Journal of Ubiquitous Systems & Pervasive Networks
Volume 1, No. 1 (2010) pp. 19-28

New Solutions for Feasible and Coherent Reconfigurations of Multi-Agent Embedded Software Architectures

Mohamed Khalgui*[a,b], Atef Gharbi[c]

[a]*Martin Luther University, Germany,*
[b]*Xidian University, China,*
[c]*University of Carthage, Tunisia,*

Abstract
The paper deals with reconfigurable multi-agent distributed embedded control systems following the component-based International Industrial Standard IEC61499 in which a Function Block (abbreviated by FB) is an event-triggered software component owning data and a control application is a distributed network of Function Blocks. To handle all possible industrial cases, we classify reconfiguration scenarios into three forms before we define an architecture of reconfigurable multi-agent systems where a Reconfiguration Agent modelled by nested state machines is affected to each device of the execution environment to apply local reconfigurations, and a Coordination Agent is proposed for any coordination between devices in order to guarantee safe and adequate distributed reconfigurations. A Communication Protocol is proposed to allow a feasible coordination between agents by using well-defined Coordination Matrices. We model the agents according to the formalism Net Condition/Event Systems (abbrev. NCES) which is a rich extension of Petri nets, and apply the model checker SESA to validate the coordination between agents. Indeed, we should verify that whenever a reconfiguration scenario is applied in a device, the remote devices should react as soon as possible according to user requirements. We developed the tool "ProtocolReconf" for the visual simulation of the inter-agent protocol which is applied in the paper to two benchmark production systems available at Martin Luther University.
Keywords: *Function Block, Embedded System, Multi-Agent, Distributed Reconfiguration, Model Checking, Simulation.*

1. Introduction

The new generation of industrial embedded control systems is addressing new criteria as flexibility and agility [1], [2]. To reduce their cost, these systems should be changed and adapted to their environment without disturbances. Several interesting academic and industrial research works have been made in recent years to develop reconfigurable embedded control systems [3]. We distinguish in these works two reconfiguration policies: static and dynamic reconfigurations such that static reconfigurations are applied off-line to apply changes before starting the system [4], whereas dynamic reconfigurations are dynamically applied at run-time. Two cases exist in the last policy: manual reconfigurations applied by users [5] and automatic reconfigurations applied by Intelligent Agents [6, 8]. Nowadays, four classes of software agents have been proposed [11]: (i) Logic-based Agents such that agents take decisions depending on logical deductions; (ii) Reactive Agents where agents make decisions according to some forms of mappings from causes to actions; (iii) Belief-Desire-Intention (BDI) Agents such that agents have beliefs, desires, and intentions in which decisions-makings are based on manipulations of data structures; (iv) Layered Architectures where agents are specified with many software layers. On the other hand, two classes of software components are clearly distinguished in [12]: the first addresses components to be instantiated, composed and deployed at run-time [7], and the second addresses components to be composed off-line for real-time embedded systems. We are interested in this paper in the second class by following the well-known component-based International Industrial Standard IEC61499 for the development of embedded control systems [14,16]. In this standard, a Function Block is an event-triggered software unit to be composed of an interface and an implementation. The interface contains data/event inputs/outputs to support interactions with the environment, whereas the implementation contains algorithms to be executed when corresponding input events occur. According to this standard, a system is implemented by a network of Function Blocks under precedence constraints.

We are interested in this research in automatic reconfigurations of distributed industrial embedded control systems following the standard IEC61499. We mean in the paper by a reconfiguration scenario any operation allowing the automatic additions, removals or updates of Function Blocks to save the whole system when hardware faults occur, or to improve also its productivity. In this case, the system is modeled by different networks of Function Blocks such that each one should be executed at run-time when a particular reconfiguration scenario is automatically applied. To handle all possible automatic reconfigurations, we propose a distributed multi-agent architecture [15] in which a Reactive and Layered Reconfiguration Agent that we model by nested state machines

* Corresponding author. E-mail: khalgui.mohamed@gmail.com

DOI: 10.5383/juspn.01.01.003

is affected to each device of the execution environment to handle local automatic reconfigurations, and a centralized Coordination Agent is defined to manage distributed reconfigurations between devices because any uncontrolled automatic reconfiguration applied in a device can lead to critical problems and/or serious disturbances in others. We propose the formal concept of "Coordination Matrix" to define for each distributed reconfiguration scenario the behavior of all concerned reconfiguration Agents that should simultaneously react. To guarantee a deterministic behavior of the whole distributed architecture, we define for each matrix a priority level according to the emergency of the corresponding reconfiguration scenario. The Coordination Agent handles all matrices to coordinate between agents according to a well-defined communication protocol: when a Reconfiguration Agent applies in the corresponding device a highest-priority reconfiguration, the Coordination Agent informs the rest of concerned agents to react and to bring the whole distributed system to safe and optimal behaviors. We model each Reconfiguration Agent and each Coordination Matrix according to the formalism Net Condition/Event Systems which is an extension of Petri nets [9], and apply the model checker SESA to exhaustively check if all system's devices react as described in user requirements to guarantee safe and correct distributed reconfigurations. We verify in this case functional properties according to the temporal logic "Computation Tree Logic"[9]. We developed in addition a software tool for graphical simulations of the inter-agents protocol in order to visually show for users the reactivity of the distributed system's agents according to environment's evolutions. We describe in the next section the benchmark production systems FESTO and EnAS that we follow as running examples in the paper to explain our contributions. We define in Section 3 a multi-agent architecture for reconfigurable embedded control systems, before apply thereafter in Section 4 a model checking of the inter-agent protocol. We present in section 5 the tool simulating the architecture, before discussing the originality of the paper's contribution in Section 6, and conclude the paper in Section 7.

2. Benchmark Production Systems FESTO and EnAS

We present two benchmark production systems following the Standard IEC61499: FESTO and EnAS available at Martin Luther University in Germany. We propose in particular some reconfiguration scenarios to be applied in these systems that we assume in the following as running examples.

