Multiply and Conquer: A Replication Framework for Building Fault Tolerant Industrial Applications

Mário de Sousa, Christos Chrysoulas, and Aydin E. Homay
INESC TEC - INESC Technology and Science, and University of Porto, Faculty of Engineering
Porto, Portugal
{msousa, chech, homay}@fe.up.pt

***Abstract*—TIEC 61499 defines an execution model for distributed industrial control applications, i.e. a single application distributed among several devices. In such an environment partial failures are likely to occur. In order to avoid probable system malfunctions and breakdowns due to partial failures, the authors have previously proposed a framework where the concept of replication may be applied to the IEC 61499 execution model. This paper focuses on describing an implementation of this replication framework on the FORTE IEC 61499 execution platform, along with the results of the first tests of the implementation. A set-up for the full validation of the approach is also described.**

Keywords—IEC 61499; fault tolerance; replica determinism

I. Introduction

The IEC 61499 [1] standard provides support for the development of distributed control applications. The main novelty brought by this standard is an event driven execution approach, which provides the synchronisation primitives between the sub-applications that compose the distributed control application, while its predecessors IEC 61131-3 [2] and IEC 61131-5 [3] never really managed to cope with the distributed nature of the modern industrial applications.

The fact that the control program is now distributed among several devices brings new challenges, especially in the timing domain as well as in the failure modes. In the first case, changes in the timing characteristics of the application occur due to the delays introduced by the network in the propagation of data and events – delays that are not present when the application is centralised. In the second case the distributed execution environment allows the occurrence of partial failures or break downs during the execution of the program.

These are the new failure modes mentioned above that the developer must take into account, especially in safety-critical applications. A typical approach to overcome these failures is based on building fault tolerant applications based on redundancy. By introducing redundancy, the failure of one sub-component of the distributed system will be masked by the continued execution of its redundant partner.

The authors define a framework in which this approach may be applied in the IEC 61499 environment. This framework takes special care in guaranteeing that all replicas are kept with the same synchronised internal state, so that changing from one replica to another does not impact the remainder of the distributed application. Additionally, the framework allows the use of partial replication, where the application developer may choose to only introduce replication in the most critical software components.

In this paper the proposed framework is summarised in section II, and a description of an implementation of the replication framework on the FORTE IEC 61499 execution environment is made in section III, together with an initial basic test application. In section IV a full validation example based on a Baggage Handling System is presented. Section V gives a brief reference to related work, and conclusions and future work are presented in Section VI.

II. IEC 61499 and Replication Framework

IEC 61499 applications can be described simply as being composed by a network of blocks that transfer data and events between them. Each building block, known as a Function Block (FB), has data and event inputs as well as data and event outputs. This FB network is typically drawn in a graphical editor, and an IEC 61499 application will include the data and event connections between the FBs.

FBs may be defined as either a network of other FBs (a composite FB), or defined using algorithms, data variables, and a state machine (a basic FB). FB instances must execute in a single execution device (computer, PLC, embedded platform, ...), but IEC 61499 applications, being composed by a network of FBs, may be distributed amongst several execution devices. In this case the data and events transferred between FBs residing on distinct devices will be transmitted over a communication network connecting these same execution devices. IEC 61499 sub-applications are similar to an application in that they are composed by a network of FBs and may be distributed amongst several devices.

FBs are executed when they receive an event, upon which the basic FB samples its data inputs and subsequently evaluates its ECC (the state machine). If this results in a change of the ECC state, any algorithms associated with the new state are executed and an output event may be generated. It is also possible for output variables to be changed when an output

This work is financed by the ERDF-European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within projects FCOMP-01-0124-FEDER-037281 and FCT EXPL/EEI-AUT/2538/2013.

event is sent if those output variables are "associated" with the output event. The ECC will then continue to be assessed until no further changes are possible.

Note that events may easily explode in a FB network, as a FB that is executing may generate several output events before its own execution has completely terminated. The FBDK [4] execution environment executes these events sequentially. The FORTE [5] execution environment identifies events generated by the IEC 61499 environment (for example due to timers or network activity) as source events, and places them in a FIFO event queue. Only when the FB completely finishes handling the input event is the next event in the queue processed.

