Exploiting Voting Strategies in Partially Replicated IEC 61499 Applications

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

***Abstract* — In a modern industrial environment control programs are distributed among several devices. This raises new issues and challenges especially in failure modes. Building fault tolerant applications can be the solution in order a failure of one sub-component not to jeopardize the execution of the whole application. The authors have proposed a framework to support replicated IEC 61499 applications. In this paper we augment this framework with the support for different voting strategies, propose an extension of the replication communication protocol, and analyse the resulting fault-tolerance semantics. A limited implementation of the framework is also described.**

Keywords—IEC 61499; fault tolerance; replication

I. Introduction

The IEC 61499 [1] standard introduces a new paradigm where a single distributed control application may be built to automate the entire process, but its execution is distributed amongst the several execution devices. 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. However, in a distributed environment partial failures during the execution of the program may occur, and the developer must take into consideration how the application will react when these partial failures happen. A typical approach to overcome these failures is based on building fault tolerant applications that exploit redundancy, where the failure of one sub-component will be masked by the continued execution of its redundant partner.

The authors have defined a framework that supports tolerating partial failures of a distributed IEC 61499 application. This framework takes special care in guaranteeing that all replicas are kept with the same synchronized internal state, so that changing from one replica to another does not impact the remainder of the distributed application [2].

The IEC 61499 standard is summarized in section II of this paper, where an updated version of the previously proposed replication framework is also presented. Section III explains the communication protocol used to support the replication framework, and section IV discusses the resulting fault-tolerance semantics that can be obtained. In Section V a slightly modified version of the protocol is presented and its advantages are discussed. An example application and an initial implementation of the proposed framework based on the FORTE [3] IEC 61499 execution environment is given in section VI. Section VII gives a brief reference to related work, and conclusions are presented in Section VIII.

II. IEC 61499 and Replication Framework

IEC 61499 defines the syntax of a graphical programming language, as well as the semantics of the platforms required to execute programs written in this language. Applications (the name given to programs in IEC 61499) are composed by a network of Function Blocks (FBs) which exchange data and events through separate data and event connections. A FB is only activated when it receives an input event, upon which it will sample its data inputs, executes its internal algorithms, update its data outputs, and eventually emit one or more output events. The semantics of IEC 61499 applications may therefore be described as event based.

Each FB may itself be constituted by another internal FB network, whose evaluation follows the same rules as the evaluation of an applications' FB network – these are known as composite FBs. The hierarchical levels of FBs based on other FBs is not limited by the standard, however there comes a time when the desired algorithm can no longer be expressed using a FB network. In this case a basic FB is used, where the internal algorithms are expressed using any other programming language (e.g.: Java, C, IEC 61131-3 languages, etc.). Basic FBs also contain an internal state machine (known as the ECC – Execution Control Chart) whose state transitions may depend on the receipt of input events, or the state of an input, output or internal variable. The activation of a state can be configured to trigger a single execution of one of more of the internal algorithms, as well as the triggering of an output event. The data outputs are changed by the execution of the algorithms.

An application (a graphical network of FBs) may be executed within a single execution device (basically any computer that implements the IEC 61499 execution semantics). In spite of this, the main advantage of IEC 61499 is to allow applications to be distributed amongst several devices, as long as each basic or composite FB is allocated inside a single device. Execution devices must be connected to a common data communication network which is used to transfer the events and data between the FBs residing on distinct devices. This exchange of data and events is not automatic – the programmer or application installer is required to manually insert special communication FBs (known as communication Service Interface FBs – SIFBs) at both ends of every data or event connection that has to traverse a data communication. The IEC 61499 standard defines a standard interface for these SIFBs, covering both the publish/subscribe and the client/server interaction models. Different implementations of each interface are allowed, therefore supporting distinct communication protocols as well as physical networks.

978-1-4799-8244-8/15/$31.00 ©2015 IEEE

We exploit this last facet of the IEC 61499 design to create a framework to support the replication of IEC 61499 applications. This framework is based on providing specific implementations of these communication SIFBs that take into account the existence of FB replicas. Note that our architecture for replicated IEC 61499 applications allows that these only be partially replicated. We expect that the application designers will choose to replicate only those FBs whose execution have a higher impact on the safety of the overall system, or those FBs executing on devices with a higher failure probability.

