Towards a New Methodology for Design, Modelling, and Verification of Reconfigurable Distributed Control Systems based on a New Extension to the IEC 61499 Standard

Dissertation
zur Erlangung des Grades des Doktors der Ingenieurwissenschaften der Naturwissenschaftlich-Technischen Fakultät der Universität des Saarlandes

Und

Tunisia Polytechnic School,

Carthage University

Von

Safa Guellouz Ep Addad

Saarbrücken 2021

Tag des Kolloquiums:	09.09.2021
Dekan:	Univ.-Prof. Dr. rer. nat. Jörn Walter
Berichterstatter:	Prof. Dr.-Ing. Georg Frey
	Prof. Dr. Alois Zoitl
	Prof. Dr. Mohamed Khalgui
Vorsitz:	Prof. Dr. Dietrich Klakow
Akad. Mitarbeiter:	Dr. -Ing. Paul Motzki
Weitere Mitglieder:	Prof. Dr. Joseph Haggege
	Dr. Adel Benzina Mannouba

ABSTRACT

In order to meet user requirements and system environment changes, reconfigurable control systems must dynamically adapt their structure and behavior without disrupting system operation. IEC 61499 standard provides limited support for the design and verification of such systems. In fact, handling different reconfiguration scenarios at runtime is difficult since IEC 61499 function blocks cannot be changed at run-time. Hence, this thesis promotes an IEC 61499 extension called reconfigurable function block (RFB) that increases design readability and smoothly switches to the most appropriate behaviour when a reconfiguration event occurs. In order to ensure system feasibility after reconfiguration, in addition to the qualitative verification, quantitative verification based on probabilistic model checking is addressed in a new RFBA approach. The latter aims to transform the designed RFB model automatically into a generalised reconfigurable timed net condition/event system model (GRTNCES) using a newly developed environment called RFBTool. The GR-TNCES fits well with RFB and preserves its semantic. Using a probabilistic model checker PRISM, the generated GR-TNCES model is checked using defined properties specified in computation tree logic. As a result, an evaluation of system performance and an estimation of reconfiguration risks are obtained. The RFBA methodology is applied on a distributed power system case study.

Keywords: *Reconfigurable control system*, *Reconfiguration*, *IEC 61499 standard*, *Reconfigurable Function Blocks*, *GR-TNCES*, *Probabilistic Model checking*, *qualitative and quantitative verification*, *PRISM*.

KURZFASSUNG

Dynamische Anforderungen und Umgebungen erfordern rekonfigurierbare Anlagen und Steuerungssysteme. Rekonfiguration ermöglicht es einem System, seine Struktur und sein Verhalten an interne oder externe Änderungen anzupassen. Die Norm IEC 61499 wurde entwickelt, um (verteilte) Steuerungssysteme auf Basis von Funktionsbausteinen zu entwickeln. Sie bietet jedoch wenig Unterstützung für Entwurf und Verifikation. Die Tatsache, dass eine Rekonfiguration das System-Ausführungsmodell verändert, erschwert die Entwicklung in IEC 61499 zusätzlich. Daher schlägt diese Dissertation rekonfigurierbare Funktionsbausteine (RFBs) als Erweiterung der Norm vor. Ein RFB verarbeitet über einen Master-Slave-Automaten Rekonfigurationsereignisse und löst das entsprechende Verhalten aus. Diese Hierarchie trennt das Rekonfigurationsmodell vom Steuerungsmodell und vereinfacht so den Entwurf. Die Funktionalität des Entwurfs muss verifiziert werden, damit die Ausführbarkeit des Systems nach einer Rekonfiguration gewährleistet ist. Hierzu wird das entworfene RFB-Modell automatisch in ein generalised reconfigurable timed net condition/event system übersetzt. Dieses wird mit dem Model-Checker PRISM auf qualitative und quantitative Eigenschaften überprüft. Somit wird eine Bewertung der Systemperformanz und eine Einschätzung der Rekonfigurationsrisiken erreicht. Die RFB-Methodik wurde in einem Softwarewerkzeug umgesetzt und in einer Fallstudie auf ein dezentrales Stromnetz angewendet.

Schlüsselworte: *Rekonfigurierbare Steuerungssysteme*, *Rekonfiguration*, *IEC 61499 Norm*, *RFB*, *Probabilistic Model checker*, *qualitative und quantitative Analyse*, *PRISM*.

Résumé

Résumé La personnalisation des produits et l'évolution continue du marché ont conduit au développement de systèmes de contrôle reconfigurables distribués (RDCSs). La reconfiguration permet au système d'adapter sa structure et son comportement suite à des changements internes ou externes. La norme IEC 61499 a été développée pour concevoir des applications de contrôle logiciel basées sur des blocs fonctionnels. Toutefois, la IEC 61499 fournit une assistance limitée pour la conception et la vérification des RDCSs. En particulier, le fait qu'un scénario de reconfiguration modifie le modèle d'exécution du système rend la gestion de plusieurs scénarios de reconfiguration assez lourde à ce niveau. Un bloc fonctionnel a également une définition statique qui ne peut pas être modifiée au moment de l'exécution. Par conséquent, cette thèse promeut une extension IEC 61499 appelée bloc fonctionnel reconfigurable (RFB) pour simplifier la conception RDCS et améliorer la reconfiguration dans la norme. Un RFB traite les événements de reconfiguration et déclenche le comportement approprié facilement en utilisant une machine d'état maître-esclave intelligente. Cette hiérarchie sépare le modèle de reconfiguration du modèle de contrôle et augmente ainsi la lisibilité de la conception. La vérification qualitative et quantitative des RDCSs conformément à la norme IEC 61499 est considérée également comme une tâche difficile, car la faisabilité du système n'est pas toujours garantie après la reconfiguration. Une approche RFBA pour la conception, la modélisation et la vérification des RDCSs est proposée. Elle vise à transformer automatiquement le modèle RFB conçu en un modèle généralisé de réseau reconfigurable temporisé de condition/événement (GR-TNCES) en utilisant un nouvel environnement appelé RFBTool. La classe de réseau de Petri utilisée est une extension reconfigurable qui s'adapte bien à RFB et en préserve la sémantique. Le modèle GR-TNCES généré est vérifié en utilisant des propriétés définies qui sont spécifiées en une logique temporelle arborescente. Un vérificateur de modèle probabiliste PRISM (model checker) est également utilisé pour vérifier les propriétés qualitatives et quantitatives. Par conséquent, une évaluation des performances du système et une estimation des risques de reconfiguration sont fournies. La méthodologie est appliquée à une étude de cas sur un système à alimentation répartie.
Mots clés: *Systèmes de contrôle reconfigurables distribués*, *Reconfiguration*, *norme IEC 61499*, *Bloc fonctionnel reconfigurable*, *modélisation*, *GR-TNCES*, *Model checker probabiliste*, *vérification qualitative et quantitative*, *PRISM*.

ACKNOWLEDGEMENT

With immense pleasure and deep sense of gratitude, I wish to express my sincere thanks to my supervisor **Prof. Mohamed Khalgui** at Carthage University for his motivation and continuous encouragement in the course of the elaboration of my PhD thesis.

I am also very grateful to my supervisor **Prof. Georg Frey** at Saarland University for his support and motivation. Moreover, I wish to acknowledge him for providing me the technical environment to achieve my goals.

I would like to thank **Dr. Adel Benzina** for his cooperation, important comments and advice during my research and especially during the review of my publications.

Place: Korschenbroich

Date: 01.09.2021 **Safa Guellouz Ep Addad**

DEDICATION

I wish to express my profound sense of gratitude to my mother, **Souad Trifi**, and my father, **Abdelfattah Guellouz**, for all the sacrifice and moral support made during my research.

I also wish to acknowledge the overwhelming presence of my husband, **Firas Addad** during my research in Germany. His constant encouragement and understanding have boosted my self-esteem.

I wish to express my gratitude to my sisters **Marwa, Dorra, Zouhour and Nourhene** and my brother **Houssem** for their support. I would like to acknowledge the help and advice of my friends and colleagues in the lab **Wafa lakhdher, Asma ben Ahmed, Dr. Fethi Belkhir, Aicha Goubaa** and **Soumoud Fkaier**.

I wish to thank also my second family: my father-in-law, **Mohamed Naceur Addad**, my mother-in-law **Asma Snani**, and my sisters-in-law, **Nada** and **Farah**, for their support.

Place: Korschenbroich

Date: 01.09.2021 **Safa Guellouz Ep Addad**

Contents

ABSTRACT i
KURZFASSUNG ii
Résumé iii
ACKNOWLEDGEMENT v
DEDICATION vii
LIST OF FIGURES xiii
LIST OF TABLES xvii
LIST OF ABBREVIATIONS xix

1 General Introduction **1**
1.1 General Context 1
1.2 Thesis Challenges 5
1.3 Contributions 8
1.4 Publications 11
1.5 Collaboration 12
1.6 Outline 12

2 State of the Art **13**
2.1 Introduction 13
2.2 IEC 61499 Standard 13
2.2.1 Concepts 13
2.2.2 Basic Function Block BFB 14
2.2.3 Service Interface Function Block SIFB 18
2.2.4 Composite Function Block CFB 20
2.2.5 IEC 61499 Models 21
2.2.6 Software Tools and Run-time Environment 24
2.2.7 Hierarchical and Concurrent Execution Control Chart HCECC . . . 25

2.3 Reconfiguration Evolution . . . 25
2.3.1 Reconfiguration Types . . . 26
2.3.2 Reconfigurable Control Systems . . . 26
2.3.3 IEC 61499 Reconfiguration Related Works . . . 27
2.4 Petri Nets for Formal Modelling . . . 29
2.4.1 Net Condition/Event System NCES . . . 30
2.4.2 Timed Net Condition/Event System TNCES . . . 32
2.4.3 Reconfigurable Timed Net Condition/Event System R-TNCES . . . 33
2.4.4 Generalised Reconfigurable Timed Net Condition/Event systems GR-TNCES . . . 34
2.5 Verification of Reconfigurable Systems . . . 37
2.5.1 Verification of IEC 61499 Architecture . . . 38
2.5.2 Temporal logic . . . 39
2.5.3 PRISM Model Checker . . . 40
2.6 Discussion and Originalities . . . 41
2.7 Conclusion . . . 43

3 New Extension to IEC 61499 based on Reconfigurable Function Block 45
3.1 Introduction . . . 45
3.2 Motivation . . . 45
3.3 Reconfigurable Function Block RFB . . . 46
3.3.1 RFB Interface Formalisation . . . 47
3.3.2 ECC Master-Slave Architecture . . . 48
3.3.3 RFB Functionality . . . 50
3.3.4 Execution Semantic and Event Consumption Policy . . . 53
3.4 Reconfigurable System . . . 54
3.5 Formal Case Study RFB model . . . 56
3.6 Dynamic RFB reconfiguration . . . 59
3.6.1 Adding a New Reconfiguration Scenario . . . 60
3.6.2 Removing an Existing Reconfiguration Scenario . . . 60
3.6.3 Modifying an Existing Reconfiguration Scenario . . . 61
3.7 Comparison of RFB with BFB . . . 61
3.8 Discussion and Originalities . . . 63

3.9 Conclusion 64

4 Formal Modelling of Reconfigurable Distributed Control Systems based on RFBs 65

4.1 Introduction 65
4.2 Motivation 65
4.3 RFBA Approach 66
4.4 Formal Modelling 67
4.4.1 Transformation Rules 69
4.4.2 Formal Example 75
4.5 Formal Verification 76
4.6 RFBTool 79
4.7 ZiZo V2 84
4.8 Discussion and Originalities 85
4.9 Conclusion 86

5 Case Study and Performance Evaluation 87

5.1 Introduction 87
5.2 Case Study Presentation 87
5.3 System Specification 89
5.4 RFBs-based Architecture 89
5.5 Automatic Transformation into a GR-TNCES Model 92
5.6 Formal Verification 94
5.6.1 Qualitative Verification 95
5.6.2 Quantitative Verification 97
5.6.3 PRISM Simulation 100
5.6.4 System Availability 101
5.7 Evaluation of Performance 102
5.8 Conclusion 105

6 General Conclusion 107

6.1 Context and Challenges 107
6.2 Contributions and Originalities 108
6.3 Perspectives 109

REFERENCES 110

List of Figures

1.1 Evolution of Manufacturing Paradigms [Mourtzis and Doukas, 2012]. . . 2
1.2 RFBA Approach for RDCS design, modelling and verification. 9
2.1 Basic Function Block (a) interface and (b) its ECC. 15
2.2 Pick and Place System. 16
2.3 Belt Conveyor Function Block Interface. 17
2.4 Conveyor Execution Control Chart. 17
2.5 SIFB interface example and its sequence diagram. 18
2.6 The state chart of a managed function block [TC 65/SC 65B, 2012a]. . . . 20
2.7 Composite Function Block. 21
2.8 IEC 61499 System and Device Model. 22
2.9 IEC 61499 Resource Model. 23
2.10 HCECC examples. 25
(a) Parallel HCECC. 25
(b) Refined HCECC. 25
2.11 Graphical Notation of an NCES Module. 30
2.12 NCES Module for a Conveyor Controller. 32
2.13 An R-TNCES Example Model. 35
2.14 A GR-TNCES Example Model. 37
2.15 Prism DTMC model. 42
3.1 A Reconfigurable Function Block Interface. 47
3.2 Master-Slave Hierarchy. 49
3.3 RFBs Network (R_1 and R_2). 52
3.4 Reconfigurable Function Block Network showing the possible paths that can be generated after reconfiguration. 56
3.5 PBROS application model based on RFBs. 57
3.6 The internal architecture of $PBROS$ RFB. 58

3.7 Reconfiguration scenarios Management. 59
3.8 Conveyor Model using RFBs. 61
3.9 Comparison Between RFB and BFB Networks. 62
3.10 Execution Scheduling Comparison. 63
4.1 Detailed RFBA Methodology Diagram. 68
4.2 RFB interface transformed into a GR-TNCES module 69
4.3 TNCES Model for Input Events Models. 71
4.4 TNCES Model of Output Events. 71
4.5 TNCES Model for Input and Output Data of Reconfiguration. 71
4.6 GR-TNCES Model of an ECC Master. 72
4.7 GR-TNCES Model of an ECC Slave. 74
4.8 A GR-TNCES module controlling memory and energy resources. 75
4.9 A system modelled with RFB and FBs. 76
4.10 GR-TNCES Model for the Formal System. 77
4.11 A generated XML Code for an RFB Example. 80
4.12 Class Diagram showing the Architecture of RFBTool. 81
4.13 “Create RFB” Sequence diagram. 82
4.14 Sequence diagram showing the RFB execution semantic. 83
4.15 RFBs Network as shown in RFBTool. 84
4.16 The Generated GR-TNCES model as shown in ZiZo v2 after exporting the RFB model from RFBTool. 85
5.1 Case Study Topology. 88
5.2 Subsystem B modelled with an RFB Network. 90
5.3 $PIOC0$ Interface and its ECC master. 91
5.4 ECC slave es_{11} $FaultOperation$. 91
5.5 $CSWI$ Interface and its slaves. 92
(a) $CSWI$ Interface 92
(b) $FaultOperation$ slave 92
(c) $NormalOperation$ slave 92
5.6 Successful Export to ZiZo v2 tool. 92
5.7 GR-TNCES Model of the $PIOC0$ ECC master. 94
5.8 PRISM Model Reconfiguration Module. 95

5.9 Property of deadlock freedom using PRISM 4.6 96
5.10 Simultaneous Scenarios Execution Property. 96
5.11 Steady state probability in the case of one, two, three and four fault/s. . . 99
(a) Case of single fault 99
(b) Case of two simultaneous faults 99
(c) Case of three successive Faults 99
(d) Case of four successive faults 99
5.12 Fault simulation using PRISM. 101
5.13 The availability of the FLISR system during 8 days. 102

List of Tables

3.1 Reconfiguration matrix of the RFB R_1. 52
3.2 Reconfiguration matrix of the RFB R_2. 52
3.3 Reconfiguration Matrix of $Recognition$ RFB. 58
3.4 Comparison between the RFB model and the BFBs network model. . . . 62
5.1 Components Size in the Generated GR-TNCES. 93
5.2 Steady-state properties. 98
5.3 The probability of failed reconfiguration properties. 100
5.4 Comparative study. 103
5.5 Comparative study about verification. 104

LIST OF ABBREVIATIONS

CFB Composite Function Block.

CTL Computation Tree Logic.

DES Discrete Event System.

DECS Discrete Event Control System.

DCTMC Discrete-Time Markov Chains.

ECC Execution Control Chart.

FB FB Function Block.

FBN Function Block Network.

GR-TNCES Generalized Reconfigurable Timed Net Condition/Event system.

HCECC Hierarchical and Concurrent Execution control chart

IE Input Event.

IEC 61131-3 Standard for Programmable Controllers.

IEC 61499 International Electrotechnical Commission standard for IPMCS.

IPMCS Industrial Process Measurement and Control Systems.

I/O Input/Output(s).

MSECC Master-Slave Execution Control Chart.

NCES Net Condition/Event System.

OE Output Event.

PCTL Probabilistic Computation Tree Logic.

PLC Programmable Logic Controllers.

PRISM Probabilistic Symbolic Model Checker.

RDCS Reconfigurable Distributed Control System.

RDECS Reconfigurable Discrete Event Control System.

RFB Reconfigurable Function Block.

RFBA Reconfigurable Function Block Approach.

R-TNCES Reconfigurable Timed Net Condition Event System.

SIFB Service Interface Function Block.

Chapter 1

General Introduction

This introductory chapter aims at presenting the control systems and their historical evolution. It also focuses on the appearance of IEC 61499 standard. The latter is highlighted to develop and implement software components for distributed, embedded, and control systems. We have addressed the challenges of this standard and proposed an RFBA approach based on IEC 61499 extension to mitigate design complexity. Then, we have worked on ameliorating the reconfiguration feature in the standard and verifying the designed model automatically. Finally, we have presented our publication list and thesis outline.

1.1 General Context

Reconfiguration has recently gained a wide attention. It has been applied in control systems to customise rapidly variable industrial needs and follow market fluctuation. The control systems, such as robots, aircraft systems, and smart grids, result from technological advances and deep research in mechatronics, electric/electronic engineering, and computer science (software and communication protocols) fields. Such accurate systems require advanced software control and new technologies to supervise complex processes and devices. They are influenced by several manufacturing paradigms. It is therefore reasonable to present the historical evolution of control systems in the automation industry. Since 1850, several manufacturing paradigms have been developed to match each period's special needs, as shown in Fig. 1.1 [Mourtzis and Doukas, 2012]. Craft production is the first paradigm where a very limited product's quantity is produced using general-purpose machine tools. After that, the mass production paradigm appears to increase product availability and decrease product cost. It offers a large scale manufacturing using several assembly lines. Most manufacturing systems are based on a centralised controller box built using hardwired relays for control logic. In order to extend a system, separate relay circuits have been required. Starting from 1980, the customer's preferences and needs have become the focus of industrial providers. The high demands of product variety have motivated the development of the mass customisation paradigm. Hardwired

relays are replaced by software codes implemented in general-purpose computing devices such as micro-controllers, or special-purpose devices such as programmable logic controllers (PLCs) [Vyatkin, 2016]. Reconfigurable Manufacturing Systems (RMS) are emerging to offer easy product changes and produce a large product variety. Since the start of the last decade (2010), the mass personalisation paradigm has been hugely applied. Manufacturers and providers intend to respond to maximum individual needs and consumer desires [Hu, 2013]. Accordingly, the distributed control systems (DCSs), i.e. distributed hardware and software modules interconnected via a network, are increasingly used in mass customisation and personalisation paradigms. Their modular and flexible architecture as well as the communication layer allow exchanging data and state information between processors and devices. Since 2014, more and more customisation has been needed. This has led to integrating computing, networking and physical processes together and hence the emergence of Cyber-Physical Systems (CPS) [Terzimehic et al., 2017, Batchkova et al., 2020]. CPSs are networked embedded systems that have a virtual representation of their physical entity, combining computation, control and communication [Zoitl and Strasser, 2017]. It includes adaptive and predictive distributed nodes [Dai et al., 2015] and aims to increase flexibility and reconfiguration in distributed control systems. Notably, flexibility allows continuous changes in product design and business strategy [Brennan et al., 2008], while reconfiguration involves every change in the software or hardware system intending to adjust the system behaviour according to environment changes and user requirements. Advanced and standardised software control is required to endow the physical system with not only more flexibility and autonomy but also intelligence and reconfiguration.

