of the delegation primitive to the communication protocol allows having a gain that evolves proportionally to the number of RAs. Consequently, the delegation allows to ameliorate the functional safety of the whole systems even if the CA is broken.

V. CONCLUSION

By assuming recall and delegation primitives, we propose in this paper a new optimization of a defined multi-agent architecture in [4] for reconfigurable DECS. We prove the gain of this extension by considering a formal example. A new protocol is proposed to guarantee safe and coherent distributed reconfigurations at run-time according to user requirements. This protocol is based on reconfiguration agents affected to devices, and a coordinator as well as coordination matrices for a useful coordination between devices after any reconfiguration scenario. Different directions can be mentioned as further work. First of all, we plan to deal with a formal verification by using UPPAAL to validate the change from one safe configuration to another. We plan also to test our approach in the context of a real-time operating system.

REFERENCES

[1] C. Angelov, K. Sierszecki, and N. Marian, “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, 2005.

[2] R. Brennan, P. Vrba, P. Tichý, A. Zoitl, C. Sünder, T. Strasser, V. Marík. “Developments in dynamic and intelligent reconfiguration of industrial automation”. Computers in Industry vol 59(6), pp.533-547, 2008.

[3] M-N. Rooker, C. Sunder, T. Strasser, A. Zoitl, O. Hummer and G. Ebenhofer, “Zero Downtime Reconfiguration of Distributed Automation Systems : The εCEDAC Approach”, *Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, Springer-Verlag, 2007.

[4] M. Khalgui and O. Mosbahi, “Intelligent Distributed Control Systems”, Information and Software Technology, vol. 52(12), pp. 1259-1271, December 2010.

[5] A. Ben Hadj Ali, M. Khalgui, and S. Ben Ahmed, “UML-Based Design and Validation of Intelligent Agents-Based Reconfigurable Embedded Control Systems”, International Journal of System Dynamics Applications, vol.1(1), pp.17, 2012, ISSN: 21609772,

[6] A. Ben Hadj Ali, M. Khalgui, A. Valentini, and S. Ben Ahmed, “Safe reconfigurations of agents-based embedded control systems”, in Proc. IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society, 2011, p. 4344.

[7] FESTO description, Martin Luther University, Germany, http://aut.informatik.uni-halle.de/forschung/testbed/, 2008.

[8] EnAS description. Martin Luther University, Germany, http://aut.informatik.uni-halle.de/forschung/enas_demo/, 2008.

[9] Y. Alsafi, V. Vyatkin, Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing. Robotics and Computer-Integrated Manufacturing, Volume 26, Issue 4, Pages 381-391, August 2010.

[10] A. Zoitl, W. Lepuschitz, M. Merdan, M. Vallee, A Real-Time Reconfiguration Infrastructure for Distributed Embedded Control Systems, IEEE International Conference ETFA, 2010.

[11] Industrial Process Measurements and Control Systems, I. S. IEC61499 2004.