Several versions of the replica supporting communication SIFBs are required, depending on whether the sender, the receiver, or both are from/to replicated FBs. Fig. 1 provides a visual description of the several possibilities, whereas Fig. 2 highlights the distinct versions of the communication SIFBs that are required to support replication.

For scenario 1 (Fig. 1), standard communication SIFBs that do not support replication may be used. For scenario 2 we provide a set of publish/subscribe communication SIFBs whose implementation implicitly assumes the one-to-many scenario. For scenario 5 we also provide a set of publish/subscribe communication SIFBs, where the subscriber FB expects to receive a message from each replica before sending out the data from both messages simultaneously. Several versions of the subscribe FB are needed as the number of data outputs must match the number of replicas – one version for a specific number of replicas. This scenario also requires the use of a voting FB, that will consolidate both outputs into a single value. Unlike previous versions of our replication framework, by separating the voter from the subscriber it now becomes possible to have different voting strategies implemented in distinct voting FBs, and therefore allow the application designer the choice of which voter to use for each specific situation (e.g., majority voting, median voting, average voting, etc.).

Scenario 4 uses a mixture of FBs already provided for scenarios 2 and 5. We re-use the publishers from the one-to-many publish/subscribe pairs, and the subscribers from the many-to-one pairs. The voter FBs are also the same as those used for scenario 5.

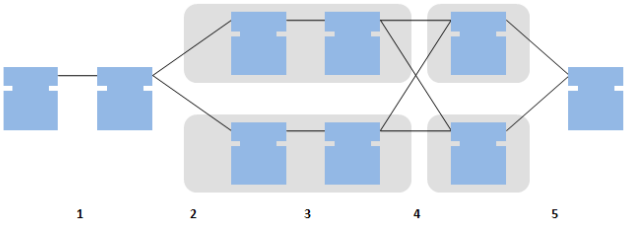

Fig. 1. Data/Event passing scenarios.

III. Communication Protocol

In the described replication framework the correct execution of the replicated IEC 61499 application depends on maintaining synchronised the internal state of all replicas of the same FB. If this were not the case, in a non-failure situation each replica would send distinct output data for the same input event making it impossible to correctly determine what should be the value of the consolidated output to be generated by the voting FBs.

Replicas may become de-synchronised when the communication delays are not identical between all replicas, or the time to execute a replica differs between execution devices. The most simple situation occurs when a FB A receives data/events from two FBs B and C. If FB A is replicated (two replicas A1 and A2), then it becomes plausible that two events sent out by FBs B and C may arrive in one order at replica A1, and in the inverse order at replica A2. Depending on the internal algorithms in FB A, the processing of the input data/events in the reverse order may lead to the replicas A1 and A2 becoming de-synchronised.

To maintain all replicas with the same internal state (i.e., replica synchronism) we must therefore guarantee that all messages are processed in the same order in all replicas, and that the algorithms executed by all replicated FBs must be deterministic (for example, may not depend on reading a random value). To achieve this we have implemented the timed messages protocol [4] on the communication SIFBs for all scenarios 2, 4 and 5.

The publish FBs all add an extra time-stamp to every message that is sent, representing the time at which the message should be accepted and processed by the subscriber. This time-stamp is determined by adding a fixed offset to the time at which the message is being sent. The fixed (per publisher) offset is determined off-line as a ceiling of the worst case transmission time of the message to each replica.

When a scenario 2 subscriber FB receives a message it stores the message until the current time matches the time-stamp of the message itself, at which time the message is processed and events/data are sent onwards. Messages arriving at a time later than their time-stamp are simply discarded. Assuming that all execution devices maintain synchronised clocks, we can guarantee that all messages are processed in the same order. For messages that happen to have the exact same

