New Multi-Token based Protocol for Flexible Networked Microcontrollers

Imen Khemaissia\textsuperscript{1,2}, Olfa Mosbahi\textsuperscript{3} and Mohamed Khalgui\textsuperscript{3}

\textsuperscript{1}Cynapsys Company, France-Germany
\textsuperscript{2}Faculty of Sciences, Tunis El-Manar University, Tunis, Tunisia
\textsuperscript{3}National Institute of Applied Sciences and Technology, INSAT, University of Carthage, Tunis, Tunisia

Keywords: Microcontroller, Networked STM32F4, Token Ring, Reconfiguration, Multi-agent Architecture, Communication Protocol.

Abstract: This research paper deals with reconfigurable networked microcontrollers following the STM32F4 technology. These controllers based on the token ring architecture, are planned to be reconfigurable according to user requirements, and should be automatically adapted at run-time to their environment. A reconfiguration scenario is assumed to be any run-time automatic addition/removal/update of OS periodic tasks to/from different STM32F4 microcontrollers. Nevertheless, if simultaneous and concurrent scenarios appear in different controllers, then we can get unpredictable critical behaviors of the whole distributed system. A multi-agent architecture is defined where Request and Coordination Agents are assigned to each microcontroller to handle local reconfiguration scenarios after coordination with remote controllers. A tool is developed to simulate a real-case study. We discuss the paper’s contribution by analyzing the experimental results that we did on Networked STM32F4 microcontrollers.

1 INTRODUCTION

The networked microcontrollers are widely used today for the new generations of embedded systems (J. García, ) especially for the modern vehicles in order to offer more functionalities in these automobiles. Due to possible external or also internal disturbances, the system can be automatically adapted by adding/removing/updating OS periodic to/from/in the microcontrollers. Nevertheless, some real-time constraints may be violated and the system becomes unfeasible. Today rich related works have been dedicated to develop reconfigurable real-time embedded control systems (M. Khalgui and Hanisch, 2011) (C. Angelov and Marian, 2005) (K. Thramboulidis and Frantzis, 2004) (George and Courbin, ) (Z. Gu, 2008). Several algorithms are proposed to schedule their changeable OS tasks at run-time (S. Baruah and Rosier, 1990) (Pellizzoni and Lipari, 2004) (Liu and Layland, 1973) (Spuri and Buttazzo, 1996). A reconfiguration can be any operation adapting the hardware or software components. A software reconfiguration is defined by the run-time addition/removal/update of OS tasks to be assumed periodic. A hardware reconfiguration means the activation/deactivation of controllers in order to optimize the energy consumption which is a major problem for embedded systems. We are interested in our work in dynamic software reconfigurations that we assume automatic to be performed at run-time. The goal is to establish a coordination between the different microcontrollers after any reconfiguration. This paper assumes that if a microcontroller \textit{mic}; applies a reconfiguration scenario to adapt its execution according to user requirements for example, the rest should perform also coherent behaviors to be adaptive to this execution. Otherwise, the whole system will be oriented to unpredictable and incoherent behaviors on different microcontrollers. The solution is to allow of course the required reconfigurations at run-time but we should also guarantee a coherent distributed behavior. We aim in this paper to allow flexible networked microcontrollers that should
be adapted to their environment at run-time. nevertheless, this flexibility should guarantee the perfect coherence between microcontrollers. We assume that any possible reconfiguration scenario can only be resumed in one of the following levels: 1) Architecture level: we add-remove OS periodic tasks in a microcontroller, 2) Composition Level: we keep the same architecture of tasks but we change their scheduling in a controller, 3) Data Level: we change the values of data according to user requirements. To allow a required coherent distributed behavior after any reconfiguration scenario, we propose a multi-agent architecture where Request and Coordination Agents (RA and CA) are defined for each microcontroller. RA defines local reconfiguration scenarios in the microcontroller and CA coordinates with the remote controllers to apply this scenario. We propose a Multi-Token based protocol (L. Gauthier and Jerraya, 2001) (Heizinger and Sifakis, 2006) for the coordination between microcontrollers at run-time. This protocol is based on the architecture, composition and data levels. Three types of tokens are proposed: Architecture, Composition and Data Tokens. First of all, if RA wants to apply a reconfiguration scenario that modifies the software architecture, the scheduling and the values of some data; the related CA is requested to perform this modification. The latter sends an Architecture Token to all the remote controllers. This token will be accepted if it has a highest priority among all tokens which are sent on the network. If CA receives the acceptance of this token, then it sends the scheduling one in order to get the acceptance of remote controllers. Then it sends the last data token and waits the acceptance of the remote controllers too. We have used three different types of tokens in order to get an optimal step-by-step adaptation of the system. Indeed, it is not significant to negotiate the scheduling with remote microcontrollers when the new architectures are not yet fixed. We assume in the current paper a distributed embedded system following the STM32F4 technology. This new technology of microcontrollers is well-used today in industry since they are cheaper than other controllers. We developed at the company Cynapsys a tool for the simulation of this protocol on real STM32F4 and we show the benefits of the paper’s contribution. The rest of the paper is organized as follows: In Section 2 the case study to be followed as a running example. We present in Section 3 the proposed Multi-Agent Architecture. The multi-microcontroller protocol is proposed in Section 4, before we implement, simulate and analyze the whole architecture in Section 5. Section 6 concludes finally this research work.

