



HAL
open science

New Solutions for Useful Execution Models of Communicating Adaptive RA2DL

Farid Adaili, Olfa Mosbahi, Mohamed Khalgui, Samia Bouzefrane

► **To cite this version:**

Farid Adaili, Olfa Mosbahi, Mohamed Khalgui, Samia Bouzefrane. New Solutions for Useful Execution Models of Communicating Adaptive RA2DL. 14th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMET'2015), Sep 2015, Naples, Italy. pp.87-101, 10.1007/978-3-319-22689-7_7. hal-02425170

HAL Id: hal-02425170

<https://hal.science/hal-02425170v1>

Submitted on 29 Dec 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

New Solutions for Useful Execution Models of Communicating Adaptive RA2DL

Farid ADAILI^{1,2,3}, Olfa MOSBAHI¹
Mohamed KHALGUI¹ and Samia BOUZEFRANE³

¹ LISI Laboratory, INSAT, University Of Carthage, Tunisia

² Tunisia Polytechnic School, University Of Carthage, Tunisia

³ CEDRIC Laboratory, National Conservatory of Arts and Crafts, France

{farid.adaili, samia.bouzefrane}@cnam.fr

{olfamosbahi, khalgui.mohamed}@gmail.com

Abstract. The paper deals with adaptive component-based control systems following the Reconfiguration Architecture Analysis and Design Language (denoted by RA2DL). A system is assumed to be composed a network of RA2DL in coordination. When a fault occurs in the plant, RA2DL component will have a lot of problems to solve such as: the management of the reconfiguration flow, the correction of execution, the synchronization of reconfiguration with the other RA2DL components and the coordination between them. A correction is proposed therefore to improve RA2DL by three layers: the first one is the Middleware reconfiguration (MR) to manage the reconfiguration of RA2DL, the second one is the Execution Controller(EC) which describes the executable and reconfiguration part of RA2DL and the third one is the Middleware Synchronization (SM) for synchronous reconfigurations. When the system is distributed on a network of RA2DL components, we propose a coordination protocol between them using a well-defined matrices to allow feasible and coherent reconfigurations. A tool is developed to simulate our approach. All the contributions of this work are applied to a case study dealing with IEEE 802.11 Wireless LAN.

Keywords: Control System, RA2DL, Reconfiguration, Execution Model, Coordination, Synchronization, Distribution.

1 INTRODUCTION

Nowadays in the academy and manufacturing industry, many research works have been made to deal with real-time reconfiguration of embedded control systems. The new generation of these systems are addressing today a new criteria such as flexibility and agility. To reduce their cost, these systems have to be changed and adapted to their environment without any disturbance.

In the literature, two reconfiguration policies exist, (i) static reconfigurations [3] to be generally applied offline: (ii) and dynamic reconfigurations that can be applied at run-time. We generally define two solutions for the second

case: manual reconfigurations to be applied by users at run-time [18], and automatic reconfigurations which are generally handled by software autonomous agents [8]. We are interested in this paper in automatic reconfigurations of embedded control systems. In order to reduce their development and consequently their time to market, these systems are based on the component-based approach [25]. A component is classically defined as a software unit to be composed with others in order to form the general control functions of the whole system [11]. Two families of components are proposed: the components to be composed at run-time such as .Net [4], Enterprise JavaBeans [12], and the components that should be composed off-line to check their respect of functional and temporal constraints such as IEC61131 [7], IEC61499 [9], Metah [15], Rapide [16], Wright [2] and AADL [23]. We are interested in this work in the AADL technology. AADL component is a software unit to be encoded with a set of algorithms that implement its control functions. Each algorithm is activated by corresponding external event-data inputs, and generally produces the results of its execution on corresponding data-event outputs. It is well-used in many industrial applications such as Avionics Software [24]. We note that a rich library is available today to develop applications in AADL. Nevertheless, these applications are not flexible and cannot be adapted to their environment since Society of Automotive Engineers (SAE) [19] does not provide technical solutions for the possible adaptation of the system based on AADL components at run-time. Moreover, no one in all related works deal with the flexibility of AADL components.

Adaptive systems such as IEEE 802.11 Wireless LAN are composed of networked components. Their logical structure is expressed by an architectural graph in which nodes represent components, and the arcs represent connections between components. In such systems, a dynamic reconfiguration means not only replacing individual components at run-time, but potentially also changing system architecture or structure by adding/removing components and/or changing the patterns of their interconnection between components.

In this work, we are interested in to extender and correct RA2DL by the execution model which are composed of three layers: (i) Middleware Reconfiguration that handles the input reconfiguration flows, (ii) Execution Controller to control the execution and reconfiguration of RA2DL and (iii) Middleware Synchronization that controls and manages the synchronization of the reconfiguration. In the other hand, we propose a new approach about the coordination between several RA2DL components in a distributed architecture. We use the well-known environment UPPAAL [5] to model and verify the correctness of the RA2DL components with its new features. The paper's contribution is applied to a case study of an IEEE 802.11 LAN Wireless system that will be followed as a running example. A tool called *ECReconf* is developed in a collaboration between LISI Lab at University of Carthage in Tunisia and CEDERIC Lab at CNAM in France to implement and simulate the case study.

