Reliability of Replicated Distributed Control Systems Applications Based on IEC 61499

Adriano A. Santos[1,2](✉), António Ferreira da Silva[1,2], António Magalhães[3], and Mário de Sousa[3]

[1] CIDEM, School of Engineering of Porto, Polytechnic of Porto, 4249-015 Porto, Portugal
{ads,afs}@isep.ipp.pt
[2] INEGI - Instituto de Ciência e Inovação em Engenharia Mecânica e Engenharia Industrial, Rua Dr. Roberto Frias, 400, 4200-465 Porto, Portugal
[3] Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
{apmag,msousa}@fe.up.pt

Abstract. The use of industrial and domestic equipment is increasingly dependent on computerized control systems. This evolution awakens in the users the feeling of reliability of the equipment, which is not always achieved. However, system designers implement fault-tolerance methodologies and attributes to eliminate faults or any error in the system.

Industrially, the increase in system reliability is achieved by the redundancy of control systems based on the replication of conventional and centralized programmable logic controllers. In distributed systems, reliability is achieved by replicating and distributing the most critical elements, leaving a single copy of the remaining components. On the other hand, given the nature of the distributed systems, it will also be necessary to ensure that the data set received by each of the replicas has the same order. Thus, any change in the order and data set received will result in different results, in each of the replicas, which may manifest in erroneous behavior.

In this paper, the interactions and the erroneous behavior of the replicas are explained, depending on the data set received, in a fault tolerant distributed system. Its tendency, behavior and possible influences on reliability are presented, considering the failure rate and availability based on the mean time to failure.

Keywords: Dependability · Distributed systems · Event-base control · Fault-tolerance · IEC 61499 · Industrial control · Real-time · Reliability · Replication

1 Introduction

Technological developments that have occurred in recent years have led to the proliferation of computerized systems both at the industrial level and for consumption and domestic use. At the industrial level, the technology used is based on a control system centralized in Programmable Logical Controllers (PLCs), sometimes redundant to ensure the reliability of the systems, running in a single machine. On the other hand,

https://doi.org/10.1007/978-3-030-79168-1_28

the use of computerized systems at the domestic and consumer level is more discreet. Embedded systems are present in our daily lives in the most diverse devices and with more visibility in traditional computers for daily use.

This proliferation makes us more and more dependent on computerized systems, becoming us more vulnerable to the occurrence of failures both in terms of the control of domestic systems and in terms of industrial systems. Failures in the control system of a reactor at a nuclear power plant will have more serious consequences than in domestic systems. The reactor, when subject to a failure, can put thousands of people in danger and seriously affect the ecosystem. On the other hand, a failure an embedded system in a washing machine or toaster, keeping the equipment energized, can also, at a lower level, put people and property at risk. Obviously, if a fire occurs due to system failure, the housing in question and the contiguous ones can be in great danger. From the point of view of the reliability of a computer system, the underlying idea is that the system behaves according to its specification, in view of the numerous problems that can occur (natural accidents, software and hardware errors, among others). In fact, what we expect from computer systems is that they will work correctly to guarantee integrity and availability.

Industrially, the process control is based on centralized PLCs, generally programmed according to the languages standardized in IEC 61131 [1]. On the other hand, with the proliferation of communication networks in the industrial domain, PLCs were interconnected with each other resulting in distributed control applications. However, according to the semantics of IEC 61131, programming languages and their execution are not presented as a good practice for the requirements used in flexible automation and distributed control applications. It was mainly for this reason that the International Electrotechnical Commission (IEC) developed the standard IEC 61499 [2] to facilitate the development of Distributed Industrial Process Measurement and Control Systems (DIPMCS). This standard proposes the use of Function Blocks (FBs) as a basis for the development of reusable software modules for the control system. Each function block is a functional software unit that encapsulates local data and algorithmic behavior within an event/data interface where operations, within a function block, are controlled by an event-driven state machine.