Access to the process (physical I/O) and communication (network) interfaces is done through specialized Service Interface FBs (SIFB). SIFBs must be provided with the IEC 61499 execution environment, as their implementation is specific to the type of device executing them. Communication SIFBs must be inserted in an application that is distributed between several execution devices, in order to propagate the events and data over the communication network.

Replication in industrial automation context is typically achieved at the hardware level. An execution device is replicated, and each replica runs an exact copy of the software of the original. We have proposed a more flexible software based replication architecture. In this approach the atomicity of replication is a FB - in other words, each FB may be replicated independently of all other FBs.

Typically, only the most critical FBs will be replicated onto several hardware execution devices, whereas the non-critical components may be executed with a single copy. The replication framework also supports the replication of sub-applications, in which case the replica may reside on another execution device, or distributed among several devices.

Taking into account that not all software components are replicated, several interaction scenarios may be identified (Fig. 1): (1) communication between two non-replicated FBs; (2) communication from a non-replicated FB to a replicated FB; (3) communication between the components of a replicated FB; (4) communication between two replicated FBs; and (5) communication from a replicated FB to a non-replicated FB.

For scenario 4 the receiving FB will receive multiple data values that need to be consolidated into a single data value. Voters are widely used with a variety of fault tolerance techniques, including design and data diverse techniques. Voters compare the results of two or more variants, and decide the correct result, if one exists. There are many type of voters [6], and the decision on which voting algorithm to use is dependent on the required application semantics. The replication framework will therefore include some basic voting algorithms that may be used by the IEC 61499 application developer, but more importantly, will provide an interface for specific voters that may need to be developed for the specific application.

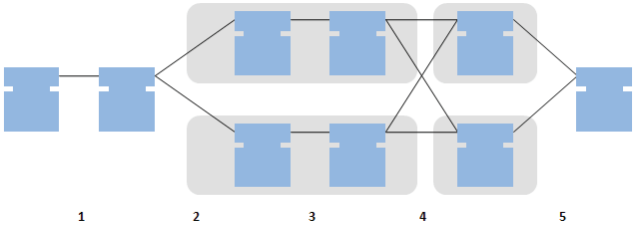

Fig. 1. Data/Event passing scenarios.

For the voting to be feasible all replicas must be kept with their internal data structures in synch – this will guarantee that replicas will all output the exact same value when no errors occur. Replica synchronism may be achieved by guaranteeing that all events and data arrive at all the replicas in the exact same order, and the the replicas implement deterministic algorithms. To guarantee the event and data ordering, alternative implementations for the communication SIFBs for scenarios 2 and 4 must be provided by the replication framework.

Due to the possible varying network delays in the transit of messages over the network these alternative SIFB introduce appropriate delays in the data that are transmitted faster, therefore guaranteeing that all replicas receive all messages in the exact same sequence. This, as explained in the previous paper [7], is known as the timed messages protocol [8].

Several versions of the SIFB can be provided, depending on the fault models that need to be tolerated. Fault models that include errors in the communication networks require more costly (slower) communication protocols when compared to communication protocols that may be used when only faults in the execution devices are considered.

III. Implementation

We have made an implementation of the replication framework on the FORTE IEC 61499 execution platform by extending FORTE's communication SIFBs.

A. 4DIAC and FORTE

FORTE is basically a virtual machine that executes IEC 61499 applications. This execution environment is developed together with the 4DIAC [5] development environment, where the programmer may design IEC 61499 applications using a graphical interface. Once programmed, 4DIAC downloads the IEC 61499 applications onto IEC 61499 execution environments using a standardized interface of these execution environments.

B. FORTE Communication FB Architecture

IEC 61499 defines communication SIFB interfaces for both the publish/subscribe and client/server interaction models. Each of these can have several implementations, typically mappings to different underlying communication protocols. Ideally we should implement a version of each client/server and publish/subscribe communication SIFB that supports replication, however, since the publish/subscribe SIFBs seem to be more often used in IEC 61499 applications, we have focused on these first.