2.1. FESTO System

The benchmark production system FESTO is a well-documented demonstrator used by many universities for research and education purposes, and is used as a running example in the context of this paper. FESTO is composed of three units: Distribution, Test and Processing Units. Distribution Unit is composed of a pneumatic feeder and a converter to forward cylindrical work pieces from a stack to the testing unit which is composed of the Detector, the Tester and the Elevator. This unit performs checks on work pieces for height, material type and color. Work pieces that successfully pass this check are forwarded to the rotating disk of Processing Unit, where the drilling of the work piece is performed. We assume two drilling machines Drill_machine1 and Drill_machine2 to drill pieces. The result of the drilling operation is next checked by the checking machine and the work piece is forwarded to another mechanical unit. Three production modes of FESTO are considered according to the rate of input pieces denoted by number_pieces into the system (i.e. ejected by the feeder).

- **Case1: High production.** If number_pieces ≥ Constant1, Then the two drilling machines are used at the same time to accelerate their production. In this case, Distribution and Testing Units should forward two successive pieces to the rotating disc before starting the drilling with Drill_machine1 AND Drill_machine2. For this production mode, the periodicity of input pieces is p = 11 seconds,

- **Case2: Medium production.** If Constant2 ≤ number_pieces < Constant1, Then we use Drill_machine1 OR Drill_machine2 to drill work pieces. For this production mode, the periodicity of input pieces is p = 30 seconds,

- **Case3: Light production.** If number_pieces < Constant2, Then only the drilling machine Drill_machine1 is used. For this production mode, the periodicity of input pieces is p = 50 seconds.

On the other hand, if one of the drilling machines is broken at run-time, then we have to only use the other one. In this case, we reduce the periodicity of input pieces to p = 40 seconds. The system is completely stopped in the worst case if the two drilling machines are broken.

2.2. EnAS System

The benchmark production system EnAS was designed as a prototype to demonstrate energy-antarcic actuator/sensor systems. We assume that it has the following behavior: it transports pieces from the production system (i.e. FESTO system) to storing units. The pieces in EnAS shall be placed inside tins to close with caps afterwards. Two different production strategies can be applied: we place in each tin one or two pieces according to production rates of pieces, tins and caps. We denote respectively by nb_{pieces} , $nb_{tins+caps}$ the production number of pieces and tins (as well as caps) per hour and by Threshold a variable (defined in user requirements) to choose the adequate production strategy. The EnAS system is mainly composed of a belt, two Jack stations (J1 and J2) and two Gripper stations (G1 and G2). The Jack stations place new produced pieces and close tins with caps, whereas the Gripper stations remove charged tins from the belt into storing units. Initially, the belt moves a particular pallet containing a tin and a cap to the first Jack station J1. According to production parameters, we distinguish two cases,

- **Case1: First production policy.** If (nb pieces/nb tins+caps ≤ Threshold), Then the Jack station J1 places from the production station a new piece and closes the tin with the cap. In this case, the Gripper station G1 removes the tin from the belt into the storing station St1.

- **Case2: Second production policy.** If ($nb_{pieces}/nb_{tins+caps}$ > Threshold), Then the Jack station J1 places just a piece in the tin which is moved thereafter into the second Jack station to place a second new piece. Once J2 closes the tin with a cap, the belt moves the pallet into the Gripper station G2 to remove the tin (with two pieces) into the second storing station St2.

3. Multi-Agent Reconfigurable Architectures

We define a multi-agent architecture for distributed reconfigurable embedded control systems following the International Standard IEC61499. A Reconfiguration Agent is affected in this architecture to a device of the execution environment to handle automatic reconfigurations of Function Blocks. It is specified by nested state machines that support all reconfiguration forms. Nevertheless, the coordination between agents in this distributed architecture is extremely mandatory because any uncontrolled automatic reconfiguration applied in a device can lead to critical problems, serious disturbances or also inadequate distributed behaviors in others. To guarantee safe distributed reconfigurations, we define the concept of *Coordination Matrix* that defines correct reconfiguration scenarios to be simultaneously applied in distributed devices, and define the concept of *Coordination Agent* that handles coordination matrices to coordinate between distributed agents. We propose a communication protocol for agents to manage concurrent distributed reconfiguration scenarios.

Running Example. *In FESTO and EnAS where a Reconfiguration Agent is defined for each one of them, the reconfiguration of the first can lead to a reconfiguration of the second in order to guarantee a coherent production in these platforms. This means:*

- *If Constant2 ≤ number_pieces, Then the FESTO's agent should apply the Medium or High Production Mode, and in this case the EnAS's agent should improve the productivity by applying the Second Production Policy in order to put two pieces in each tin.*
- If *Constant2 > number_pieces, Then the FESTO's agent should decrease the productivity by applying the Light Mode (i.e. only Drill_machine1 is used), and in this case, the EnAS's agent should also decrease the productivity by applying the First Production Policy in order to put only one piece in the tin.*

On the other hand, when a hardware problem occurs at run- time in a platform, a reconfiguration of the second is required as follows:

- If *one of the Jack stations J1 and J2 or the Gripper station G2 is broken in the production system EnAS, Then the corresponding agent should decrease the productivity by applying the First Production Mode, and in this case the FESTO's agent should also follow the Light Production Mode in order to guarantee a coherent behavior.*
- *If one of the drilling machines Drill_machine1 and Drill_machine2 is broken, Then the FESTO's agent should decrease the productivity, and in this case the EnAS's agent should follow the First Production Mode where only one piece is put in a tin.*

3.1. Architecture of the Reconfiguration Agent

We define for each device of the execution environment a unique agent that checks the environment's evolution and takes into account user requirements to apply automatic reconfiguration scenarios. We define the following units that belong to three hierarchical levels of the agent's architecture:

– **First level:** (Architecture Unit) this unit checks the system's behavior and changes its architecture (adds/removes Function Blocks) when particular conditions are satisfied. We note that *Standardized Manager Function Blocks* are used in this unit to load or unload such blocks into/from the memory [14].

– **Second level:** (Control Unit) for a particular *loaded* architecture, this unit checks the system's behavior and: reconfigures compositions of blocks (i.e. changes the configuration of connections), or adds/removes data/event inputs/outputs, or reconfigures the internal behavior of blocks (i.e. updates of algorithms),

– **Third level:** (Data Unit) this unit updates data if particular conditions are satisfied.