We present in the next section the background of RA2DL, and define in Section 3 the case study of the IEEE 802.11 LAN Wireless. Section 4 proposes the execution model of RA2DL, Section 5 presents the coherent execution models

of the communication between RA2DL components, and Section 6 defines the modelling and verification where an UPPAAL-based model checking is applied. We propose in Section 7 an implementation and simulation of our solution and conclude the paper in section 8.

2 BACKGROUND

We defined in a previous paper [1] the concept of RA2DL dealing with the reconfigurable AADL components. RA2DL is composed of a controller and a controlled modules where the first one is a set of reconfiguration functions applied in AADL, and the second one is a set of input/output events, algorithms, and data as represented in the four reconfiguration modules. We concentrate on three hierarchical reconfiguration levels in RA2DL: **(i) Form 1:** Architectural level: modifies the component architecture when particular conditions are met. This is done by adding new algorithms, events and data or removing existing operations in the internal behaviors of the component. **(ii) Form 2:** Compositional level: modifies the composition of the internal components (algorithms) for a given architecture. **(iii) Form 3:** Data level: changes the values of variables without changing the component algorithms.

In [14] the authors describe the ADL features which permit the description of dynamic software architectures in which the organisation of components and connectors may change during the system execution, taken from Darwin language, a language used to describe the distributed system structure. In [6] the authors expose the *RUNES* approach (reconfigurable, ubiquitous, and networked embedded systems) which has the general goal of developing an architecture for networked embedded systems that encompasses dedicated radio layers, networks.

The other authors do not provide solutions to develop flexible RA2DL components of adaptive embedded systems. We mean by flexibility the facility to change correctly the behavior of a component according to user requirements and evolution of the environment. The current paper proposes new extension solutions to a correct execution and reconfiguration of a RA2DL component. However, in this work we want to extend this study by considering a distributed system controlled by several interacting RA2DL components.

3 CASE STUDY

IEEE 802.11 Wireless LAN [21] is used as a running example in this paper in order to highlight the contributions of our work. It represents a collection of sensor nodes represented by two RA2DL components: *RA2DL – sender* and *RA2DL – receiver*, connected by a multihop backbone to a channel described by a *RA2DL – channel* component, which is in turn connected to a wired network. The description of the components of case study is as follows:

RA2DL-sender: is for sending packets in the network. It begins with a data packet ready to send, and senses the *RA2DL – channel*. If the channel remains free, then the *RA2DL – sender* enters its vulnerable period and starts sending

a packet (event send), otherwise the *RA2DL – sender* enters a backoff via an urgent transition. The time taken to send a packet is nondeterministic (within T_{min} and T_{max}).

RA2DL-Channel: plays an intermediary role in the network, as a transmission canal of packets between components in the network. The success of the transmission depends on whether a collision has occurred, and is recorded by setting the variable status to the value of the *RA2DL – channel* variable $c1$. The *RA2DL – sender* then immediately tests the *RA2DL – channel* (represented by the urgent location TEST C). If the channel is busy, the *RA2DL – sender* enters the backoff procedure, otherwise it waits for an acknowledgement. If the packet was sent correctly ($status = 1$), then the destination *RA2DL – receiver* waits and sends the acknowledgement; the *RA2DL – sender* then receives this acknowledgement. On the other hand, if the packet was not sent correctly ($status = 2$), then the destination *RA2DL – receiver* does nothing. In this case, the *RA2DL – sender* time out and enters the backoff procedure.

RA2DL-Receiver: represents the component for receiving the packets delivered by *RA2DL – sender*. Messages from *RA2DL – sender* component need to be transmitted across the wired network. If the wired network is busy, these messages should be stored in the *RA2DL – channel* delaying their processing and increasing the buffer space requirements in the *RA2DL – channel*.

The implementation of this case study with the classic RA2DL presents a set of problems: (i) problem of management of the reconfiguration flow if a component receives several reconfigurations at the same time. (ii) Execution problem to resolve the deadlock and the ambiguity when a reconfiguration execution of each RA2DL component occurs. (iii) Synchronization problem when synchronous reconfigurations occur between RA2DL components and (iv) Coordination problem, when RA2DL components are interconnected and communicated between them by reconfiguration flows, data and events.

The component *RA2DL – channel*, receives a set of reconfiguration flows as input at run-time from *RA2DL – sender* and *RA2DL – receiver* components. For example, changing the frequency of sending (S_f) by giving a maximization or a minimization. *RA2DL – channel* has two variables $c1$ and $c2$ which record respectively the status of the packet being sent by *RA2DL – sender* and *RA2DL – receiver*, and which are updated both when a station starts sending a packet (*event send*) or finishes sending a packet (*event finish*). These variables have the following interpretation: $ci = 0$ - nothing being sent by a station i ; $ci = 1$ - packet being sent correctly from station i ; $ci = 2$ - packet being sent garbled from station i . Let show some reconfiguration scenarios that can adapt and coordinate RA2DL components of this application to its environment at run-time. Let suppose that the *RA2DL – sender* sends M packets received by *RA2DL – receiver* at a particular time. Let us denote also by (i) P_i the i -th packet ($i \in [1:M]$) to be sent from the *RA2DL – receiver* with a frequency F_i and a transmission speed TS between T_{min} and T_{max} . Each P_i has a size S_e at the transfer moment from *RA2DL – sender* and S_r in reception from *RA2DL – receiver* (ii) P_i the i -th packet ($i \in [1:P]$) which passes through the