In FORTE, communication SIFBs are implemented using a layered architecture [20]. This allows FORTE to provide a single version of the publish and subscribe SIFB – instead of choosing a specific SIFB implementation for each situation, the user merely needs to configure the communication layers that each communication SIFB instance will use. This is done through a standard input parameter, using a syntax layer1[layer1_parameters].layer2[layer2_parameters].

We have implemented several new "replication" layers that work between the top layer (typically fbdk) and the bottom layer (usually ip layer). This means that, for example, to the usual syntax fbdk[].ip[225.0.0.1:61499], the user merely needs to add a middle replication layer: fbdk[].replication[T#5ms].ip[225.0.0.1:61499]. In reality, several versions of the replication layer are needed, to cover the interaction scenarios mentioned in Fig. 1. The most representative are scenarios 2 and 5.

1) Scenario 2

For scenario 2 (Fig. 1), the replication layer on the publishing side adds a time-stamp specifying the time at which the event will become valid, by reading the current time and adding a fixed offset. This offset is a configuration parameter for this layer, and as was described in [7], should be determined off-line by taking into account the maximum delays for the transmission of this message to all replicas. The subscribing side of this layer strips the time-stamp, and waits until the event becomes valid. Only at this time is the event sent to the upper layer.

2) Scenario 5

For scenario 5 (Fig. 1) the publishing layer sends the time-stamp of the event that was received from the subscribing layer of (2), and adds another fixed offset. The corresponding replication layer on the subscribing side once again waits until the validity time comes to pass, and at that time will vote on all the messages it has received with the same validity time. Currently the voting mechanism simply chooses the first message to have arrived.

3) Scenario 4

No new replication layer is required for scenario 4 as this scenario is identical to multiple many-to-one connections (scenario 5). Simply creating several instances of the FBs required for scenario 5 is sufficient.

C. Message Queueing

Delaying of messages has to be done in a non/blocking fashion - while waiting for the validity time of a message, other messages that arrive in the meantime cannot be lost, and must be handled using the exact same algorithm. Therefore, delaying of messages is implemented as a queue, where all outstanding messages are placed, and a timer event is set for the validity time of the message at the top of the queue.

Following the design pattern already present in FORTE, the timer is set using the CTimerHandler service that runs as an independent execution thread. The replication layer registers the communication SIFB (registerTimer(FB) in Fig. 2) to receive the timer expiration notification. When the timer expires, CTimerHandler notifies the CEventChainExecution service (that also runs as an independent thread) to start a new event chain (startEventChain()). The event chain calls the communication SIFB (CComFB) through the executeEvent() call, who in turn forwards this notification to the replication layer (processInterrupt()). Notice that the replication layer must first register with the CComFB to be called with processInterrupt(), as the CComFB has now way of knowing which layer initially set the timer that has just expired.

The described mechanism guarantees that events and data being sent to several replicas will be released in the replicas all

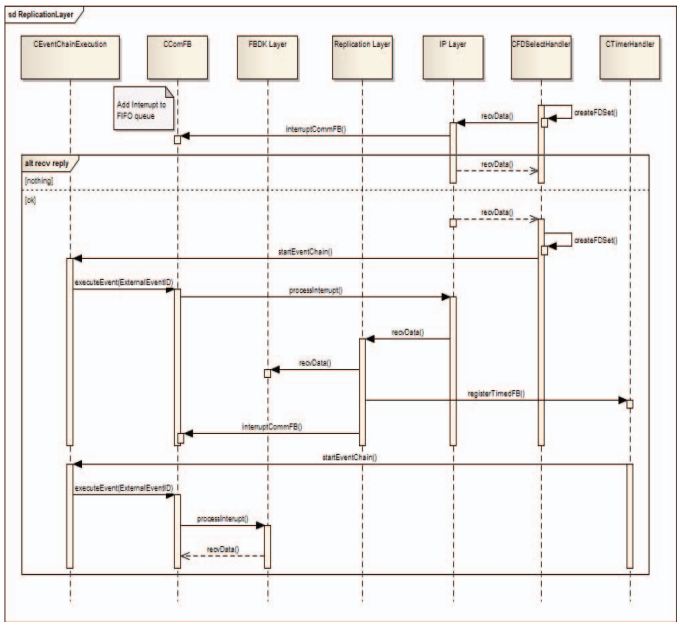

Fig. 2. Message queuing.