Given the nature of distributed control applications, many new problems must be considered. Therefore, when developing a distributed application, the designer must be aware of the possibility of partial failures, that is, the possibility of some device stopping its execution and the others continue with their normal processing. So, when developing IEC 61499 applications with high dependability requirements, the implementation of solutions to tolerate these partial failures should be considered. The increase in the reliability can be obtained by masking the failed devices, introducing fault-tolerance in the application architecture.

The purpose of this text focuses on the presentation and definition of some concepts related to the dependability attributes of computer systems. In this way, it is also intended to expose some of the classic mechanisms that allow to solve the dependability problems, namely the faut-tolerance techniques. The analysis presented in the next sections are an adaptation of traditional approaches to partial fault-tolerance. Some of the problems associated with the reliability of replicated systems will be explained considering the various possibilities of communication between the devices.

2 Dependability

Replication is widely used in distributed systems as a Fault-Tolerance (FT) mechanism to maintain the desired availability and reliability. Among other reasons for its use, the fact that replication fits naturally in the topology of distributed systems stands out. So, it is for these reasons that fault-tolerance techniques are generally used to satisfy dependability requirements. Dependability of systems engineering – computational, mechanical, physical, and human systems – must be expressed by a relatively small number of failures. On the other hand, it is necessary to consider that the acceptable levels of failure of the systems vary according to their measured specifications, in percentage, according to the *Mean Time Between Failures* (MTBF) and the *Availability* (A).

Dependability systems has been defined by [3] as "*…the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers …*". Another similar definition is given by Avizienis *et al.* [4] "*… the ability to deliver service that can justifiably be trusted …*". However, the main idea of dependable systems is that they must be able to ensure that the service provided meets the specifications and, as such, does not fail. This means that the use of dependable systems is always desirable, as they are "trustworthy" [5]. The perception of dependability can be very generic and understood in different ways by different people, however this must integrate the following attributes: Availability, Reliability, Safety, Integrity, Maintainability, and Confidentiality.

Availability is used to measure the probability that a system or component will become operational and perform its functionality after a failure. Mathematically, the "*availability is a measure of the fraction of time that the item is in operating condition in relation to total or calendar time.*" [6]. More formally, availability can be defined as the "*… proportion of time a system is in a functioning condition*" [7]. The availability of the system at time t is defined by $A(t)$ which represents the fraction of time that the system is available. It is also called inherent availability or steady-state availability and can be expressed at time t by:

$$A(t) = \frac{u}{u + d} \tag{1}$$

where u is the mean uptime and d is the mean downtime of the system.

The next attribute, reliability, is define as "*… the ability of an item (a product or a system) to operate under designated operating conditions for a designated period of time or number of cycles.*" [6]. More formally, reliability can be defined as "*… the ability of a system to function under stated time and conditions*" [7]. So, reliability is the probability that the system is operating normally in the range $[t_0, t]$, therefore, reliability at time t is denoted as $R(t)$. On the other hand, the abnormal operating condition of the system at $[t_0, t]$ is defined as unreliability degree at time t and it denoted as $F(t)$. The relationship between reliability and unreliability can be expressed at time t by:

$$R(t) = 1 - F(t) \tag{2}$$

Safety is defined as "*the nature of a system not to endanger personnel and equipment.*" [7], however Avizienis *et al.* [4] define safety considering the environment as the "…

absence of catastrophic consequences on the user(s) and the environment". Note that availability and reliability are related, however they are not synonymous with security. The availability or unavailability and the reliability or unreliability of a system does not translate into security. Therefore, high reliability results in high security, but high security does not necessarily result in high reliability [7]. Safety is denoted as $S(t)$.

Maintainability is defined as "*the ability of a system to recover its required function*" [7]. It is the ability of the system to be successfully repaired under certain conditions. The degree maintainability is denoted as $M(t)$. Integrity is the non-occurrence of undue changes to the information in the systems, they cannot be modified without authorization, while confidentiality is defined as the absence of unauthorized disclosure of information, that is, the information is not disclosed without authorization.