RA2DL – channel and has a boolean variable C (0 if busy, 1 if free), and (iii) we assume in the current paper that we have two threads allowing the emission of packets with two periods according to the channel conditions: the first sends the packet each 6ms when the channel is busy ($C=0$) whereas the second sends each 2ms when the channel is free ($C=1$). We assume the following 6 reconfiguration scenarios:

1. **Reconfiguration 1:** if *RA2DL – sender* sends the packet by *RA2DL – channel* to *RA2DL – receiver*, then the content of the packet must not modify or change the *RA2DL – channel* component.
2. **Reconfiguration 2:** if the reconfiguration of the *RA2DL – channel* component is synchronous or dependent of *RA2DL – sender* and *RA2DL – receiver*,
3. **Reconfiguration 3:** if *RA2DL – channel* receives conflicting reconfigurations to minimize or maximize frequency sending (S_f), then the *RA2DL – channel* component reconfigures the parameter (S_f).
4. **Reconfiguration 4:** if *RA2DL – channel* is busy ($C=0$) then the packet is periodically sent from the *RA2DL – receiver* by a thread $EV - T1$ each 6ms.
5. **Reconfiguration 5:** if *RA2DL – channel* is free ($C=1$) then the packet is periodically sent from the *RA2DL – receiver* by a thread $EV - T2$ each 2ms.
6. **Reconfiguration 6:** if the packet size Se is large when the emission is done, then Se should be compressed in Sr by the *RA2DL – receiver*.

4 EXTENSION TO RA2DL : New Execution Model of RA2DL

We define in [1] a new reconfiguration component RA2DL to control and adapt AADL-based systems to their environment. This RA2DL reacts when an error occurs in the plant and the decision taken may vary from changing the set of RA2DL components that constitute the system, adding-modifying-deleting the internal algorithms/ports, substituting the behavior of some RA2DL components by other behaviors or even modifying data. According to these functionalities, a RA2DL component presents some gaps which remain unresolved like the management of different reconfigurations, the synchronization, the coordination and the distribution. In this paper, we enrich RA2DL by an execution model that undergoes such a failure and to ensure better distribution between RA2DL components. The execution model is composed of three layers (Middleware reconfiguration, Execution Controller and Middleware synchronization) as presented in Fig.1.

4.1 Middleware Reconfiguration layer

The Middleware Reconfiguration layer (MR) is dedicated to receive all the reconfiguration Flows (RF) from the input port (IRF). Each RF has a token RT

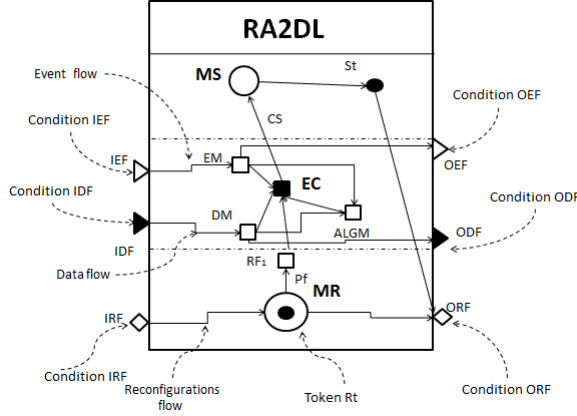


Fig. 1. Execution Model of RA2DL

containing the necessary information such as the address of RA2DL destination DA , a binary variable V ($V = 1$ if the reconfiguration is synchronous, otherwise $V = 0$) and a priority factor PF gave by the user. Secondly this layer represents the RA2DL manager; it decides whether the RF is associate for it or not, if not associated, it sends to its successor by ORF output. In the case it receives concurrent or contradictory RF , the layer decides which reconfiguration will be accepted using the PF .

Running Example: We suppose two reconfiguration scenarios $RS1$ and $RS2$. $RS1$ is assumed to amplify frequency F_s for $RA2DL - channel$ and $RS2$ is assumed to minimize the size of the packet for the $RA2DL - receiver$. The token RT is represented in Table 1.

	DA (Destination Address)	Synchronous	Asynchronous	PF
RS1	RA2DL-receiver	0	1	1
RS2	RA2DL-channel	1	0	3
RR	RA2DL-channel	0	1	2
RC	RA2DL-sender	0	1	4

Table 1. Token Informtion

According to Table 1, we have two problems: (i) two contradictory reconfigurations appear at the same time in $RA2DL - channel$: $RS2$ from $RA2DL - sender$ to amplify frequency F_s and RR from $RA2DL - receiver$ to minimize F_s . In this case, RM compares the two priority factors $PF(RS1 = 3) > PF(RR = 2)$. $RS1$ will be accepted and RR will be rejected. (ii) We have synchronous and asynchronous reconfigurations. When the packet is well transmitted, the reconfiguration is asynchronous when $RA2DL - channel$ changes the variable $c1$ to

