ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 5, No. 6, 531-537 (2020)

www.astesj.com

Second International Virtual Conference on Multidisciplinary Research 2020

ASTESJ

ISSN: 2415-6698

Determinism of Replicated Distributed Systems–A Timing Analysis of the Data Passing Process

Adriano A. Santos[*,1], António Ferreira da Silva[1], António P. Magalhães[2], Mário de Sousa[3]

[1]*Centre for Research & Development in Mechanical Engineering (CIDEM), Mechanical Department, School of Engineering (ISEP), Polytechnic of Porto, 4249-015 Porto, Portugal*

[2]*Mechanical Engineering Department, Faculty of Engineering, University of Porto (FEUP), 4200-465 Porto, Portugal*

[3]*Electrical and Computer Engineering Department, Faculty of Engineering, University of Porto (FEUP), 4200-465 Porto, Portugal*

ARTICLE INFO

Article history:
Received: 10 August, 2020
Accepted: 07 October, 2020
Online: 20 November, 2020

Keywords:
Distributed Systems
Event-Base Control
Fault-Tolerance (FT)
IEC 61499
Industrial Control
Real-Time (RT)
Replication
Commercial Off-the-Shelf (COTS)

ABSTRACT

Fault-tolerant applications are created by replicating the software or hardware component in a distributed system. Communications are normally carried out over an Ethernet network to interact with the distributed/replicated system, ensuring atomic multicast properties. However, there are situations in which it is not possible to guarantee that the replicas process the same data set in the same order. This occurrence will lead to inconsistency in the data set produced by the replicas, that is, the determinism of the applications is not guaranteed.

To avoid these inconsistencies, a set of Function Blocks has been proposed which, taking advantage of the inherent properties of Ethernet, can guarantee the synchronism and determinism of the real-time application. This paper presents this set of Function Blocks, focusing our action on the development of reliable distributed systems in real-time. This demonstrates that the developed Function Blocks can guarantee the determinism of the replicas and, as such, that the messages sent are processed, in the same order and according to the time in which they were made available.

1. Introduction

Fault-tolerance or replication is implemented, in most cases, by replication of one or more critical components, by replication of the hardware, the software or both. Therefore, regardless of the approach used, the main objective is to ensure that if one of the replicas fails, the remaining replicas will continue to function and therefore mask the existence of the failed replica before the remaining application, making it as transparent as possible. So, we must consider a fault-tolerant distributed system as an interconnection of several unitary components that, in each call of an event, process data, generating new events and/or data. On the other hand, event and data, generated on a different replica, located to different nodes, must be synchronized to assure that the replicas receive the same set of data in the same order. All replicas must have the same perception of the data and this perception will be obtained through a multicast protocol.

To support the Distributed Computer Controller System (DCCS) in real-time and reliably replicated over Commercial Off-The-Shelf components (COTS), it is essential to provide a simple and transparent programming model. So, programmers should be unaware of implementation problems and the details of distribution and replication. However, it is important that the replication mechanism allows us to develop a generic and transparent approach without concerns the inherent requirements of the distributed system and the replication issues. Therefore, to overcome the problems inherent to the development of distributed/replicated systems, we opted for the use of a framework that guarantees not only the required abstraction capacity but also the ability to carry out the distribution and replication of real-time systems as well as, ensure determinism. Thus, taking as a starting point the new standard IEC 61499 [1], we opted for the use of an application that allows perform the development of distributed systems and, consequently, their replication, which, in parallel, supports the requirements of this same standard, i.e., based on the Open Source PLC Framework for Industrial Automation &

[*]Corresponding Author: Adriano A. Santos, Centre for Research & Development in Mechanical Engineering (CIDEM), Department of Mechanical, School of Engineering (ISEP), Polytechnic of Porto, ads@isep.ipt

Control, *Eclipse 4diac™* [2] (infrastructure for distributed Industrial Process Measurement and Control Systems – IPMCS).

This paper is organized as follows: Section 2 presents a literature review, where ideas from different authors for the distributed and replicated systems problems will be exposed. Section 3 presents an overview of IEC 61499 standard. Section 4 presents the proposed implementation for reliable real-time SIFB communication and their decision timing analysis. A numerical example is presented in Section 5 and the conclusions are outlined in Section 6.

2. Related work

Replicated systems is based in the replication of the critical components of software or hardware, connected by a network. So, to use a networks communication to support DCCS applications requires not only a bounded times transmission services, but also ensuring the dependability for the applications with real-time needs.