We design the agent by nested state machines where the Architecture Unit is specified by an Architecture State Machine (denoted by ASM) in which each state corresponds to a particular architecture of the system. Therefore, each transition of the ASM corresponds to the load (or unload) of Function Blocks into (or from) the memory. We construct for each state *S* of the ASM a particular Control State Machine (denoted by CSM) in the Control Unit. This state machine specifies all reconfiguration forms to possibly apply when the system's architecture corresponding to the state *S* is loaded (i.e. modification of FB compositions or of their internal behavior). Each transition of any CSM should be fired if particular conditions are satisfied. Finally, the Data Unit is specified also by Data State Machines (denoted by DSMs) where each one corresponds to a state of a CSM or the whole ASM.

Notation. We denote in the following by,

– n_{ASM} the number of states in the state machine ASM (i.e. the number of possible architectures implementing the system). ASM_i ($i \in [1, n_{ASM}]$) denotes a state of ASM to encode a particular architecture (i.e. particular FB network). This state corresponds to a particular state machine *CSM* that we denote by CSM_i ($i \in [1, n_{ASM}]$),

– n_{CSM_i} the number of states in CSM_i and let $CSM_{i,j}$ ($j \in [1, n_{C\ SM_i}]$) be a state of CSM_i,

– n_{DSM} the number of Data State Machines corresponding to all possible reconfiguration scenarios of the system. Each state $CSM_{i,j}$ ($j \in [1, n_{CSMj}]$) is associated to a particular DSM state machine DSM_k ($k \in [1, n_{DSM}]$),

– n_{DSM_k} the number of states in DSM_k. $DSM_{k,h}$ ($h \in [1, n_{DSM_k}]$) denotes a state of the state machine DSM_k which can correspond to one of the following cases: (i) one or more states of a state machine *CSM*, (ii) more than one *CSM* state machine, (iii) all the *ASM* state machines. The agent automatically applies at run-time different reconfiguration scenarios such that each one denoted by $Reconfiguration_{i,j,k,h}$ corresponds to a particular network of Function Blocks $fbn_{i,j,k,h}$ as follows: (i) the architecture ASM_i is loaded in the memory, (ii) the control policy is fixed in the state $CSM_{i,j}$, (iii) the data configuration corresponding to the state $DSM_{k,h}$ is applied.

Running example. *We present in Figure 1 the nested state machines of our FESTO agent. The ASM state machine is composed of two states ASM1 and ASM2 corresponding to the first (i.e. the Light Production Mode) and the second (the High and Medium modes) architectures. The state machines CSM1 and CSM2 correspond to the states ASM1 and ASM2. In CSM 2 state machine, the states CSM21 and CSM22 correspond respectively to the High and the Medium Production Modes (where the second architecture*

is loaded). To fire a transition from CSM21 to CSM22, the value of number_pieces should be in [Constant2, Constant1 [. We note that the states CSM12 and CSM25 correspond to the blocking problem where the two drilling machines are broken. Finally the state machines DSM1 and DSM2 correspond to the state machines CSM1 and CSM2. In particular, the state DSM21 encodes the production periodicity when we apply the High production mode (i.e. the state CSM21 of CSM2), and the state DSM 22 encodes the production periodicity when we apply the Medium mode (i.e. CSM22 of CSM2). Finally, the state DSM23 corresponds to CSM 23 and CSM24 and encodes the production periodicity when one of the drilling machines is broken. We design the agent of our EnAS Benchmark Production System by nested state machines as depicted in Figure 2. The first level is specified by ASM where each state defines a particular architecture of the system. The state ASM_1 (resp. ASM_2) corresponds to the second (resp. first) policy where Control Components that control J_1 , J_2 and G_2 (resp. only J_1 and G_1) are loaded in memory. We associate for each one of these states a CSM in the Control Unit. Finally, Data Unit is specified by DSM which defines the values that Threshold takes under well-defined conditions. Note that if we follow the Second Production Policy (state ASM_1) and the gripper G_2 is broken, then we should change the policy and also the system architecture by loading the Control Component $G1\_CTL$ to remove pieces into $Belt1$. On the other hand, we associate in the second level for the state ASM_1 the CSM CSM_1 that defines the different reconfiguration forms to apply when the first architecture is loaded in the memory. In particular, If the state CSM_{11} is active and the Jack station J_1 is broken, Then we activate the state CSM_{12} in which the Jack station J_2 is running alone to place only one piece in the tin. In this case, the internal behavior of the block $Belt\_CTL$ should be changed (i.e. the tin should be transported directly to the station J_2). In the same way, If we follow the same policy in the state CSM_{11} and the Jack station J_2 is broken, Then we should activate the state CSM_{13} where the behavior of J_1 should be changed to place a piece in the tin that should be closed too (i.e. the behavior of the Control Component $J1\_CTL$ should be reconfigured). We specify finally in Data Unit a DSM where we change the value of Threshold when Gripper G_1 is broken (we suppose as an example that we are not interested in the system performance when this Gripper is broken). By considering this hierarchical model of agents, we specify all possible reconfiguration scenarios that can be applied in embedded control systems: Add-Remove (first level) or Update the structure of Control Components (second level) or just Update data (third level).