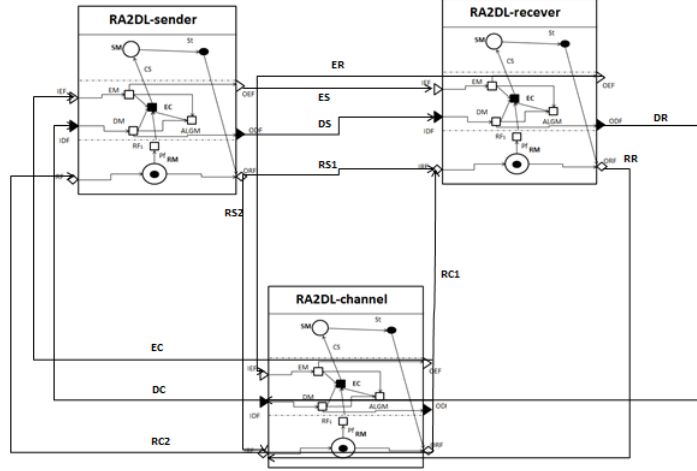


Fig. 2. RA2DL components

c2. And when the reconfiguration *RC2* adds a new byte in *RA2DL – sender*, this reconfiguration should be synchronized by also adding a byte with *RS1* in *RA2DL – receiver* and automatically passes to the middleware synchronization that will be explained later.

4.2 Execution Controller layer

The *ExecutionController* (*EC*) layer is responsible of the reconfiguration execution part of RA2DL having two input/output ports (the first for data flow and the second for events flow) and algorithms (*Alg*) of RA2DL. The *EC* is assumed to be encoded in three hierarchical levels (a) Architecture Level (to be denoted by *AL*), (b) Composition Level (to be denoted by *CL*), and (c) Data Level (to be denoted by *DL*). We define in *AL*, all the possible architectures that can implement the RA2DL component at run-time. An architecture in *AL* is a set of algorithms (*Alg*) that perform control activities. A reconfiguration scenario can change the architecture of the *RA2DL* component by adding or also removing algorithms. For each architecture in *AL*, we need to define an execution model of the corresponding algorithms. A composition is then defined in *CL* to affect a priority to each algorithm. For each architecture and for each composition of the corresponding algorithm, we define also in Data level, all the possible corresponding values of data to be handled at run-time.

Running Example: We have two architectures in the IEEE 802.11 Wireless LAN as depicted in Fig.3. The first one is when the *RA2DL – channel* is busy ($C = 0$), and is implemented with the first architecture (*ASM1*). The second one is when the *RA2DL – channel* is free ($C = 1$) and is implemented with the second architecture (*ASM2*).

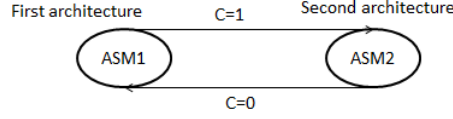


Fig. 3. Architecture Level

4.3 Middleware Synchronization layer

The reconfiguration of each RA2DL component in each round is independent of the other RA2DL components and the output generated by a RA2DL component in a round is the input of the next round. In RA2DL technology, such RA2DL components should be executed asynchronously. However, since the RA2DL components are independent in each round, the final states in each round are the same in both asynchronous and synchronous reconfigurations. This layer has a Synchronization Token (ST). If the reconfiguration is synchronized with the other RA2DL components, ST sends with the address, the RA2DL components involved in the reconfiguration pass to semaphore state (S). If the reconfiguration is asynchronous, this layer is not considered.

Running Example: The reconfiguration of the *RA2DL – channel* component is synchronous or dependent with *RA2DL – sender* and *RA2DL – receiver*, when the *RA2DL – channel* has collision problems. In this case, a synchronous flow of reconfiguration is sent to *RA2DL – sender* and *RA2DL – receiver* to inform for a new reconfiguration to apply. Automatically, *RA2DL – sender* and *RA2DL – receiver* pass to semaphore state (S).

```

Semaphore (RA2DL-sender, RA2DL-receiver)
RA2DL-sender(s:)
RA2DL-receiver(s):
n(s) := n(s) - 1;
if n(s)<0 then;
State(RA2DL-sender) := blocked;
State(RA2DL-receiver) := blocked;
enter (RA2DL-sender, f(s))
enter (RA2DL-receiver, f(s))
  
```

5 COHERENT RECONFIGURABLE EXECUTION MODELS IN DISTRIBUTED ARCHITECTURES

In this section, we define a new component named *RA2DL – coordinator* for the coordination between all RA2DL components. Each RA2DL is specified by nested state machines that support all reconfiguration forms. Nevertheless, the coordination between execution models in this distributed architecture is extremely mandatory because any uncontrolled automatic reconfiguration applied

in a RA2DL can lead to critical problems. To guarantee safe distributed reconfigurations, we define the concept of *Coordination Matrix (CM)* that defines correct reconfiguration scenarios.