From the above, we can say that these attributes are interconnected, so, a product or system will be considered dependable if it has all the attributes. On the other hand, we must also consider that the degree of dependability for a system is not a binary phenomenon but is based on the degradation of these attributes and the limits that are considered acceptable [5]. It is a combination of availability, confidentiality, and integrity.

2.1 Relationship Between Fault, Error, and Failure

According to the above, a service will only be performed correctly when it meets the conditions mentioned in the specifications. The existence of an error (see Fig. 1), is due to the occurrence of a component failure caused by physical phenomena of mechanical or electrical origin, internal or external. This can spread, manifesting itself in the degradation of the service, such as, for example, limiting services, decreasing speed of service provision, etc., causing a failure. In software, faults resulting from wrong programming will translate into the existence of a latent fault that, when activated, called the wrong instructions or the use of the wrong data. This event will give rise to an error that in turn will propagate to produce a failure [8].

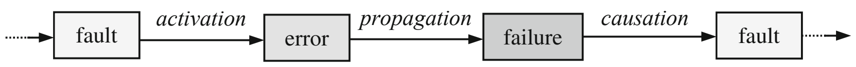

Fig. 1. Error propagation chain [4].

A *fault* is a static feature of a system and is the cause of an error. The *error* is an incorrect internal state of the system that can lead to failure. A *failure* is considered as the occurrence of an unexpected behavior of the system, that is, when the system does not fulfill its mission. This is caused by an error. So, if we look at a car tire, we can say that the failure in this system can be caused by an external event (a nail, a wire, or a glass) that causes a puncture or an internal cause like the tube's trail (*causation*). This fault will give rise to an error (*activation*), deflating the tire, compromising its use for the specified purposes (*propagation*) (see Fig. 1). The tire deflating causes the failure, preventing the vehicle moving forward. Therefore, we can say that the faults, however small, they can cause major errors, however the presence of an error will not necessarily be considered the cause of a failure [9].

Note, however, that the classification of the several internal states of a system for *fault*, *error* or *failure* situations will depend on the system and its specific characteristics. Therefore, based on the characteristics of the systems, for example, centralized, distributed and/or replicated, will be in the presence of organic software and/or hardware failures. In this perspective, we must consider them different since the software does not wear out. Thus, it can be said that there will be a software error or failure if consider the definition presented in [10] "… *work according to the original contract or according to the requirements documentation*…" whenever its MTBF, for example, exceeds the default value. In this assessment, will be in the presence of a fault, but the system will remain operational.

2.2 Means to Attain Dependability

Over the years, various means have been used to achieve the attributes of dependability. These attributes can be grouped into four groups [4]: *Fault Prevention* (use of design methodologies, techniques, and technologies to avoid faults), *Fault-Tolerance* (use of means to provide service even in the presence of faults), *Fault Removal* (use of review, analysis, and testing techniques to reduce the number and severity of faults) and *Fault Forecasting* (estimate of the number, incidence, and future consequences of faults).

The development of flawless computer systems is rarely achieved, so some of the attributes of dependability, such as fault-tolerance, are often used. Fault-tolerance is used to increase the probability that the final design of the application will show acceptable behaviors and thus produce correct outputs. On the other hand, correctness of the system is closely linked to the structure of the application, so, the greater or lower level of fault-tolerance will directly depend on the characteristics of itself and its design. Dependability based on fault-tolerance is generally achieved by redundancy. Redundancy is the duplication (triplication, quadruplication, etc.) of a specific component, considered critical, for which the failure of the component or subcomponent will not result in the failure of the entire system.