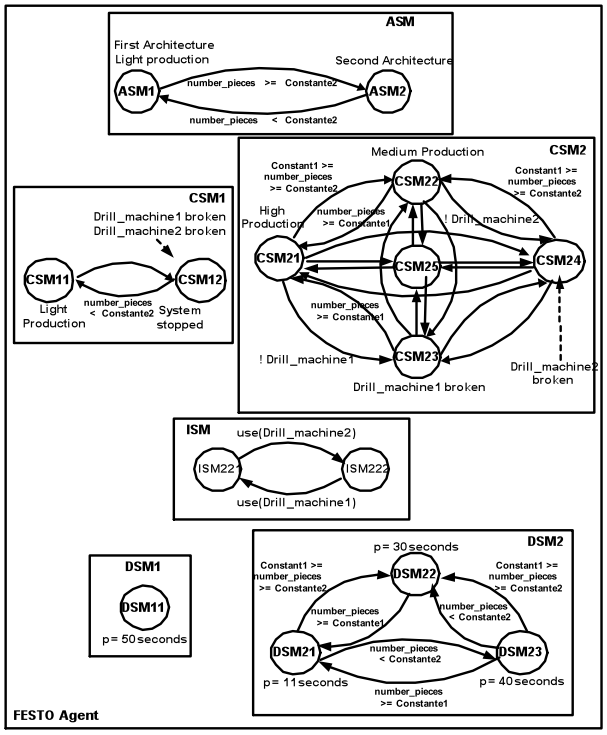

Fig.1. Specification of the FESTO's Agent

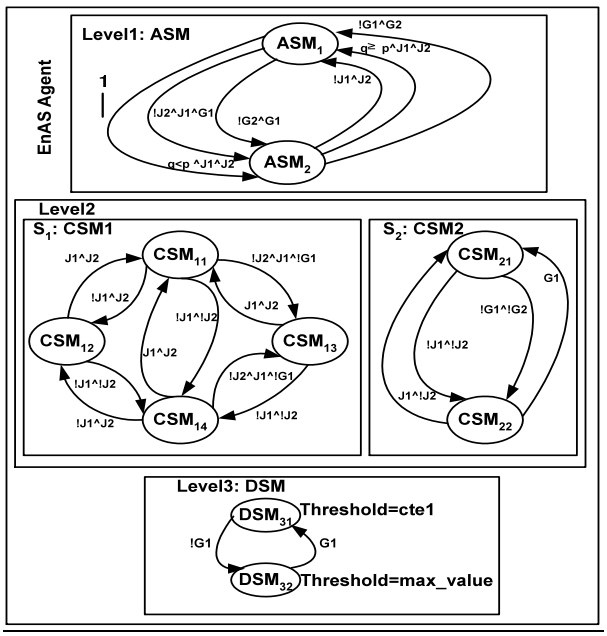

Fig.2. Specification of the EnAS's Agent

3.2. Reconfiguration in a Distributed Architecture

We are interested in automatic reconfigurations of Function Blocks to be distributed on networks of devices where coordination between agents is important because any uncontrolled automatic reconfiguration applied by any agent can lead to critical problems or serious disturbances in others. We define in this section the concept of Coordination Matrix to handle coherent reconfiguration scenarios in distributed devices, and propose thereafter an architecture of multi-agent distributed reconfigurable systems where a communication protocol between agents is defined.

3.2.1. Distributed Reconfigurations

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* $a_{ia,ja,ka,ha}$ a reconfiguration scenario applied by Ag_a ($a \in [1, n]$) as follows:

- The corresponding *ASM* state machine is in the state ASM_{ia}. Let $cond^a_{ia}$ be the set of conditions to reach this state,
- The state machine *CSM* is in the state $CSM_{ia,ja}$. Let $cond^a_{ja}$ be the set of conditions to reach this state,
- The state machine *DSM* is in the state $DSM_{ka,ha}$. 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" of size (*n,4*) that defines coherent scenarios to be simultaneously applied by different agents. Let *CM* be a such matrix that we characterize as follows: each line a ($a \in [1, n]$) corresponds to a reconfiguration scenario $Reconfiguration^a_{ia,ja,ka,ha}$ to be applied by Ag_a as follows:

$$CM[a,1]=i_a ; CM[a,2]=j_a ; CM[a,3]=k_a ; CM[a,4]= h_a$$

According to this definition: **If** an agent Ag_a applies the reconfiguration $Reconfiguration^a_{CM[a,1],CM[a,2],CM[a,3],CM[a,4]}$, **Then** each other agent Ag_b ($b \in [1, n] \setminus \{ a \}$) should apply the scenario $Reconfiguration^b_{CM[b,1],CM[b,2],CM[b,3],CM[b,4]}$. We denote in the following by "idle agent" each agent Ag_b ($b \in [1,n]$) which is not required to apply any reconfiguration when the others perform scenarios defined in CM. In this case:

$$CM[b,1] = CM[b,2] = CM[b,3] = CM[b,4] = 0$$
$$cond^a_{CM[a,1]} = cond^a_{CM[a,2]} = cond^a_{CM[a,3],CM[a,4]} = True$$

We denote in addition by $\xi(Sys)$ 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 Ag_a ($a \in [1, n]$) the following conditions are satisfied:

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

On the other hand, we define "Concurrent Coordination Matrices", CM_1 and CM_2 two matrices of $\xi(Sys)$ that allow different reconfigurations of same agents as follows: $\exists\ b \in [1,n]$ such that:

- $CM_j[b, i] \neq 0\ \forall\ j \in \{1, 2\}$ and $i \in [1,4]$,
- $CM_1[b, i] \neq CM_2[b, i]\ \forall\ i \in [1, 4]$.

In this case, the agent Ag_b is disturbed because it should 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)* the set of concurrent matrices of $CM \in \xi(Sys)$,
- *level*(*CM*) the priority level of the matrix *CM* in the set $Concur(CM) \cup \{CM\}$.

Running Example. *In the Benchmark Production Systems FESTO and EnAS, we show in Figure 3 the Coordination Matrices to be applied in order to guarantee coherent distributed reconfigurations at run-time. According to Figures 1 and 2:*

- *The first matrix CM1 is applied when the FESTO's agent applies the Light Production Mode (i.e. the states ASM_1, CSM_{11} and DSM_{11} are activated and $Reconfiguration_{1,1,1,1}$ is applied), and the EnAS's agent is required to decrease the productivity by applying the First Production Policy to put only one piece in each tin (i.e. the states ASM_2, CSM_{21} are activated and $Reconfiguration_{2,1,0,0}$ is applied),*
- *The second matrix CM2 is applied when the FESTO's agent applies the High Production Mode (i.e. the states ASM_2, CSM_{21} and DSM_{21} are activated and $Reconfiguration_{2;1;2;1}$ is applied) and the EnAS's agent is required to increase the productivity by applying the Second Production Mode to put two pieces into each tin (i.e. the states ASM_1, CSM_{11} are activated and $Reconfiguration_{1,1,0,0}$ is applied),*
- *The third matrix CM3 is applied when the FESTO's agent applies the Medium Production Mode (i.e. the states ASM_2, CSM_{22} and DSM_{22} are activated and $Reconfiguration_{2;2;2;2}$ is applied). In this case the EnAS system is required to apply the Second Production Policy (i.e. the states ASM_1, CSM_{11} are activated and $Reconfiguration_{1,1,0,0}$ is applied),*
- *The fourth matrix CM4 is applied when the Jack station J1 in the EnAS system is broken (i.e. the states ASM_1, CSM_{12} are activated and $Reconfiguration_{1;2;0;0}$ is applied). In this case the FESTO system has to decrease the productivity by applying the Light production Mode (i.e. the states ASM_1, CSM_{11} and DSM_{11} are activated and $Reconfiguration_{1,1,1,1}$ is applied),*
- *The matrix CM5 is applied when the Jack station J2 and the Gripper station G1 are broken in the EnAS system (i.e. the states ASM_1 and CSM_{13} are activated and $Reconfiguration_{1,3,0,0}$ is applied). In this case the FESTO system is required to decrease the productivity by applying the Light Production Mode,*
- *The matrix CM1 is applied at run-time when the Gripper station G2 is broken (i.e. the states ASM_2, CSM_{21} are activated and $Reconfiguration_{2,1,0,0}$ is applied). In this case the FESTO's agent has also to decrease the productivity by applying the Light Production Mode,*
- *The matrix CM6 is applied when the drilling machine Drill_machine1 is broken in FESTO (i.e. the states ASM_2, CSM_{23} and DSM_{23} are activated and $Reconfiguration_{2;3;2;3}$ is applied). In this case, the EnAS system is required to decrease the productivity by applying the First Production Mode (i.e. the states ASM_2, CSM_{21} are activated and $Reconfiguration_{2,1,0,0}$ is applied),*
- *The matrix CM7 is applied at run-time when the second drilling machine is broken at run-time. In this case, the EnAS system is required also the decrease the productivity by applying the First Production Mode,*
- *Finally, the matrix CM8 is applied at run-time to stop the whole production when the two drilling machines Drill_machine1 and Drill_machine2 are broken. In this case, the EnAS's agent has to reach the halt state (i.e. the states ASM_1, CSM_{14} are activated and $Reconfiguration_{1,4,0,0}$ is applied).*