5.1 Distributed RA2DL architecture

Let Sys be a distributed reconfigurable system composed of n RA2DL components, and let $RA2DL_1, \dots, RA2DL_n$ be n RA2DL components to handle automatic distributed reconfiguration scenarios of these components. We denote in the following by $Reconfiguration_{ia,ja,ka}^a$ a reconfiguration scenario applied by $RA2DL_n$ ($a \in [1, n]$) as follows: (i) the corresponding *ASM* state machine is in the state ASM_{ia} . Let $cond_{ia}^a$ be the set of conditions to reach this state, (ii) the *CSM* state machine is in the state CSM_{ja} . Let $cond_{ja}^a$ be the set of conditions to reach this state, (iii) the *DSM* state machine is in the state DSM_{ka} . Let $cond_{ka}^a$ 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 $(n, 3)$ that defines coherent scenarios to be simultaneously applied by different *RA2DL* components as presented in Fig.4 . Let CM be such a matrix that we characterize as follows: each line a ($a \in [1; n]$) corresponds to a reconfiguration scenario $Reconfiguration_{ia,ja,ka}^a$ to be applied by $RA2DL_a$ as follows:

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

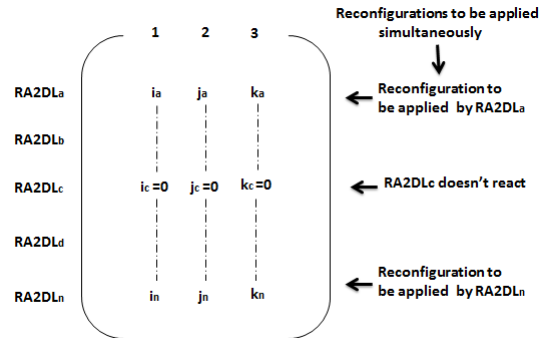


Fig. 4. Coordination Matrix

5.2 Coordination Between Distributed RA2DL components

We propose a new architecture for control systems following the Standard RA2DL to handle automatic distributed reconfigurations of components. To guarantee a coherent behavior of the whole distributed system, we define *RA2DL* –

coordinator (denoted by $CR(\Omega(Sys))$) which handles the Coordination Matrices of $\Omega(Sys)$ to control the rest of RA2DL components (i.e. $RA2DL_a \in [1:n]$) as follows: when a particular $RA2DL_a$ ($a \in [1:n]$) should apply a reconfiguration scenario $Reconfiguration_{ia,ja,ka}^a$ (i.e. under well-defined conditions), it sends the following request to $CR(\Omega(Sys))$ to obtain the authorization.

$$request(RA2DL_a, CR(\Omega(Sys)), Reconfiguration_{ia,ja,ka}^a)$$

When $CR(\Omega(Sys))$ receives this request that corresponds to a particular coordination matrix $CM \in \Omega(Sys)$ and if CM has the highest priority between all matrices of $\Omega(Sys)$, then $CR(\Omega(Sys))$ informs the Control RA2DL that it should react simultaneously with RA2DL as defined in the CM . The following information is sent from $CR(\Omega(Sys))$:

For each $RA2DL_b$, $b \in [1, n] \setminus \{a\}$ and $CM[b, i] \neq 0 \forall i, i \in [1, 3]$:

$reconfiguration((CR(\Omega(Sys)), RA2DL_b, Reconfiguration_{CM[b,1], CM[b,2], CM[b,3]}^b)$

Running example: In the communication IEEE 802.11 Wireless LAN, we distinguish three kinds of participating components:

- The *RA2DL – sender* (station 1): it starts the communication and when an error occurs in a specific plant, the associate *RA2DL – receiver* tries to correct it and if it decides the necessity of reconfiguration in the whole network (i.e. the other RA2DL components must be aware of this modification) it informs the *RA2DL – coordinator*.
- The *RA2DL – coordinator* (CR): it is the main component which aims to coordinate between the different RA2DL components. When it receives a reconfiguration request, it searches the list of RA2DL components which should be informed. It sends a request to these RA2DL components and waits for their response.
- The *RA2DL – receiver* (station 2): it is the component that receives a reconfiguration request from the *RA2DL – coordinator* component. Firstly, it checks the possibility to apply a reconfiguration. If it is possible, it sends a positive answer, otherwise it sends a negative answer.

Fig.5 shows the coordination between these three RA2DL components when the packet size is large. In this case, the CR uses the Matrix CM to compress the size of packets in the *RA2DL – receiver*.

6 MODELING AND VERIFICATION OF DISTRIBUTED ARCHITECTURE RA2DL

We propose in this section the modelling and verification of RA2DL by using UPPAAL. Firstly, we model the execution model of RA2DL with the three layers (RM, EC, SM) by Nested State Machines. Secondly, we model the coordination part with CR and the coordination matrix CM . Thirdly, we check a set of properties to ensure the security and flexibility of our case study.

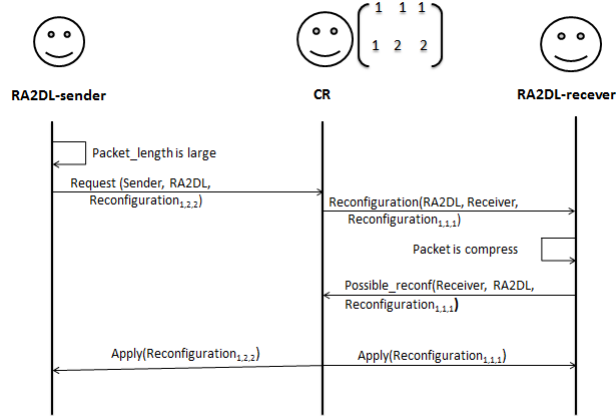


Fig. 5. Coordination between the RA2DL-sender and RA2DL-receiver

Modelling of the execution model The modelling of the execution model with the three layers is described by the state machine presented in Fig.6.