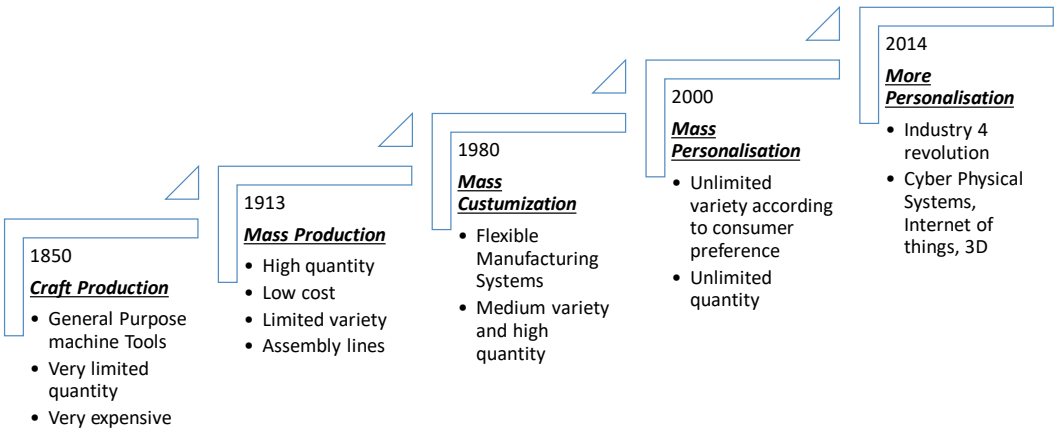


Figure 1.1 Evolution of Manufacturing Paradigms [Mourtzis and Doukas, 2012].

Control software occupies an essential position in the product life cycle and an increasing cost portion (around 50%) in the automation systems development [Kan, 2012, Vyatkin, 2013]. It provides autonomy and intelligence to devices. The reusability of

software components, applications and sub-systems is a crucial need since it reduces the development time and engineering effort. In the literature, two fundamental standards have been developed to implement such software: IEC 61131-3 and IEC 61499. IEC 61131-3 is the standard for programmable logic controllers PLCs. It defines a basic software structure based on low-level languages such as ladder diagram and instruction list. The reuse of developed control software components is not easily applicable due to some details in vendor specifications. Moreover, IEC 61131-3 standard depends mostly on the platform and is designed basically for centralised control systems [Dai and Vyatkin, 2012]. A centralised control system faces several challenges such as its no suitability for wide systems and its strong dependency. A breakdown in any part of the system implies a central controller's failure [Dai and Vyatkin, 2009]. Therefore, IEC 61131 does not address the new requirements of complex industrial systems like portability and distribution. An abstraction in the design model is crucial for reusing existing software elements and communicating remote services, devices, and controllers from different vendors together. A high-level oriented programming language is demanded for the development of complex industrial applications. IEC 61499 standard emerged to overcome the shortcomings of the previous standard IEC 61131-3 [Dai and Vyatkin, 2009] and to extend it. The first edition was published in 2005 and the second edition in 2011, aiming at designing and developing industrial process measurement and control systems (IPMCSs). These systems are defined as a set of interconnected devices that can communicate together via a communication network in order to control a process.

The IEC 61499 standard is an application of the Holonic theory [Leitão et al., 2012], [Kruger and Basson, 2013], [TC 65/SC 65B, 2012a] where a holon is an autonomous and cooperative building block that is able to transfer, convert and validate some information or physical object. A holon can take a decision and communicate with higher control levels since it has a functional component represented by a software entity, and a communication/cooperation component [Leitão et al., 2012]. The international electrotechnical commission defined the IEC 61499 standard as an approach based on software components that can be instantiated through a defined interface in a distributed system. The component is called function block FB that can model a mechatronic component or a service to ensure reusability during the design and development process. The standard incorporates particular features such as encapsulation of algorithms and data in FB units, component-based design, event-driven execution, and distribution. As a result, specific implementations available from different providers of field devices, controller hardware, human-machine interfaces, communication networks, can be integrated into component-based heterogeneous systems.

IEC 61499 is based on event Model-Driven Development (MDD) [Thramboulidis and Frey, 2011], [Vyatkin, 2003]. This paradigm defines the function block as an event-

triggered software unit where an event is an abstraction of an instantaneous happening that does not have any duration. The FB is enabled when an input event occurs. MDD substitutes the traditional cyclic computation framework applied in IEC 61131 based systems where a scan refers to a time-triggered control [Vyatkin, 2003]. The most important features provided by the IEC 61499 standard compared to the previous standard IEC 61131-3 [Thramboulidis and Frey, 2011] are interoperability between different devices, software portability, and configurability [Pang et al., 2014a, Hopsu et al., 2019] where:

(i) Interoperability is the fact that functional units from distinct vendors in a system can operate together to perform the required functionality [Jhunjhunwala et al., 2020, Dai et al., 2015],

(ii) Portability is the fact that a software or library element can be accepted and correctly interpreted by other software tools. Portability is supported in IEC 61499 by Extensible Markup Language (XML) [Arenas et al., 2002],

(iii) and configurability is the fact that a system, such as PLCs and software components, can be configured by tools of other vendors. IEC 61499 supports configurability by selecting functional units and assigning locations and parameters to them [Christensen et al., 2012b].

IEC 61499 suits perfectly the mass personalisation and future requirements. It meets the challenges of modern industrial enterprise [Lyu and Brennan, 2020] and it is applied in cyber-physical systems [Yan and Vyatkin, 2011, Yoong et al., 2013, Malik et al., 2017, Batchkova et al., 2020] and service-oriented architecture [Dai et al., 2015, Demin et al., 2015, Homay et al., 2019]. It provides an important degree of flexibility needed to extend and change existing systems. Such flexibility aim at producing competitive and innovative products in a short time. Interestingly, IEC 61499 has succeeded in promoting software control in many academic and industrial applications. Some examples of industrial control systems implemented based on IEC 61499 standard are Eco-Friendly heating control systems [Pang et al., 2014b], conveyor belt control system, smart distributed power system automation [Mousavi and Vyatkin, 2015, Zhabelova et al., 2015], baggage handling system [Sinha et al., 2015, Black and Vyatkin, 2010], intelligent manufacturing Systems [Jakovljevic et al., 2017], aerospace control systems [Insaurralde, 2016], robotic medical systems [Sorouri et al., 2013, Guellouz et al., 2016b]. However, with increasing customised demand and personalisation in industry, traditional control systems do not efficiently satisfy the newly imposed requirements such as variable demand, minimum energy waste, minimum time execution, minimum cost, maximum energy efficiency, real-time execution, etc. Reconfigurability becomes a crucial need to switch from a system configuration to another or change immediately the system strategy according

to the new requirements. It guarantees a qualitative change in the structure, functionality and algorithm of the control systems. However, reconfiguration increases the design complexity of distributed systems due to the need for handling different reconfiguration scenarios.

1.2 Thesis Challenges

Reconfigurable distributed control systems (RDCSs) can be designed in IEC 61499 as a set of control applications, i.e. software applications that are distributed and configured on multiple devices and resources interconnected via a network. It is responsible for controlling system processes and system changes. A control application in IEC 61499 is a network of function blocks that can be distributed in several devices from heterogeneous providers. The main component in an application is the basic function block. Its control logic is defined through a finite state machine called execution control chart (ECC). It simplifies the representation of application logic and reduces the complexity of algorithms [Christensen et al., 2012b]. Despite its proven usefulness in many applications, the execution control chart remains complex and not flexible when designing reconfigurable distributed systems. The encapsulated ECC has a static structure. It uses predefined sets of inputs and outputs which cannot be changed or configured without reprogramming them [Angelov et al., 2005]. RDCSs must handle all system changes and reconfiguration scenarios. A reconfiguration scenario is any operation that changes the execution model of an existing system depending on the environment, architecture or user requirements changes. The handling of reconfiguration scenarios inside the function block is quite cumbersome on this level since the reconfiguration and control models are combined in just one ECC. For example, error handling, initialisation and reconfiguration are overlapped. There is no separation between control level and high levels within the ECC. This engenders a large number of states and transitions, even a low readability and maintainability. Moreover, it is difficult to extend an ECC or maintain it by the developer due to the high dependency between the states. A hierarchical state machine HCECC is proposed in [Sinha et al., 2015] to model concurrent behaviours. It is a multilevel hierarchical state machine which aims to include concurrent and hierarchical behaviours into basic function blocks. These blocks based on HCECC allows to decrease the system complexity problem, but the authors do not address the reconfiguration problem. In reconfigurable distributed control systems, the execution semantic depends on environmental changes and user requirements. Therefore, a separation between reconfiguration and control logic is required inside the FB. Two hierarchy levels are needed to define reconfiguration scenarios: one level for the reconfiguration and the other for executing the corresponding scenario in a dynamic schedule. Moreover, the function block based on HCECC does not switch directly between the reconfiguration scenarios and does not

decide which suitable scenario corresponds to a particular input event.

Various reconfiguration approaches have been developed in the last years to reduce cost and development time by automatically adapting the system evolution [Duhem et al., 2012]. Most studies related to reconfiguration in IEC 61499 standard have focused on executing reconfiguration to handle disturbance or errors. However, a minority focuses on self-adapted reconfiguration that is required when a system requirement changes. The authors in [Zoitl et al., 2006, Brennan et al., 2008] propose an online reconfiguration approach. Their goal was to reconfigure an existing IEC 61499 system by executing commands, i.e. adding, removing or changing an existing function block instance in the control application to execute changes when the system is running. Despite the benefits of the developed approach, it is not possible to add a new function block type that does not exist before deployment in any existing IEC 61499 development software tools. Additionally, the reconfiguration commands are executed manually by the user to manage FB instances, connections and types, i.e. configure an IEC 614999 system. The user must master the system configuration and then face all the possible reconfiguration risks. Despite the benefits of IEC 61499, reconfigurable distributed control systems (RDCSs) are still facing design complexity challenges. The increasing number of function blocks, events and data interconnection is progressively increasing the size of application and complexity. To address the complexity problem, the authors in [Zhabelova and Vyatkin, 2015] developed a set of metrics for evaluating the quality of IEC 61499 automation software. However, the metric cannot be easily applied since the developer can use IEC 61131-3 languages, object-oriented programming and/or state machines to implement its application.

On the other hand, there is a gap between IEC 61499 standard and verification engineering. RDCSs verification is a fundamental step to detect deadlocks, and check both real-time constraints and concurrent tasks. The dynamic behaviour of such reconfigurable systems makes verification a challenging and complicated task. Importantly, in the existing IEC 61499 compliant tools, simulation is supported, but verification of the control logic is not proved. Several authors model an IEC 61499 system with Petri nets (for example Signal Net System SNS [Vyatkin, 2000], NCES [Vyatkin and Hanisch, 2000a]) to verify it with model checker such as Signal/Event systems analyser (SESA) [Vyatkin and Hanisch, 2000b]. A study in [Mironovich et al., 2017] proposes an automatic generation of function block applications using evolutionary computation verified with UPPAAL. Most of them guarantee the system verification. However, the task is still more difficult for large scale reconfigurable systems because each reconfiguration scenario is designed as a new controller module. This strongly overbids the final system verification and necessitates a long verification time [Zhang, 2015]. A reconfigurable model switching from one configuration to another and minimizing the verification time

is required. Additionally, since most reconfigurable distributed control systems are under time, memory and energy constraints, it is necessary to supervise resources and energy before executing reconfiguration in IEC 61499 systems at run-time. Hence, an adequate formalism that controls these three constraints is highly recommended.

Functional safety is another challenge in the design of IEC 61499 because the standard does not grant any verification process. The system safety, availability and stability can be disturbed by unpredictable events such as faults, environment changes, and architecture changes. The authors in [Bhatti et al., 2017] have combined the qualitative and quantitative analyses in one approach using IEC 61508 standard. Their endeavour was to manage functional safety in software with IEC 61499 standard so that to measure and manage risks. The study shows an exponential analysis time due to the combinatorial state space explosion despite the failure risk estimation. Modular verification of each module is required to reduce the state space explosion problem. Additionally, the detection of the worst cases before deployment is not considered. It is a significant virtue that companies can estimate and avoid unpredictable risks.

Many methodologies have been developed to reduce the software cost, time, and resources during the product life-cycle, including the design, development, testing, deployment, and maintenance process. An agile development methodology is exhibited in the study of [Flora and Chande, 2014]. It is an iterative incremental development approach that allows providing and testing an incremental delivery, i.e. part of the product, in a short time and with a minimal cost. The adaptation to changing business requirements in the agile process is faster than traditional software development approaches that deal with the whole project life cycle [Theocharis et al., 2015]. However, the agile approach is not suitable for large scale projects. In fact, it is more useful for management and planning than for developing reconfigurable distributed control system. In other words, such a methodology is not suitable for reconfigurable embedded systems. This increases the need for an approach for RDCSs design, modelling and verification.

Finally, most tools that comply with the IEC 61499 standard only offer simulation techniques. There is no integral environment that supports the design, development, formal modelling, and verification phases. However, companies and engineers have a crucial need for a unique tool that integrates all the processes and automates them since each process takes much time to be finalized.

To overview, this thesis treats several challenges faced by the standard. These challenges are summarised as follows:

(i) The function block has a static definition that cannot be changed at run-time. This makes a considerable number of function blocks are required to implement all reconfiguration scenarios needed during the execution process. For each change, the existing control software has to be adjusted. Therefore, if basic function blocks

are used to model a reconfigurable control system, the number of events, data, states, and transitions increases the system complexity progressively and permits low readability and maintainability.

(ii) Verification of distributed reconfigurable control systems following IEC 61499 is a challenging task. The reconfiguration process changes the system behaviour and execution model. Consequently, the system feasibility cannot be guaranteed after executing a reconfiguration process. Formal verification is required, but most IEC 61499 compliant development tools offer simulation techniques that prove neither system correctness nor deadlock.

(iii) Most reconfiguration events are unpredictable. A probabilistic reconfigurable formal model is needed. To the best of our knowledge, there is no approach checking the probabilistic reconfiguration events and estimating risks. A combination of qualitative and quantitative analyses of IEC 61499 systems is thus necessary to provide an insight about system performance and an estimation of reconfiguration risks.

(iv) No integrated development environment supports the design, development, formal modelling, and verification of reconfigurable distributed control systems following IEC 61499. This increases the need of companies and engineers for a unique tool that integrates and automates all the processes.

1.3 Contributions

The thesis brings into light a new approach for the design, development and verification of reconfigurable control systems called RFBA, as illustrated in Fig.1.2. Reconfiguration is a required dynamic solution that ensures the system flexibility at run-time, as it allows to adapt the system behaviour to the related environment under different constraints described in the user requirements. IEC 61499 is the best standard in the industry due to its design and implementation of distributed control systems. It provides a rich library allowing the reuse of function blocks in different applications and reducing the development time.

However, the guarantee of the IEC 61499 system feasibility when stochastic reconfiguration events occur seems to be a hard task in reconfigurable systems. To address the IEC 61499 challenges and limitations, we propose to extend the basic function block to support reconfiguration. Reconfigurable function block RFB is a new design pattern that separates the reconfiguration from the control model in the component. While a control model defines the hardware and software behaviours, a reconfiguration model manages unpredictable changes in the related environment to supervise the system behaviour. The

aim of this separation is to add more flexibility, readability and simple maintainability. RFB also intends to enhance the reconfiguration and simplify the design process. It introduces a new "reconfiguration event" having a higher priority than the other events in order to adapt the system under such constraints. Such event is dependent on the external environment such as the weather, temperature, earthquake, storm, or user requirements and internal or external failures. It is characterised by its probability of occurrence. The increasing complexity of reconfigurable distributed control applications and low readability and static behaviour of ECC have all motivated the investigation of a dynamic master-slave state machine called MSECC. The latter represents the RFB control logic. MSECC describes each reconfiguration scenario in a slave state machine. While the slave defines the behaviour of a particular configuration, the master processes the input events of reconfiguration, and triggers the suitable slave smoothly. Each slave ECC is supervised by the master ECC. It can be activated or deactivated according to the coming events. It results in a dynamic execution order in the function blocks network.

The new RFBA approach intends to simplify and improve the life cycle of RDCSs. It also aims at capturing the system specification and determining the reconfiguration scenarios. After that, the RFBs, the reconfiguration events and the reconfiguration data in each RFB will be defined. Hence, a control application is designed using RFBs that are interconnected with other FBs. As a result, the IEC 61499 extended model is generated in XML format.

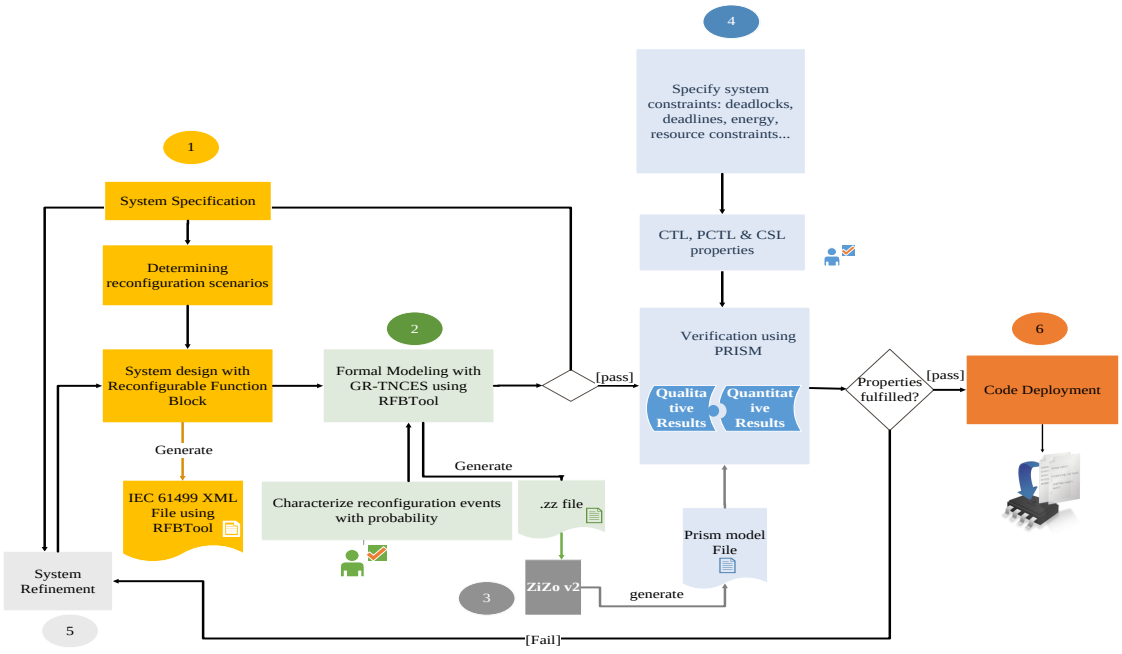


Figure 1.2 RFBA Approach for RDCS design, modelling and verification.