2 CASE STUDY

We assume as a running example a reconfigurable distributed system to be composed of two microcontrollers. In this work, the different LEDs will be used to differentiate the different reconfiguration steps that will be detailed in the next sections. We assume that each microcontroller contains several periodic OS tasks such that each one produces periodic jobs according to (Liu and Layland, 1973). It is characterized by: (1) A release time $R$, (2) A period $T$, (3) A deadline $D$, (4) A WCET $C$ and (5) A static priority $S$. We assume that these tasks are periodic, synchronous, i.e. $R=0$ and with precedence constraints. Table 1 represents the initial tasks of $mic_1$ and $mic_2$. Table 2 represents the added tasks of $mic_1$ and $mic_2$, respectively.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>$C_i$</th>
<th>$D_i/T_i$</th>
<th>$mic_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>10</td>
<td>$mic_1$</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>15</td>
<td>$mic_1$</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>7</td>
<td>30</td>
<td>$mic_1$</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>3</td>
<td>20</td>
<td>$mic_2$</td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>5</td>
<td>20</td>
<td>$mic_2$</td>
</tr>
<tr>
<td>$\tau_6$</td>
<td>6</td>
<td>20</td>
<td>$mic_2$</td>
</tr>
</tbody>
</table>

Table 1: The characteristic of the initial tasks of $mic_1$ and $mic_2$.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>$C_i$</th>
<th>$D_i/T_i$</th>
<th>$mic_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_4$</td>
<td>1</td>
<td>15</td>
<td>$mic_1$</td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>2</td>
<td>30</td>
<td>$mic_1$</td>
</tr>
<tr>
<td>$\tau_6$</td>
<td>1</td>
<td>30</td>
<td>$mic_1$</td>
</tr>
<tr>
<td>$\tau_{10}$</td>
<td>6</td>
<td>20</td>
<td>$mic_2$</td>
</tr>
<tr>
<td>$\tau_{11}$</td>
<td>3</td>
<td>30</td>
<td>$mic_2$</td>
</tr>
<tr>
<td>$\tau_{12}$</td>
<td>9</td>
<td>30</td>
<td>$mic_2$</td>
</tr>
</tbody>
</table>

Table 2: The characteristic of the added tasks of $mic_1$.

We assume the following precedence constraints between the tasks to be uploaded on both $mic_1$ and $mic_2$ in order to offer the global service of the distributed system. We assume that $\tau_4$ must be executed after $\tau_2$ and $\tau_6$ must be executed before $\tau_{10}$. We assume that the latters are reconfigurable. For each controller, we define two different implementations of tasks. Note also that these controllers apply each time a unique distributed reconfiguration scenario.

Running example 1 At a particular time $t_1$, if $mic_3$ decides to apply a reconfiguration scenario by removing $\tau_2$ and $mic_2$ wants to keep $\tau_4$. It is one of the incoherence that can be happen if there is no coordination between the microcontrollers. For that, a protocol should be proposed in the current paper to coordinate between the different microcontrollers and avoid the incoherence in a network of microcontrollers.
3 MULTI-AGENT BASED ARCHITECTURE

We formalize in this section the networked real-time microcontrollers that we assume adaptive to their environment. The distributed system is composed of n STM32F4 controllers to be linked with a token ring topology and serial ports. After applying a reconfiguration scenario, the coherence between the microcontrollers can not be guaranteed. We propose a multi-agent architecture where we propose for each microcontroller a Request Agent RA that defines the reconfigurations to be applied locally, and a Coordination Agent CA which manages any coordination with remote controllers after any reconfiguration scenario.