The states of execution model are described as follows: *start* to start the reconfiguration, *test* to test the condition *condition – IRF* entering at the port *IRF*, *RT* state to test the reconfiguration acceptance with the reconfiguration token in *MR* if it corresponds to the greatest priority factor *PF*. *EC* state corresponds to the execution controller layer with *EM* for events and *DM* data of RA2DL, *Reconfiguration* state to apply the reconfiguration request. It tests whether if the reconfiguration is synchronous or asynchronous. *SM* state describes the middleware synchronization. If the reconfiguration of the current component is synchronous with another RA2DL component, the semaphore (*S*) is used in *waiting* state. *St* is a state to send the reconfiguration to the target component. *ORF* corresponds to the final state of the execution model.

6.1 Modeling coordination

We present in Fig.7 the modeling of the RA2DL coordinator between the RA2DL components, and the communication process between them. *RA2DL – server* state describes the packet transmission and *RA2DL – receiver* state describes the packet receiving when an error occurs in *RA2DL – server*. This component sends a reconfiguration request to *RA2DL – coordinator*. The latter with the coordination matrix *CM* ensures the passage of the requests between the two components while a reconfiguration is executing.

6.2 Verification

We check a set of properties for the correct behavior of the execution model and better coordination between the RA2DL components in the systems after any reconfiguration scenario in order to avoid any unpredictable execution.

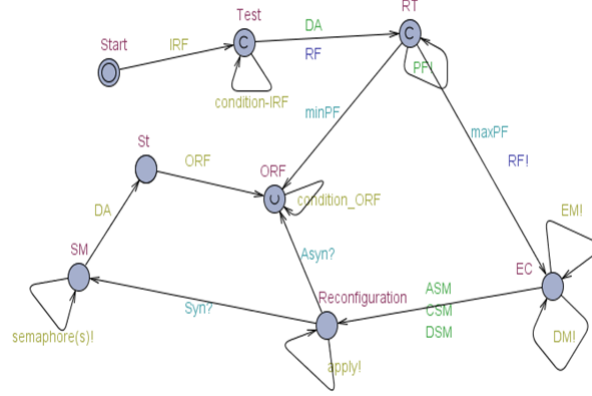


Fig. 6. Modeling of Execution Model

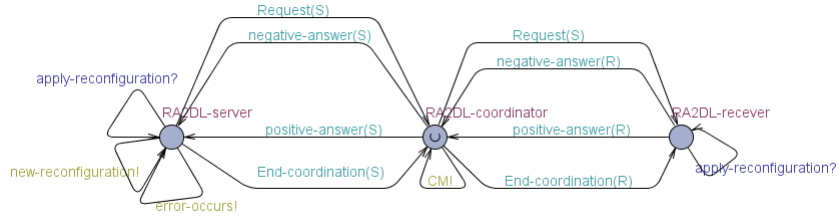


Fig. 7. Coordination between RA2DL-server and RA2DL-receiver

Running Example: In the assumed IEEE 802.11 Wireless LAN, we check simple reachability, safety, liveness and deadlock-free properties. The simple reachability properties are checked if a given location is reachable:

Property 1: $RT[] . PF! . maxPF(RF)$: the execution model has to manage competition reconfigurations according to the PF and has to decide which reconfiguration should be executed the first.

Property 2: $EC[] . DM!$ and $EM!(ASM.CSM.DSM)$: when a reconfiguration RF is selected in the RM layer, we must specify the architecture, composition and data levels in the execution controller level EC .

Property 3: $Reconfiguration[] . apply!$: if the reconfiguration is synchronous, it passes to the SM layer otherwise it will be sent to the input port of the next RA2DL component.

Property 4: $SM[] . semaphore(s) \Rightarrow St[]$: if the reconfiguration is asynchronous, the SM must necessarily send a token to block the target with a $semaphore(S)$.

Property 5: $RA2DL-server \square .error-occurs \Rightarrow RA2DL-coordinator$: this property means that when an error occurs in RA2DL, the $RA2DL-coordinator$ is informed.

Property 6: $NOT(RA2DL-sender \square \text{ AND } RA2DL-receiver \square)$ This property means that we could not receive two different notifications from the $RA2DL-coordinator$ at the same time (i.e. notification that the other $RA2DL-receiver$ accept or refuse the new reconfiguration).

Property 7: $(RA2DL-sender \square \text{ AND } RA2DL-coordinator \square) \text{ AND } RA2DL-receiver \square$ *not deadlock*: the system is deadlock-free.

The verification of these properties is summarized in Table.2

Property	Result	Time (sec)	Memory (Mo)
Property 1	True	12.03	5.34
Property 2	True	5.9	3.56
Property 3	True	10.23	5.78
Property 4	True	11.34	4.23
Property 5	True	7.24	3.96
Property 6	True	8.12	3.45
Property 7	True	14.39	6.78

Table 2. Verification result

7 IMPLEMENTATION AND SIMULATION

To simulate the behavior of a distributed Architecture RA2DL, we develop a prototype tool called ECRconf. First, we present the different graphical interfaces of the ECRconf with the execution model and its three layers as presented in Fig.8.

We start by showing the simulations of the coordination and the communication between the different RA2DL components, the $RA2DL-coordinator$ and the *coordination - matrix* in Fig.9.