After the design phase, we verify the generated RFB model. In order to guarantee system correctness, formal verification is proposed. We consider the reconfiguration events

in the verification step as probabilistic events since they depend mainly on the occurrence of unpredictable events. With the probability aspect of the reconfiguration event, we can estimate performance and risks. A modular verification is proposed to reduce the total verification time and mitigate the state space explosion problem. It is important to note that in formal verification, Petri nets are an effective method for formal modelling. Modularity, temporal conditions of event occurrence, reconfiguration as well as the ability to associate a probability to reconfiguration events are special features supported by the Generalised Reconfigurable Timed Net Condition/Event systems (GR-TNCS) [Khlifi et al., 2015]. This formalism fits well with reconfigurable distributed control systems and preserves the MSECC execution semantic. Moreover, GR-TNCES allows to model and supervise the energy and resources consumption at run-time. Therefore, an RFB can switch from one reconfiguration scenario to another only after checking its feasibility, memory and energy availability. For verification purposes, an automatic transformation from an RFB based model to a GR-TNCES model is developed, and a set of transformation rules is defined.

After the automatic formal modelling, the system designer characterises the unpredictable events by probabilities of occurrence in GR-TNCES model. The main objective of this modelling is to check the system performance through a model checking. The latter is the most widely used technique for verification. Compared with simulation, it grants an exhaustive search through the design state space [Clarke et al., 2009]. In this work, a unified verification is presented. We attempt to analyse risks in reconfigurable IEC 61499 systems not only qualitatively but also quantitatively using a PRISM model checker. PRISM [Kwiatkowska et al., 2002, 2006] is an effective probabilistic and symbolic model checker based on the construction of a precise mathematical system model. It permits the verification and analysis of properties that are expressed formally with temporal logic. While a computation tree logic (CTL) is used for specifying qualitative properties, probabilistic computation tree logic (PCTL) is used for specifying probabilistic quantitative properties. If the verification is successful and the risk estimation is acceptable, then the code will be deployed in a real system under real-time constraints. If the verification fails, the designer should rectify the original model. To support the RFBA approach, a complete toolchain RFBTool is developed, i.e. called also ”ZiZo v3” in [Guellouz et al., 2019]. RFBTool supports an end-to-end process from the specification to the code deployment. It offers a drag and drop interface to design the new system model based on RFBs and translate it automatically to a GR-TNCES model. It provides an RFB library that supports reconfigurable events and master-slave execution control chart. RFBTool is integrated with ZIZO software second version [Khlifi et al., 2016] to automatically convert a GR-TNCES model, i.e. a “.zz” file, into a Markov chain compatible with PRISM which is advantageous for engineers since they can estimate the limit

of reconfiguration in IEC 61499 model when unpredictable events occur. The model is implemented in C++ language.

The originality of this thesis, compared with existing researches, resides in several aspects, including:

1. Extending IEC 61499 by proposing a reconfigurable function block separating the control model from the reconfiguration model based on master-slave execution control chart. This reconfigurable function block permits an easy switching from a reconfiguration scenario to another and a self-decision of the best reconfiguration scenario,

2. Automatic formal modelling of RFB model with GR-TNCES model that models time, reconfiguration, memory and energy constraints,

3. Adding a probabilistic aspect in the generated GR-TNCES formal model to estimate risks after reconfiguration,

4. Qualitative and quantitative analyses using model checker PRISM to give an insight into system performance,

5. The development of a complete environment for design, modelling and verification of reconfigurable distributed control systems following the new extension to IEC 61499.

1.4 Publications

- Journal Paper

  [Guellouz et al., 2019] S. Guellouz, A. Benzina, M. Khalgui, G. Frey, Z. Li, and V. Vyatkin. Designing efficient reconfigurable control systems using iec61499 and symbolic model checking. IEEE Transactions on Automation Science and Engineering, 10.1109/TASE.2018.2868897 July 2019, **(IF. 5.9) Q1**

- Conference Papers

  [Guellouz et al., 2016b] S. Guellouz, A. Benzina, M. Khalgui, and G. Frey. Reconfigurable function blocks: Extension to the standard IEC 61499. In Proc. 13th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2016), November 2016, Agadir, Morocco, **(Best Paper Award) C ranked conference**.

  [Guellouz et al., 2016a] S. Guellouz, A. Benzina, M. Khalgui, and G. Frey. Zizo: A complete toolchain for the modeling and verification of reconfigurable function blocks. In Proc. 10th International Conference on Mobile Ubiquitous

Computing, Systems, Services and Technologies (UBICOMM), pages 144–151, 2016- Venice, Italy. **(Best Paper Award) C ranked conference**.

1.5 Collaboration

This thesis is carried out at Tunisia Polytechnic School, with the LISI Lab INSAT, Carthage University Tunisia and also with Saarland University Saarbrücken at Automation and Energy System Lab, Germany.

1.6 Outline

Apart from this introductory chapter, this thesis comprises four other chapters structured as follows:

- The second chapter introduces useful preliminaries needed. It defines briefly the concepts used to achieve this research work such as IEC 61499 concept, reconfiguration, Petri net extensions and model checking. It presents and also analyses the related works of the studied topic.
- The third chapter formalizes both the proposed reconfigurable function block framework and its different interactions with other standard function blocks. It highlights the execution semantic of an RFB model and the system dynamic. Furthermore, a formal case study is exposed in order to showcase the originality of the framework.
- The fourth chapter underlines the proposed RFBA methodology for RDCSs. In this chapter, an automatic transformation of an RFB model to a GR-TNCES model is implemented through a well-defined set of transformation rules. At the end of Chapter four, we expose the developed RFBtool environment and its functionalities to support the RFBA approach.
- The fifth chapter presents a case study on a distributed power system to illustrate the RFBA approach with RFBTool and PRISM Model Checker. Therefore, a performance evaluation and a comparative study are carried out.
- Finally, a general conclusion summarises the thesis challenges, highlights its contributions, and opens new perspectives.

Chapter 2

State of the Art

2.1 Introduction

This chapter gives an overview of the IEC 61499 standard for the design and development of reconfigurable distributed control systems. The chapter analyses the researches related to IEC 61499 and reconfiguration as well as presents definitions of the major concepts tackled such as such as formal modelling and model checking technique.

2.2 IEC 61499 Standard

2.2.1 Concepts

International Electrotechnical Commission (IEC) has come with IEC 61499 standard for the design and development of distributed control systems. The latter are characterised by their modularity, flexibility, extensibility, and reconfigurability. The IEC 61499 standard is developed to overcome the incompatibility between different PLC brands and centralised control of the IEC 61131 standard [Dai and Vyatkin, 2009]. The new standard introduces a component architecture based on function blocks concept [Dai and Vyatkin, 2012]. IEC 61499 architecture represents a component solution for distributed industrial automation systems targeting the portability, reusability, interoperability, and reconfiguration of distributed applications. In addition to the programming languages of IEC 61131 (Ladder Diagram, Function Block Diagram, Sequential Function Chart, Instruction List and Structured Text), IEC 61499 provides the option of implementing large cooperating applications on one or more controllers in the network using object-oriented programming (C++, Java). The latter is known for its efficiency and reliability thanks to its abstraction and encapsulation concept.

An IEC 61499 controller not only performs control tasks but also communicates with external intelligent modules via industrial buses and network protocols. It executes both real-time-critical tasks and non-real-time-critical tasks.

IEC community has provided a function block library used to create IEC 61499 com-

pliant control applications using Function Block Development Kit FBDK [HOLOBLOC, 2020], the commercial environment ISaGRAF [Vyatkin and Chouinard, 2008], 4DIAC [Eclipse, 2020], and nxtSTUDIO [nxtControl GmbH, 2021] development tool. The developer can define a new function block type in the library and then freely instantiate it in the application. Instantiation makes the development and maintenance process easier for engineers and reduces effectively the time needed for building a control application. Furthermore, the use of IEC 61499 components does not require a background in automation [Zoitl and Strasser, 2017].

IEC 61499 is based on Function Block (FB) concept that is the core of the standard. A function block is a unit of execution that encapsulates data and algorithms as well as processes signal events. It represents an independent, self-contained entity which can be instantiated to be reused in the application. It hides the internal implementation but shares the interface with other FBs in an application [Wozniak et al., 2018]. It is also an event-triggered component. The function block becomes active when an event occurs. The standard defines three types of function blocks:

- Basic Function Block (BFB),
- Service Interface Function Block (SIFB),
- and finally, Composite Function Block (CFB) that is a network of interconnected BFB or SIFB or other CFB instances.

2.2.2 Basic Function Block BFB

Each basic function block contains an interface and a body. The interface represents events and data flow, while the body describes the entire behaviour of the FB using a Moore-type state machine called execution control chart ECC, as illustrated in Fig. 2.1.

A Basic Function Block f is formalised as a tuple of interface I, a set of internal variables V, a set of algorithms identifiers Alg and a state execution control chart ECC, as defined in [Dubinin and Vyatkin, 2006, Wozniak et al., 2018], where

$$I = (EI, EO, VI, VO, IW, OW), \text{ with}$$

(i) EI (respectively, EO) is a set of event inputs (respectively, outputs);

(ii) VI (respectively, VO) is a set of data inputs (respectively, outputs);

(iii) $IW \subseteq EI \times VI$ associates input events with input data variables (respectively, $OW \subseteq EO \times VO$ associates output events with output data variables). IW (respectively, OW) is a set of WITH-associations for inputs (respectively, outputs).

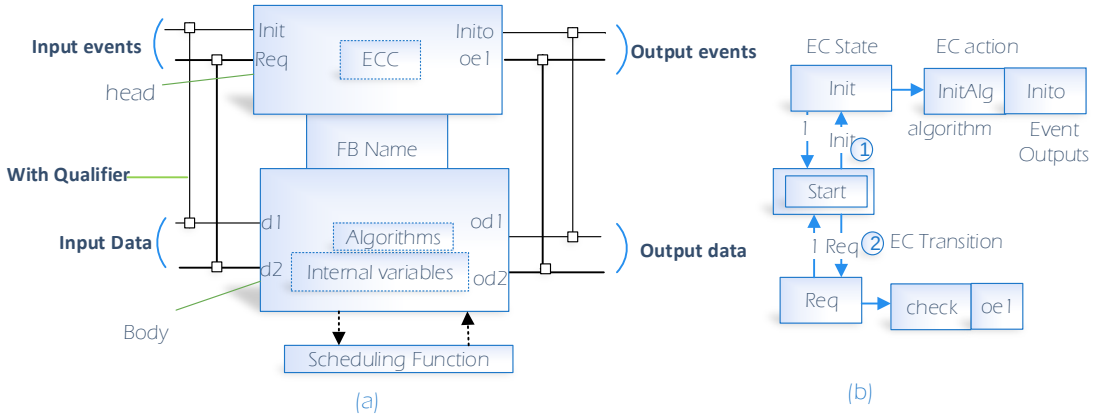


Figure 2.1 Basic Function Block (a) interface and (b) its ECC.

The set of internal local variables V is declared inside a function block algorithm. The encapsulated algorithms Alg can initialise or update internal variables V and output data VO of the function block. Fig. 2.1 depicts an FB example that has input events $\{init, Req\} \in EI$, output events $\{inito, oe1\} \in EO$, input data $\{d1, d2\} \in VI$, output data $\{od1, od2\} \in VO$, and algorithms $\{initAlg, check\} \in Alg$.

The Execution Control Chart ECC is a Moore machine determined by a tuple:

$$ECC = (State, s_0, Transition, Condition, Action, Priority), \text{ where}$$

– $State$ is a set of states,

– s_0 is the initial ECC state that is not assigned to any action,

– $Transition \subseteq State \times State$ is a set of transitions. A transition is enabled when the condition associated with it is fulfilled,

– $Condition$ is a Boolean formula associated with a transition, defined using variables and/or input event. If the condition is "1" on the transition arc, then the transition is always activated. According to the standard, no more than one event input can be contained in a condition.

– $Action = Alg \cup EO$. An action can have zero or more algorithms, and one or zero output event. Each EC $state$ can have zero or more actions.

– $Priority$: $Transition \rightarrow \{1, 2, \cdots\}$ is an enumeration function that prioritises EC transitions where 1 is the highest priority and vice-versa.

When an event occurs, a sequence of steps is executed as follows: a data variable is available, and an input event arrives at BFB from another block. The data inputs variables associated with the incoming event are sampled. The execution control chart ECC

is notified. When the transition condition (event and data) is fulfilled, the corresponding transition is fired. Accordingly, the next corresponding state is enabled. The corresponding algorithm encapsulated in EC action will be executed. After executing the algorithm, output data will be updated, and output events will be sent to another function block in the application. If a data variable is not associated with any event, it always keeps its initial value. For each input/output data, a buffer is used locally within the function block to store the changes [Wozniak et al., 2018]. Furthermore, global variables are not allowed in IEC 61499 to simplify the access to distributed variables and the reallocation of function blocks to other devices in a distributed system. The execution control chart ECC is the main part in a function block. However, it is a static defined state machine in such a way that it cannot be modified at run-time.

A simple pick and place station is illustrated to show function blocks functionalities. The system in Fig. 2.2 includes a belt conveyor moving the workpieces "A" for assembly, and the complete workpieces "AB" out the conveyor, a manipulator inserting a workpiece "B" from a gripper into a workpiece "A". The conveyor stops when the separator detects the arrival of workpiece "A" for assembling. After that, the conveyor transports the complete pieces to packaging.

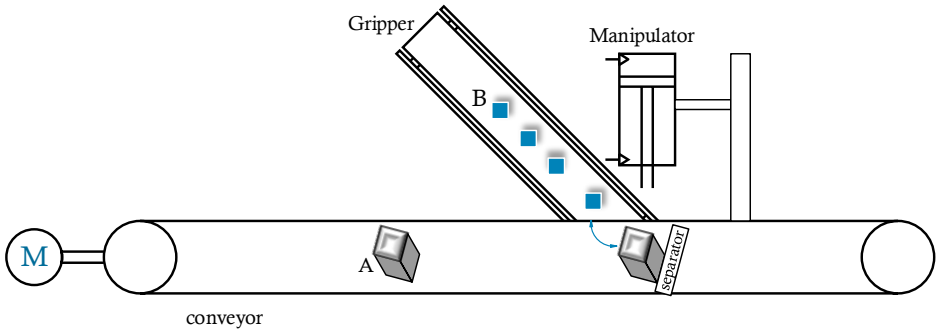


Figure 2.2 Pick and Place System.

To further illustrate this system, a sensor detecting the workpiece arrival at the belt conveyor, and another one near the separator detecting the workpiece arrival for assembly are considered. A motor starts and stops the conveyor. The belt conveyor controller is modelled by a basic function block as shown in Fig. 2.3 using *4DIAC IDE*.

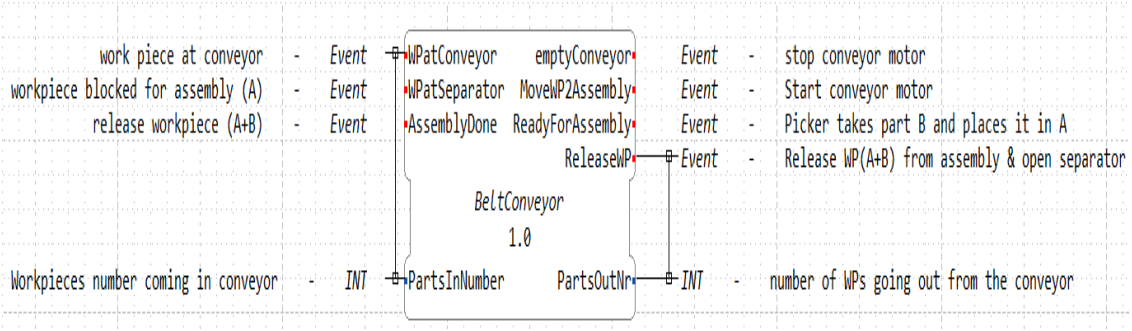


Figure 2.3 Belt Conveyor Function Block Interface.

The logic part of the FB is represented in the ECC as depicted in Fig. 2.4. The conveyor function block switches to the *stopEmptyConveyor* state when the *partsInNumber* is null and sends an *emptyConveyor* output event to stop the conveyor. Upon the arrival of the input event *WPatConveyor*, the associated input variable *partsInNumber* is updated, if it is greater than zero, then the FB switches to *moveToAssembly* state and triggers *moveWP2Assembly* output event to notify the separator function block that a workpiece "A" is coming. If the separator detects the workpiece arrival, it sends a *WPatSeparator* event to the conveyor controller. When this event arrives, the ECC switches to *stopConveyorForAssembly* state and sends an output event *readyforAssembly* to the manipulator FB. After assembling part "A" with "B", the manipulator sends an *AssemblyDone* event to the conveyor. Upon the arrival of this event, the ECC switches to *ReleaseWP* state to decrement *partsInNumber* by running a *decrementPartNumber* algorithm and release the complete workpiece by sending a *ReleaseWP* event to the separator FB. If any workpieces are left on the conveyor belt, ECC returns to *moveToAssembly* state and repeats the same steps. Otherwise, it returns to the initial state.

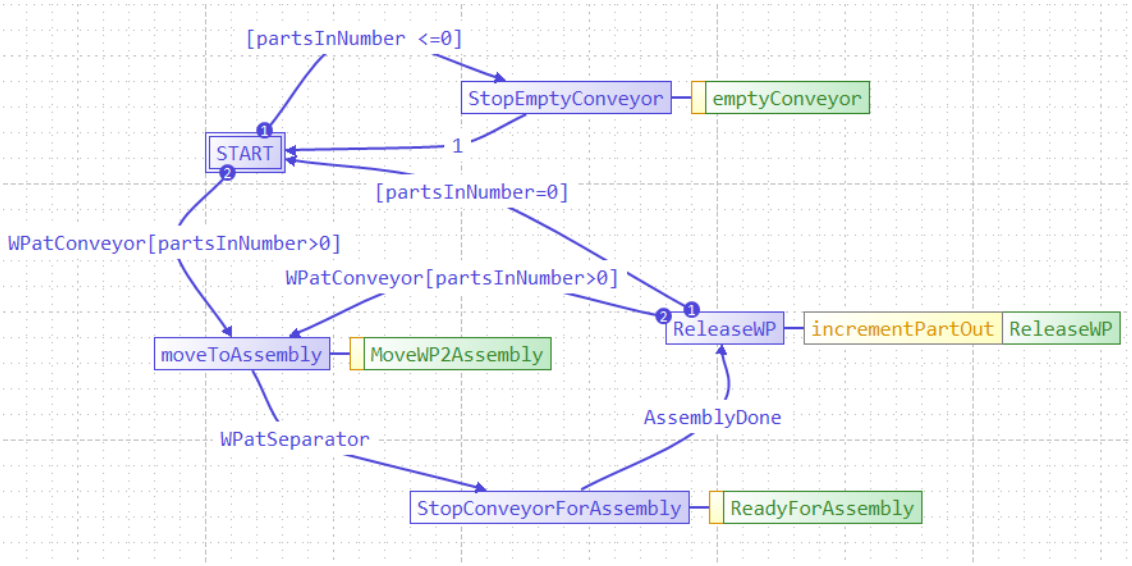


Figure 2.4 Conveyor Execution Control Chart.