at the same time, as long as all the devices in the distributed system have synchronized clocks. It should be noted that several standardized protocols for clock synchronization already exist, some of which are already widely used in the industrial automation domain, and can achieve synchronization errors down to the nano second range [9].

D. Test Application

Some simple tests were run based on the trivial XPlus3 sample application that comes with the 4DIAC development environment in order to make an initial validation of this implementation. The original sample application (read input, add 3, print result) simultaneously uses the FORTE and FBDK execution environments. FBDK is used in order to provide a graphical interface to the user (read input, print result), whereas FORTE is used to execute the FB containing the main application logic (add 3). This is possible since the FBs of the IEC 61499 application are distributed between two devices (or execution environments), and these FBs use communication SIFBs in each execution environment to exchange data and events.

Using the new replication layers a trivial replicated version of this application has been tested. In the replicated version (Fig. 3), the FB that adds a constant value to the input has been replicated. In principle, both replicas should add the same value, but in our test we are adding distinct values in each replica (3 and 4) so it becomes possible to identify which result was chosen by the voter.

Since the replication framework is being implemented solely for FORTE, all the replication related FBs must run on FORTE (and not FBDK). In other words, the FBDK device cannot execute the replication specific communication SIFBs, so the 'B' and 'B_0' were introduced to serve merely as intermediaries (or proxies) between the graphical user interface and the main replicated algorithm.

The sample replicated application therefore uses 5 devices (Fig. 4); one device runs on FBDK to provide the user interface

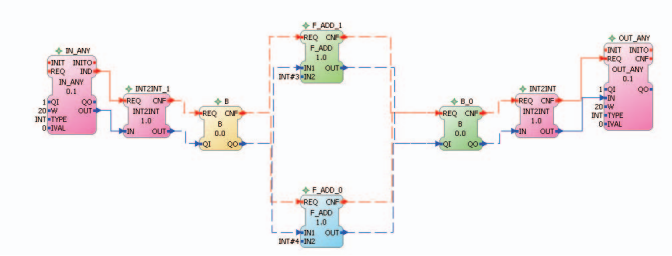

Fig. 3. Replication in XPlus3 example.

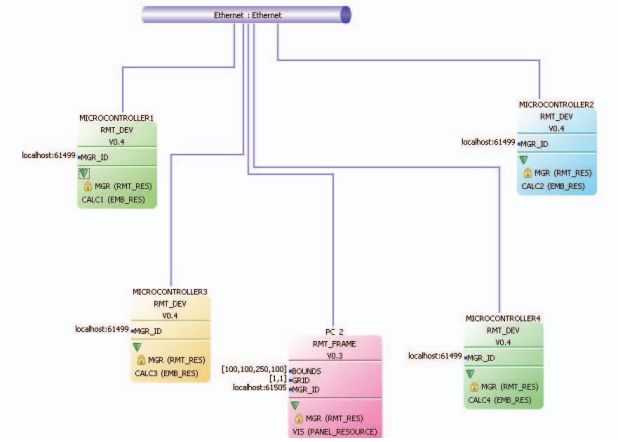

Fig. 4. System view and Communication Network in XPlus3 example.

(both input and output), whereas each of the other four devices is a distinct instantiation of the FORTE execution environment. One FORTE device runs the 'B' proxy FB, another runs the other proxy FB 'B_0', and the remaining two FORTE instances each execute one of the replicas of the main FB ('F_ADD').