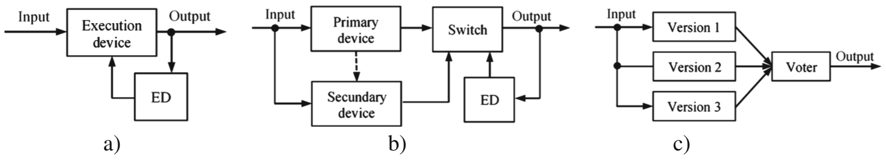

Fig. 2. Fault-tolerance architecture: a) Normal implementation; b) Standby implementation (Replica in standby, rum in parallel); c) Parallel implementation, voting validation.

From the perspective of the operational requirements of fault-tolerance systems, they can have classified into two classes of operability (Continuous and Non-continuous operations) and three main approaches. In the first case, the systems remain operational even in the presence of faults (provides a continuous service) while in the second case the systems interrupt their operation. On the other hand, the fault-tolerance implementation architectures will be based on the use of error detection (ED), with modules operating in standby or in parallel, and the use of voters that validate and consolidate the output values of the redundant elements (see Fig. 2).

The criteria for choosing one of these implementations will depend on the system itself. It will be obvious, and therefore logical, that the application designer only replicates devices with a high influence on the application, while maintaining a single copy of the remaining devices. However, regardless fault-tolerance approach used, the availability, reliability, and safety properties (see item 2) for the system, must be maintained.

The measurement of these properties involves knowledge of MTBF based on the Mean Times To Failure (MTTF) and Mean Time To Repair (MTTR). So, MTBF can be expressed as:

$$MTBF = MTTF + MTTR \tag{3}$$

then availability, according to (1), will translate into the following expression:

$$A = \frac{MTTF}{MTTF + MTTR}(\%) \tag{4}$$

On the other hand, from a reliable point of view, it can be said that the system's reliability represents the probability that there will be no fault during a given period, in which more than one failure is possible [11]. In this sense, it can be considered that the ratio of the number of faults (fault rate) will also be an indicator of reliability showing the portion of components or equipment that must survive in an instant *t*. The fault rate (λ) will be given by the following expressions:

$$\lambda = \frac{Number\ of\ faults}{Time\ of\ use} \quad \text{or} \quad \lambda = \frac{1}{MTBF} \tag{5}$$

Safety must be a propriety for the entire system, including hardware and software, and for its entire life cycle. Thus, when addressing security issues, a set of attributes must be considered that must be verified at the same time: 1) availability for authorized actions, 2) confidentiality and 3) integrity [4].

3 Overview of IEC 61499 Replications

In the industrial context, control processes are dominated by PLCs. With the increase in demand for new, more flexible productive markets, flexibility requirements become preponderant and lack real-time responses. On the other hand, it must be considered that these new needs for flexibility demand greater availability of productive systems and, consequently, greater reliability of them. Reliability or redundancy in centralized applications using PLCs, is usually achieved using a second or more identical PLCs. However, if the same criterion were used in a distributed system, the replication of all devices would become very complex and with exaggerated dimensions.

To address the problems associated with distribution and reliability based on device replication, the International Electrotechnical Commission (IEC) has developed the IEC

61499 standard. The distributed nature of this standard allows each device to perform a simple replica or a sub-application of other devices that are running on the same system.

Based on a several FB – Basic FB (BFB – which runs an Execute Control Chart (EEC) on the head and several algorithms on the body), Composit FB (CFB – several FBs running inside of the FB) and Special Interface FB (SIFB – communication FB) –, the application designer decides which component, as a software unit (FB), to be replicated. The designer replicates the FBs or sub-applications that have greatest influence on the dependability of the system (defining new input and output events and data port, (see Fig. 3a), keeping a single copy of the remaining (see Fig. 3b).