3.2.2. Inter-Agent Protocol for Reconfigurable Systems

We propose a multi-agent architecture for embedded control systems following the Standard IEC61499 to handle automatic distributed reconfigurations of devices. To guarantee a coherent behavior of the whole distributed system, we define a "Coordination Agent" (denoted by $CA(\xi(Sys))$) which handles

the Coordination Matrices of ξ(*Sys*) to control the rest of agents (i.e. Ag_a , $a \in [1, n]$) as follows:

- When a particular agent Ag_a ($a \in [1, n]$) should apply a reconfiguration scenario $Reconfiguration^{a}_{ia,ja,ka,ha}$ (i.e. under well-defined conditions), it sends the following request to *CA*(ξ(*Sys*)) in order to obtain its authorization:

$$request(Ag_a, CA(\xi(Sys)), Reconfiguration^{a}_{ia,ja,ka,ha})$$

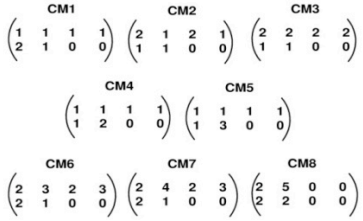

Fig.3. Coordination Matrices For FESTO and EnAS

- When CA(ξ(Sys)) receives this request that corresponds to a particular coordination matrix CM ∈ ξ(Sys) and if CM has the highest priority between all matrices of Concur(CM) ∪ {CM}, then CA(ξ(Sys)) informs the agents that have simultaneously to react with Ag_a as defined in CM. The following information is sent from CA(ξ(Sys)):

For each Ag_b, $b \in [1, n] \setminus \{a\}$ and $CM[b, i] \neq 0$, $\forall\, i \in [1, 4]$:

$$Reconfiguration\ (CA(\xi(Sys)), Ag_b, Reconfiguration^{b}_{CM[b,1],CM[b,2],CM[b,3],CM[b,4]})$$

According to well-defined conditions in the device of each Ag_b, the CA(ξ(Sys)) request can be accepted or refused by sending one of the following answers:

- If $cond^{b}_{ib} = cond^{b}_{jb} = cond^{b}_{kb,hb}$ = True, Then the following reply is sent from Ag_b to CA(ξ(Sys)):

$$Possible\_reconfig(Ag_b , CA(\xi(Sys)), Reconfiguration^{b}_{CM[b,1],CM[b,2],CM[b,3],CM[b,4]})$$

– Else the following reply is sent from Ag_b to CA(ξ(Sys)):

$$Not\_possible\_reconfig(Ag_b, CA(\xi(Sys)), Reconfiguration^{b}_{CM[b,1],CM[b,2],CM[b,3],CM[b,4]})$$

If CA(ξ(Sys)) receives positive answers from all agents, Then it authorizes the reconfiguration in the concerned devices:

$$For\ each\ Ag_b,\ b \in [1, n]\ and\ CM[b,i] \neq 0,\ \forall i \in [1, 4],\ apply(Reconfiguration^{b}_{CM[b,1],CM[b,2],CM[b,3],CM[b,4]})\ in\ device\ b$$

Else If CA(ξ(Sys)) receives a negative answer from a particular agent, Then

– If the reconfiguration scenario $Reconfiguration^{a}_{ia,ja,ka,ha}$ allows an optimization of the whole system's behavior, Then CA(ξ(Sys)) refuses the request of Ag_a by sending the following reply:

$$Refused\_reconfiguration(CA(\xi(Sys)), Ag_a , Reconfiguration^{a}_{CM[a,1],CM[a,2],CM[a,3],CM[a,4]}).$$

4. Model Checking of the Coordination Agent

The model checking of a distributed reconfigurable system is mandatory to check the reactivity of distributed agents when reconfiguration scenarios should be applied in the corresponding devices. We propose a NCES-based model for each Coordination Matrix to be handled by the Coordination Agent, and propose thereafter the verification of the whole system's behavior by applying the model checker SESA and the temporal logic CTL.

4.1. NCES-Based Models for Coordination Agent

We model each Coordination Matrix CM ∈ ξ(Sys) to be handled by the Coordination Agent CA(ξ(Sys)) by a NCES-based Coordination Model in which the conditions $cond^{a}_{ia}$, $cond^{a}_{ja}$ and $cond^{a}_{ka,ha}$ are verified for each non idle agent Ag_a ($a \in [1, n]$) (i.e. application of the reconfiguration scenario $Reconfiguration^{a}_{ia,ja,ka,ha}$) before an authorization is sent into all non idle agents in order to effectively apply corresponding reconfigurations.