3.1 Request Agent Modeling

The request agent RA is responsible of the local software reconfigurations in the microcontroller. We consider three forms of reconfigurations to be locally applied at run-time: (i) Architectural Reconfiguration allowing the addition-removal of OS tasks at run-time to-from the controller, (ii) Scheduling Reconfiguration allowing the modification of the composition of tasks without modifying their architecture, (iii) Data Reconfiguration allowing the modification of values of data while keeping the same architecture and scheduling. The idea of our paper is as follows: suppose that a Request Agent of a controller wants to apply a deep reconfiguration that changes the architecture (adds-removes tasks), the scheduling (the composition of some tasks) and the values of some data. This reconfiguration should be applied with the agreement of remote controllers in order to apply a coherent and correct distributed behavior of the whole system. We model the agent RA by nested state machines which are studied in several researches before (Rausch and Hanisch, 1995) (Hanisch and Luder, ). We assume that the architecture level is specified by an architecture state machine $SAm$ for each microcontroller $m \in [1,n]$ where each state represents a particular proposed architecture. We define for each state $SAm$ of $SAm$ a composition state machine to be denoted by $SCm$. It represents the different schedules that can be applied if this particular architecture $SAm$ is applied. The data level is specified by data state machine state machines such that each one $SDm_k$ corresponds to a particular state $SCM_{i,j}$ of $SCM$. The agent automatically applies at run-time different reconfiguration scenarios where each scenario is denoted by $\varphi_{i,j,k,h}$ and corresponds to (i) an architecture state in the top level, (ii) scheduling state in the second level, (iii) data state in the bottom level as follows:

- The architecture State $SAm$: a state of $SAm$ that corresponds to a particular set of tasks to be locally executed in the microcontroller,
- The Composition State $SCm_{i,j}$: a state describing a possible scheduling of these tasks according to user requirements. This state belongs to a state machine that corresponds (in the second level) to $SAm_i$,
- The Data State $SDm_k$: a state that corresponds in the third level to particular values of data that we attach to the tasks of $SAm_i$.

3.2 Coordinator Agent

The coordination agent CA of a microcontroller coordinates with remote controllers to look for the application of the desired reconfigurations from RA. In a particular controller $i (i = 1...n)$, the controller CA sends a request to the remote coordinator agents of the rest of controllers. These coordinators analyze this request with their RA agents to possibly decide if the request of $CA_i$ is acceptable or not. If all the coordinators of the different remote microcontrollers agree the request of $CA_i$, then this reconfiguration is applied since the coherence is guaranteed.

3.3 Token

To apply this protocol, we propose three types of tokens: (i) Architecture Token, (ii) Scheduling Token, (iii) Data Token. The Coordination Agent sends first a Token Agent in order to have an authorization from remote controllers. This token has a priority and can be rejected from the network if another Architecture Token with a higher priority is sent. The Coordination Agent waits the authorization from all remote controllers before applying effectively this reconfiguration. The architecture of the different distributed tasks is fixed at this level in the different controllers. CA sends now a new Scheduling Token in order to define the scheduling of the local tasks and also in the rest of controllers. When an agreement is received, CA sends then a Data Token in order to ask the application of reconfigurations allowing the modification of data. This step-by-step reconfiguration is useful since it is not important to apply architectural, scheduling and data reconfigurations at the same time. We note finally that these tokens are described in the transitions of the nested state machines of each RA.

3.3.1 Token Architecture

The architecture token is defined by a Matrix where the lines represent the microcontrollers of the system
and the columns are as follows (Three columns):

- Column 1: Architecture "added/removed tasks",
- Column 2: Priority of the architecture,
- Column 3: Decision of $CA_i$ (1 or 0). After a reconfiguration request, $CA_i$ will answer by 1 or 0 where:
  - 1, if the reconfiguration is authorized,
  - 0, if the reconfiguration is not authorized.

### 3.3.2 Token Composition

For a particular architecture, a token composition is defined. The columns are as follows:

- Column 1: The given schedule,
- Column 2: The priority,
- Column 3: Decision of $CA_i$.

Note that the priority of each schedule is modified according to user requirement.

### 3.3.3 Token Data

When an agreement is received after the composition level, $CA_i$ sends then a Data Token in order to ask the application of reconfigurations allowing the modification of data. A Data Token is composed two columns that represent the new values of data and its priority that will be applied for a given architecture after scheduling.