2.2.3 Service Interface Function Block SIFB

Service interface function block is developed to create an interface between the function block (BFB, CFB) and external services as depicted in Fig. 2.5. It allows a function block to communicate with another device via a network. Furthermore, SIFB has access to hardware data, such as real-time clock [Sadeghi, 2010]. SIFB has a service sequence

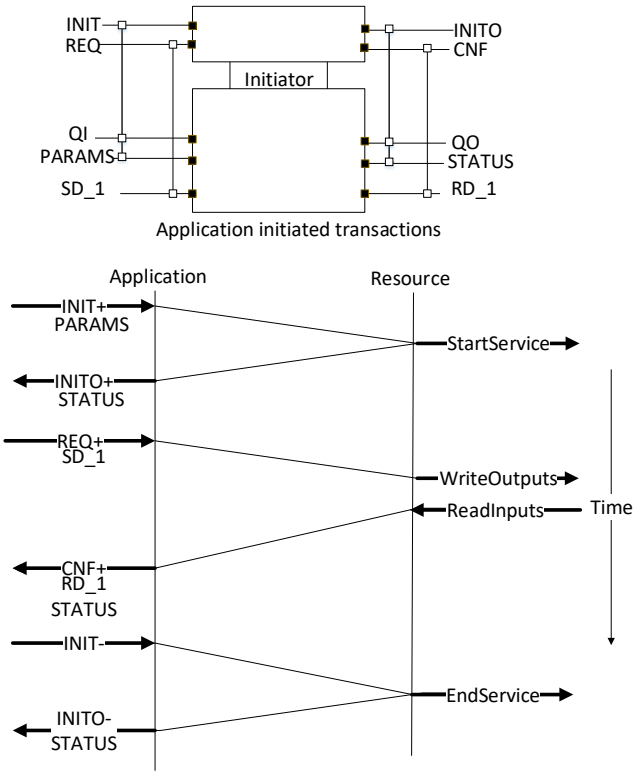


Figure 2.5 SIFB interface example and its sequence diagram.

diagram that hides the entire code of the function block. The sequence diagram illustrated in Fig. 2.5 shows the timing and sequential relationships between input and output events of SIFBs. It describes the dynamic part of a function block and the execution order. A service transaction occurs only when the previous transaction is valid.

SIFBs are also driven by events occurrence. A SIFB executes special algorithms that interact with the underlying hardware system (sensors, actuators, etc.). The algorithms are differently implemented for each service and hardware. The vendors generally provide such function blocks. In Fig. 2.5, the input data QI is an event input qualifier linked to $init$ and req input events. It indicates the event arrival. The service parameters are saved in the $PARAMS$ input data. QO indicates the success of request or service if it is equal to true and the failure of request/service otherwise. The $STATUS$ output data returns a message related to the QO event if the service fails.

2.2.3.1 Standard Event Function Blocks

All operations with primitive events are presented as SIFBs event function blocks found in the function block library. The designer can easily instantiate them in his application. The most important event function blocks are as follows. E_START FB is a cold start event function block which initiates the first function block in an FB network when the system is initially started. E_SPLIT FB generates simultaneous output events after the reception of a single input event to activate the parallel execution of several FBs. E_MERGE FB allows merging several events in one event. E_SWITCH FB permits the switching between output event when an incoming event occurs. It generates two different event paths depending on a Boolean input data. E_Cycle event function block generates a stream of events at fixed time intervals. For more details, read Chapter 6 Event function blocks in [A. Zoitl, 2014] book.

2.2.3.2 Communication Service Interface Function Block CSIFB

Several communication protocols (UDP, TCP, IP, MQTT, etc.) have been developed in IEC 61499 standard to communicate the distributed application with devices (sensors, actuators, PLCs, micro-controllers, etc.). Communication service interface function blocks (CSIFBs) are special SIFB that exchange data with devices in a unidirectional (respectively, bidirectional) way via *PUBLISH/SUBSCRIBE* (respectively, *CLIENT/SERVER*) function block types [A. Zoitl, 2014, HOLOBLOC, 2019b]. The *PUBLISH/SUBSCRIBE* FBs are used to establish *UDP/IP* communications, while the *CLIENT/SERVER* function blocks are used to exchange data with external resources such as databases, and it is used in an application to establish TCP/IP communication. The *CLIENT* FB waits for a response from the *SERVER* FB, in contrast to the *PUBLISH* function block.

2.2.3.3 Management Function Block

The management function block is a particular service interface function block ensuring reconfigurability. The IEC 61499 standard facilitates function blocks management using reconfiguration commands (creating or deleting FBs, adding or removing connections, starting or stopping elements, etc.) on the device and resource control layer. As shown in Fig. 2.6, the device manager provides several services. It allows function block instances to be created, started, deleted, killed, or stopped. The device manager also provides information about function block status such as in $READY$, $INVALIDSTATE$ or $OVERFLOW$ state [Brennan et al., 2008].

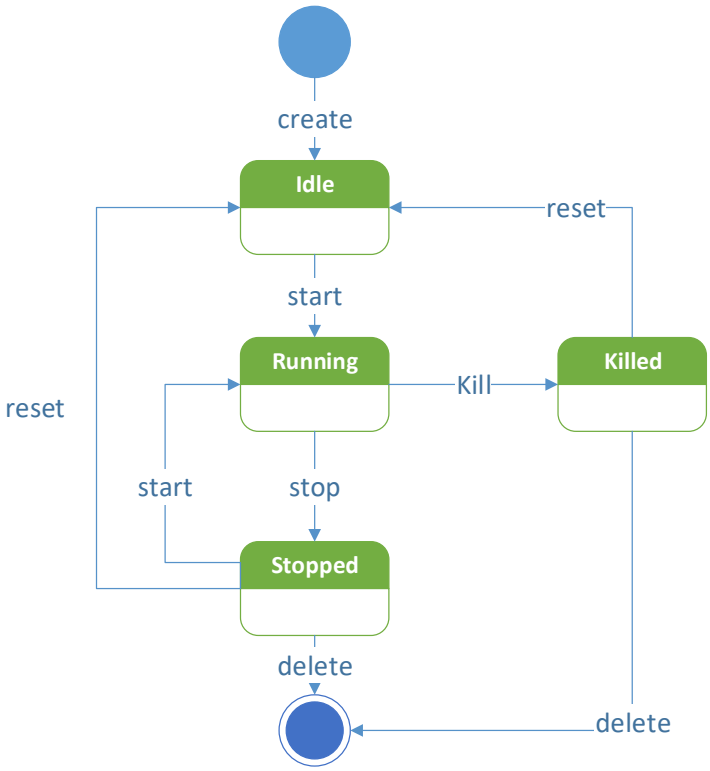


Figure 2.6 The state chart of a managed function block [TC 65/SC 65B, 2012a].

However, the commands are executed by the user at his own risk. Therefore, system inconsistencies can appear after a command execution and disturb the system. The user must be familiar with the system and must have excellent engineering skills. To the best of our knowledge, no verification tool has yet been developed to check the reconfiguration feasibility before and after reconfiguration execution. Additionally, the available debugging tools do not show the logical errors inside an application. As a result, a formal verification method is demanded.

Generally, SIFBs read events and send them to BFBs and CFBs. However, SIFBs do not classify events with priorities in such a way that errors and reconfiguration events are treated as standard events.

2.2.4 Composite Function Block CFB

A composite function block (CFB) is a network of function block instances linked by data and event connections as depicted in Fig. 2.7. CFB provides a hierarchical functionality. It is a container of multiple function blocks having a joint interface. The events occurrence defines the execution order of the encapsulated function blocks inside the composite FB. The CFB inputs, i.e. events and data inputs, are transferred to the internal FBs. Output events and data of internal FBs are transferred to the CFB outputs.

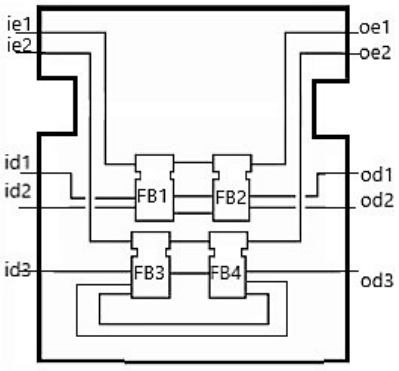


Figure 2.7 Composite Function Block.

The composite function block CFB has a hierarchical-structure and can describes multi-functionalities [Kim et al., 2018]. However, a mixture of basic and service interface function block increases the design complexity. A separation of the control level from the reconfiguration level is proposed.

2.2.5 IEC 61499 Models

According to the IEC 61499 standard, a system model is a set of applications distributed in several devices. The latter are connected by a communication network as illustrated in Fig. 2.8 where communication delays are considered. An application is a collection of interconnected function blocks which is developed independently of devices. FBs can be distributed over resources of one or several devices. During distribution, SIFBs are required to communicate the application with distributed devices in the network. A sub-application is a network of function blocks like CFB but its content can be distributed across different devices space. A sub-application is different from a composite function block in two points. The first point is that a sub-application can be executed on multiple resources. The second point is that it does not have any WITH qualifier between events and data.

In general, a device is a hardware unit capable of interacting with automation equipment and processes information [Vyatkin, 2016]. It can be a PLC, an on-chip controller, an intelligent sensor/actuator, a hub, a gateway, etc. However, a device in IEC 6149 is defined as an abstract model of such computing unit. A device model is a container of resources. It represents the information processing properties of hardware such as IP address and ports, etc. A device is also defined as an element of a system configuration in IEC 61499. It has a process interface and a communication interface in order, as shown in Fig. 2.8. The process interface allows a mapping function between the physical process and the resources. Any information coming from the physical process is presented to the resource as data/events. At the same time, the communication interface provides mappings between resources and information transferred via communication networks.

CSIFB and SIFBs function blocks are used to implement such communication interface.

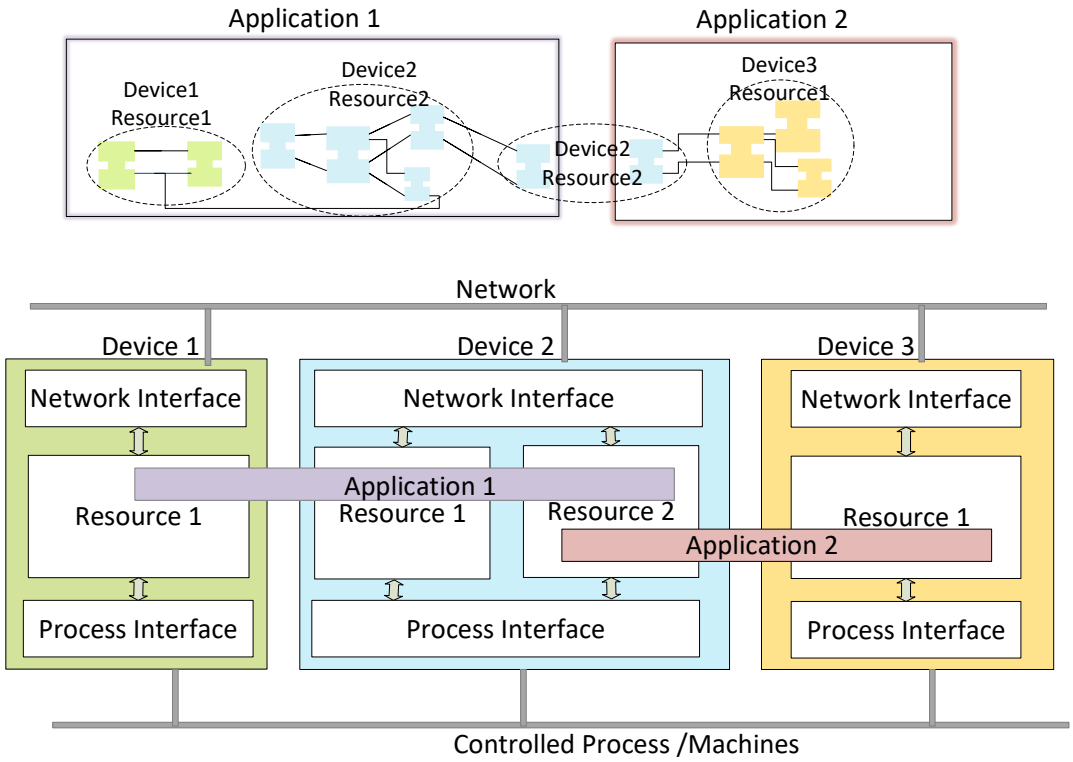


Figure 2.8 IEC 61499 System and Device Model.

A Resource model, as shown in Fig. 2.9, is considered as a kind of an execution container for FB networks. It is responsible for managing and controlling the behaviour of FBs, such as scheduling and executing function blocks algorithms. It also stores data, algorithms, execution control, and events. Furthermore, it maps communications and process input/output functions to service interface function blocks SIFBs. Each resource in IEC 61499 is independent of other resources within a device. It can control its operation without affecting other resources such as resource creation, configuration, parameterisation, start-up, deletion, etc. The scheduling function in a resource is a required function to schedule algorithms for execution and determine the sequence order in which such algorithms will be executed.

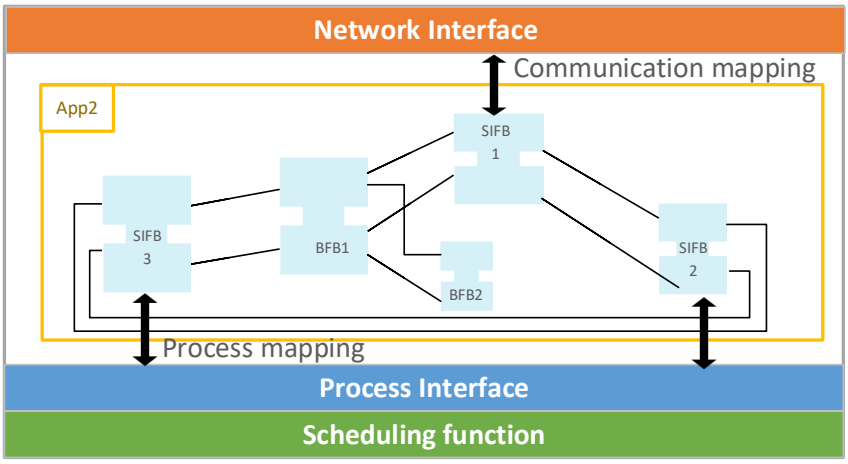


Figure 2.9 IEC 61499 Resource Model.

In the engineering cycle, in order to develop a distributed system, the developer should start with the top-level functionalities independent from the hardware structure beginning by modelling the control applications, regardless of whether the application will be executed by one device or distributed to multiple devices [Strasser et al., 2014]. Then, each function block can be mapped to the corresponding device/s.

A device management model is introduced in the IEC 61499 standard supporting reconfigurability at run-time [Strasser and Froschauer, 2012]. It consists of a distribution model, communication interfaces between devices, a set of commands, and suggested protocols [Dai et al., 2015]. The management function blocks belong to a configuration interface of a device. They invoke eight configuration commands (create, delete, read, write, start, stop, kill and query) [Strasser et al., 2014]. The function block manager controls the life cycle of FB instances. In [Vyatkin, 2016], IEC 61499 proposes three device classes to manage and configure the system:

(i) Class 0 gives the possibility to create and change connections between FB instances in an existing application, i.e. re-connecting the function block instances,

(ii) Class 1 permits to add new FB instances from the FB library and connect them to other FBs in the application. It also allows deleting FB instances and connections.

(iii) Class 2 allows to reprogram an existing application by adding new FB types to the library, instantiate them and connect them to the FB network.

In order to change a system configuration, a device may be configured by adding or deleting resources. Removing a device does not affect the application. Resources can also be configured using the reconfiguration commands. However, the commands are executed by users and not self-adapted. The logic behaviour of an existing application cannot be automatically reconfigured without user intervention. Moreover, only

some commands are implemented in the existing run-time environments such as creating/deleting FB instances during execution [Dai et al., 2015].

2.2.6 Software Tools and Run-time Environment

Several software tools and run-time environment have been developed in the last two decades [Prenzel et al., 2019]. They aim at providing an implementation for IEC 61499 architecture in accordance with the four parts of IEC 61499. The first part of IEC 61499 standard (IEC 61499-1) [TC 65/SC 65B, 2012b] defines function blocks architecture and provides guidelines for using function blocks in IPMCSs. The second part [TC 65/SC 65B, 2012c] of the standard defines software tools and IDE requirements. The third part (IEC 61499-3) presents tutorial information. Indeed, it is an obsolete part due to the changes done in the second version. The fourth part [TC 65/SC 65B, 2013] defines requirements for compliance profiles. The distributed devices and software tools must ensure the IEC 61499 requirements described in part 2 and 4 such as portability, interoperability and configurability.

The most important tools are FBDK, ISAGRAF, Fuber, 4DIAC and nxtStudio. Each software tool proposes an execution model. A cyclic execution semantic Cyclic-Buffered Execution Model (CBEM) is implemented in ISAGRAF and "ICARU FB" tools [Christensen et al., 2012a, Vyatkin and Chouinard, 2008, Pinto et al., 2016]. CBEM semantic supports the cyclic computation framework implemented in IEC 61131-3. ICARU FB does not allow multitasking implementation in the application. A Buffered sequential execution model (BSEM) is implemented in Fuber tool [Christensen et al., 2012a, Cengic and Akesson, 2008].

Archimedes and FBDK implement a non-preemptive multithreaded (NPMTR) execution semantic. In FBDK, multitasking is allowed, but threads are non-preemptible.

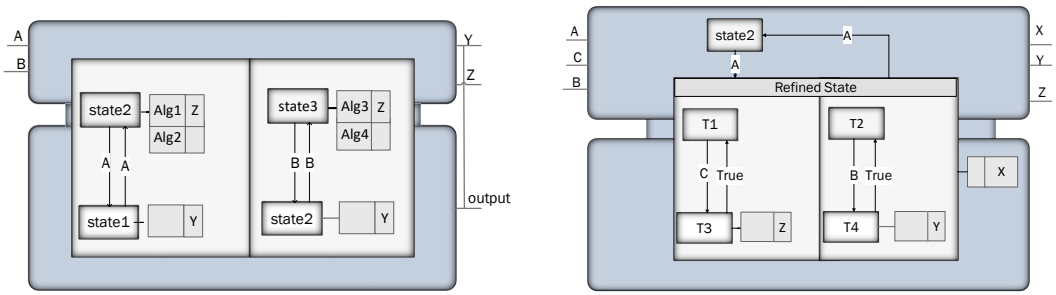
In other researches, a preemptive multithreaded (PMTR) execution model is developed. It is used in FBBeam, 4diac/FORTE, nxtControl/nxtIECRT, and RTFM-RT software tools. Multitasking is possible, and threads execution is preemptible. Nevertheless, all these tools do not implement the whole configuration commands defined in [Zoitl et al., 2006]. For these reasons, there is an increasingly crucial need to prioritise reconfiguration events compared to standard events and to execute the best reconfiguration scenario automatically without disturbing the execution process. Moreover, the existing IDEs and RTEs do not offer verification support. Only simulation technique is provided. In [Vyatkin and Hanisch, 2003], a verification environment for distributed application (VEDA) is developed. It focuses mainly on modelling and verifying the function block execution control using net/condition event systems. However, this tool models the plant manually, and it is not integrated with the compliant tools. Furthermore, system analysis and performance evaluation are not possible with the existing tools. For the

above-mentioned reasons, an environment supporting design, modelling and verification of RDCSs is recommended.