Running Example. *We show in Figure 4 the Coordination Module Module($CM_{7,8}$) to be applied when the drilling machines Drill_machine1 or Drill_machine2 are broken (i.e. the states PF4 and PF7 of CSM1(PF1) of FESTO). In this case, the EnAS's agent should reduce the productivity by applying the First Production Mode (i.e. the state PE2 of ASM(EnAS)). We show in Figure 5 the module Module($CM_{4,5}$) that defines the behavior of the Coordination Matrix when the Jack stations J1 , J2 or the Gripper G1 are broken. In this case, the FESTO's agent should reduce the productivity by applying the Light Production Mode (i.e. the state PF1 of ASM(FESTO) of FESTO). We show in Figure 6 the module Module(CM_1) that defines the behavior of the Coordination Matrix when the Light Production Mode is applied by the FESTO's agent (number_pieces < Constant2). In this case, the EnAS's agent should apply the First Production Mode in which only one piece is put in the tin. On the other hand, the module Module($CM_{2,3}$) defines the behavior of the Coordination Matrix when the FESTO's agent should apply the High or the Medium Modes. In this case, the EnAS's agent should change the production strategy to the Second Production Mode where two pieces are put in the tin. To manage Concurrent Coordination Matrices, the resolution of hardware problems is assumed to have a higher priority than any optimization of the system productivity. Therefore, the Coordination Matrix CM7,8 (CM 4,5 ,resp) has a higher priority than CM1 (CM2,3 , resp). According to Figure 6, the matrix CM1 (CM2,3 , resp) is applied if and only if the drilling machines Drill_machine1 and Drill_machine2 (the Jack stations J1 , J2 and the Gripper station G1) are not broken.*

4.2. Verification of Distributed Reconfigurations

We verify with the model checker SESA the behavior of the whole control system when distributed reconfigurations are applied by the Coordination Agent. Indeed, we have to check for each Coordination Matrix CM ∈ ξ(Sys) that whenever an Agent Ag_a ($a \in [1, n]$) applies a reconfiguration scenario under well-defined conditions, the other non-idle agents should react by applying required the corresponding reconfigurations.

Running Example. *In our the benchmark production systems FESTO and EnAS, we apply the model checker SESA to verify the correct feasibility of distributed reconfiguration*

scenarios on NCES-based models of agents. Our objective is to check that whenever one of these demonstrators improves or decreases the productivity, the other applies the same strategy. In addition, we have to check that each one reacts when any hardware problem occurs in the second. This verification is mandatory in order to guarantee coherent behaviors of these complementary demonstrators. The model checker SESA generates for the NCES-based models of the considered agents a reachability graph composed of 162 states. We specify the following functional properties according to the temporal logic CTL:

– ***Property1:*** *whenever the drilling machines Drill_machine1 or Drill_machine2 are broken in the FESTO Benchmark System (i.e. the states PF7 or PF4 are reached), the EnAS system has therefore to decrease the productivity (i.e. the state PE2 is reached). The following formulas are proven to be true by the model checker SESA:*

Formula1: z0 |= AGAte7XAGAtc1XPF7

Formula2: z0 |= AGAte7XAGAtc1XPF4

– ***Property2:*** *whenever the Jack stations J1 and J2 or the Gripper station G1 are broken in EnAS, the FESTO system has to react by decreasing the productivity to the Light Production Mode. The following formulas are proven to be true by the model checker SESA:*

Formula3 (J1 is broken):

z0 |= AGAtf2XAGAtc3XPF12

Formula4 (J2 and G1 are broken):

z0 |= AGAtf2XAGAtc3XPF13

– ***Property3:*** *If the condition number_pieces ≥ Constant2 is satisfied and FESTO improves in this case the productivity to the Medium or the High Modes (i.e. the place PF2 is reached), EnAS should improve also the productivity by applying the Second Production Policy where two pieces are put in each tin. The following formula is proven to be true by the model checker SESA:*

Formula5: z0 |= AGAtf8XAGAtc4XPF2

– ***Property4:*** *If the condition (nb pieces /nb tins+caps > Threshold) is satisfied and EnAS should improve the productivity by applying the second mode (i.e. the place PE1 is reached), FESTO should also increase the productivity by applying the Medium or High Modes (i.e. the state PF1 is reached). The following formula is proven to be true by the model checker SESA:*

Formula6: z0 |= AGAtf1XAGAtc6XPE1

5. Implementation

We developed a complete tool *ProtocolReconf* at INSAT Institute of Carthage University in Tunisia by using Qt Creator 2.0.0 (for more information we refer to [10]). We firstly present its different graphic interfaces, before we show a simulation running the communication protocol. The tool *ProtocolReconf* offers the possibility to create the Reconfiguration and Coordination Agents by introducing their parameters. For the Reconfiguration Agents, it is necessary to define Data, Devices, and all possible Reconfigurations (Figure 7). Each data must be defined by indicating the name and the value, and each device is characterized also by its identifier and state (functional or broken). It is required in addition to define the different scenarios that the Reconfiguration Agent can support so that when a modification occurs in the system, it should look for the convenient reconfiguration. For the Coordination Agent, it is necessary to define the set of Coordination Matrices and especially the current matrix to apply to the whole system (Figure 8). The communication between the different reconfiguration agents follows the specific protocol defined before.