### 3.4 Running Example

We define a nested state machine for $RA_1$ and $RA_2$.

#### The architecture level of $mic_1$ is characterized by two different architectures:

- $SA_1$: composed of $τ_1$, $τ_2$ and $τ_4$
- $SA_2$: composed of $τ_3$, $τ_5$ and $τ_6$

For $SA_1$, we have 2 possibilities of schedules:

- $SC_{11}$: $τ_1$, $τ_2$ and $τ_4$
- $SC_{12}$: $τ_2$, $τ_4$ and $τ_1$

For $SA_2$, we have 3 possibilities of schedules:

- $SC_{21}$: $τ_1$, $τ_3$, $τ_5$ and $τ_6$
- $SC_{22}$: $τ_1$, $τ_5$, $τ_6$ and $τ_3$
- $SC_{23}$: $τ_1$, $τ_6$, $τ_3$ and $τ_5$

For $SD_{11}$, the periods must be equal to 20 but for $SD_{12}$, they will be equal to 30 Time Units. For a given $SC_{11}$, $SC_{12}$ and $SC_{13}$ the periods will be equal to 20, 30 and 40, respectively.

#### The architecture level of $mic_2$ is characterized by two different architectures:

- $SA_1$: composed of $τ_7$, $τ_5$ and $τ_9$
- $SA_2$: composed of $τ_8$, $τ_{10}$, $τ_{11}$ and $τ_{12}$

Since $τ_7$ is executed after $τ_2$ and $τ_9$ is only in $SA_1$. Then $τ_2$ will be in $SA_2$ and not $SA_2$. Similar for $τ_7$, it will be in $SA_2$ since it has a precedence constraint with $τ_9$ which is in $SA_1$. For $SA_1$, we have 2 possibilities of schedules:

- $SC_{11}$: $τ_7$, $τ_8$, $τ_{10}$ and $τ_9$
- $SC_{12}$: $τ_{10}$, $τ_8$ and $τ_9$

For $SA_2$, we have 2 possibilities of schedules:

- $SC_{21}$: $τ_{10}$, $τ_8$, $τ_{11}$ and $τ_{12}$
- $SC_{22}$: $τ_8$, $τ_{10}$, $τ_{12}$ and $τ_{11}$

The composition states $SC_{11}$ and $SC_{12}$, the periods are modified to be equal to 30 and 40, respectively. For a given $SC_{11}$, the period is equal to 30 Time Units.

At a particular time $t_1$, $RA_1$ chooses to apply $SA_1$ and $RA_2$ chooses to apply $SA_2$. We assume that $SA_1$ has the highest priority. A token architecture is represented by table 3. If $CA_1$ receive the acceptance of this token from $CA_2$, then it sends the composition one which is defined by table 5 in order to get the acceptance of $CA_2$. If the latter give its authorization, then $CA_1$ sends the last data token shown by table and waits for the answer of $CA_2$.

#### Table 3: Architectural Matrix.

<table>
<thead>
<tr>
<th>$Mic_i/Rec_j$</th>
<th>Architecture</th>
<th>Priority</th>
<th>Decision of $CA_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$mic_1$</td>
<td>$SA_1$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$mic_2$</td>
<td>$SA_2$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Table 4: Data Matrix.

<table>
<thead>
<tr>
<th>$CA$</th>
<th>$data$</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CA_1$</td>
<td>$SD_{11}$</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Table 5: Composition Matrix.

<table>
<thead>
<tr>
<th>$CA$</th>
<th>$Schedule$</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CA_1$</td>
<td>$SC_{11}$</td>
<td>1</td>
</tr>
<tr>
<td>$CA_1$</td>
<td>$SC_{12}$</td>
<td>0</td>
</tr>
</tbody>
</table>

### 4 COMMUNICATION PROTOCOL

We propose a protocol for coherent distributed reconfigurations of networked microcontrollers. Let we assume that this protocol is based on the following three parts:

- **Part 1**: $RA$ of a particular microcontroller detects a reconfiguration request.
Part 2: relation RA – CA: CA receives this request and verifies according to its conditions if RA can apply this reconfiguration or not. CA will check if this applied reconfiguration lead it to apply another corresponding reconfiguration. If it is the case, it will inform the correspond RA to request the new reconfiguration.