2.2.7 Hierarchical and Concurrent Execution Control Chart HCECC

A hierarchical and concurrent execution control chart (HCECC) has been proposed in [Sinha et al., 2015]. It is a syntactic extension to the IEC 61499 standard that directly expresses statecharts-like hierarchy and concurrency within basic function blocks. It is a multilevel hierarchical state machine. Its hierarchy enhances function blocks readability and decreases the system complexity. Thus, a parallel and refined HCECC are proposed in this research to support hierarchy and parallelism within basic blocks, as illustrated in Fig. 2.10.

In reconfigurable distributed control systems, the execution strategy depends on environmental changes and user requirements. Therefore, the HCECC notation proves to be effective since it can separate the reconfiguration model from the behaviour model. However, HCECC is not applied for reconfiguration purposes. Indeed, it can enable several sub-state machines in parallel that is not suitable for reconfiguration scenarios. Moreover, the function blocks defining reconfiguration scenarios require just two hierarchical levels: one for the reconfiguration and the other for executing only the corresponding scenario in a dynamic schedule.



(a) Parallel HCECC. **(b)** Refined HCECC.

Figure 2.10 HCECC examples.

2.3 Reconfiguration Evolution

Due to the unpredictable customer demand and international market, industrial and manufacturing systems must be reconfigured. Reconfiguration incorporates customising old products, changing the automation process, producing more or fewer products. Accordingly, many researchers work on this point and define reconfiguration as the ability to change an existing system configuration to a new one. Generally, reconfiguration is every change in software or hardware configuration (system parameters, adding devices,

resources) [Sinha et al., 2014]. In [Zoitl, 2009], the reconfiguration process transforms an existing control system into another. In [Vyatkin, 2016], reconfiguration is any change in the hardware or substitution of a vendor that affects the software. Two reconfiguration scenarios are treated in [Vyatkin, 2016]: the first is to change hardware configuration, and the second is to substitute a mechatronic component with its functional equivalent.

2.3.1 Reconfiguration Types

In literature, there are three types of reconfiguration. First, static reconfiguration is every change/configuration made before using the system components accordingly before system cold starts. Second, online reconfiguration is every change made at start-up time, mainly through reading configuration files. Finally, dynamic reconfiguration is every change applied automatically at run-time and requires hard time requirements, not like static and online reconfiguration. The satisfaction of time constraints is a crucial requirement of correctness [Strasser et al., 2014]. Hence, the imposed time constraints in the software application must be maintained while parts of the software are modified. Dynamic reconfiguration can also change the system without shutting it down and rebooting it [Zoitl, 2009]. A reconfiguration scenario in such dynamic reconfiguration is defined as any operation that changes the system execution model according to changes in its environment, system architecture, or user requirements. The main benefit of dynamic reconfiguration is that quick maintenance or upgrade can be enabled without stopping the whole system. Due to its benefits, dynamic reconfiguration has attracted much attention to increasing the lifetime and utility of systems in several fields due to its benefits. However, it has severe challenges such as preserving the system stability after reconfiguration. In this context, this dissertation focuses on the dynamic reconfiguration and verifies its feasibility while respecting real-time restrictions and maintaining system stability.

2.3.2 Reconfigurable Control Systems

A distributed control system is characterised by its control logic deployed in several interconnected devices and resources. The main goal of such a system is to control system processes and plants. Each change in the system status at the time of execution is primarily triggered by an event such as sensor detection, actuator change, message reception, etc. A reconfigurable distributed control system (RDCS) is a distributed control system where hardware and/or software components can adapt its structure and behaviour according to internal and external changes [Zoitl et al., 2006]. RDCS meets well the new flexibility and personalisation compromises.

A practical example of RDCS is a reconfigurable manufacturing system (RMS) proposed by [Koren et al., 1999, 2018, Vyatkin and Lastra, 2003] providing fast changes in structure and control system. This system allows the adjustment of production capacity

and system functionalities [Koren et al., 1998]. The main characteristics of RMS [Koren et al., 2018] are its scalability, convertibility, diagnosability, modularity and integrability.

2.3.3 IEC 61499 Reconfiguration Related Works

Several researchers have shifted attention to developing reconfigurable control systems following the IEC 61499 standard to resolve industrial changes, such as integrating new products without changing the current system.

A reconfigurable concurrent function block model and its implementation in real-time JAVA is proposed in 2002 in [Brennan et al., 2002b] for dynamic reconfiguration experimentation. The proposed model presents hierarchical reconfiguration management that specifies two concurrent control paths: (i) control of the process, and (ii) configuration control. This implementation is addressed to a particular project.

[Brennan et al., 2002a] focused only on reconfiguration in case of faults. They proposed an agent-based reconfiguration method to reconfigure the systems dynamically when faults occur at run-time. They had recourse to coordinator, mobile, and cohort agents modelled with IEC 614999 function Blocks for reconfiguration. [Angelov et al., 2005] propose design models of a reusable and reconfigurable state machine that can be configured using a state transition table. The latter consists of multiple-output binary decision diagrams that define the next state mappings of various states and the associated control actions. However, the proposed reconfigurable state machine considers a predefined state transition table. It develops a static reconfiguration applied offline before the system start-up, which does not fulfil the new industrial requirements. In [Zurawski, 2004], the authors propose to use the international standard IEC 61499 to achieve reconfigurability of automation systems by defining basic function blocks independent from a particular execution platform. Moreover, control applications are defined independently from a particular hardware architecture. They intended to apply IEC 61499 by defining a type for each level (system configuration type, device type, resource type, application and sub-application type, and function blocks type) and then instantiate and map them. Moreover, they highlight the use of adapter interfaces to minimise interblock connections. In fact, an adapter connection combines the bidirectional interaction of two function blocks encapsulating several events and data connections in both directions to decouple the interaction between function blocks [A. Zoitl, 2014]. [Zoitl, 2009] elaborate a management model that is based on a reconfiguration manager component and a user-programmable application. The limitation of this approach appears in the fact that the manager component is not a programmed entity deployed in every control device.

[Lepuschitz et al., 2011] propose to reconfigure software components by changing program sequences or component parameters in the application. They propose to add, delete, relocate, and replace instances of software components. The approach is based

on ontological knowledge bases [Lepuschitz et al., 2011]. Notably, an ontology is composed of an interpreted language and explicit assumptions to provide a formal representation of a real system [Ameri et al., 2012]. It provides a shared vocabulary needed for knowledge management and information exchange among distributed agents. The authors foreground an ontological representation of low-level agents to permit high-level control and initiate a reconfiguration process for low-level control. Nevertheless, this approach, based on dynamic software reconfiguration, has not been applied by any run-time execution environment.

[Strasser et al., 2014] highlight the need for an appropriate engineering approach and execution environment supporting dynamic reconfiguration in IEC 61499. They propose management commands to create and initialise function blocks at the run-time level as well as to transmit data and state from one resource to another. This IEC 61499 approach also allows deleting and updating a function block instance in a real-time system and changing its encapsulated algorithms to fix a bug or optimise it [Brennan et al., 2008, Zoitl and Strasser, 2017]. However, the authors assume that the function block type definitions must exist in the target execution environment. This imposes a serious restriction to the reconfiguration of systems at run-time.

Another methodology executing dynamic reconfiguration in discrete systems is proposed in [Pinto et al., 2017]. It swirls around the application of the supervisory control theory. The latter allows changing the system control logic at run-time while maintaining global and local consistency of the application. To the best of our knowledge, the ability to switch from one configuration, i.e. reconfiguration scenario, to another inside a function block has not yet been studied.

The authors in [Sünder et al., 2013] propose an Evolution Control Application named ECA. This application models the new changes occurring during the system life cycle. It is loaded if a change occurs and removed once the change is over. Once the application is loaded, it runs with the existing application. However, the ECA requires changes in the function block network.

[Sinha et al., 2014] proposed a framework to automatically reconfigure evolving IEC 61499 systems for deployment onto an available set of resources. Any change in the system architecture and high-level configuration requirements are formally specified as Satisfiability Modulo Theory (SMT) constraints to generate a valid subsystem reconfiguration. Nevertheless, the framework requires user-provided configurations.

A Virtual Function Block (VFB) mechanism is developed in Cloud-based Manufacturing Systems [Wang and Xu, 2013], i.e. service-oriented systems. The authors suggest to package existing and future manufacturing resources in a VFB to manipulate and integrate them via event states and data flows. Several studies focus on service-oriented architecture (SOA) for dynamically reconfiguring IEC 61499 systems. [Koumoutsos and

Thramboulidis, 2009] propose a knowledge-based framework based on ontologies, SOA, and semantic Web languages. The proposed SOA framework is integrated within software agents aiming to achieve negotiation between support systems. It seems that the framework is limited only to the design level, and no run-time environment is implemented. [Dai et al., 2015] proposes an SOA-based run-time architecture to apply the SOA concept implementing the IEC 61499 standard at the device level.

[Strasser et al., 2011] propose a complete Zero downtime reconfiguration approach based on the user intervention to reconfigure control applications based on IEC 61499 standard dynamically. The authors present an enhanced IEC 61499 device management that shows the dynamic change of a function block algorithm without perturbing the whole system. Even so, the user intervention is required to reconfigure the control system.

[Alsafi and Vyatkin, 2010] put forward automatic reconfigurations applied by intelligent agents that adapt to changes in the requirement and environment. The research has focused on studying control systems reconfiguration when hardware faults occur at run-time. [Khalgui et al., 2010] define a multi-agent architecture for distributed reconfigurable systems following the IEC 61499 standard. They come up with not only a reconfiguration agent to handle automatic local reconfigurations under well-defined conditions but also a coordination agent to manage distributed reconfigurations between devices. However, the reconfiguration using agents makes the design more complex.

Several approaches are applied mainly in the last decade to customise the industrial automation such as smart grids and cyber-physical systems. These approaches increase engineering efficiency, but the issue is that reconfiguration increases development time and design complexity due to the need for handling different reconfiguration scenarios, i.e. behaviours. In this thesis, we focus on optimising dynamic software reconfiguration. In most reconfiguration studies, the authors focus on the system architecture changes, such as adding new devices/resources or removing devices/resources instances from the system, which are translated to system configuration. To the best of our knowledge, none of the related works propose to reconfigure the function block itself by defining a reconfigurable dynamic state machine executing reconfiguration scenarios according to an intelligent decision. Moreover, rare approaches separate the reconfiguration model from the control model inside a function block despite the fact that such separation is beneficial for enhancing readability, speeding up the switching from one reconfiguration scenario to another, simplifying the design and reducing reconfiguration time.

2.4 Petri Nets for Formal Modelling

Modelling and verification of reconfigurable manufacturing systems have attracted the attention of many researchers. [Dubinin et al., 2006] model IEC 61499 function block

systems with the language of logic programming Prolog to verify liveness, reachability and safety properties. Stanica and Guéguen [Stanica and Guéguen, 2004] have modelled function block network with timed automata. Moreover, several researchers have shared the ideas of modelling and analysing distributed control systems with Petri nets to achieve formal verification. Indeed, Petri nets are used for formal modelling, analysis and verification of distributed systems, introduced in the early sixties by Carl Adam [Petri, 1962]. It is developed to verify mutual exclusion, deadlocks and time constraints. IEC 61499 FBs are based on event-driven activation mechanism that preserves the causality of distributed control systems [Drozdov et al., 2016]. Several event-conditions Petri net formalisms are introduced in this section. Most of them are used to model IEC 61499 designs except two reconfigurable Petri nets model R-TNCES and GR-TNCES. Their concepts are event-driven and very close to the IEC 61499 execution control chart.

2.4.1 Net Condition/Event System NCES

Net Condition/Event System (NCES) is a modular Petri nets extension with extra condition and event signals, making them different from other Petri nets. NCES models reactive systems. It was introduced by Rausch and Hanisch [Rausch and Hanisch, 1995] and further developed in [Hanisch and Lüder, 1999]. Each NCES module has a particular dynamic behaviour and is connected via special condition/event signals to other modules to constitute a system model. Condition signal is a constant signal coming from a place and going to a transition, whereas event signal is defined as the description of a trigger signal, coming from a transition and going to another transition as illustrated in Fig. 2.11. The token is removed from the place when the condition is fulfilled, and the next transition(s) is fired. If a transition fires, then it removes the token away from the last place and places it in the next place.

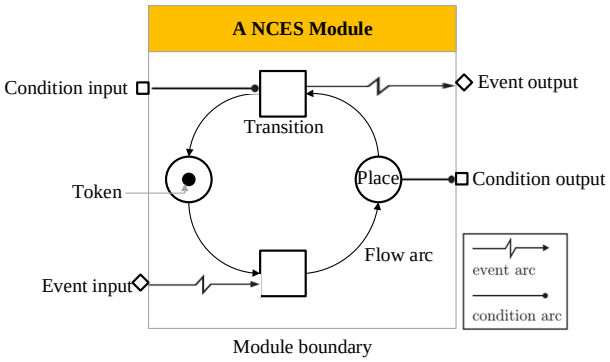


Figure 2.11 Graphical Notation of an NCES Module.

NCES is formally represented as a tuple:

$$NCES = (P, T, F, CN, EN, Cin, Ein, Cout, Eout, Bc, Be, Cs, Dt, m_0)$$

where:

- P (respectively, T) is a non-empty finite set of places (respectively, transitions),
- F: $(P \times T) \cup (T \times P)$ is a set of flow arcs,
- $CN \subseteq P \times T$ is a set of condition arcs,
- $EN \subseteq T \times T$ is a set of event arcs,
- C_{in} is a set of condition inputs,
- C_{out} (respectively, E_{out}) is a set of condition (respectively, event) outputs,
- Bc (respectively, Be) is a set of condition (respectively, event) input arcs in an NCES module, $Bc \subseteq (C_{in} \times T)$ (respectively, $Be \subseteq (E_{in} \times T)$),
- Cs (respectively, Dt) is a set of condition (respectively, event) output arcs, $Cs \subseteq (P \times E_{out})$ (respectively, $Dt \subseteq (T \times E_{out})$),
- m_0: $P \rightarrow \{0, 1\}$ is the initial marking.

In NCES, a transition is enabled when all pre-places of the condition signals are marked by one token, i.e. condition concession, and the pre-places of the transition are marked by one token, i.e. token concession. Every transition having no incoming event arcs, it is called a spontaneous transition. Otherwise, it is a forced transition. The forced transition is enabled by token, condition and event signals.

NCES formalism has been used in [Vyatkin and Hanisch, 2000a, Missal et al., 2007, Ivanova-Vasileva et al., 2008, Stanica and Guéguen, 2004] to model function blocks for verification reason due to its structure and execution which is very close to the one of the function block. A condition signal in FB corresponds to a condition related to a place in NCES, and a FB event signal corresponds to an event transition in NCES. NCESs are used in [Zhang et al., 2018a] to control dynamic reconfigurable discrete event systems.

The same example of Section 2.2.2 is modelled with NCES, as illustrated in Fig. 2.12. An assembly module is activated when the input event "pieceReadyForAssembly" is active. After assembling piece "A" with "B", an output condition is sent to the conveyor module.

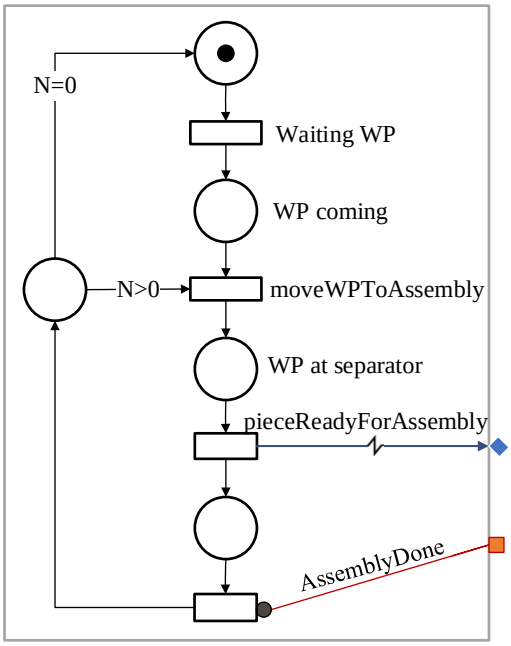


Figure 2.12 NCES Module for a Conveyor Controller.

2.4.2 Timed Net Condition/Event System TNCES

TNCES formalism is an extension to Net Condition/Event System NCES with extra time constraints attached to input arcs of transitions. TNCESs have been widely applied in distributed control systems due to their suitable structure to such systems [Zhang et al., 2018b].

A TNCES is a 2-tuple $\Gamma = (N_\Gamma, z_0)$ where $N_\Gamma = (P, T, F, CN, EN, em, DC)$ is its net structure with P, T, F, CN and EN having the same meaning as in NCES.

1. $z_0 = (m_0, d_0)$ is the initial state of Γ where m_0: $P \to \{0, 1\}$ is a mapping function which is called the initial token state of Γ, and $d_0 : P \to \mathbb{N}$ is the initial clock position of Γ. Every pre-arc of a transition has an accepted time interval between [l,h] where $l, h\mathbb{N}$ and by default they are equal to 0 that is the minimum value.

2. em: $T \to \{\vee, \wedge\}$ is a mapping function which attaches a (signal-processing) mode to every transition where "$\vee$" is OR mode and "$\wedge$" is AND mode.

3. DC is a mapping that assigns an integral time interval to each output flow arc (flow arcs from places to transitions). DC: $F(P \times T) \to \{[l, h]\}$ is a super-set of time constraints on output arcs where $l, h \in \mathbb{N}$.

The clock measures the time of the token states. Each place has a clock that runs when the place is marked by a token. When a transition fires, the clock is reset, and the

token is removed from the place [Hanisch et al., 1997]. The token can be removed from a place p by a transition t only when the clock at the place p is between the earliest and the latest firing time. In TNCES, a set of transitions can fire simultaneously, i.e. concurrently.

TNCES is used in several studies to model PLCs following IEC 61131 and IEC 61499 standard. [Thieme and Hanisch, 2002] proposed a method that automatically translates a TNCES model into control code based on IEC 61131 Function Blocks. TNCES can model uncontrolled plant behaviours. [Vyatkin and Hanisch, 1999] have modelled IEC 61499 function blocks with TNCES model to verify discrete control applications using SESA model checker and temporal logic properties. An approach in [Ivanova-Vasileva et al., 2008] aims to generate a formal model for basic and composite function blocks at the application level. However, TNCES model is not a reconfigurable Petri net model since it does not fit the changes in the states and transitions in a reconfigurable distributed control system.

2.4.3 Reconfigurable Timed Net Condition/Event System R-TNCES

R-TNCES is a formalism for modelling and verifying dynamic reconfigurable discrete event systems developed in 2013 [Zhang et al., 2013]. It is a reconfigurable extension to timed net condition/event systems (TNCESs). R-TNCES separates the behaviour from the control module. The behaviour module B is a set of imbricated TNCESs which correspond to the model of a configuration of the underlying reconfigurable system. The control module R corresponds to the reconfiguration control mechanism of a reconfigurable system. R is a set of reconfiguration functions that deal with structural changes of configurations and state coherence during reconfigurations [Zhang et al., 2013].