Second, we present a graph showing gains compared to the old RA2DL in response time and how the RA2DL component becomes correct. The result of the simulation after using the execution model is represented in Fig.10.

Runing Example: In IEEE 802.11 Wireless LAN, the $RA2DL-coordinator$ applies the reconfiguration for minimizing the packet size from Se in $RA2DL-sender$ to Sr in $RA2DL-receiver$, the result is shown in Fig.11.

8 CONCLUSION

The paper deals with new solutions for a required flexibility of adaptive control systems. Firstly, we define an execution model which represents a correct model

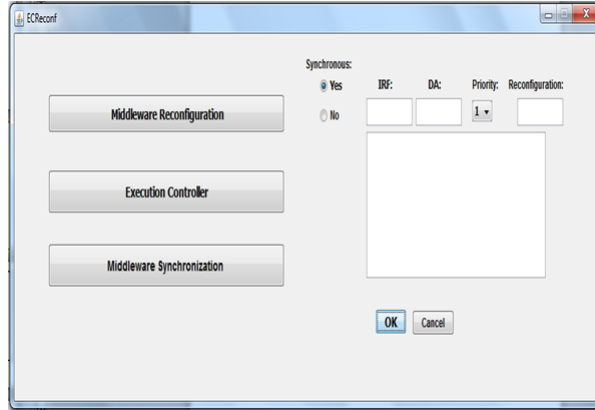


Fig. 8. ECreconf Tool

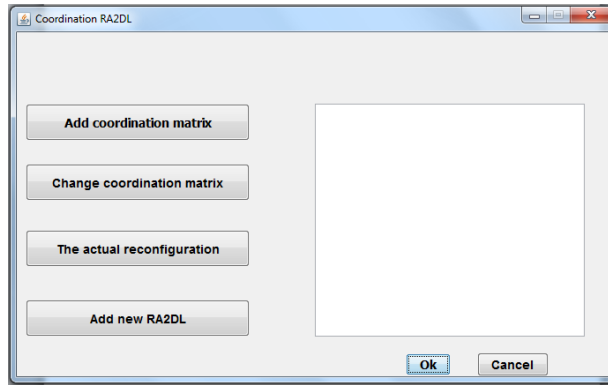


Fig. 9. Coordination RA2DL

of RA2DL in three-layer: (i)Middleware Reconfiguration (MR), Execution controller (EC) and Middleware Synchronization (MS). Secondly, we propose a coordination and communication between execution models for each RA2DL for the distributed reconfigurations, In this case, we define an RA2DL architecture where each RA2DL (associated to a defined component) controls the environment evolution and applies automatic reconfigurations when errors occur at run-time to guarantee a functional safety. Besides, the Coordination RA2DL is responsible for the interaction between each RA2DL component of the system in order to ensure a mutual agreement with the others by the applied reconfiguration. These components use the coordination matrix to define for each RA2DL the applied reconfiguration according to predefined conditions. The model checking is used to prove the correctness of the execution model. The coordination model is represented by Nested state machine. Finally, the "ECreconf" tool is

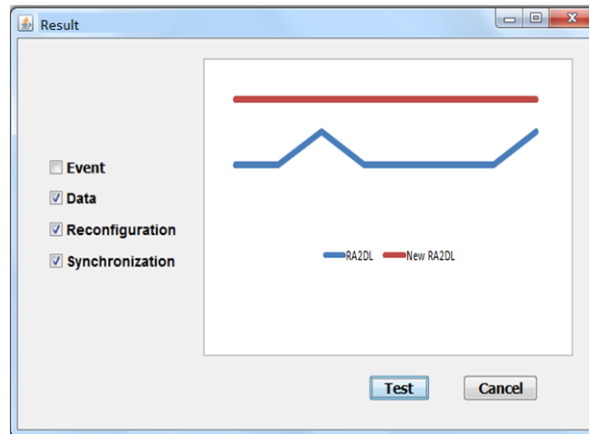


Fig. 10. Result of simulation

used to simulate the execution model with the three layers and the coordination between RA2DL components, it is applied to an IEEE 802.11 Wireless LAN.

The future works will deal with the security of RA2DL-based systems where the reconfigurable security will be an issue to be discussed on RA2DL.

References

1. F. ADAILI, O. Mosbahi, m. khalgui, and S. Bouzefrane. Ra2dl: New flexible solution for adaptive aadl-based control components. *5th international conference on Pervasive and embedded computing and communication systems*, 2015.
2. Robert Allen, Rmi Douence, and David Garlan. Specifying and analyzing dynamic software architectures, 1998.
3. Christo Angelov, Krzysztof Sierszecki, and Nicolae Marian. Design models for reusable and reconfigurable state machines. In *in L.T. Yang et al. (Eds.): Proc. of EUC 2005, LNCS 3824*, pages 152–163, 2005.
4. B. Baudry, F. Fleurey, J.-M. Jezequel, and Y. Le Traon. Automatic test case optimization using a bacteriological adaptation model: application to .net components. In *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on*, pages 253–256, 2002.
5. Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal;a tool suite for automatic verification of real-time systems. pages 232–243, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.
6. P. Costa, G. Coulson, C. Mascolo, G.P. Picco, and S. Zachariadis. The runes middleware: a reconfigurable component-based approach to networked embedded systems. In *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on*, volume 2, pages 806–810 Vol. 2, Sept 2005.
7. M. de Sousa. Data-type checking of iec61131-3 st and il applications. In *Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on*, pages 1–8, 2012.