Fault-tolerance architectures based on software have been proposed by many authors, all of them exploring the diversity of implementation, diversity of data and temporal diversity [3]. Consequently, two approaches can be taken to tolerated or recovery faults: the forward recovery (N-Version Programming [4] and its variations, N-Self Checking Programming [5], N-Copy Programming such as Distributed Recovery Block (DRB) [6] or Extend DRB [7]), and backward recovery (use of checkpoints from which recovery is attempted). So, when using in a replicated system forward recovery it is necessary that all replicas remain synchronized in order to produce the same set of data and events outputs in the same order (replicas will have to be deterministic) [8] and, somewhere, the replicas outputs need to be consolidated. On the other hand, replica determinism it will be possible achieved through the use of clocks synchronization, atomic multicast protocol and consensus agreement protocols [9] but also by timed messages [10]. Base in these concepts, a similar technic for a DEAR-COTS framework based on Ada 95 has already been proposed by Pinho *et al*. [11]. These authors using a Network Time Protocol (NTP) to synchronize replicas and messages transmission time are set offline.

Likewise, considering replication systems, [12] presents an approach based on a Fast and scalable Byzantine Fault-Tolerance protocol (FBFT) using message aggregation technique combined with reliable hardware-based execution environments. Based on a multicasting replication the aggregation reduces the complexity of messages and computation overhead. In this turn, Pinho *et al.* [13] using a Stat Machine approach to develop fault-tolerant distributed system. Replication is based on a priority algorithm (total order and consensus) like Raft (PRaft). Incoming messages are executed according to the priority level, so the processes do not have to wait to the confirmation of the request. Messages are executed at the moment they are received. Hu *et al.* [14] propose a standard for fault-tolerance modulation based on the programming of N-Versions that can be integrated, transparently, into existing applications, improving its operation, maintaining the characteristics of time. The model, developed in C language, consists of a set of components (initiator, member versions and the voter) where they encapsulated several alternative algorithms to obtain the same outputs.

Some works based on the IEC 61499 fault tolerance system have already been tested and presented in [15], where a distributed replication structure, similar to the work presented in this document, is presented. In this case, a timed message protocol is used to ensure synchronization of the internal states of the replicas and was also implemented in *Eclipse 4diac™* and validated in a FORTE runtime multicast environment. In the same line of reliability [16], it presented a formal modeling methodology to validate and evaluate the reliability of IEC 61499 applications applied to critical safety situations. On the other hand, the works developed by Batchkova *et al*. [17] and Dai *et al*. [18], which are related in some way, focus on the development of reconfigurable IEC 61499 control systems. The methodology proposed by these authors allows IEC 61499 applications to be reconfigured during execution, replacing one Function Block (FB) with another. The substitution, done in real time, does not create a significant impact on the execution of the system and as such can be used as an approach to fault-tolerance. However, they focused on reconfiguration of the system and not on a fault-tolerance scenario, where time to bring up the system is not considered.

In the same area of the IEC 61499 Lednicki *et al.* [19] presents a model to calculate the Wast-Case Execution Time (WCET) for the software FB execution. This model works with a set of events, considering the information associated with the inputs, in which the execution is initiated by the arrival of the input event to the function blocks. The WCET value represents the maximum time that a FB needs to execute its functionality, from the entry of an event until its internal activity is completed (exit of the event). The WCET is independent of the internal path or the execution paths. This model gives us the time needed to activate the next FB.

3. Overview of the IEC 61499

IEC 61499 applications are made up of interconnections of FBs that exchange information, data and events, based on a graphical representation. Each of these graphic representations (FB) consists of a rectangular structure that incorporates an upper part, called the head, and a lower part, called the body. The head is the interface for receiving events, initializing the FB and activating the internal algorithms. It is also used for sending events, confirming the initialization of the FB, as well as the execution of the internal algorithm or algorithms. The body is the interface for received and sending data and it is also de base for the internal algorithms. Data and events inputs are on the left and outputs on the right.

An FB network is assumed to be an event exchange process in which each outgoing event is linked to an incoming event, and each outgoing data is linked to an incoming data. FBs are executed when they receive events so, from that moment on data can be processed, reading inputs data. However, it is only after the execution of the internal algorithms that the data is updated, placed on the output link, and one or more output events can be generated. Input and output data are defined according to the type (Int, Real, etc.) so, in IEC 61499 applications, it will only be possible to establish connections between data of the same type, while the events can be considered as the base event type used in the FB activation.

As stated earlier, the central structure of the IEC 61499 is the FB, which interconnected in a network may represent a device, like a Personal Computer (PC) or a Programmable Logical Controller

(PLC), for example, connect in a control node. In this sense, we will be able to fit the FBs in the object-oriented paradigm where each of the FBs can be considered as a single object (Basic Function Block – BFB, with an Execution Control Chart – ECC, consisting of one or more algorithms), as a Composit Function Block (CFB), constituted by an interconnected BFBs or CFBs and Special Interface Function Block (SIFB) used in communication. The program algorithm for the BFB, CFB and SIFB may be developed in any languages defined by the IEC 31131-3 [20] and also in all additional languages supported by the IEC 61499 (e.g., Java, C, C++, C#, etc.). Developers are free to choose language since the standard does not specify a recommended language.