A reconfiguration function allows switching between TNCESs through enabling or disabling control components, changing condition or event signals, modifying the communications among them, and dealing with state feasibility before and after reconfiguration process. This transformation is in reason to reply to environment changes, new user requirements, new manufacturing strategy or error handling [Zhang et al., 2013]. In R-TNCES, the concurrence of standard events and reconfiguration events problem is treated, and the dynamic system behaviour can be described, analysed, and controlled during reconfiguration to guarantee system correctness [Zhang et al., 2013].

An R-TNCES is formalised as a control module R and a behaviour module B. The behaviour module B is a set of superposed Timed Net Condition/Event Systems ($B = \sum TNCES$). The control module is a set of reconfiguration functions ($R = \sum r_i$) which control the switching between TNCESs inside the behaviour module B. A reconfiguration function is defined as $r_i = (pcond, S, x)$ where:

- $pcond$ is its pre-condition. The pre-condition $pcond$ is a boolean variable that can be a specific external instruction or a specific system state. If $pcond$ is true, then

the corresponding reconfiguration function r_i is executable.

- S is its structure modification instruction S: $TN(*r) \rightarrow TN(r*)$ that transforms TN before reconfiguration $TN(*r)$ to a TN after reconfiguration $TN(r*)$,
- and x: $laststate(*r) \rightarrow laststate(r*)$ is the state correlation function that maps the last state before applying the reconfiguration function r_i to an initial state after applying r_i.

In order to implement the reconfiguration inside R-TNCES, the author [Zhang, 2015] defined a state machine called a $structure_{changer}$ and a set of actuators responsible for synchronising the $structure_{changer}$ with the behaviour module of the R-TNCES. In the $structure_{changer}$, each place corresponds to a specific TNCES of an R-TNCES, and each transition corresponds to a reconfiguration function. If a place gets a token, then the corresponding TNCES is selected. If the linked transition is fired, i.e. the reconfiguration function is applied, then the token is removed from the previous place and inserted in the next places. The authors propose an actuator for each place in the Structure changer to reactivate the changed TNCES, as shown in Fig. 2.13. Each actuator is composed of a single place marked by only one token and a single transition. When a place in $structure_{changer}$ receives a token, the corresponding actuator (TNCES) will be enabled, and its transition will constantly send event signals to the corresponding control components in the TNCES. Only the control components in the active TNCES are executable, all the remaining control components still disabled.

R-TNCES is gaining a growing interest in the last five years since it proposes optimised modelling of reconfigurable discrete event systems [Zhang et al., 2015b]. It is applied perfectly in a flexible medical robotic platform [Salem et al., 2015] and on reconfigurable assembly systems [Zhang et al., 2018b]. It also models reconfigurable systems under memory and energy constraints [Khlifi et al., 2015]. A resource sharing solution in R-TNCES is discussed in [Salem et al., 2014]. Moreover, a new R-TNCES rebuilding method for Reconfigurable Systems has been proposed by [Ramdani et al., 2019] to repair the R-TNCES model when a property is unsatisfied during model checking. Furthermore, a new temporal logic called RCTL has been developed to improve formal verification of Reconfigurable Discrete-Event Systems based on R-TCNES [Ramdani et al., 2020].

2.4.4 Generalised Reconfigurable Timed Net Condition/Event systems GR-TNCES

A GR-TNCES is an R-TNCES extension for modelling reconfigurable discrete systems under energy and resources constraints. It is a set of $|G|$ superposed R-TNCES g_i where $g_i = (B, R)$ with B being the behaviour module and R being the control module that is a set of reconfiguration functions. The behaviour module B is a union of multiple TNCES defined as follows:

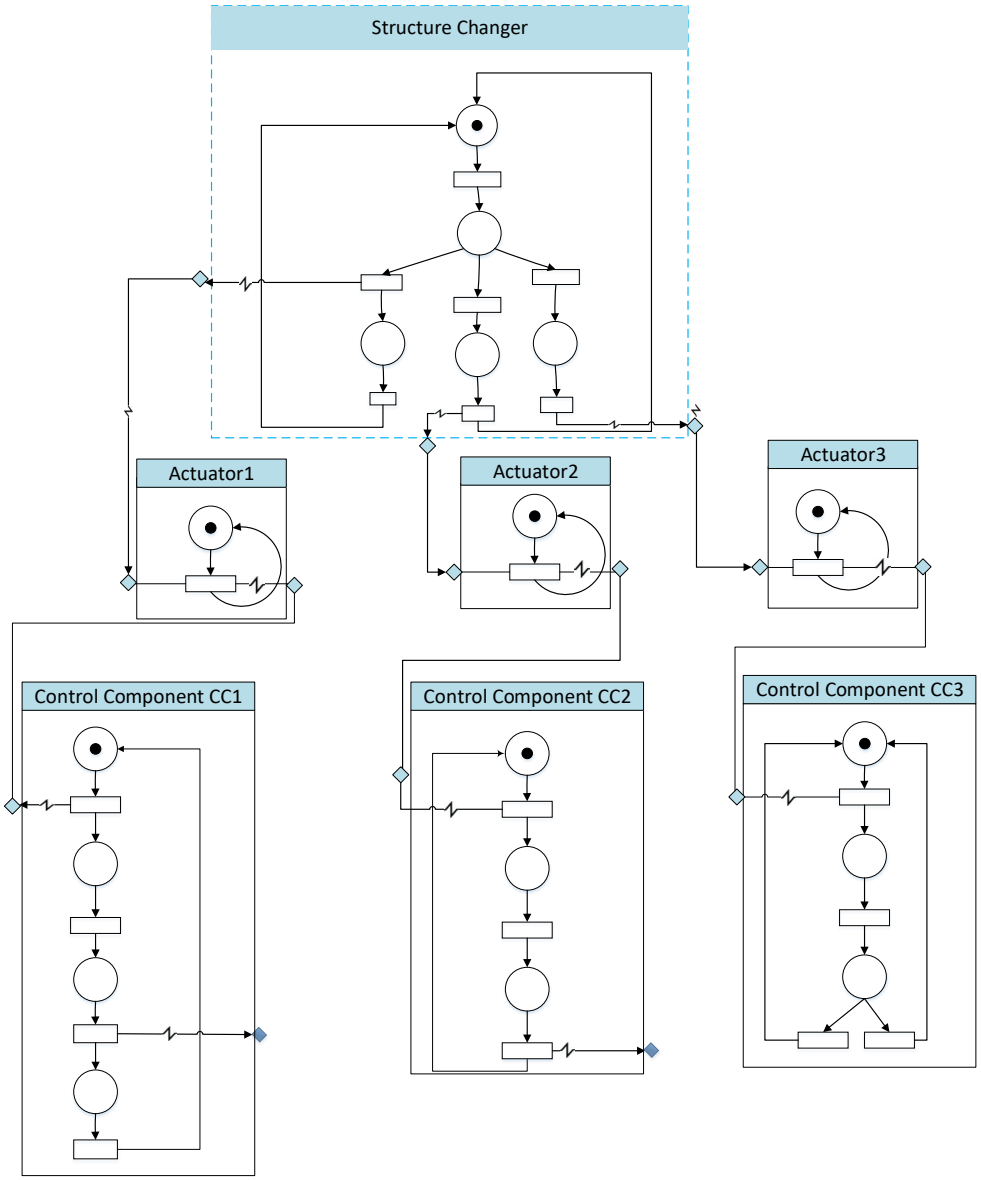


Figure 2.13 An R-TNCES Example Model.

$$B = (P, T, F, W, CN, EN, DC, V, Z_0)$$

where P, T, F, CN, EN, and DC have the same meaning as in TNCES and

(i) W: $(P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ maps a weight to a flow arc: $W(x, y) > 0$ if $(x, y) \in F$, and $W(x, y) = 0$; otherwise, if $x, y \in P \cup T$,

(ii) V: $T \rightarrow \{\vee, \wedge\}$ maps an event-processing mode (AND or OR) to each transition,

(iii) $Z_0 = (T_0, D_0)$ where T_0: $P \rightarrow \{0, 1\}$ is the initial marking and D_0: $P \rightarrow \{0\}$ is the initial clock position.

The control model R is a set of reconfiguration functions r_k where

$$r_k = (cond, P_0, E_0, M_0, Q, L) \tag{2.1}$$

with:

- $cond \rightarrow \{true, false\}$ is the precondition of r_k,
- P_0: $F \rightarrow [0..1]$ is the TNCES probability,
- E_0: $P \rightarrow [0..max]$ is the number of tokens controlling the energy requirements,
- M_0: $P \rightarrow [0..max]$ is the number of tokens controlling the memory requirements,
- Q: $TN(*r) \rightarrow TN(r*)$ is the structure modification instruction for a reconfiguration scenario, it transforms TN before reconfiguration $TN(*r)$ to a TN after reconfiguration $TN(r*)$,
- and L: $l_s \rightarrow i_s$ is the state processing function. l_s denotes the last state before applying the reconfiguration function r_k and i_s denotes the initial state after applying r_k.

The union of multiple TNCESs $\cup N$ is a set of all feasible net structures that can be performed by an R-TNCES where $\cup N$ = ($P \times T \times F \times W \times CN \times EN \times DC \times V$). Each $N_o \in \cup N$, $o \in [1 \cdots |N|]$ is a behaviour in the system.

A state machine called structure changer manages the dynamic reconfigurations of the system. It describes the control module R. Each place sp_i in the structure changer corresponds to an R-TNCES module, while each transition corresponds to a reconfiguration function. A reconfiguration is achieved according to the highest R-TNCES branch's probability and the system resources at the desired instant.

The structure changer SC = (P, T, F, Q, M, E) is a particular state machine where M is the memory resources, and E is the energy resources of each R-TNCES. $Q \in [0, 1]$ is the probability for each R-TNCES branch where the sum of $Q_i = 1$. It designs the chance to fire such reconfiguration scenario that is depicted as a branch. The control module R has a memory reserve Mem and an energy reserve Eng. The executed reconfiguration scenario should not violate/exceed the global memory and energy, i.e. $E <= Eng$ and $M <= Mem$.

Each state in TNCES consumes one token from the energy reserve Eng and one token from the memory reserve Mem. Before executing any probabilistic reconfiguration scenario, the availability of reserved energy and memory has to be checked. If the memory reserve is enough, i.e. $M < Mem$, then M is subtracted from the global memory Mem. After executing the reconfiguration process, the memory tokens are added back

to the model's memory reserve. In parallel, the energy reserve will be removed from the battery and will be recharged periodically.

A GR-TNCES example is illustrated as follows in Fig. 2.14. R-TNCES1 needs 15 tokens and 12 energy units. The needed resource should respect the global resources Eng and Mem. In GR-TNCES, the most probabilistic reconfiguration will be executed if the required resources are provided. Otherwise, the next probabilistic scenario satisfying the time, memory and energy constraints will be carried out. In this case, $RTNCES4$ will be achieved.

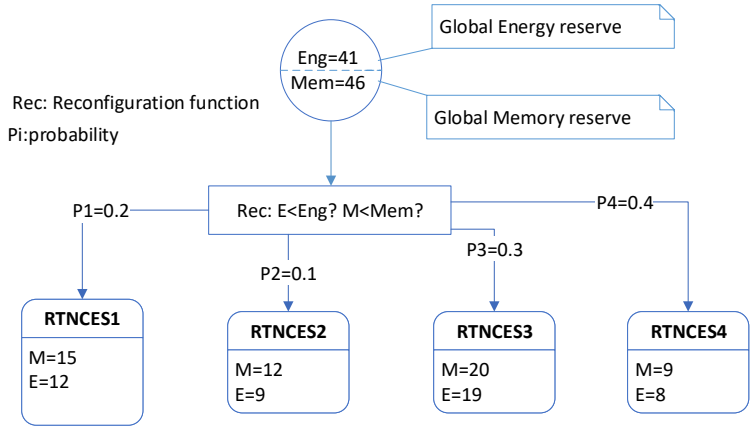


Figure 2.14 A GR-TNCES Example Model.

2.5 Verification of Reconfigurable Systems

Because of the increasing number of components and modules in the reconfigurable distributed systems, logic correctness cannot be guaranteed. IEC 61499 standard does not show the way an application is verified. Simulation is one of the most used techniques for evaluating software application. It is supported by several IEC 61499 development tools such as Function Block Development Kit FBDK, 4DIAC, NxtSTUDIO and ISaGRAF [Christensen et al., 2012a]. Matlab is well known for its efficiency in simulation, which can be integrated with the cited IEC 61499 tools [Liu, 2018]. This technique injects thousands or millions of inputs to the system model and analyses only the outputs. Nevertheless, simultaneous tasks execution, parallel control logic, events behaviour, temporal logic, and deadlocks cannot be proved with simulation. The latter is also not adequate for testing critical industrial applications and does not guarantee safety property. To address this problem, reachability analysis has been developed by [Starke and Roch, 2002] to analyse the dynamic behaviour of a system model. They have introduced a model checking technique that checks whether the system model satisfies a formal requirement. The reachability graph generates all possible behaviours of the system. Compared to simulation, theorem proving and equivalence checking techniques, model checking is the best

technique. The latter stems its importance from the fact that it enables the unsupervised automatic system verification and identifies system failure via counterexamples [Patil et al., 2015].

In order to use model checking, formal modelling of the target system in specific formalism is required, such as Petri net formalism as discussed in the previous section. The generated models can be automatically verified against formal properties specified in Computational or Linear Tree Logic (LTL or CTL). A model checking algorithm, encapsulated in the model checker tool, searches all the reachable states automatically in the model and generates counterexamples if the system cannot satisfy the property.

2.5.1 Verification of IEC 61499 Architecture

Most studies on IEC 61499 verification use qualitative analysis and formal modelling. Some studies have recourse to model checking in order to automatically verify properties of finite-state systems. Indeed, model checking uses computation tree logic for specifying such properties. This technique is known for its ability to handle complex problems. [Zhang et al., 2015a] used model checking to automatically verify properties of an R-TNCES model. They checked through the model checker SESA some properties related to reconfiguration such as the system correctness after concurrent reconfiguration requests and the valid behaviour of subsystems after applying a reconfiguration scenario. They used Computation Tree Logic CTL detailed in the next section. SESA is a model checker that proves properties of desired or prohibited behaviour in the reachability graph of the system model. SESA has been used with NCES Petri nets [Vyatkin and Hanisch, 2000a] to verify IEC 61499 systems.

Qualitative verification is an excellent technique to validate system correctness and several properties of reconfigurable distributed control systems. In recent research, probabilistic verification has taken an important place in the verification process since it allows to drive the system to satisfy the wanted quality of service requirements. In [Forejt et al., 2012], an incremental probabilistic technique has been proposed for the run-time analysis of adaptive software systems. Probability is characteristic of reconfiguration events. To the best of our knowledge, only the work of [Bhatti et al., 2017] introduces unified analysis checking qualitative and quantitative properties using IEC 61499 standard. They convert the IEC 61499 model to a specific PRISM model, and then they inject some faults to the model to check safety properties using the PRISM model checker [Kwiatkowska et al., 2002]. However, the last approach exacerbates the state explosion problem for large scale reconfigurable systems because it translates IEC 61499 model directly into a PRISM model. The total time of system verification grows exponentially with the total number of possible states in the system. Consequently, a modular verification is required in complex systems to minimise the state explosion risk. Moreover, a

unified qualitative and quantitative analysis is needed to take advantage of its benefits by verifying the feasibility of reconfiguration, estimating reconfiguration risks and maintaining system stability. PRISM is the best tool which combines qualitative and quantitative verification using computation tree logic.

2.5.2 Temporal logic

Temporal logics [Starke and Roch, 2002] describe and specify properties that the system behaviour must fulfil. Computation Tree Logic CTL and Probabilistic Computation Tree Logic PCTL are well known temporal logics used for the specification of model checking. This section has as objective to give a brief presentation of the most important temporal logic: CTL, PCTL, and Timed Computation Tree Logic TCTL.

2.5.2.1 Computation Tree Logic CTL

The Computation Tree Logic CTL is defined by Clarke and Emerson in [Clarke et al., 1986]. In CTL, all formulae specify qualitative behaviours of the system starting from an assigned state in which the formula is evaluated by taking paths into account. A path is a sequence of states. The semantics of formulae are defined according to a reachability graph where states and paths are used for the evaluation. A reachability graph M consists of all global states that the system can reach from a given initial state. It is formally defined as a tuple $M[Z, E]$ where Z is a finite set of states and E is a finite set of transitions between states, i.e. a set of edges (z, z_0) where $z, z_0 \in Z$ and z_0 is reachable from z. The designer can verify the system using such computation tree logic formulae:

1. $z_0 \models AF(p)$ which expresses that we will always reach p from z0, and
2. $z_0 \models EF(p)$ which expresses that it is possible to reach p from z_0.

They enable to express that a given state property must hold for all paths starting from the state. Thereby, witnesses for existence-quantified sub-formulae and counterexamples for all-quantified sub-formulae can be determined and displayed. CTL allows the use of atomic propositions to express the properties of some states.

2.5.2.2 Timed Computational Tree Logic TCTL

Timed Computational Tree Logic TCTL is an extension to CTL with extra clock variables and constraints, allowing to analyse and evaluate clocks in the system. The semantics of temporal formulae is defined following a reachability graph, and it is defined as follows:

$$\varphi ::= a|g|\varphi_1 \wedge \varphi 2|\neg\varphi|E(\varphi_1 U^J \varphi_2)|A(\varphi_1 U^J \varphi_2) \tag{2.2}$$

where:

- a is an atomic action;
- g is a clock constraint;
- E means ”for some path”;
- A means ”for all paths”;
- J is an interval whose bounds are $\in \mathbb{N}$

The reachability graph consists of all global states that the system can reach, starting from a given initial state. The basic structure can be seen as a directed graph. States and paths of the reachability graph are used in the evaluation of functional and temporal properties. Qualitative analysis using CTL and TCTL properties allows the analysis of system behaviours, but it does not give an insight into system performance and an estimation of reconfiguration risk. Quantitative analysis is thus required.

2.5.2.3 Probabilistic Computation Tree Logic PCTL

PCTL [Kwiatkowska et al., 2002] is a probabilistic specification formalism which is an extension to the temporal logic CTL. It was introduced by Hansson and Johnson [Hansson and Jonsson, 1994] to analyse discrete-time probabilistic systems such as Markov Decision Processes MDPs [Puterman, 2014] and discrete-time Markov chains DTMCs.

A PCTL property is of the form “the probability that event “A” occurs is at least p under any scheduling of processes”. The P operator allows computing the actual probability that a model behaviour is observed or the probability of an event occurrence. The operator F means “eventually”, U means “until” and $U^{\leq k}$ means “bounded until a period k”. PCTL excludes the next “X” operator. An example of PCTL property $P =?[Fphi]$ means ”what is the probability of reaching a state where phi is true from the initial state of the model?”.

2.5.3 PRISM Model Checker

PRISM offers a probabilistic model checking for run-time verification of adaptive systems and analyses systems that exhibit random or probabilistic behaviour [Kwiatkowska et al., 2002]. A PRISM model is a set of modules $M = \{M_0, ..., M_n\}$, where a module M_i consists of local variables V_i and commands C_i, i.e. $M_i = (V_i, C_i)$. The variables V_i describe the possible states in the module. The values of these variables at any given time constitute the states of the module. The space of reachable states is calculated using the range of each variable and its initial value. Each module can read variables of other modules but only write to its own. The local state of all modules determines the global state of the whole model. The behaviour of each module is described by commands C_i that define the way the state changes over time. A PRISM command is formalised as

$$([a]g \rightarrow \lambda_0 \colon u_0 + \cdots + \lambda_n : u_n) \text{ where}$$

[a] is an action-label, g is a guard condition, u_i is an update statement, and λ_{i} is a probability value ($0 \leq \lambda_i \leq 1$) such that ($\sum_{i=0}^{n} \lambda_i$ = 1) and n is the number of statements. The command is enabled when the boolean expression g is equal to true.