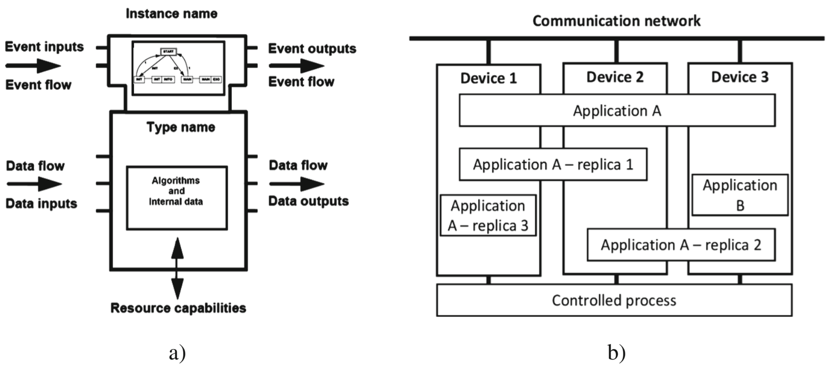

Fig. 3. a) Basic Function Block (BFB) with input and output ports; b) Distribution of replicated and non-replicated applications among divices.

A distributed IEC 61499 application can be distributed or replicated between divices. Thus, the replica of an application can be performed on a single device (application A – replica 3), while other replicas of the same application are distributed between two or more devices (application A – replicas 1 and 2), see Fig. 3b. Based on these distribution possibilities, several interaction scenarios can be identified.

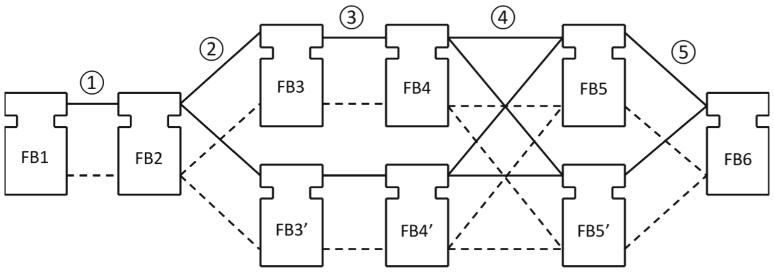

Fig. 4. Distributed system with the several interaction scenarios of the replicated FBs [12].

The interaction scenarios depend on the system distribution and the replication performed by the application designer, that is, on the components that send events and/or data ② *(one-to-many)*, the receivers ⑤ *(many-to-one)* or both ④ *(many-to-many)* are replicated components or not, as you can see in Fig. 4 and explained in [12] and [13].

4 Example for Redundancy Implementation

To implement redundancy, all replicas must be identical. Therefore, for this requirement to be met, all replications must be performed assuming a deterministic behavior, that is, the algorithm of each of the replicated FBs must be deterministic. In this sense, when executing the FBs, in IEC 61499 execution environment, if it is guaranteed that they will be executed in the same way, the determinism of the replicas will be guaranteed. This means that the order in which each event is run will be the same in all replicas and will produce the same results.

To illustrate the implementation of redundancy, based on IEC 61499, we will present a simple example. The fault tolerance approach will be based on simple active redundancy of software and hardware. This application is a small part of a more complex application for conveyor-based pieces distribution systems. In this sense, the case study will be limited to the analysis of the interconnection of two types of conveyors (linear, C1 and C2, and rotating, C3) that working, perpendicularly. C3 receives parts from both conveyors considering that all conveyors are relatively small and will only be able to transport one part at a time and its work according to the layout shown in Fig. 5.

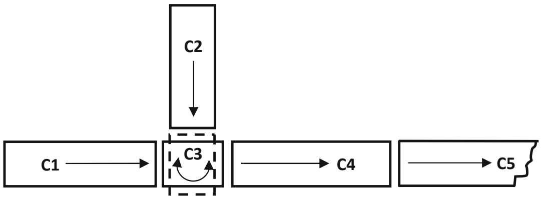

Fig. 5. Example conveyor layout.

Workpieces arrive from the left (C1) and the top (C2) and must be transferred to the rotating conveyor belt (C3) at the center. Each conveyor belt is controlled by a single FB based on the conveyor type (linear or rotating) that runs on an independent low-cost device (Raspberry Pi) that controls the entire mechatronic device of the conveyor.