A distributed IEC 61499 or a replicated fault-tolerance system, split among several computer devices (e.g., PC, microcomputer or PLC), needs to send, over the network, events and data to each one of the FB distributed or replicated into the devices. To do this, the developer, insert a SIFB communication which will allow communication between FBs allocated to the same device or distributed by remote devices. The purpose of this article is to define a methodology to guarantee replicas determinism, using standard communication interfaces (SIFB), which will communicate between replicas, on remote devices, ensuring that the replicas process the same data set in the same order. On the other hand, industrial redundancy is typically archived at the hardware level, where the access to physical I/Os is done by communication SIFB Publish/Subscribe pair over UDP/IP – User Datagram Protocol/Internet Protocol (unidirectional communication), or Client/Server pair over TCP/IP (Transmission Control Protocol/Internet Protocol), bidirectional data/event communication [21].

4. Proposed IEC 61499 Implementation

An IEC 61499 application is seen as a combination by several devices, sub-applications or interconnected unitary processing elements (FBs), which at each invocation of events, process the data, generating new events and/or data. So, to tolerate individual faults, ensure the reliability of the application, only the critical components of the application must be replicated. Components is defined as an atomic and indivisible component (FB) which can include tasks and resources replicated on multiple nodes or allocated in just on node. As an example, Figure 1 shows a real-time sub-application "C" with 2 FBs (FB1 e FB2) distributed over nodes 1 and 2 and replicated over nodes 2 and 3 or, alternatively, replicated entirely in a single node (node 4).

So, the communication infrastructure of the proposed replication framework (base on active replication, i.e., all replicas are active and running at all times), based on the same communication structure used by IEC 61499, must guarantee that all messages sent by computer devices, delivery to all receiver, is correctly received. However, it will also be necessary that replicas agree with the order of the data set sent and consolidate data from replicated inputs into a single value that will be propagated to the subsequent FB.

Component replication can be performed using multiple replicas, however, the most common is the use of two ($f + 1$) or three replicas ($2 * f + 1$), to tolerate f failures. Therefore, replication based on these assumptions places us in the presence of several communication/interaction scenarios in which the exchange of messages can be defined according to the following four approaches [22]: *1-to-1* (communication from a nonreplicated FB to another nonreplicated or communication inside of the replica, base of the IEC 61499 communication); *1-to-many* (communication from a nonreplicated FB to a group of replicated FB. An atomic multicast protocol [9] must be used to ensure that all the replicas received the same set of information (data/events) in the same order. Replicas need to maintain internal state synchronized); *many-to-1* (a replicated FB sends data/events to a nonreplicated FB. Nonreplicated FB receives a set of inputs from all replicas and vote, consolidate mechanism [3], on the output value to process continue) and *many-to-many* (a mix of the last two cases where each received replica need to agree on the value to forward process. An atomic multicast protocol is used to disseminate values and the agree decision can be performed by one of the received values or on some value calculate based on majority, average, median, etc.). Voting mechanism can itself be replicated or only a single copy can be executed.

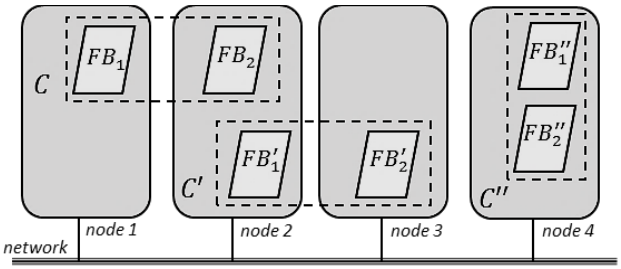

Figure 1: Replicated real-time sub-application

Replication model was implemented using *Eclipse 4diac™*. It is developed using the 4DIAC-IDE (Integrated Development Environment), graphical platform, and the FORTE runtime execution environment [2]. The graphical platform is used to develop the application, perform the interconnections between the instances, create, compile (in C ++) and integrate (in FORTE) the new types of FBs, once FORTE is compiled and executed in each computer device. Communication between replicas was carried out using standard communication FBs, made available by the 4diac repository and in accordance with IEC 61499. Data and events connection, between computer device and all instantiated FBs is supported by the FORTE runtime environment.

4.1. Communication architecture

The communication architecture is based on standards SIFB (Publish/Subscribe or Client/Server) interactions using Internet Protocol (IP) and FORTE runtime [23]. So, each of communication layers needs to be configured by the addressing schema accomplished by the identifier parameter (ID). Communication protocol is implemented in FBDK, inside a multicast group, using an Internet address and a unique port number [IP:port, e.g., 239.0.0.100:61023]. In a multicast communication scenarios (SIFB Publish/Subscribe) any of the subscribe blocks, located in the same network segment as the publish, can receive all published data/events [24]. On the other hand, a real-time industrial control application requires clocks synchronization, use of the timed messages to ensure determinism and the analyses of the worst-case execution times of the replicated FBs (including clock lag and communication delays). In fact, the developer needs to create a structure that supports IEC 61499

replication based on the existent communication SIFB considering scenarios presented above, in other words, it will only be necessary to use only two of the presented layers: *1-to-many* and *many-to-1* communication scenarios. Figure 2 shows the interface of the pair Publish/Subscribe used in multicast protocol communications.