This trivial replicated application was successfully tested with all device instances running on the same computer. However, since communication between all instances is all done over a TCP/IP multicast network connection, we are convinced that the application would work the same were each FORTE and FBDK execution environment to run on a distinct hardware platform, with the platforms connected on a network.

Device failures were tested by simply stopping and starting each of the execution devices. The results were as expected, where the application was able to produce an output result as long as at least one of the replica devices were executing.

IV. Full Validation Setup

We are currently starting to build a more complex application that will fully validate the replication framework.

A. Work-piece Transportation System

An example of where the replication framework will be useful is an airport baggage handling system, which is a classic example of a complex distributed automation system that requires high performance, reliability, and flexibility. These systems come with sensors and actuators, embedded control devices and machines, all combined and working in a seamless way. The applicability of using the IEC 61499 reference model in implementing this control architecture has been studied by Yan, and Vyatkin in [10], who created a test bed of more than 50 networked control nodes simulating a small sized airport. The results from the experiments proved that IEC 61499 can meet the needed levels of flexibility.

Instead of an airport baggage handling system, we are building an application to control the flow of work-pieces in a manufacturing cell (Fig. 5). This transport system is also based on conveyors, and for simplicity of the control application we assume that work-pieces always follow a single direction: linear conveyors simply transfer pieces from one end to the other (action (i)), while rotating conveyors can rotate about their vertical axis, therefore either transferring the piece to one of two destinations (action (ii)) or receiving it from one of two origins (action (iii)) and transferring it to the same destination.

Every conveyor is assigned a FB, to be executed by a PLC type device. The most critical conveyors for the correct functioning of the whole transport chain are controlled by at least two PLC type devices, so that a possible failure in one device will not affect the functionality of the chain. In this paper we use just a small part of a transportation system to illustrate the replication functionality (Fig. 6).

This example contains 4 conveyors. Work-pieces enter this sub-system through C0, and are transferred to C1. From here, pieces can either go towards C2 or C3. The decision depends on the feedback the controller of C1 receives from the Distance Acceptance Stations residing on C2 and C3. The first one declaring that there is no work-piece in its visual field will be the one qualified to accept a new piece. Fig. 7 presents the FB networks controlling this physical layout.

B. Adding Replication

Also for simplicity of presentation in the paper, we consider that in this layout only conveyor C1 plays a critical role. This being the case, only a new instance of the FB

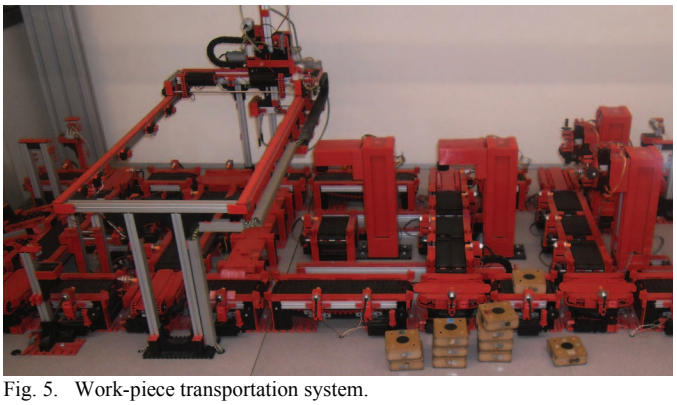

Fig. 5. Work-piece transportation system.

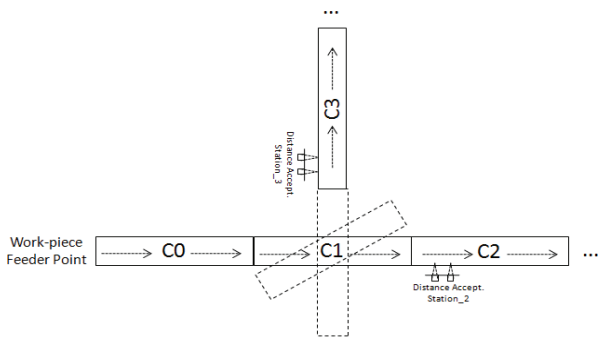

Fig. 6. Work-piece spacing control and diverting system.