The pieces to be transferred are simulated by a simple algorithm in each of the FBs. The linear conveyors feed (C1 and C2) provides two integers value corresponding to each of the conveyors, data 1 and 2, respectively. The rotating conveyor (C3) receives data from the feed conveyors. This conveyor processes the data according to the event's availability time (the oldest first) and transfers the part and data information to the following conveyor. Each of these timed messages will be ordered according to the time they were available. A conveyor will be free to receive a new piece as soon as the piece in transport is transferred to the next conveyor.

The design of the application and the replication of elements considered critical of the system are shown in Fig. 6. Note that the conveyors need to receive events and data from the previous ones and send events and data to the following ones. So, conveyors belts on the left, which are feed conveyors, are triggered by a start event sent simultaneously to C1 and C2 and the data produced are sent to the subsequent replicas. Device 3 and 4 are replicas of conveyor C3 that will receive information from the clients C1 and C2. The information received in each of the replicas is an indicator of availability for delivery a work piece. The data received in each of the replicas must belong to the same data set and in the same order.

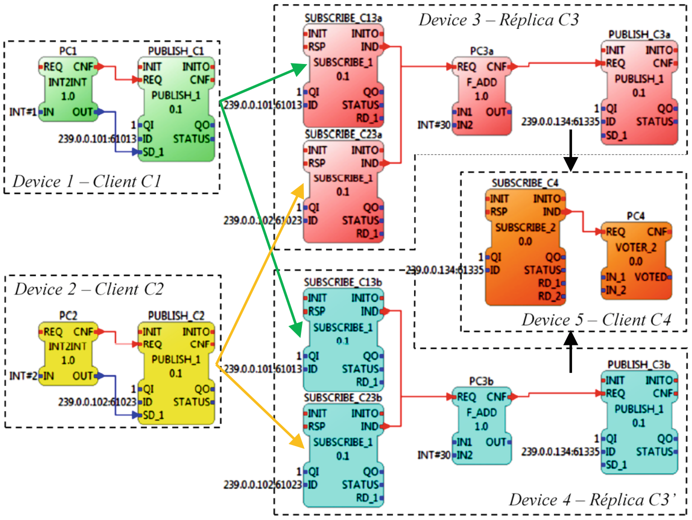

Fig. 6. Design application and replicated elements, interconnection with Publish/Subscribe pair.

4.1 Execution Semantics

In this item, a brief description of the design process and how the replication/distribution was performed in IEC 61499 application will be made. The application design is very simple, and it is focuses on the fault-tolerance approach. To develop the replicated distributed system, we used the open-source *Eclipse 4diac*™ and the FORTE IEC 61499 execution environment [14]. Application was replicated according to scenario ② *(one-to-many)*, shown in Fig. 4 and the final design of it is shown in Fig. 6.

As already explained, to maintain replicas determinism, it will be necessary to ensure that the execution of each replicated FBs occurs in a synchronized manner, that is, that each event received generates the same sequence of actions in all replicated FBs. So, to ensure communications between all replicated FBs and maintain design application, you must use Publish/Subscribe communication SIFBs pairs to implement timed-messages protocol (*one-to-many* and *many-to-one*) as well internal synchronization. On the other hand, all instances of the FORTE runtime environment and devices must be in the same multicast group and be synchronized [15], as well as internal clocks by Network Time Protocol (NTP) synchronization, for example. However, when we consider that "*FORTE* *is relatively well suited for supporting the replication design …*" [13] the expected results will not be guaranteed, since "*… its execution semantics almost guarantee a deterministic execution of event sequences.*" [13] as demonstrated in [16].