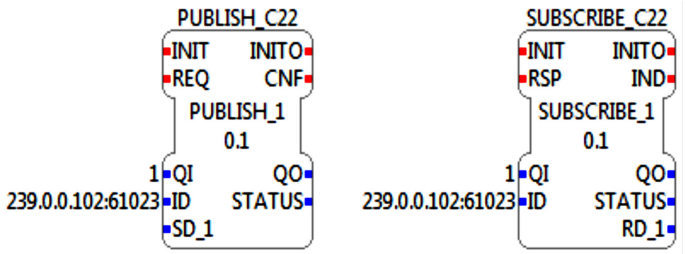

Figure 2: Interface of the Publish/Subscribe communication pair

4.2. Consolidation and voting replicate inputs

Voters can be developed according to the most varied techniques of fault-tolerance. However, these have the ultimate purpose of comparing the results of two or more variables and deciding which is the correct result, if any. There are in fact many types of voters [3] and the decision on which voting algorithm to use will depend on the semantics required by the application. For the voting to be viable, all replicas must send the data in the expected time. So, voting mechanism or consolidate module is built in top of the atomic multicast protocol, to ensure that all replicated FB receives the same set of data in the same order. The subscribe will wait until the set of events and data from the publishes are received. It is only at this point that the data is consolidated (the data is chose) or when it know that you will no longer receive messages at the δ_{voting} specific delay, Figure 3.

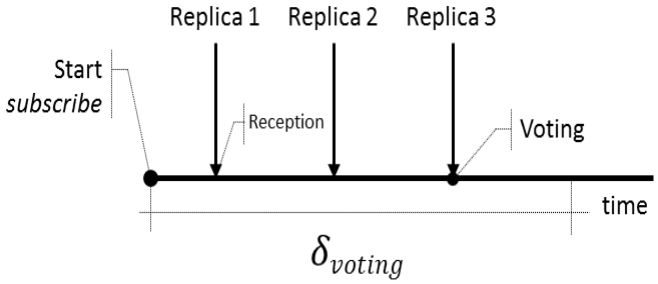

Figure 3: Data consolidate without faults

Note that this procedure must be implemented in a *many-to-1* context and that the decision time will depend on the worst-case response time (WCRT) of the last data received. On the other hand, we should consider that it will not be necessary to use underlying protocols that solve the problems of inconsistency by omitting messages, as it will be enough that only one node delivers a message.

In this approach, as we use a Triple Modular Redundancy (TMR) to mask faults [25] a majority voter was used. In the case of the three replicates, is done by simply comparison of the values received in A and C. The voting mechanism determines which value should be chosen according to the pseudocode shown in Algorithm 1.

Algorithm 1 VOTER algorithm pseudo code

```
1:  /* Initialization */
2:  void FORTE_VOTER
3:  switch(pa_nEIID){
4:  case input event:
5:          if (A() == C())
6:              VOTED() = A();
7:          else
8:              VOTED() = B();
9:          output event;
10:     break;
```

On the other hand, it is also necessary to consider that the determinism of the replicas must be guaranteed by the active replication. Therefore, the concept of timed messages [10] must be implemented to define the correct order of the execution of the received data. Thus, according to the mapping of the replication scenarios presented in [22], the application clocks must be synchronized. Data to be disseminated are associated with the availability times, defined by the execution times of the FBs to which the events/data are linked, that is, immediately after their execution. This validation time will be, in reality, the worst execution time of the FB that makes the data available (since in the IEC 61499 framework, the output data are only available when the algorithm finishes its execution) plus the sending times, which can be determined *offline* [26]. In this sense, when the data manager, a software element that guarantees the achievement of determinism, reads the received values (sent only once), works with the most recent values that have the oldest validation moments associated with the task validation. Figure 4 shows the scheme of how determinism can be obtained in the replicated components depending on the treatment of the timed messages received and treated according to the concept developed in the ordering FB.