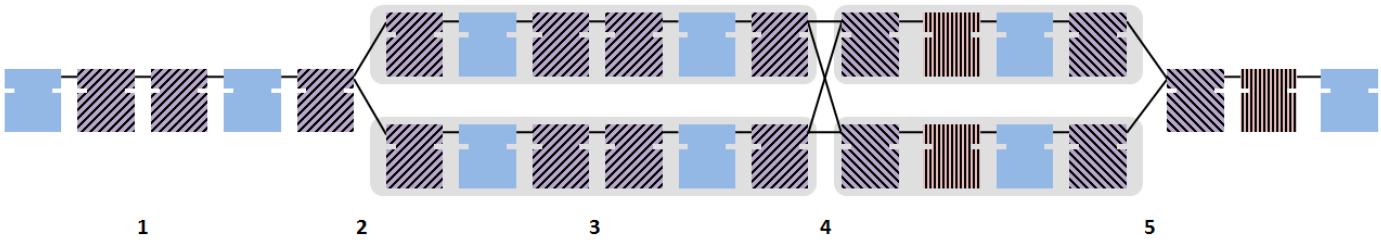

Fig. 2. Communication SIFBs and voting FBs used in each Data/Event passing scenario.

time-stamp, a second parameter is used to break the tie (in our case, the network address of the sender).

Similarly, when a scenario 5 subscriber receives a message, it will store that message until the clock reaches the value in the time-stamp. At this time all messages with the same time-stamp are collected, and their outputs placed on the FBs output to be voted upon by the subsequent voting FB.

The pairing of received messages based on time-stamps will only work if both replicas send the message with the exact same time-stamp. For this reason, publishers of scenario 5 cannot use the local time to generate the time-stamp as there is no guarantee that both replicas will execute at the exact same instant. In this case the time-stamp is determined by adding a fixed offset to the time-stamp of the message that arrived at the subscriber FB at the beginning of the FB network, and that started the execution of the event sequence. This source event may be substantially distant from the publishing FB, and may have gone through several intermediary FBs (scenario 4 in Fig. 1). Passing the time-stamp information along with the internal events therefore requires an extension to IEC 61499.

IV. Tolerated Fault Models

Assuming that no faults occur on the communication networks, the fault models and the number of faults that can be tolerated will depend on the voting algorithms in use.

When using non-majority voters that generate an output event after receiving a single valid message (i.e., that is received before the valid time expires), then only fail-stop faults in the execution devices are tolerated. This is the situation where a faulty device simply stops executing and therefore also abstains from sending any new requests. With this voting algorithm more complex fault models cannot be masked. Consider for example the failure mode in which an execution device stops receiving all messages, but is still able to send out messages. A replica running on this execution device may become de-synchronised with all other replicas as soon as the remaining replicas receive an input message. The de-synchronised replica may however still generate messages if the replicated algorithm contains a local source of events (e.g., periodic timer). In this case the voting algorithm will receive a single message from this failed replica, which will be incorrectly passed onwards as if it were correct.

When voting uses a majority based algorithm then failed execution devices do not need to become silent - any messages generated by failed devices will be ignored by the voter as long as they are a minority.

Note that execution devices may also control physical outputs to which actuators are connected. In order to tolerate faults in the execution devices each actuator must be controllable by more than one execution device, in which case several physical outputs (one from each execution device) will be connected to the same actuator. These outputs must also be voted upon, following whatever voting algorithm may be considered appropriate for the application. Reliable voting algorithms (e.g., at least one) can easily be built using relays in series or parallel. Majority voting based on relays requires circuits that grow exponentially with the number of replicas.

In summary, when using n replicas, the number of faults that may be tolerated may be n-1 when using the fail-stop failure model together with 1 out of n voting. When using more permissive fault models and majority voting, then the number of faults that can be tolerated reduces to floor((n-1)/2).

V. Augmented Communication Protocol

The proposed framework is also sufficient to tolerate faults in the communication network itself – in fact, many network faults are equivalent to a fault in the execution device affected by the network fault. For example, when a node becomes permanently disconnected from the network it will behave as if it had a fail-stop fault. An intermittent fault in the network that results in a replica not receiving a message will make this replica's internal state become de-synchronised, and therefore all output messages will be voted out by subsequent voters in the case that majority voting is being used.

However, in this last case it is possible to augment the communication protocol so that incorrect internal state is detected by the replica itself. This is done by adding a unique identifier to each message based on a counter that is incremented for each consecutive message sent by each FB. When a message arrives at a replica, that replica will be able to independently determine whether any previous messages have failed to be delivered to itself, and in this case will take itself out of commission (basically stop sending output messages).