4.2 System Reliability

A system is usually composed of several components that will have different reliability and that, in principle, will be known. In this sense, knowing the reliability and trend of each component, determining the reliability of the system will be relatively simple, if the system can be represented by a reliability block diagram (RBD). However, when it is only possible to know the failure rate, without knowing the system's tendency (decreasing, constant or increasing), the adoption of any distribution to quantify the reliability will become an almost impossible task. The RBD representation of the implemented redundant system is shown in Fig. 7. This is a complex system composed of components in series and in parallel. Determining the value of system reliability is a succession of mathematical operations that combine redundancy in series and in parallel to obtain an overall value for it.

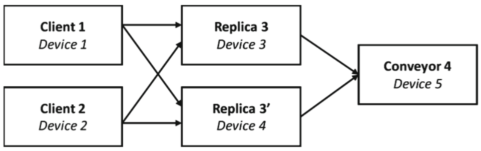

Fig. 7. Reliability for scenario 2, communication *one-to-many*.

However, the study carried out aims, essentially, to analyze the deterministic behavior of the Publish/Subscribe communication pairs, that is, quantify their guarantee of determinism. Therefore, all communication errors between clients 1 and 2 and the replicas C3 and C3′ were recorded and based on this analysis, it was possible to build the graph shown in Fig. 8. The trend of the system can be determined through a process of statistical inference based on hypothesis tests, that is, to verify if it is possible to confirm or deny the formulated hypothesis. To verify H_0 (constant rate) we will have to confirm it or not using the statistical process called Laplace Test (ET). Following equation:

$$ET = \sqrt{12N}\left(\frac{\sum_{i=1}^{N} t_i}{N.t_0} - 0,5\right) = \sqrt{12 \times 567}\left(\frac{2711967}{567 \times 10000} - 0,5\right) = -1,789 \quad (6)$$

where N is the number of faults, t_i the time of fault i, and t_0 the total time. So, in this case the test is conclusive, as there is evidence of acceptance of H_0, since the ET value is outside the rejection region $[-ET(\alpha/2) < ET < +ET(\alpha/2)]$, that is, $-1,960 < ET < +1,960$ with $\alpha = 5\%$. When accepting H_0, rejecting H_1 (non-constant), we consider that the fault rate is constant, and the occurrences are Independent and Identically Distributed (IID).

The fault rate of the system is $\lambda = 0,0567 fault/s$ (5) so, reliability components, considering only replicas, is given by $R(t) = e^{-\lambda t}$. According to Fig. 7, the reliability system (R_S) will be obtained, based on the received components, considering the rest with high reliability, by the expression: $R_s = 1 - \left(1 - e^{-\lambda t}\right)^n$. Data received in the active replicas, have an $R_S^{10000} = 0,951$, a MTTF = 17,637 s, and $F(t) = 0,049$ (2).

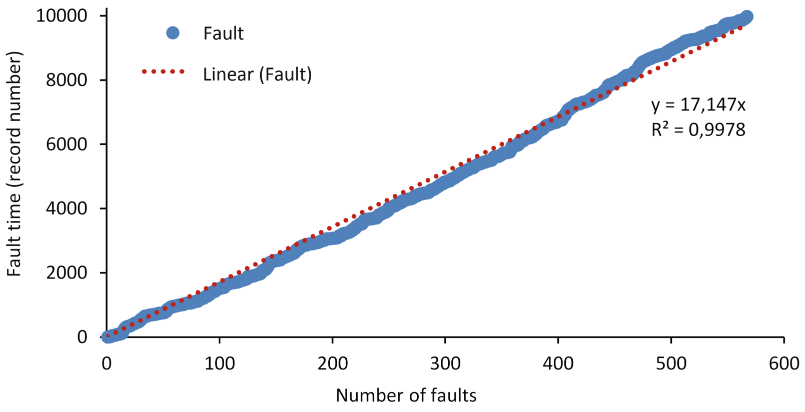

Fig. 8. Relation between fault number and time (F-T).