A discrete-time Markov chain DTMC [Arora and Rao, 2016] is defined as a tuple

$$D{=}(S, s_{init}, P, L) \text{ where}$$

(i) S is a non-empty set of states,

(ii) $s_{init} \in S$ is the initial state,

(iii) $P \colon S \times S \rightarrow [0, 1]$ is the transition probability matrix where $\sum_{s' \in S} P(s, s')$ = 1 for all $s \in S$,

(iv) $L \colon S \rightarrow 2^{AP}$ is a function labelling states with atomic propositions taken from a set AP.

The file extension used for DTMC model is ".pm". A DTMC example of two mutual processes is written in PRISM as depicted in Fig. 2.15. The system example [of Oxford, 2020] comprises two processes x and y that operate under the mutual exclusion. Each process can be in one of its three states: $\{0, 1, 2\}$. From state 0, a process will move to state one with a probability equal to 0.2 and remains in the same state with a probability equal to 0.8. It tries to move from state one to the critical section state two. The process can only occur if the other process is not in its critical section. Finally, from state two, a process will remain in state two or move back to state 0 with equal probability.

2.6 Discussion and Originalities

IEC 61499 standard faces challenges in the design and verification of reconfigurable distributed control systems. The existing control applications are implemented using a huge number of function blocks and/or a complex execution control chart composed of several states and transitions that increase complexity. For every new requirement or environment change, the design model should be adjusted and adapted. The fact that the existing approaches do not adjust the system automatically accentuates the need for a dynamic reconfiguration in IEC 61499 standard. The latter then needs to be extended and automatically applied. The basic function block types do not handle the reconfiguration scenarios easily. The control logic encapsulated in the execution control chart of the basic function block is static. Moreover, to avoid cumbersomeness in ECC, a separation of reconfiguration scenario from the control logic is necessary. Additionally, there is a need for a smart hierarchical execution control chart that can ensure more modularity

```
dtmc

module M1

    x : [0..2] init 0;

    [] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);
    [] x=1 & y!=2 -> (x'=2);
    [] x=2 -> 0.5:(x'=2) + 0.5:(x'=0);

endmodule

module M2

    y : [0..2] init 0;

    [] y=0 -> 0.8:(y'=0) + 0.2:(y'=1);
    [] y=1 & x!=2 -> (y'=2);
    [] y=2 -> 0.5:(y'=2) + 0.5:(y'=0);

endmodule
```

Figure 2.15 Prism DTMC model.

in the control logic, increase readability, and provide an automatic switching from one reconfiguration scenario to another.

Formal modelling using Petri net is an interesting approach in formal verification. IEC 61499 systems have been modelled with NCES and TNCES which are not perfectly suitable for reconfigurable distributed control systems. Reconfigurable timed net condition event system R-TNCES has recently emerged to model and verify such systems. It can be seen as the best formalism that fits well with reconfigurable control systems following the IEC 61499 architecture. Indeed, R-TNCES permits the activation of places/-transitions and the switch from one TNCES to another to execute reconfiguration. Most reconfiguration events are probabilistic since they are related to unpredictable events. A probabilistic R-TNCES extension called GR-TNCES has been developed to model RDCS under memory and energy constraints executing the most probabilistic scenarios. Most RDCS run under limited memory and energy resources. Therefore, translating an IEC 61499 model into a GR-TNCES model grants a modular verification, an estimation of reconfiguration risks, and reduction of the total number of possible states in the system.

Most IEC 61499 software environments integrate simulation and do not deal with model checking that provides an exhaustive analysis. In the formal verification of IEC 61499 control systems, none associate probabilities to reconfiguration scenarios. Such probabilities are useful for evaluating reconfiguration risks and estimating system performance. In the same context, it is crucial to draw on the fact that PRISM model checker

proves its usefulness in allowing qualitative and quantitative analysis using CTL, TCTL and PCTL properties. PRISM also ensures the verification of system safety and reconfiguration feasibility. It gives an insight into system performance. Another worth mentioning idea is that PRISM is nearly not used in other researches to verify systems following IEC 61499 architecture except the research of [Bhatti et al., 2017] which exacerbates the state explosion problem.

2.7 Conclusion

In this chapter, we have defined the main terminologies of the scope. Several related works about IEC 61499 standard, reconfiguration and formal modelling and verification have been discussed. As a result, the related works are not able to deal with reconfiguration in reconfigurable distributed control systems. This is mainly related to the fact that IEC 61499 lacks dynamic reconfiguration inside the execution of the function block control chart. For this reason, an extension of reconfiguration feature in IEC 61499 standard is proposed. Additionally, the existing IEC 61499 tool does not integrate the verification of the control logic. Hence, a simplified approach for design, modelling and verification of RDCSs is elaborated to verify reconfiguration and estimate risks.

Chapter 3

New Extension to IEC 61499 based on Reconfigurable Function Block

3.1 Introduction

In this chapter, a new framework called reconfigurable function block (RFB) is introduced to support reconfiguration in IEC 61499. We have formalised the RFB and its interconnection with other function blocks inside an industrial application. The execution semantic of RFB is presented, and a formal case study is applied to show RFB functionalities. Notably, parts of this chapter are presented in the ACS/IEEE International Conference on Computer Systems and Applications (AICCSA) [Guellouz et al., 2016b], and in IEEE Transactions on Automation Science and Engineering (TASE) journal [Guellouz et al., 2019].

3.2 Motivation

By analysing existing approaches on reconfigurable distributed control systems in the last chapter, a separation between reconfiguration and control logic inside a function block model becomes mandatory. The aim of such separation is to optimise the design and increase readability and maintainability. Another emphasis is put on reducing design complexity and the number of function blocks which minimises the execution and reconfiguration time.

The creation or deletion of FB instances, proposed by [Strasser et al., 2014], facilitates a system reconfiguration. However, these operations are carried out commands executed by a user. That is why we aim at extending the reconfiguration feature in IEC 61499 standard by defining a new reconfigurable function block model called Reconfigurable Function Block (RFB). The latter takes into consideration all possible reconfiguration scenarios related to the changes in the controlled process and switch easily from one reconfiguration scenario to another without user intervention. A self-decision algorithm

called “DcisionAlg” is thus proposed to give the component the possibility to make the best decision without user intervention and select the adequate scenario. Accordingly, we suggest to change the implementation of the execution control chart model and introduce new reconfiguration events and data types in the RFB interface. The proposed solution must offer a dynamic reconfiguration as well as probabilistic annotation for events. The probabilistic aspect is needed to add a degree of uncertainty to events in the verification phase. Thus, it will be possible to evaluate the probability of some unwanted states or scenarios like deadlocks and estimate reconfiguration risks.

Hierarchical state machines fit with the continuous changes in the system since they permit modularity and easy switch from a sub-state machine to another while minimising the overlapping between states. Hence, we propose a master-slave execution control chart to encapsulate each reconfiguration scenario in a sub-state machine slave in RFB. All salves are controlled and supervised by a state machine master. The encapsulation of many reconfiguration scenarios sharing the same events and data, and enabling only one scenario in a reconfigurable function block minimises the number of function blocks and performs efficient reconfiguration management. In addition, a reconfiguration matrix is proposed to endow RFB with a cognitive knowledge base in which rules are stored to select the right reconfiguration scenario to be executed.

An RFB requires a special execution semantic at the resource level. The reconfiguration event must be more prioritised than standard events. The arrival of a reconfiguration event can stop the execution of an old reconfiguration scenario and trigger another. The switching from one scenario to another must be safe. For this reason, the master must wait for the current slave until it returns to its idle state to perform the reconfiguration process.

In this chapter, a network of reconfigurable function blocks is formalised where the interconnection between standard and reconfigurable function blocks is ensured thanks to events and data flow. RFB formalism is the main component which is used to design reconfigurable control systems in the RFBA approach. In order to reconfigure an existing RFB unit, the addition of a new scenario and the deletion or modification of an existing reconfiguration scenario inside an RFB are supported. This facilitates the task for the developer, increases system flexibility, and reduces the development time.

3.3 Reconfigurable Function Block RFB

A reconfigurable function block (RFB) is a new event-triggered software component introduced to control and execute reconfiguration tasks. It separates the reconfiguration model from the control model inside the state execution control chart to avoid cumbersomeness. An RFB is denoted by R_i, as illustrated in Fig. 3.1. It includes an interface that controls events and data flow in which extra reconfiguration events and data are de-

fined. Moreover, RFB defines the logical part as a Master-Slave Execution Control Chart (MSECC). It also has a set of algorithms $Algo$ used in the MSECC to update data.

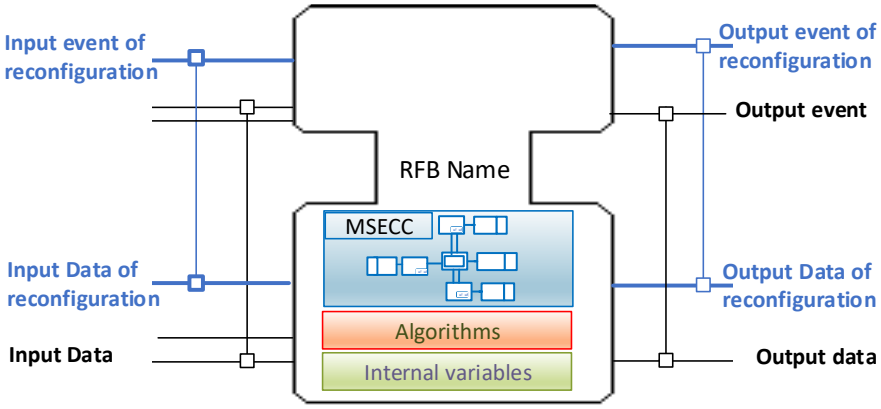


Figure 3.1 A Reconfigurable Function Block Interface.

The RFB interface and master-slave ECC are formalised as follows.

3.3.1 RFB Interface Formalisation

The interface is a set of events and data flow, denoted as:

$$R_i = (E_i, D_i, W_i, IV_i) \text{ where}$$

- E_i is a set of arrived and issued events that trigger the RFB R_i. It includes standard and reconfiguration events: $E_i = (IE_i, OE_i, ier_i, oer_i)$ where IE_i (OE_i, respectively) is a set of standard input (output, respectively) events in R_i. The arrival of an input event can fire a transition in MSECC. The input event of reconfiguration ier_i (the output event of reconfiguration oer_i, respectively) is an event that launches modification in the execution model of the RFB R_i (the next RFB R_m, respectively). A reconfiguration is a dynamic change in the RFB triggered when an event of reconfiguration occurs. It can be a change in the algorithm execution order, a switch from one behaviour to another by activating/deactivating slaves in the MSECC, or an activation/deactivation of next FBs in the network. An event of reconfiguration can be generated when an error occurs, production mode is changed, a product type is added, memory or energy are insufficient, etc.

  Every event is characterised by its priority pr and its probability of occurrence $p \in [0, 1]$. The priority p_r: $ie \rightarrow \{H, L\}$ is a function that prioritises events where H is the highest priority and L is the lowest. The probability p is just for modelling purpose and does not affect the deterministic control logic.

- D_i is a set of data that present the RFB information received from/transferred to other FBs in the network (for example current value, delay, execution period, etc.). The set of data D_i is structured as follows D_i = (ID_i, OD_i, idr_i, odr_i) where ID_i (respectively OD_i) is a set of input (respectively output) data in R_i used by the algorithms of the slave E_s^{ij} and idr_i (respectively odr_i) is the input (respectively output) data of reconfiguration of the RFB R_i that is equal to the index of a slave E_s^{ij} in the RFB R_i (respectively E_s^{mq} in the next RFB R_m) when the corresponding reconfiguration scenario is triggered where $idr_i = ij$ (respectively $odr_i = mq$).

- W_i is a set of WITH-associations of R_i that relate an event with data. We formalise the set of WITH associations W_i of R_i by W_i = (IW_i, OW_i, IWR_i, OWR_i) where

  [(i)] $IW_i \subseteq IE_i \times ID_i$ (respectively $OW_i \subseteq OE_i \times OD_i$) is a set of WITH-associations for inputs (respectively outputs) of R_i,

  [(ii)] $IWR_i = (ier_i, idr_i)$ (respectively $OWR_i = (oer_i, odr_i)$) is the WITH-association between input (respectively output) event and data of reconfiguration. The input (respectively output) event of reconfiguration is associated with the input (respectively output) data of reconfiguration idr_i (respectively odr_i). The reconfiguration data corresponds to the index of the adequate slave E_s^{ij} that will be activated in R_i (respectively R_m).

- IV_i represents a set of internal variables in the RFB R_i used by the algorithms of an ECC slave in E_s^i.

3.3.2 ECC Master-Slave Architecture

State machines play an important role in industrial software design. The execution control chart architecture of an RFB differs from that of a basic function block, and the hierarchical and concurrent execution control chart HCECC syntax [Sinha et al., 2015]. The choice of a master-slave state machine is based on two reasons. Firstly, modular architecture increases readability and maintainability. Secondly, every control behaviour should be separated from the other one.

MSECC contains an ECC master denoted by E_m^i to supervise several elementary ECC slaves E_s as depicted in Fig. 3.2. The ECC master defines the reconfiguration model. It controls the slaves activation/deactivation $E_s^{ij} \in E_s^i$, $j \in [1 \cdots |E_s^i|]$. Notably, it is always sensitive to any change in the system such as user requirements, fault tolerance or environment changes. According to the coming events and data, and the stored rules in a reconfiguration matrix, the master E_m^i smartly decides which is the most suitable reconfiguration scenario, i.e. behaviour, to be activated and executed. At the end of every reconfiguration, the master notifies the interconnected function blocks about the result of reconfiguration using the output event and data of reconfiguration.

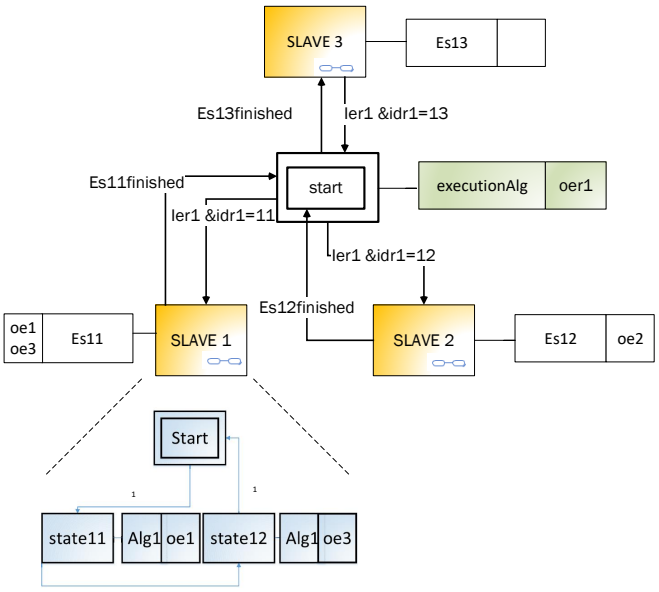


Figure 3.2 Master-Slave Hierarchy.

MSECC also has a set of ECC slaves, denoted by E_s^i. It defines the control model of an RFB R_i. Each slave E_s^{ij} encapsulates a unique behaviour and executes a particular sequence of algorithms. An example of a slave can be a particular state machine for error handling or managing a production mode (low, medium or high).

- The structure of the master E_m^i is formalised as follows:

$$E_m^i = (S_m^i, S_0, Tr_m^i, Cd_m^i, Ac_m^i) \text{ where}$$

S_m^i is a set of states encapsulating a slave. Each state is associated with an action Ac_m^i;

- $S_0 \in S_m^i$ is the initial state. It is associated with an action $A_0 \in Ac_m^i$ which is an operation that activates a slave E_s^{ij}. The action is defined as $A_0 = (DecisionAlgi, oeri)$ where the decision algorithm selects the adequate reconfiguration scenario to be executed and generates an output event of reconfiguration oer_i at the end of the active slave;

$Tr_m^i \subseteq S_m^i \times S_m^i$ is a set of arcs that represent the transitions. Each transition in the master is fired by a condition;

Cd_m^i is a guard condition defined by events and data enabling a slave;

Ac_m^i is a set of actions. Each action contains the name of the slave to be executed E_s^{ij} and the generated output events in the slave.

- E_s^i is a set of ECC slaves E_s^{ij} supervised by E_m^i of the RFB R_i where

$$E_s^{ij} = (S_s, Tr_s, C_s, Ac_s, Alg_s),\ j \in [1 \cdots |E_s^i|].$$

S_s represents a set of states;
$Tr_s \subseteq S_s \times S_s$ is a set of arcs that represent the transitions from a state to another;
C_s is a set of guard conditions defined over input, internal and output variables of R_i;
Ac_s is a set of actions sequences. Each action is related to an algorithm that can change only internal variables and output data of the RFB R_i;
and Alg_s is a set of algorithms related to the slave E_s^{ij}.

After formalising the RFB structure, the next session traces the manner with which a reconfigurable function block reacts to treat the events and selects the most adequate reconfiguration scenario to be executed.

3.3.3 RFB Functionality

A reconfigurable function block [Guellouz et al., 2019] is a model that includes an autonomous master-slave based execution control chart. This hierarchy aims to avoid cumbersomeness by separating the reconfiguration model from the control model. The ECC master E_m^i is always in listening to the coming events. It is activated once an input event arrives. If a reconfiguration event occurs, the "DecisionAlg" (Algorithm 1) reads the associated reconfiguration data and checks the corresponding rule in the reconfiguration matrix (Table 3.2). The latter is a knowledge base including all the possible cases that can happen in RFB. It is defined by combining all events and data ranges included in this component, and then assigning the suitable scenario for each case. The reconfiguration matrix is defined mainly by the designer in cooperation with an expert in the domain.

If the associated input data of reconfiguration is not empty (is issued from a previous RFB), then the master reads the reconfiguration matrix (Table 3.2). The master selects the adequate slave having an index equal to idr_i to execute. If the index of the current slave is different from the index of the selected slave, then a reconfiguration is required. Accordingly, the master waits for the current slave to return to its initial state to execute the adequate reconfiguration scenario. Before activating the corresponding slave E_s^{ij}, the index of the current slave is saved.

If a standard event occurs and idr_i is zero, then the decision algorithm checks the rules in the reconfiguration matrix related to the coming events and data. The decision algorithm updates the input data of reconfiguration idr_i got from the reconfiguration matrix. After that, the corresponding slave will be executed.

The slave E_s^{ij} attempts to (i) select and schedule the algorithms sequence to execute, (ii) update the internal and output data, and (iii) send output events at the end of execution.

When the slave E_s^{ij} finishes its execution, it notifies the master about its status. It sends an event indicating the execution termination of all the encapsulated algorithms. Finally, the master updates the output data of reconfiguration depending on the matrix and generates an output event of reconfiguration that ensures the communication with the next RFBs. The output data of reconfiguration includes the index of the next slave to be executed in the next RFB.

Algorithm 1: DecisionAlg to Execute the Best Scenario according to the Inputs.