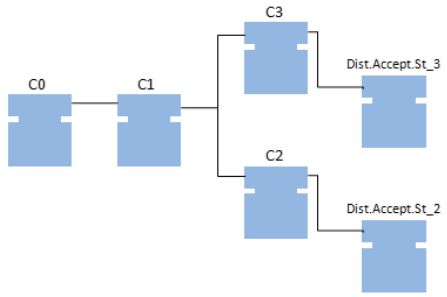

Fig. 7. FBs connected following the physical layout.

controlling this conveyor (Fig. 9) is inserted. Each replica is allocated to a distinct execution device, in this case a Raspberry Pi [11] running FORTE execution environment. These same devices also control other conveyors, so the addition of a FB replica does not necessarily imply an increase in the number of physical devices.

Fig. 8 presents the simplified state machine diagram of the FB controlling C1. Notice that C1 is capable of receiving work-pieces, and depending on the events it receives from C2 and C3, it passes the piece to the first conveyor that becomes free.

The replication framework must guarantee that both replicas always maintain a synchronised state. In this case, this means that both replicas must always make the same decision as to the destination of the work-piece (C2 or C3), so that if one replica fails, the other replica continues providing coherent control signals to the physical hardware. In the case of the C1 conveyor, this decision will depend on the order of the events arriving from conveyors C2 and C3, so we need to guarantee that all messages from the down-stream conveyors reach the C1 replicas in the exact same order.

As was previously stated, maintaining replica determinism will be guaranteed by the new communication SIFBs provided by the replication framework. We therefore merely replace the version of the publish/subscribe used when sending data and events to the replicated FB (in FORTE this actually consists on adding the appropriate replication layer as a parameter to the already existing communication SIFBs). In the example application, this means changing all the publish/subscribe SIFBs used to transmit data from C0, C2 and C3 to C1.

Data and events going out from the replicas must be consolidated into a single data point or event. This is also achieved by changing the publish/subscribe SIFB used to transfer data from C1 to C2 and C3. Note that each FB

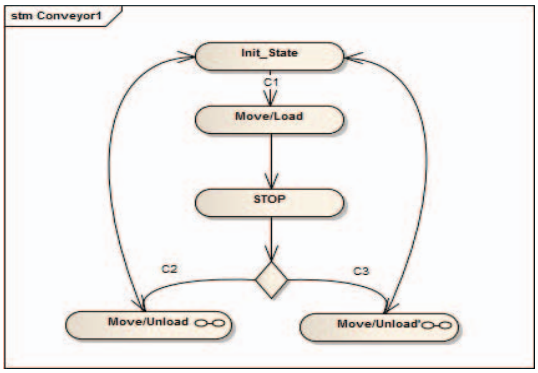

Fig. 8. Conveyor 1 state machine diagram.

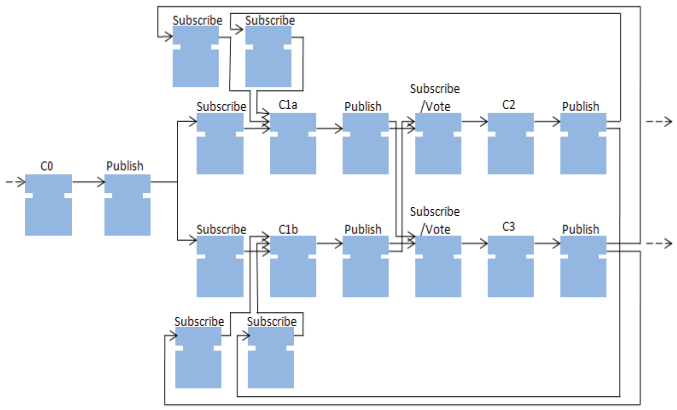

Fig. 9. Replication of the most critical function block; adding data communication function blocks.

receiving data from the replicas will independently vote on the received data/events. In this application we could use majority voting, which in the case of only two replicas reduces to a simple comparison (if not equal, ignore data/event).