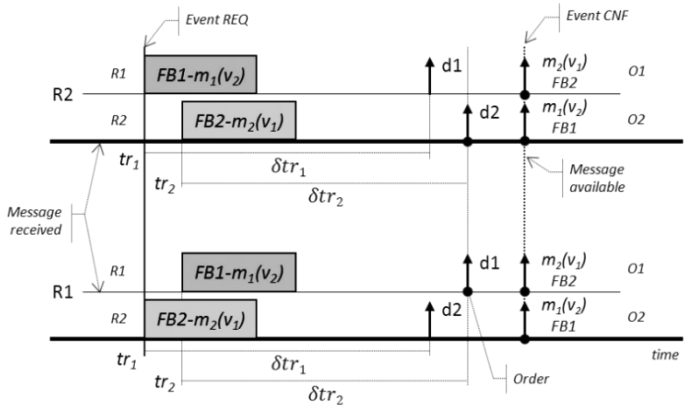

Figure 4: Replicas consistence, adapted to IEC 61499

Each FB is associated with a task, so they will send a message of order m_k where, in this implementation, the k index is associated with the number of the respective FB or task. δt_r is the limit of the predefined time for its execution (time that is activated after receiving the first message). $m_k(v_i)$ is a message of order k made available at the time of validity v_i. The instant tr_i represents the times of recession in processes *P1* and *P2* so, δtr_i is the waiting time, associated with event i received. When a new event is received the δtr_i waiting time is restarted, associated with the new event i, at the end of which the events will be ordered according to

the instant of validation v_i. Message that has the most recent value and that has the oldest validation time associated with its validation will be allocated to O1 (OUT1) while the second message will be allocated to output O2 (OUT2). CNF event confirms the execution of the FB and the availability, at the same time, of data *d1* and *d2*.

4.3. Consolidate time analysis

The analysis of the consolidate protocol execution time, at the receiving replicas, aims to define the delay time in the decision phase ($\delta_{decision}$), necessary for the FB to consolidate the received data, i.e., guarantee that FB will not receive any more messages. This time is dependent on the worst-case response time of the replicated messages as well as their best-case response time (BCRT), message processing, having as reference the initial time common to all sending and receivers nodes. However, we must consider that the time to send messages is common to all nodes (synchronization of local clocks) there were small variations or errors in the clock readings (*jitter*) that, like the *offset*, made the clocks only approximately synchronized.

Knowing the worst-case response time, we can determine the $\delta_{decision}$ assuming that the first message received has the best-case transmission time and the last has the worst-case response time. Therefore:

$$\delta_{decision} = \max_{\forall m \in res(m)} \{W_m\} - \min_{\forall m \in res(m)} \{W_m\} + \varepsilon \qquad (1)$$

where $max\{W_m\}$ is the worst-case response time of message and $min\{W_m\}$ is the best-case response time of message, considering a common time reference. $res(m)$ is the set of replicated messages received and ε is the maximum deviation between nodes synchronized local clocks. Figure 5 shows the time relationships referred.

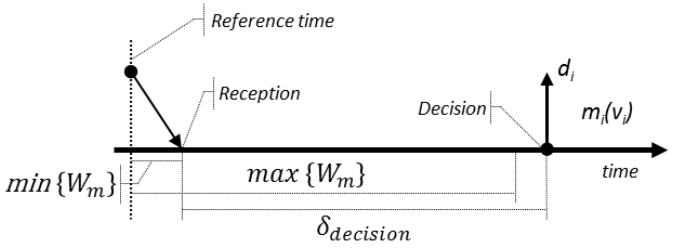

Figure 5: Relationship of times consolidate protocol

5. Numerical example

In order to explain the use of the presented model, a simple example of application is used. In Figure 6 is presented a real-time distributed system replication scheme using in the considered example. System application is constituted by five nodes, connected by TPC/IP network based on multicast protocol.

The application used in this example is constituted by five components (C_1 to C_4), each one with a task (τ_1 to τ_4) which are distributed over the nodes. A simple replication of critical components was performed by interconnecting the distributed and replicated components, with a switch, over an Ethernet TCP/IP network at a rate of 10/100 Mbps. Component C_1 (FB1) encapsulate tasks τ_1 at node 1, component C_2 (FB2) encapsulate task τ_2 at node 2, components C_{3a} and C_{3b} (replicated components, FB3 e FB3', at nodes 3 and 4) encapsulate τ_3 and τ_3', and finally component C_4 (FB4) encapsulated τ_4 (used to synchronize the start of components C_1 and C_2) at node 5.