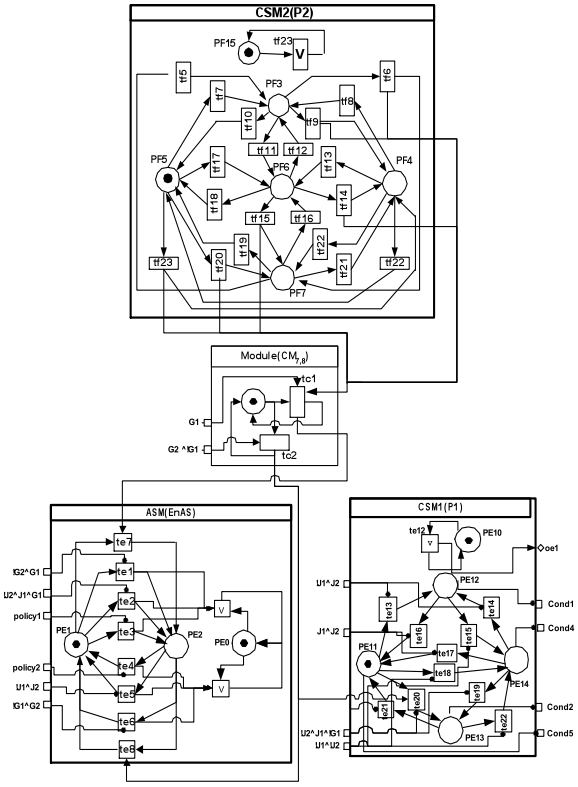

Fig.4. Reconfigurations when the drilling machines are broken

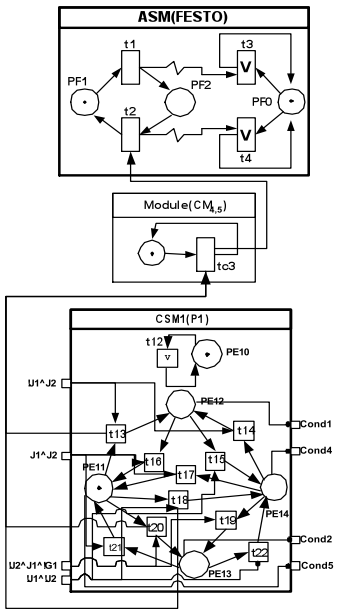

Fig.5. Reconfigurations after when hardware problems

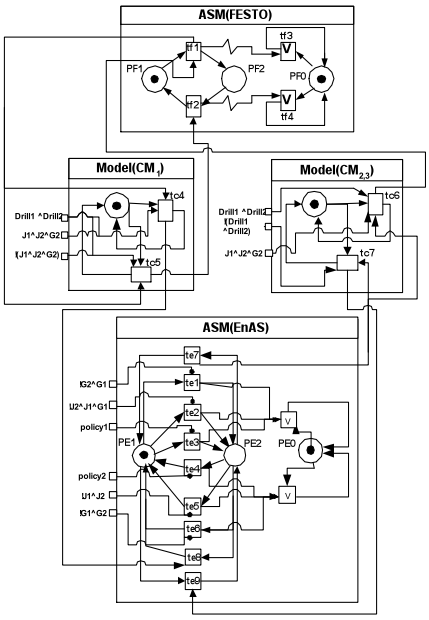

Fig.6. Reconfigurations to regulate the performance

To ensure a new adaptation of the system, a Reconfiguration Agent sends a request to the Coordination Agent indicating the new reconfiguration to apply. Consequently, this coordinator searches the right Coordination Matrix and sends a request to the rest of Reconfiguration Agents. After receiving all the feedbacks, the Coordination Agent decides to apply this new Coordination Matrix (if all Reconfiguration Agents accept this modification) or to cancel the corresponding required reconfiguration scenario.

Running Example1. *In FESTO and EnAS (Figure 9), we assume that the matrix CM2 is applied i.e. the FESTO's agent applies the High Production Mode and the EnAS's agent applies the Second Production Strategy. To verify the interaction between these agents when a particular hardware problem occurs, we change the state of the device Driller1 which becomes broken. Consequently, the FESTO's agent should decrease the production by sending a request to the Coordination Agent in order to look for the most convenient matrix which is CM6. The Coordination Agent sends a request to decrease the production in EnAS. The EnAS's agent studies the feasibility of this new reconfiguration in order to accept the decrease of production. In this case, the Coordination Agent sends a final confirmation to officially apply this new coordination matrix.*

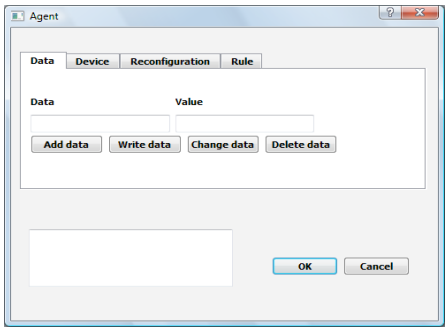

Fig.7. Interface for Reconfiguration Agent

Running Example2. *We assume that the matrix CM6 is applied i.e. the FESTO's agent applies the Low Production Mode and the EnAS's agent applies the First Production strategy (Figure 10). When the state of the device Driller2 becomes broken, the FESTO's agent should stop the production by sending a request to the Coordination Agent in order to halt the second agent. The Coordination Agent decides to apply the matrix CM8 and sends a request to stop the production in EnAS. The EnAS's agent accepts this new reconfiguration. Consequently, the Coordination Agent sends a confirmation to stop the production in both EnAS and FESTO systems.*