Note that the voter is itself a single point of failure, and therefore must have a higher reliability than each of the replicas individually for the replication scenario to achieve a higher reliability than a non-replicated solution. In the current scenario where we decided that FBs for conveyors C2 and C3 do not need to be replicated, then the fact that the voter for each of these conveyors is not replicated is not an issue, as we can consider that a failure of C2's voter is equivalent to the failure of C2 itself (which is not considered critical, since we decided not to replicate C2's FB).

In the validation set-up we are building, the controllers for C2 and C3 are also replicated. In this case each of the C2 FB replicas have its own voter – i.e., the voter for each downstream conveyor is replicated, for a total of 4 voters (2 for C2, and 2 for C3). In truth, the set-up includes many more conveyors, but these are not mentioned for simplicity of exposition.

C. *Hardware*

The FBs controlling the conveyors must send out signals to the conveyor's actuators. However, when using replicas these physical signals are also replicated, and must therefore also be consolidated. In our set-up the physical conveyors include a Remote Terminal Unit (RTU), which basically functions as a physical I/O interface that may be controlled remotely over a network. In our case, we use the Modbus/TCP communication protocol, with the RTUs functioning as Modbus servers.

Notice that Modbus/TCP is a client/server protocol that allows multiple clients accessing the same server. In our validation application we therefore have all replicas of the same conveyor sending data to that conveyor's RTU. The RTU will function as a voter, placing on the outputs the most recent information received from either replica. By considering that all execution devices follow the fail-stop failure-mode (either work correctly, or stop completely), we can conclude that all working replicas always provide correct outputs, so all information sent to the RTUs will always be correct. If one replica fails, the RTU simply keeps on listening to the single replica that continues sending updated information.

The RTU, being a voter, may also be considered a single point of failure. In our validation set-up we simply consider that these RTUs are highly reliable, and therefore never fail. In a real-world application, voters can be built out of electrical relays in series or parallel, removing all programmable equipment from the voter, and therefore increasing the reliability of the voter to the required level.

To correctly test if the replication mechanism guarantees synchronised state between the replicas, we must make sure that message transmission over the network differs between pairs of executing devices. To achieve this we will use distinct execution devices (Raspberry PI, Personal Computers, and low powered embedded X86 devices), as well as build a (in principle unnecessarily) complex network architecture, with some nodes using a wired network interface and others using wireless. Even this might not be enough if the difference in time in which the two conveyors C2 and C3 (Fig. 6) become free is usually large when compared to the usual network delays. To really force the issue, we physically wire the controllers of C2 and C3 to both use the same physical sensor that indicates that they are free to receive a new work-piece. Controllers for C2 and C3 will therefore consider both conveyors to be free in the exact same instant. Even in this strict situation, all replicas (controlling C1) must always make the same decision of where to send the work-piece. Since all replicas are constantly writing to the Modbus server the values of their outputs, any discrepancy in the values they send is immediately detected by a shaking of the physical conveyor.

V. Related Work

To the authors' knowledge, little work has been done regarding the use of fault tolerance in the context of IEC 61499 applications. However, somewhat related is the use of IEC 61499 in safety critical applications - [12] applies formalisms to model and validate IEC 61499 applications. Although not in the context of safety-critical application, a lot of work has been done on formalising the IEC 61499 execution semantics (as a limited example [13][14]).

Another approach focuses on on-line reconfiguration of IEC 61499 control applications while guaranteeing the control application's real-time requirements. [15][16][17][18]. This work does not however consider the requirement of keeping the internal state synchronised between replicas, which is fundamental to our approach.

VI. Conclusions and Future Work

The paper presents the implementation of a replication framework for IEC 61499 on the FORTE execution environment. An initial assessment of this implementation is made, and from these initial limited tests we can conclude that the replication framework works as intended.

The paper also describes the setup we are using to run more stringent tests of both the replication framework, as well as its implementation. We expect to present the results of these tests and a final evaluation of the framework as a journal paper that will be a continuation of this same paper.