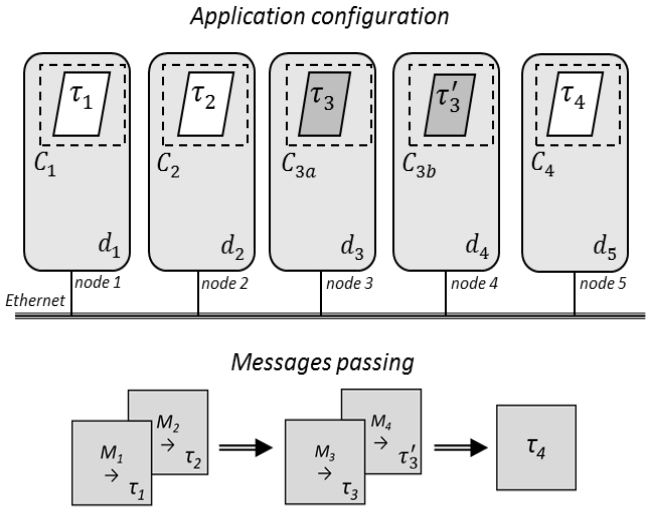

Figure 6: Application structure (replication and messages passing)

Table 1: Tasks characteristics

Tash	Type	Component	Node
τ_1	*Periodic*	*C1*	*1*
τ_2	*Periodic*	*C2*	*2*
τ_3	*Periodic*	*C3a*	*3*
τ_3'	*Periodic*	*C3b*	*4*
τ_4	*Spor/Per*	*C4*	*5*
τ_5	*Periodic*	*C5*	*6*

Table 1 presents each of the task's characteristics of the distributed application, while Table 2 presents the messages exchanged between FB (all values are in milliseconds).

Table 2: Messages passing characteristics

Msg	Bytes	Period (*ms*)	From	To	Protocol
M_5	---	---	τ_4	τ_1, τ_2	*Multicast*
	56(84)	*ping -f*	τ_4	τ_5	*Unicast*
M_1	*24*	*1000/500*	τ_1	τ_3, τ_3'	*Multicast*
M_2	*24*	*1000/500*	τ_2	τ_3, τ_3'	*Multicast*
M_6	*32*	*ping -t*	τ_5	τ_4	*Unicast*

Note that message from component C_1 and C_2 (M_1, M_2) is a *1-to-many* communication (multicast protocol) and also the message of system initialization (event synchronization starts of the C_1 and C_2) M_5. Messages M_5 is also used to increase the network traffic, in the switch, to component C_5 (Windows PC). Message M6 from C_5 to C_4 is the task response and also a contribute to the network traffic, it is a *1–to–1* communication. Messages M_1 and M_2 are messages from nonreplicated components to a replicate's components (C_{3a} and C_{3b}), therefore, they will have to be consolidate in each of the receiving replicas. This consolidation mask node failures of the sender's components.

5.1. Determinism testing

In order to clarify the possible determinism of the application, using only standard elements of the *4diac framework*, Publish/Subscribe communication pairs, 10000 events and data were launched in the network with frequency ranges of 1Hz and 0.5Hz, that is, with 1000 ms and 500 ms intervals. Each of the components (C_1 and C_2) sends messages to the replicas, in the same frequency ranges, associated with the availability time (*tsa - timestamp to availability* in *sec* and *nsec*) in the format [*data, sec, nsec*]. Replicas receives data associating the reception time (*tsr – timestamp to receiving*) in the same format.

The purpose of this example is to test the possibility of guaranteeing the determinism of the replicas according to the proposed and applied communication protocols (Publish/Subscribe pairs, inherent to *4diac*), for both the delivery and response time of messages. Thus, we consider the response time as the time interval between the instant when the message is sent and the instant when it is received by replies. A multicast protocol is used, to propagate messages, assuming that all the messages are delivered and received correctly. Table 3 shows the worst-case response time (WCRT) for a set of messages sent to the network using a 1000 ms trigger frequency. These experiments are carried out considering three conditions of operation: no additional traffic, with daily traffic directed simply to the switch (*tp-link, TL-SF1008D*) and additional traffic directed from a nonreplicated component to a replicated. Table 4 shows the results obtained for the WCRP considering a 500 ms trigger frequency.

Table 3: Messages response time, 1000 ms trigger frequency

Msg	Period	WCRT (*ms*)		Reception order
		C3a	C3b	
M_5	-	-	-	
M_1	1000	6,720	3,979	93,35%
M_2	1000	6,765	3,806	
M_1	1000	6,527	2,602	94,33%
M_2	1000	6,464	2,942	
M_1	1000	6,208	1,556	97,09%
M_2	1000	6,448	1,941	

As can be seen, the WCRP is obtained for M_2 messages. This carryout the messages sending from component C_2 (nonreplicated) to the replicated component C_{3a}, response time of 6.765 ms. The average order value of the received data (*Reception order*) is 94.92%, which demonstrates the absence of the determinism of the replicas (C_{3a} and C_{3b} components) due to the frequency of the experience triggering.

Table 4: Messages response time, 500 ms trigger frequency

Msg	Period	WCRT (*ms*)		Reception order
		C3a	C3b	
M_5	-	-	-	
M_1	500	6,760	6,236	97,94%
M_2	500	7,207	8,045	
M_1	500	5,843	2,457	93,43%
M_2	500	6,248	2,569	
M_1	500	7,196	3,178	94,34%
M_2	500	6,575	3,235	

As can be seen, the WCRP is obtained for M_2 messages. This carryout the messages sending from component C_2 (nonreplicated) to the replicated component C_{3b}, response time of 8.045 ms. The average order value of the received data (*Reception order*) is 95.24%, which demonstrates the absence of the determinism of the replicas (C_{3a} and C_{3b} components) due to the frequency of the experience triggering.