If this mechanism is used, and if the replicated FB only sends out messages (event or data) as a consequence of receiving an input message (event or data), then the replica will behave according to the fail-stop model. Notice that if the replicated FB contains an internal event source, or if it handles input events from physically connected inputs, then the previous conditions are not met. For example, if a FB contains an internal source of periodic events, then it may generate output events without having received any input message. In this case one replica may miss the receipt of a message, and before it can detect this failure (by the receipt of the subsequent message skipping an identifier) the internal event source fires and an erroneous output message is sent. The same argument applies if the event source comes from a physically connected input – the physical inputs may be viewed as an out-of-band communication channel between the replicas.

If the above conditions are met, using this mechanism together with the use of the simpler 'one out of n' voting, a higher number of faults may be tolerated using the same number of replicas. Note however that the higher number of tolerated faults is in the network itself, and not in the execution devices – the execution devices must be working correctly to detect that it has reached an inconsistent internal state, and therefore stop sending output messages.

VI. Implementation - Test Application

We have implemented this framework on FORTE, an open source IEC 61499 execution environment which runs as a virtual machine executing IEC 61499 applications. Each instance of this virtual machine is therefore an execution device from the point of view of an IEC 61499 application, and in real-world use are expected to run on distinct

computing hardware. FORTE is developed in conjunction with 4DIAC, a graphical development environment for IEC 61499 applications.

Our sample replication framework has been developed as an extension of the communication SIFBs in FORTE. Since FORTE implements the communication SIFBs using a layered architecture, what we actually implemented were two new layers – one for one-to-many replication, and the other for many-to-one. We have also augmented the code in FORTE that handles the transmission of events inside the FORTE virtual machine, so that the time-stamp is implicitly transferred from the first subscriber in a replica, to the final publisher of that same replica. As explained previously, this is required so that all messages arriving at the many-to-one subscriber can be grouped together.

Some simple tests were run based on the trivial XPlus3 sample application that comes with the 4DIAC development environment. The objective of these tests is only to make an an initial validation of the implementation. The original sample application (read input, add 3, print result) simultaneously uses the FORTE and FBDK [3] execution environments. Using the new replication layers a trivial replicated version of this application has been tested, where the Add FB is now replicated. This trivial replicated application was successfully tested with all device instances running on the same computer. Device failures were tested by simply stopping and starting each of the devices running one of the replicas. 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.

Future tests of the replication framework will be run using conveyors whose I/O is accessible over Modbus/TCP. The Modbus servers in the conveyors will be used as voters for the physical outputs.

VII. 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 - [5] 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 [6-7]. Other somewhat related work focuses on online reconfiguration of IEC 61499 control applications, while guaranteeing the control application's real-time requirements [8-10].

VIII. Conclusions

As was stated in the introduction, our work focuses on providing support for tolerating partial failures of the execution control devices. At first we only considered that the execution devices would follow the fault-stop fault model. In this paper we explore the changes required to the framework to support the fault-silent fault model in the execution devices, as well as intermittent failures of the communication network itself. We do not yet consider how faulty execution devices that have been repaired may be brought back into the running application without rebooting the application itself, as this requires that these devices be somehow reloaded with the current internal state of a working replica. Although we have not yet focused our attention to this issue, we currently believe that the result of previous work regarding the on-line reconfiguration of IEC 61499 applications is likely to be applicable to this scenario.

In this paper we have also provided an updated version of the IEC 61499 framework, that supports the existence of several distinct voting algorithms from which the application designer may choose from.

Acknowledgements

This work is financed by the ERDF-European Regional Development Fund through the COMPETE Programme and by National Funds through the FCT-Portuguese Foundation for Science and Technology within project FCT EXPL/EEI-AUT/2538/2013.

References

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

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

[3] FORTE Homepage: http://www.fordiac.org/8.0.html (accessed April 20, 2015).

[4] 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.

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

[6] G. Cengic, O. Ljungkratz, and 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 '06), Prague, September 2006, pp. 1269-1276.

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

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

[9] 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.

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