Future work will focus on adding other communication protocols capable of tolerating faults on the communication network itself (e.g., [19]).

Acknowledgement

Dr. Alois Zoitl helped in defining the sequence of actions for handling the delaying of messages [Fig. 2].

References

[1] International Electrotechnical Commission, “IEC61499-1 ed2.0 Function blocks - Part 1: Architecture”, 2012-11-07.

[2] International Electrotechnical Commission, “IEC 61131-3 ed3.0 Programmable controllers-Part 3: Programming languages“, 2013-02-20

[3] International Electrotechnical Commission, “IEC 61131-5 ed1.0 Programmable controllers - Part 5: Communications ”, 2000-11-15

[4] FBDK: http://www.holobloc.com/doc/fbdk/ (accessed 2015/02/17).

[5] FORDIAC & FORTE: http://www.fordiac.org (accessed 2015/02/17).

[6] L. L. Pullum, “Software Fault Tolerance Techniques and Implementation”, Artech House computing Library, 2001.

[7] M. Sousa, “Guaranteeing Replica Determinism on IEC 61499”, in 19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Barcelona, September 2014.

[8] S. Zhang, A. Burns, J. Chen, and E.S. Lee, “Hard real-time communication with the timed token protocol: Current State and Challenging Problems”, Real-Time Systems, Volume 27, Issue 3, pp.271-295, 2004.

[9] IEEE, “1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems”, 2008 (also available as “IEC 61588 ed2.0 Precision clock synchronization protocol for networked measurement and control systems “, 2009-02-27).

[10] Y.Yan, V. Vyatkin, “Distributed execution and cyber-physical design of Baggage Handling automation with IEC 61499”, IEEE International Conference on Industrial Informatics (INDIN), 26-29 July 2011, pp.573-578.

[11] Raspberry PI: http://www.raspberrypi.org/ (accessed 2015-02-17).

[12] L. Yoong, “Modelling and Synthesis of Safety-Critical Software with IEC 61499”, PhD Thesis submitted for Electrical and Electronic Engineering, University of Auckland, 2010

[13] G. Cengic, O. Ljungkratz, K. Akesson, “Formal modeling of Function Block applications running in IEC 61499 execution runtime”, in 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Prague, September 2006, pp. 1269-1276.

[14] V. Dubinin, V. Vyatkin, “On Definition of a Formal Model for IEC 61499 Function Blocks”, EURASIP Journal of Embedded Systems, 2008.

[15] A.R. Sardesai, O. Mazharullah, and V. Vyatkin, “Reconfiguration of Mechatronic Systems Enabled by IEC 61499 Function Blocks”, in Australasian Conference on Robotcks and Automation (ACRA '06), Aucкland, 6-8 December 2006.

[16] T. Strasser, I. Müller, C. Sünder, O. Hummer, and H. Uhrmann, “Modeling of Reconfiguration Control Applications based on the IEC 61499 Reference Model for Industrial Process Measurement and Control Systems”, in IEEE Workshop on Distributed Intelligent Systems (DIS '06), Prague, June 2006, pp. 127-132.

[17] A. Zoitl, C. Siinder, and I. Terzic, “Dynamic Reconfiguration of Distributed Control Applications with Reconfiguration Services based on IEC 61499”, in IEEE Workshop on Distributed Intelligent Systems (DIS '06), Prague, June 2006, pp. 109-114.

[18] A. Zoitl, “Real-time execution for IEC 61499”, Durham, North Carolina, International Society of Automation, 2008.

[19] X. Defago, A. Schiper, P. Urban, “Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey”, ACM Computing Surveys, Vol 36, No.4, December 2004, pp. 372-421.

[20] M. Hofmann, M. Rooker, and A. Zoitl,, “Improved communication model for an IEC 61499 run time environment”, Emerging Technologies &FactoryAutomation (ETFA), 2011 IEEE 16th Conference on,vol., no., pp. 1, 7, 5-9 Sept. 2011.