Part 3: communication between CA and the remote microcontrollers: every CA asks the rest of the CA if it has the authorization to apply a reconfiguration or not. The reconfiguration can be halted when architecture or composition or data tokens are not authorized. The goal of this protocol is to have useful and optimal distributed reconfigurations: if the microcontrollers do not agree a suggested new architecture, then it is not important to go to scheduling and so on. Algorithm 1 is developed to show the behavior of an RA when it sends a request to CA. Algorithm 2 shows the behavior of the remote CA when they receive this request. This algorithm is applied in all the different level of reconfiguration.

Algorithm 1: Communication Protocol.

Variables:
int CA j ; // Coordination agent for each mic j
int RA j ; // Request agent for each mic j

1. if (RA j sends an architecture request) then
   if (answer(CA) == 1) then
     Invoke algorithm 2;
   end if
   else
     break;
   end if
2. if (RA j sends a composition request) then
   if (answer(CA) == 1) then
     Invoke algorithm 2;
   end if
   else
     break;
   end if
3. if (RA j sends a data request) then
   if (answer(CA) == 1) then
     Invoke algorithm 2;
   end if
   else
     break;
   end if

Algorithm 2: Communication CA-CA.

Variables:
int i=0; // counter for microcontrollers
boolean RR; // reconfiguration request for each level
Matrix of integers:
Mat decision // Matrix containing the decision of the other microcontrollers for each level,
for each CA i , i != j do
  CA i gives its answer:
  Put this value in Mat decision;
end for
for (l = 1; l <= size(Mat decision); i++) do
  if l != j then
    if Mat decision[l] == 1 then
      Message="Reconfiguration is authorized";
      RR = 1;
    else
      RR = 0;
      break;
    end if
  end if
end for

5 EXPERIMENTATION

The contribution of this paper is applied to a ring that will be composed of 2 STM32F4 as said in the sections below. A tool is developed to apply this new simple strategy in order to manage the coordination between the different devices of the system at runtime. We expose in this section our case study.

5.1 Architectural Level 1

The microcontroller mic1 asks mic2 to apply a reconfiguration scenario. It implements the architecture SA11. The microcontroller mic1 activates the yellow led when it asks the reconfiguration. Two cases may happen.

5.1.1 Case 1: Reconfiguration is Authorized

According to table 6 the reconfiguration is authorized since CA2 gives its authorization. Then, mic2 activates the yellow led.

Table 6: Architectural Decision.

<table>
<thead>
<tr>
<th>Mic</th>
<th>Rec</th>
<th>CA1</th>
<th>CA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA1</td>
<td>-</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CA2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.1.2 Case 2: Reconfiguration is not Authorized

Table. 7 shows that the reconfiguration SA11 is not authorized. In this case, mic1 will be light in red.
Table 7: Architectural Decision.

<table>
<thead>
<tr>
<th>MiCm/Reci</th>
<th>CA1</th>
<th>CA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>CA2</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

5.2 Composition Level

The microcontroller $mic_1$ will ask to apply $SD_1$. The microcontroller $mic_2$ will give the authorization to apply this schedule. In this level $mic_1$ will be light in orange after the task addition. Table 8 shows the decision of $CA_2$.

Table 8: Composition Decision.

<table>
<thead>
<tr>
<th>MiCm/Reci</th>
<th>CA1</th>
<th>CA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>CA2</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

5.3 Data Level

The microcontroller $mic_1$ must update the periods of the tasks for a given architecture $SA_1$ and a schedule $SD_1$. $mic_2$ must gives its authorization as shown in table 9. If it accepts this reconfiguration. Then, it will be light in blue. In our example, $mic_1$ will update the relative periods to be equal to 20. The microcontroller $mic_1$ informs the microcontroller $mic_2$ that a particular reconfiguration is applied. As a consequence, the microcontroller $mic_2$ will react and apply a forced reconfiguration. $mic_2$ reacts by lighting in blue like the other microcontrollers.

Table 9: Data Decision.

<table>
<thead>
<tr>
<th>MiCm/Reci</th>
<th>CA1</th>
<th>CA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>CA2</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

6 CONCLUSION

This paper presents a new solution for feasible coordination between $STM32F4$ microcontrollers of a reconfigurable distributed system. A protocol is applied in step-by-step for optimal distributed reconfigurations or the system. We plan in future works to deal with low-power and low-memory reconfigurations of the same architecture by assuming heuristics-based strategies since the problem is NP-Hard.

REFERENCES