Table 5 shows the WCRT (maxims time) and BCRT (minimums times), *calculated offline*, for a set of messages exchanged between components C_1, C_2 and C_{3a}, C_{3b} characterizing the messages of stream M_1 and M_2. These values are the result of the data and events received in the replicas, considering the experiences defined above. The incidence based is 10 k events processed in each of the experiments. These values are the result of the *offline* analysis of the 60 k records processed.

Table 5: Messages times characteristics, maxims and minimums value

Msg	WCRT (*ms*)	BCRT (*ms*)	from→to	Average order
M_1	7,169	0,170	$\tau_1 \rightarrow \tau_3$	95,08%
M_2	8,045	0,131	$\tau_2 \rightarrow \tau_3'$	

As can be seen, the average order for data received in the replicas has a value less than 100%, which induces the existence of failures in the ordering of the data. On the other hand, since the messages must be consolidated and ordered according to the time validity requirement, to guarantee determinism, it is necessary to determine the $\delta_{decision}$ parameter of the consolidation protocol as defined in (1). The maximums and minimums times for transmission data/events are defined in Table 5 and the maximum deviation between synchronized clocks (ε), also calculated *offline*, is 164 μs. Therefore, using (1):

$$\delta_{decision} = 8.045 - 0.131 + 0.164 = 8.078\ ms \quad (2)$$

The worst-case decision time for consolidation, considering $\delta_{decision}$, can be defined considering all messages received in the set *res(m)* so, that the worst-case decision time will be given by:

$$W^{decision} = \max_{\forall m \in res(m)} \{W_m\} + \delta_{decision} = 16.123\ ms \quad (3)$$

where $W^{decision}$ is the worst time of the consolidation, i.e., time decision when a new event arrives at the end of the $\delta_{decision}$ time.

6. Conclusions

The *Eclipse 4diac*™ tool was developed in accordance with IEC 61499 and it is specially directed to the development of distributed industrial applications portables and modular.

The Publish/Subscribe pair, provides by the 4diac object repository, are a communication FB, fundamental for the interconnection of the distributed components, guaranteeing not only the synchronization of the system but also the implementation

of the atomic multicast protocol. However, the introduction of software or hardware component replication introduces new problems that were not anticipated in IEC 61499 as well as in 4diac. These pairs, using different communication interfaces, guarantee replication synchronization, but cannot guarantee their determinism and, as such, promote the occurrence of failures that must be masked.

Based on the experiences carried out we can conclude that the Publish/Subscribe pair, most used in the interconnection of distribute/replicated computer devices, has a failure rate, independent and identically distributed, of 4.92%. Therefore, it is not possible, by itself, to ensure determinism, so the programmer will need to develop a FB, also replicable, which ensures that all replicas process the same data set in the same order.

Acknowledgment

We acknowledge the financial support of CIDEM, R&D unit funded by the FCT – Portuguese Foundation for the Development of Science and Technology, Ministry of Science, Technology and Higher Education, under the Project UID/EMS/0615/2019 and ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme, and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project SAICTPAC/0034/2015- POCI-01-0145-FEDER-016418.

References

[1] International Standard IEC 61499-1, Function Block-Part 1: Architecture, 2nd Edition. International Electrotechnical Commission, 2012.

[2] Eclipse 4diac, Open Source PLC Framework for Industrial Automation & Control [Online], https://www.eclipse.org/4diac/.

[3] L. Laura, Software Fault Tolerance – Techniques and Implementation, Artech House Publishers, ISBN: 9781580531375, 2001.

[4] L. Chen, A. Avizienis, "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation" in Proceedings of FTCS-8, Toulouse, France, 1978. DOI**:** 10.1109/FTCSH.1995.532621.

[5] J. C. Laprie, J. Arlat, C. Beounes and K. Kanoun, "Definition and analysis of hardware- and software-fault-tolerant architectures," in Computer, **23**(7), 39-51, July 1990. DOI: 10.1109/2.56851.

[6] K. H. Kim. The Distributed Recovery Block Scheme, in Software Fault Tolerance, M. R. Lyu (Ed.), New York: John Wiley & Sons, 1995.

[7] D. Nguyen and Dar-Biau Liu, "Recovery blocks in real-time distributed systems," in Annual Reliability and Maintainability Symposium. 1998 Proceedings. International Symposium on Product Quality and Integrity, Anaheim, CA, USA, 1998, 149-154. DOI: 10.1109/RAMS.1998.653703.

[8] R. Guerraoui and A. Schiper, "Software-based replication for fault tolerance" in Computer, **30**(4), 68-74, April 1997. DOI: 10.1109/2.585156.