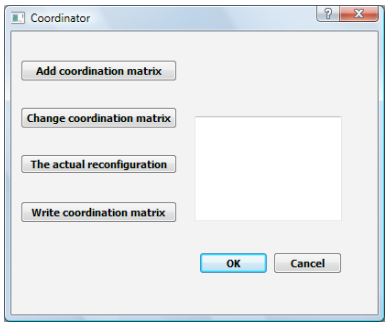

Fig.8. Interface for Coordination Agent

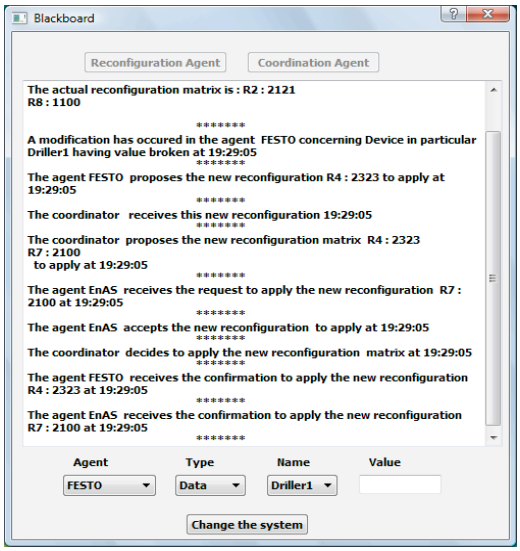

Fig.9. First Example of Communication Protocol

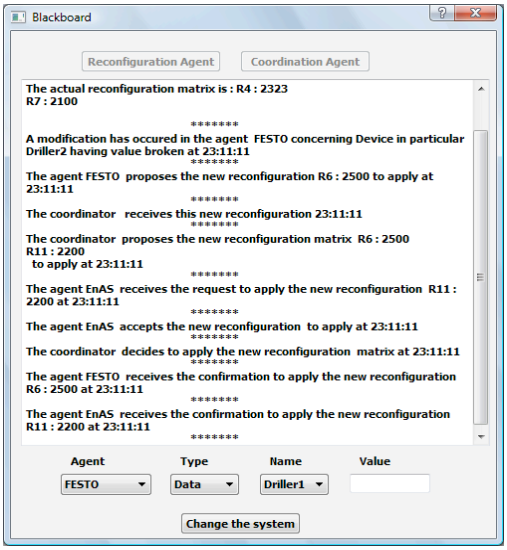

Fig.10. Second Example of Communication Protocol

6. Related work

After presenting our contribution, we discuss in this section the originality of our paper by analyzing related works. The Authors in [6] propose an ontology-based reconfiguration agent to dynamically reconfigure (without any human intervention) the manufacturing system by applying an intelligent semantic. Our work differs because it deals with feasible and coherent distributed reconfigurations to be applied to a distributed architecture. The authors in [17] present a component model named MALEVA to encapsulate different behaviours or activities of a software agent. In this work, there are two types of ports: data ports to transfer data and control ports to activate specific behaviours. Although this work is interesting, the proposed component model is limited and doesn't allow all possible forms of distributed reconfigurations. In [18], the authors define SAASHA as a Self-Adaptable Agent System for Home Automation. The authors distinguish two types of components: graphical user interface components and control device components. A SAASHA agent is proposed to evaluate the environment's evolution before generating convenient control components. Although this work is important, it is restricted to Home Automation unlike our work intended to be used for any embedded system. In addition, the authors do not address reconfigurations in distributed architectures. In [19], the authors propose PROSA as a holonic architecture for manufacturing systems. PROSA offers a set of standard components to be used by the architect to design a manufacturing system where holons are used as autonomous entities. The architecture of a holon is defined as follows: physical processing component, physical control component, decision making component, inter-holons interface and human interface. However in our work, we are mainly interested in the software level (the interaction with the hardware level is ensured through sensors and actuators). The contribution that we propose in the current paper addresses the correct application of coherent distributed reconfigurations of embedded control systems. To our knowledge, this is the first contribution addressing this problem.

7. Conclusion

This paper deals with multi-agent distributed reconfigurable embedded control systems following the standard IEC61499. We classify all possible reconfiguration scenarios into three forms: the first deals with the system architecture, the second with the internal structure of blocks or with their composition, and the third deals with the reconfiguration of data. We define an architecture of reconfigurable multi-agent systems where a Reconfiguration Agent modelled by nested state machines is affected to each device of the execution environment to handle local automatic reconfigurations, and a Coordination Agent handling Coordination Matrices is defined to guarantee safe distributed reconfigurations. We use NCES to model the whole architecture which is verified by applying the model checker SESA. We developed the tool "ProtocolReconf" to simulate the inter-agents communication protocol. The paper's contributions are applied to two Benchmark Production Systems FESTO and EnAS. In our future work, we plan to study the schedulability of distributed reconfigurable Function Blocks in order to meet real-time constraints. We plan also to study their simulation which is not an exhaustive approach. In this case, a technique based on injections of faults should be studied to improve a such verification.

References

[1] Vyatkin, V., Christensen, J., Lastra, J-L.: OOONEIDA: An Open, Object-Oriented Knowledge Economy for Intelligent Distributed Automation. IEEE Transactions on Industrial Informatics, Vol. 1, N. 1, pp.04-17, (2005)

[2] Pratl, G., Dietrich, D., Hancke, G., Penzhorn, W.: A New Model for Autonomous, Networked Control Systems. IEEE Transactions on Industrial Informatics, Vol. 3, N. 1. (2007)

[3] Gehin, A-L., Staroswiecki, M.: Reconfiguration Analysis Using Generic Component Models. IEEE Transactions on Systems, Machine and Cybernetics, Vol.38, N.3. (2008)

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

[5] Rooker, M.N., Sunder, C., Strasser, T., Zoitl, A., Hummer, O., Ebenhofer, G.: 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)

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

[7] Crnkovic, I., Larsson, M.: Building reliable component-based software systems. Artech House, UK, (2002)

[8] Theiss, S., Vasyutynskyy, V., Kabitzsch, K.: Software Agents in Industry: A Customized Framework in Theory and Praxis. IEEE Transactions on Industrial Informatics, Vol. 5, N. 2. (2009)

[9] Khalgui, M., Hanisch, H-M.: NCES-based modelling and CTL-based verification of reconfigurable Benchmark Production Systems. SIES2008 Third international symposium on industrial embedded systems, June 11. (2008)

[10] Qt, cross-platform application and framework, http://qt.nokia.com/products.

[11] Wooldridge, M. Intelligent agents. In G. Weiss (Ed.), Multi-agent systems. MIT Press. (1999).

[12] ROADMAP: Component-based Design and Integration Platforms, Ed Brinksma and all, 2003, http://www.artist-embedded.org.

[13] Rainer Faller, Project experience with IEC 61508 and its consequences, Safety Science, Vol. 42, pp. 405-422, 2004. http.dx.doi.org/10.1016/j.ssci.2003.09.008

[14] Thramboulidis, K., An Architecture to Extend the IEC61499 Model for Distributed Control Applications, 7th International Conference on Automation Technology, 2003

[15] Gharbi, A., Khalgui, M., Ben Ahmed, S.: Inter-Agents Communication Protocol for Distributed Reconfigurable Control Software Components, The International Conference on Ambient Systems Networks and Technologies (ANT-10), 2010

[16] Minhat, M., et al. : A novel open CNC architecture based on STEP-NC data model and IEC 61499 function blocks. Journal of Robotics and Computer-Integrated Manufacturing, Vol. 25, N. 3, (2009)

[17] Briot, J.-P., Meurisse, T. : A Component-based Model of Agent Behaviors for Multi-Agent based Simulations, Proceedings of the 7th International Workshop on Multi-Agent Based Simulation (MABS'06), 5th International Joint Conference on Autonomous Agents and Multi-Agent Systems, pp. 183–190, 2006.

[18] Hamoui, F., Urtado, C., Vauttier, S., Huchard, M., SAASHA : a Self-Adaptable Agent System for Home Automation, Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on, pp. 227 – 230, 2010

[19] Verstraete, P., Germain, B. S., Hadeli, K., Valckenaers, P., Brussel, H. V., On applying the PROSA reference architecture in multi-agent manufacturing control applications, Proceedings of the Multi-agent Systems and Software Architecture, Special Track at Net. Object Days. Erfurt, Germany, Sep 19, 2006 - Sep 19, 2009 (pp. 31-47).