5 Conclusion

In a distributed system, there are usually partial failures that must be considered. Critical component redundancy is one of the measures that must be used for fault-tolerance. However, the adoption of redundancy measures will lead to replication with or without voters in which the results outgoing replicas must present the same values. This restriction will require that all replicas remain synchronized.

The use of timed messages protocol and NTP are an important contribution to ensure the synchronization of the replicas. However, it will be necessary to consider the different communication scenarios between replicated and non-replicated components and how to guarantee the determinism of IEC 61499 applications, considering their restrictions. Thus, to ensure the ordering of data in a FORTE execution environment, it will be necessary to ensure that the Publish/Subscribe communication pairs send and receive the same data set in the same order in which they are made available.

The analyses carried out shows that the design of replicated systems, using the FBs defined in the IEC 61499 standard is not able to guarantee its determinism. Thus, based on the lead time and the ordering of data received, it was possible to quantify the reliability of the system, considering only the data received in the replicated pairs, so the measured reliability was around 95%. These results define a 5% rate of unreliability (base in 10 K data received in each replica), which will indicate that the replication based on the IEC 61499 FBs and FORTE runtime do not guarantee reliability by itself. Reliability will be guaranteed by the development of additional FBs that implements it.

Acknowledgments. 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 this work was supported by FCT, through INEGI and LAETA, project UIDB/50022/2020.

References

1. IEC 61131: Programmable Logic Controllers Part 3 (IEC 61131-3). International Electrotechnical Commission, 3rd edn. IEC (2013)
2. IEC 61499: International Standard IEC 61499-1, Function Block Architecture Part 1, 2nd edn. International Electrotechnical Commission. IEC (2012)
3. IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance. http://wg10.4.dependability.org/. Accessed 07 Dec 2020
4. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secure Comput. **1**(1), 11–33 (2004)
5. Farrukh Khan, M., Paul, R.A.: Chapter 4 - pragmatic directions in engineering secure dependable systems. In: Advances in Computers - Dependable and Secure Systems Engineering, vol. 84. Elsevier, USA (2012)
6. Modarres, M., Kaminskiy, M.P., Krivtsov, V.: Reliability Engineering and Risk Analysis, a Practical Guide. 3rd edn. CRC Press, Taylor & Francis Group (2017)
7. Yang, M., Hua, G., Feng, Y., Gong, J.: Chapter 1 - introduction. In: Fault-Tolerance Techniques for Spacecraft Control Computers. Wiley (2017)
8. Bloomfield, R., Lala, J.: Safety-critical systems: the next generation. IEEE Secur. Priv. **11**(4), 11–13 (2013)
9. Abdulhameed, O.A., Jumaa, N.K.: Designing of a real time software fault tolerance schema based on NVP and RB techniques. Int. J. Comput. Appl. **180**(26), 35–4 (2018)
10. ISO 9000-3:1997: Quality management and quality assurance standards - Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software. ISO (1997)
11. O'Connor, P.D.T., Kleyner, A.: Practical Reliability Engineering, 5th edn. Wiley, UK (2012)
12. Santos, A.A., de Sousa, M.: Replication strategies for distributed IEC 61499 applications. In: IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, pp. 2225–2230. IEEE. Washington, DC (2018)
13. de Sousa, M.: Chapter 9 – fault-tolerance IEC 61499 applications. In: Distributed Control Applications: Guidelines, Design Patterns, and Applications Examples with the IEC 61499, 2nd edn. CRC Press, Boca Raton (2016)
14. Eclipse 4diac. https://www.eclipse.org/4diac/. Accessed 29 Dec 2020
15. Pinho, L.M., Vasques, F., Wellings, A.: Replication management in reliable real-time systems. Real-Time Syst. **26**(3), 261–296 (2004)
16. Santos, A.A., da Silva, A.F., Magalhães, A.P., de Sousa, M.: Determinism of replicated distributed systems-a timing analysis of the data passing process. Adv. Sci. Technol. Eng. Syst. J. **5**(6), 531–537 (2020)