[9] V. Hadzilacos, S. Toueg, "Fault-Tolerant Broadcasts and Related Problems" in Distributed Systems (2nd Ed.), ACM Press/Addison-Wesley Publishing Co, USA, 97-145, 1993.

[10] S. Poledna, A. Burns, A. Wellings and P. Barrett, "Replica determinism and flexible scheduling in hard real-time dependable systems" in IEEE Transactions on Computers, **49**(2), 100-111, Feb. 2000. DOI: 10.1109/12.833107.

[11] L.M. Pinho, "Replication Management in Reliable Real-Time Systems", in Real-Time Systems, **26**, 261–296 (2004). https://doi.org/10.1023/B:TIME.0000018248.18519.46.

[12] J. Liu, W. Li, G. O. Karame and N. Asokan, "Scalable Byzantine Consensus via Hardware-Assisted Secret Sharing" in IEEE Trans. on Computers, **68**(1), 139-151, 1 Jan. 2019. DOI: 10.1109/TC.2018.2860009.

[13] R. Paulo, et al., "Replicação de Máquina de Estado Baseada em Prioridade com PRaft" in XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, Salvador, Brazil, 2016.

[14] H. Tingting, Bertolotti, Ivan C. and Navet, Nicolas, "Towards seamless integration of N-Version Programming in model-based design", in 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA2017), Cyprus, 1-8, 2017. DOI: 10.1109/ETFA.2017.8247678.

[15] M. de Sousa, C. Chrysoulas and A. E. Homay, "Multiply and conquer: A replication framework for building fault tolerant industrial applications" in 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), Cambridge, 1342-1347, 2015. DOI: 10.1109/INDIN.2015.7281930.

[16] L. Yoong, "Modelling and Synthesis of Safety-Critical Software with IEC 61499", PhD Thesis submitted for Electrical and Electronic Engineering, University of Auckland, 2010. http://hdl.handle.net/2292/6691.

[17] I. Batchkova, G. Popov, H. Karamishev and Grigor Stambolov, "Dynamic reconfigurability of control systems using IEC 61499 standard", in 15th Workshop on International Stability, Technology, and Culture the International Federation of Automatic Control, **46**(8), 256-261, 2013. https://doi.org/10.3182/20130606-3-XK-4037.00050.

[18] W. Dai, V. Vyatkin, J. H. Christensen and V. N. Dubinin, "Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Interoperability," in IEEE Transactions on Industrial Informatics, **11**(3), 771-781, June 2015. Doi: 10.1109/TII.2015.2423495.

[19] Lednicki, Luka, Carlson, Jan e Sandstrom, Kristian, "Model level worst-case execution time analysis for IEC 61499", in Proc. of the 16th International ACM Sigsoft symposium on Component-based software engineering, June, 169–178, 2013. https://doi.org/10.1145/2465449.2465455.

[20] International Electrotechnical Commission, "International Standard IEC 61131-3, Programmable Controllers - Part 3: Programming Languages", 3rd Edition, 2013.

[21] H.-M Hanisch and V. Vyatkin, "Achieving Reconfigurability of Automation Systems by Using the New International Standard IEC 61499: A Developer's View", in The Industrial Information Technology Handbook, CRC PRESS, Section 4, 66, 2004. https://doi.org/10.1201/9781315220758.

[22] A. A. Santos and M. de Sousa, "Replication Strategies for Distributed IEC 61499 Applications", in *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, Washington, DC, 2018, 2225-2230. Doi: 10.1109/IECON.2018.8592737.

[23] M. Hofmann, M. Rooker and A. Zoitl, "Improved Communication Model for an IEC 61499 Runtime Environment", Proc. of IEEE 16th Conference on Emerging Technologies & Factory Automation (ETFA2011), Toulouse, France, 2011, 1-7. Doi: 10.1109/ETFA.2011.6059121.

[24] A. A. Santos, A. F. da Silva, M. de Sousa and P. Magalhães, "An IEC 61499 Replication for Distributed Control Applications", in IEEE 16th International Conference on Industrial Informatics (INDIN), Porto, 2018, 362-367. Doi: 10.1109/INDIN.2018.8471958.

[25] M. Yang, G. Hua, Y. Feng, and J. Gong, "Fault-Tolerance Architectures and Key Techniques", in Fault-Tolerance Techniques for Spacecraft Control Computers (1st. ed.). Wiley Publishing, pp 29-76, 2017. DOI: 10.1002/9781119107392.

[26] K. Mohamed, R. Xavier, S. Françoise, "A Schedulability Analysis of an IEC-61499 Control Application", in 6th IFAC International conference on Fieldbus Systems and their Applications (FeT'2005), **38**(2), 71-78, 2005. https://doi.org/10.3182/20051114-2-MX-3901.00011.