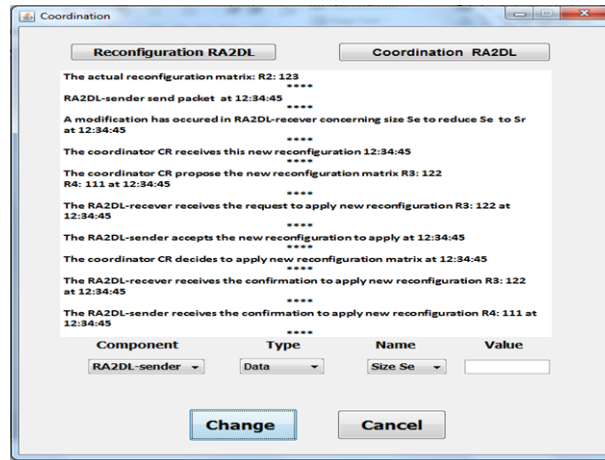


Fig. 11. Example of communication and reconfiguration

8. Mohamed Khalgui. Nces-based modelling and ctl-based verification of reconfigurable embedded control systems. *Comput. Ind.*, 61(3):198–212, April 2010.
9. Mohamed Khalgui. Distributed reconfigurations of autonomous iec61499 systems. *ACM Trans. Embed. Comput. Syst.*, 12(1):18:1–18:23, January 2013.
10. Dries Kimpe, Philip H. Carns, Kevin Harms, Justin M. Wozniak, Samuel Lang, and Robert B. Ross. Aesop: Expressing concurrency in high-performance system software. In *NAS*, pages 303–312, 2012.
11. Jihyun Lee and Jin-Sam Kim. A methodology for developing component-based software with generation and assembly processes. In *Advanced Communication Technology, 2004. The 6th International Conference on*, volume 2, pages 696–699, Feb 2004.
12. Yan Liu, I. Gorton, A. Liu, and Shiping Chen. Evaluating the scalability of enterprise javabeans technology. In *Software Engineering Conference, 2002. Ninth Asia-Pacific*, pages 74–83, Dec 2002.
13. F. Luders. Adopting a software component model in real-time systems development. In *Software Engineering Workshop, 2003. Proceedings. 28th Annual NASA Goddard*, pages 114–119, Dec 2003.
14. Jeff Magee and Jeff Kramer. Dynamic structure in software architectures. In *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering, SIGSOFT '96*, pages 3–14, New York, NY, USA, 1996. ACM.
15. Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.
16. Karen Palma, Yadrn Eterovic, and Juan Manuel Murillo. Extending the rapide adl to specify aspect oriented software architectures. In *SEDE*, pages 170–167, 2006.
17. A. Prijic, Z. Prijic, D. Vuc-kovic, and A. Stanimirovic. Aadl modeling of m2m terminal. In *Microelectronics Proceedings (MIEL), 2010 27th International Conference on*, pages 373–376, May 2010.
18. Martijn N. Rooker, Christoph Sünder, Thomas Strasser, Alois Zoitl, Oliver Hummer, and Gerhard Ebenhofer. Zero downtime reconfiguration of distributed au-

- tomation systems: The εcedac approach. In *Proceedings of the 3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems: Holonic and Multi-Agent Systems for Manufacturing, HoloMAS '07*, pages 326–337, Berlin, Heidelberg, 2007. Springer-Verlag.
19. SAE. Architecture analysis & design language (standard sae as5506). September 2004.
 20. Young-Jun Seo, Young-Jae Song, and Hwa-Young Jeong. Acme-based connector interface considering component important factor. In *SKG International Conference on Semantics, Knowledge and Grid (SKG 2005), 27-29 November 2005, Beijing, China*, page 54. IEEE Computer Society, 2005.
 21. Oleg Sokolsky and Alexander Chernoguzov. Performance analysis of AADL models using real-time calculus. In *Foundations of Computer Software. Future Trends and Techniques for Development, 15th Monterey Workshop 2008, Budapest, Hungary, September 24-26, 2008, Revised Selected Papers*, pages 227–249, 2008.
 22. Teng teng Zhang, Jian min Wu, Lin Qi, and Hai yu Xu. Architecture analysis and design language amp; harmony system engineering process. In *Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st*, pages 7D2–1–7D2–12, Oct 2012.
 23. Thomas Vergnaud, Laurent Pautet, and Fabrice Kordon. Using the aadl to describe distributed applications from middleware to software components. In *Proceedings of the 10th Ada-Europe International Conference on Reliable Software Technologies, Ada-Europe'05*, pages 67–78, Berlin, Heidelberg, 2005. Springer-Verlag.
 24. Ying Wang, Dianfu Ma, Yongwang Zhao, Lu Zou, and Xianqi Zhao. An aadl-based modeling method for arinc653-based avionics software. In *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual*, pages 224–229, July 2011.
 25. Lin Zhu, Xinran Li, Hui Ouyang, Yanan Wang, Weijian Liu, and Kun Shao. Research on component-based approach load modeling based on energy management system and load control system. In *Innovative Smart Grid Technologies - Asia (ISGT Asia), 2012 IEEE*, pages 1–6, May 2012.