Input: E_s, IE, ID, idr_i, ier_i
Output: oer_i, odr_i
listen events;
Read current data;
Read the corresponding line in the reconfiguration matrix;
$currentSt$ ←GET current slave state;
$currentSlaveId$ ←GET current slave id;
save $currentSlaveId$;
if ier_i & *(associated* $idr_i \neq 0$) & $idr_i \neq currentSlaveId$ **then**
    while *isInitialState(currentSt)* **do**
        execute E_s^{i,idr_i};
        $id = odr$ from the reconfiguration matrix;
        $odr_i \leftarrow id$;
    end
end
else
    if $idr_i = 0$ **then**
        $idr_i \leftarrow$ index from the matrix;
        Execute E_s^{i,idr_i};
    else
        if *error* **then**
            HMI message ← "Reconfiguration is not possible";
        end
    end
end
if E_s^{i,idr_i} *finishes* **then**
    E_s^{i,idr_i} send a notification to the master;
    if *Master receive error from slave* **then**
        Rollback to $currentSlaveId$;
    end
    else Master send oer_i, $odr_i \leftarrow id$;
end

Let assume two RFBs connected via events and data of reconfiguration as shown in Fig 3.3. The reconfiguration matrix of RFB R_2 is shown in Table 3.2.

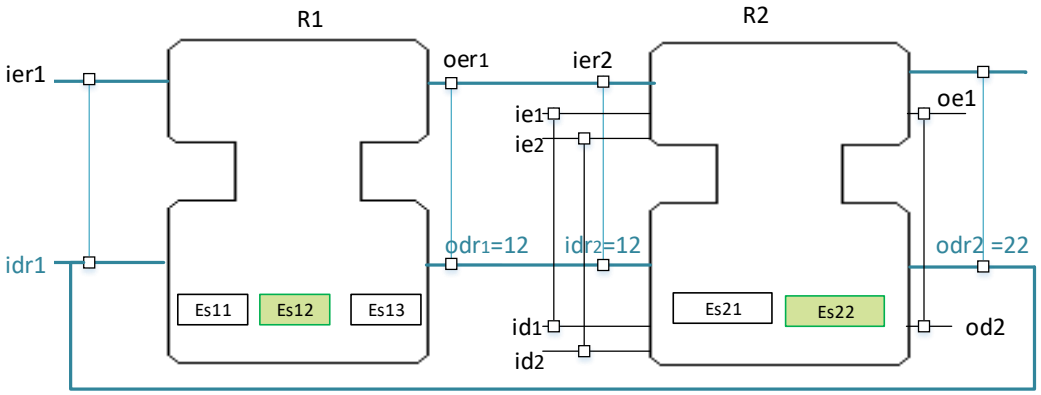


Figure 3.3 RFBs Network (R_1 and R_2).

Table 3.1 Reconfiguration matrix of the RFB R_1.

Rules	Inputs		Outputs	
	ier_1	idr_1	oer_1	odr_1
1	1	11	1	21
2	1	12	1	22
3	1	13	1	22

Table 3.2 Reconfiguration matrix of the RFB R_2.

Rules	Events			Data		Reconfiguration	
	ie_1	ie_2	ier_2	id_1	id_2	idr_2	odr_2
1	0	0	1	0	0	21	31
2	0	0	1	0	1	22	32
3	0	1	0	1	0	0	31
4	0	1	0	1	1	0	31
5	1	1	0	1	1	0	32
6	0	1	1	0	1	22	32

Three cases in the reconfiguration matrix can happen:

1. Case of an input event of reconfiguration coming from other RFB (ier_2 =1): if ier_2 occurs with $idr_2 = 12$ in the local matrix R_2 coming from the RFB R_1, then the master of RFB R_2 executes the suitable slave E_s^{12}, and then generates an output data of reconfiguration odr_2 equal to 22.

2. Case of no input event of reconfiguration ($ier_2 = 0$): RFB analyses the current data and compares them with the previous ones. If it is equal to (or in the same range as) the current data, no reconfiguration is required. If there is a big difference, it goes to the local data stored in the reconfiguration matrix (line 3, 4, 5) and selects the best scenario index predefined by experts, as illustrated in Table 3.2.

3. Case of two concurrent events: like in line 7 in the reconfiguration matrix, an input event of reconfiguration ier_2 is coming simultaneously with an input ie_1. The input event of reconfiguration is always more prioritised than standard events. Therefore, slave 22 will be executed like in the case 1. If there is no rule for standard events in the matrix, then they are always executed with low priority.

An RFB activates the best control behaviour thanks to the dynamic switching from one reconfiguration scenario to another. Once the execution of the suitable slave $E_s^{i,idr}$ finishes, the master is notified and a reconfiguration output event oer_i associated with the output data of reconfiguration odr_i is issued. It propagates the events and data to other function blocks in the system to notify them about the changes. Accordingly, reconfiguring the rules in the reconfiguration matrix is easier than reconfiguring the state machine directly.

3.3.4 Execution Semantic and Event Consumption Policy

Several execution models are proposed in IEC 61499 Run-time Environment (RTE) since the standard does not specify any execution semantic. A buffered sequential execution model (BESM), non preempted multi-threaded resource (NPTMR), preempted multi-threaded resource (PTMR), and cyclic buffered sequential model (CBEM) [Kim et al., 2018, Prenzel et al., 2019] are suggested as discussed in Section 2.2.6 Chapter 2. The fact that the existing semantics do not classify and prioritise reconfiguration events motivates us to propose a preempted multitasking execution semantic. The latter assigns the highest priority to the reconfiguration event and performs the preemption of a slave execution only when the slave is in the idle state.

A multi-thread model of invocation is implied in the RFB framework: a thread for executing the master and another thread for executing the corresponding slave. Only the master with one slave can be concurrently executed. Executing multiple slaves at the same time is prohibited. Furthermore, the switching from one slave to another requires that the new thread waits until the end of the current thread execution.

Regarding the event scheduling policy in the resource model, all external and internal events are stored in a global FIFO event buffer with the calculated order and priority. The events are consumed by the respective FBs and RFBs in a sequential manner according to their priorities. Two levels of priority are introduced in the buffer: (i) a high-priority level

for the reconfiguration events, and (ii) a low-priority level for the remaining events. The sequence of emitted events is preserved in the invocation order of target FBs or RFBs with giving high-priority to reconfiguration events. Indeed, the events prioritisation is ensured by a scheduling function SF executed in the resource model. The scheduling function assigns priorities to the coming events according to their event types, and after that sorts the events according to their arrival time and priorities, i.e. assign them orders after sorting. If a reconfiguration event occurs in parallel with a standard input event, the scheduling function SF gives the reconfiguration events a low order in the queue and then calls the master to execute the decision algorithm. The scheduling function also executes the algorithms in the defined ECC slaves. It ensures that each phase of the reconfigurable function block execution occurs in the correct order. A reconfiguration event can be consumed from the queue only when the ECC master reads it, and the corresponding event action is executed. Once the action is accomplished, then the event will be removed from the buffer.

3.4 Reconfigurable System

A reconfigurable distributed control system Sys is a network of $|Sys|$ function blocks deployed in several devices. The included function blocks are interconnected together with their events and data as shown in Fig. 3.4, i.e., Sys = $\{F_1, \cdots, F_{|Sys|}, LinkE, LinkD, LinkEr, LinkDr\}$ where $|Sys|$ is the cardinality of Sys. A function block F_i can be a basic, composite or service interface function block (BFB, CFB, SIFB) or a reconfigurable function block RFB where $i \in [1 \cdots |Sys|]$.

RFB is linked with standard FBs by events in $LinkE \subseteq OE_i \times IE_j$ and data in $LinkD \subseteq OD_i \times ID_j$, whereas RFB is linked with another RFB by reconfiguration events in $LinkEr$ = (oer_i, ier_j) and reconfiguration data in $LinkDr$ = (odr_i, idr_j).

RFB has a particular relation with SIFB since it is a specific function block that can access hardware and external changes. RFB can directly read events an data from basic, composite and service interface function blocks and generates output events of reconfiguration. At the end of each reconfiguration scenario that is encapsulated in a slave E_s^{ij}, each reconfiguration output data contains the index of the executed slave to determine the next slave E_s^{mk} in the next RFB R_m where $k \in [1 \cdots |E_s^m|]$. When its value is updated, the associated reconfiguration output event occurs to trigger the suitable RFBs.

It is plausible at this stage to detail the dynamic part of the system using RFB which is featured by

$$Dynamic(Sys) = (\mathcal{B}_{sys}, R_{sys})$$

where $\mathcal{B}_{sys}$ is a set of $|\mathcal{B}_{sys}|$ system behaviours and R_{sys} is a set of reconfiguration func-

tions. A behaviour $\mathcal{B}_u = \{s_i, s_j, \cdots, s_w\}$, $u \in [1, |\mathcal{B}_{sys}|]$, $i, j, w \in [1, |s_{Sys}|]$, is a set of organised states, encapsulated in RFB slaves or FBs, to be executed for adapting the system to the environment change. A behaviour can be depicted as a path in the system, as shown in Fig. 3.4.

A reconfiguration function $r_f \in R_{sys}$ deals with the automatic transformations of the execution model from a set of states to another set in response to the changes caused by faults, or user requirements via enabling/disabling control components. It permits to guarantee the system correctness after reconfiguration. It is characterised by r_f = (Cd_f, S_f, U_f), $f \in [1 \cdots |R_{sys}|]$, where

(i) Cd_f is the condition for the reconfiguration which is an input event of reconfiguration associated with data inputs;

(ii) S_f: $\mathcal{B}_a \rightarrow \mathcal{B}_b$ the structure defining the reconfiguration from the behaviour $\mathcal{B}_a$ to another behaviour $\mathcal{B}_b$, ($a, b \in [1 \cdots |\mathcal{B}_{sys}|]$);

(ii) U_f is the initial state of $\mathcal{B}_0$ before reconfiguration.

The system Sys changes its behaviour to another one when an input event of reconfiguration occurs to an RFB in the system and the condition of the chosen slave E_s^{ij} is fulfilled, where $i \in [1 \cdots |Sys|]$ and $j \in [1 \cdots |E_s^i|]$. Moreover, the reconfiguration is executed only when the scheduling function checks the available memory and energy in a device. These resources should not be violated during the reconfiguration process.

We assume that the execution time for a path T_p is the result of adding the sum of RFBs' execution time T_{RFB} with the sum of the execution time for each function block in the path:

$$T_p = \sum T_{RFB_j} + \sum T_{FB_i} \text{ where}$$

T_{RFB} is the time of RFB execution. $T_{RFB} = T_{DAlg} + T_{ID} + T_{IE} + T_{executedSlave} + T_{OD} + T_{OE}$ with:

- $\diamond$ T_{DAlg} time of decision algorithm;
- $\diamond$ T_{ID} time of reading associated input data;
- $\diamond$ T_{IE} time of input event arrival;
- $\diamond$ $T_{executedSlave}$ time of slave execution which is $T_{executedSlave} = \sum T_{Alg} + T_t$ where T_t the time of transitions;
- $\diamond$ T_{OD} time of updating associated output data;
- $\diamond$ T_{OE} time of generating output events;

In Fig. 3.4, the time of path $T_{Path1} = T_{RFB1} + T_{FB1} + T_{FB3} \simeq T_{slave3} + T_{FB1} + T_{FB3}$

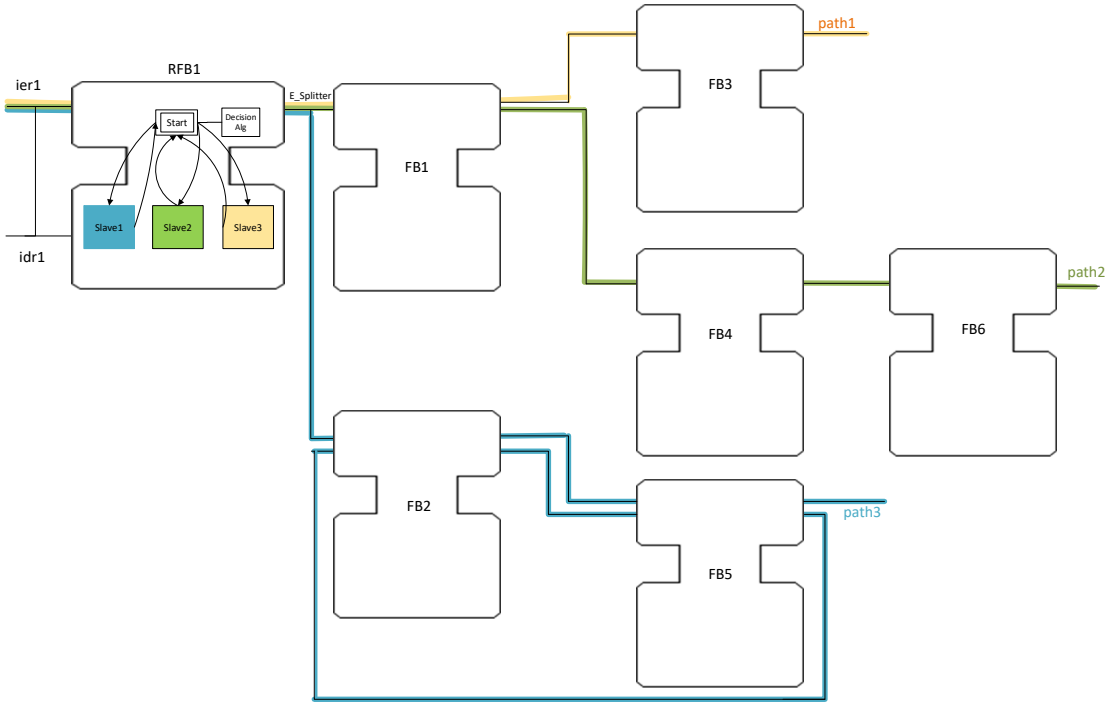


Figure 3.4 Reconfigurable Function Block Network showing the possible paths that can be generated after reconfiguration.

3.5 Formal Case Study RFB model

A simplified surgical robotised system BROS [Ben Salem et al., 2016] is illustrated to show RFB functionalities. A pinning robot PBROS fixes one or two pins in the bone to correct a supracondylar fracture. The number of pins depends on the fracture displacement that defines the fracture type. However, if the fracture type is equal to IIB or IIC, then PBROS needs a single pin to fix the fracture. Otherwise, if the fracture type is equal to IIA or III, then the PBROS requires double pins to fix the fracture.

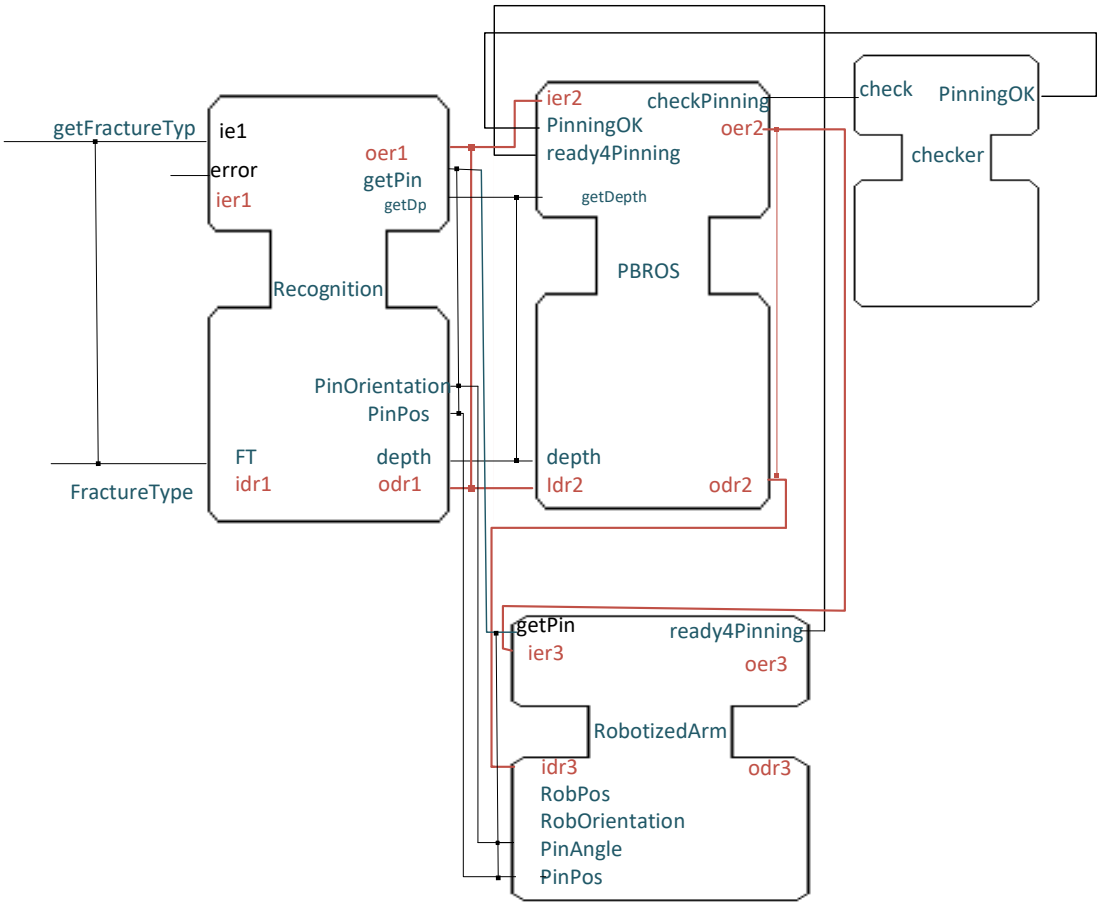


Figure 3.5 PBROS application model based on RFBs.

The control application is modelled by a network of RFBs and BFB as shown in Fig. 3.4. The standard events and data are marked in blue, while the reconfiguration events and data are marked in red. The recognition system in BROS determines first of all the fracture type by imagery and sends it to $Recognition$ RFB via the network. The $Recognition$ RFB has a standard input event $ie1$ that reads the fracture type FT from an imagery controller, and then calculates the coordinates of pinning $PinPos$ (X, Y, Z), the angle to determine the pin orientation $PinOrientation$, and the pin depth $depth$. It sends $PinOrientation$ and $PinPos$ to $RobotizedArm$ RFB to move the robotised pinning arm to the pinning position and rotating it to be in the right orientation. In $Recognition$ RFB, the decision algorithm executes the best scenario according to the fracture type. It reads the reconfiguration matrix and emits an output event of reconfiguration $oer1$ associated with $odr1$ data of reconfiguration to the RFB $PBROS$ according to the fracture type FT. The output data of reconfiguration $odr1$, can be equal to 1, 2 or 3, is the corresponding slave indexes as shown in the reconfiguration matrix in Table 3.3. The pinning arm $PBROS$ reads the value of $idr2$ and determines which scenario to execute after receiving $ready4Pinning$ from the $RobotizedArm$. The $PBROS$ has three ECC slaves controlled by an ECC master: (i) slave1 for single pinning, (ii) slave2

for double pinning and (iii) slave3 for manual pinning. The third slave is performed when an error occurs, such as the defected recognition system case or when a fracture type or coordinates cannot be provided. Slave1 is encapsulating the single pinning algorithms that require the depth of the pin to pierce. In fact, it pierces one pin if the fracture type is equal to IIB or IIC. If the fracture type is IIA or III, then the ECC master will execute the double pinning scenario (slave2).

In the case of double pinning, $PBROS$ should be executed twice with different coordinates $PinPos$ and $PinOrientaion$. After first pinning, a basic function block $checker$ checks the pinning. If it is successful, it sends $pinningOk$ to $PBROS$.

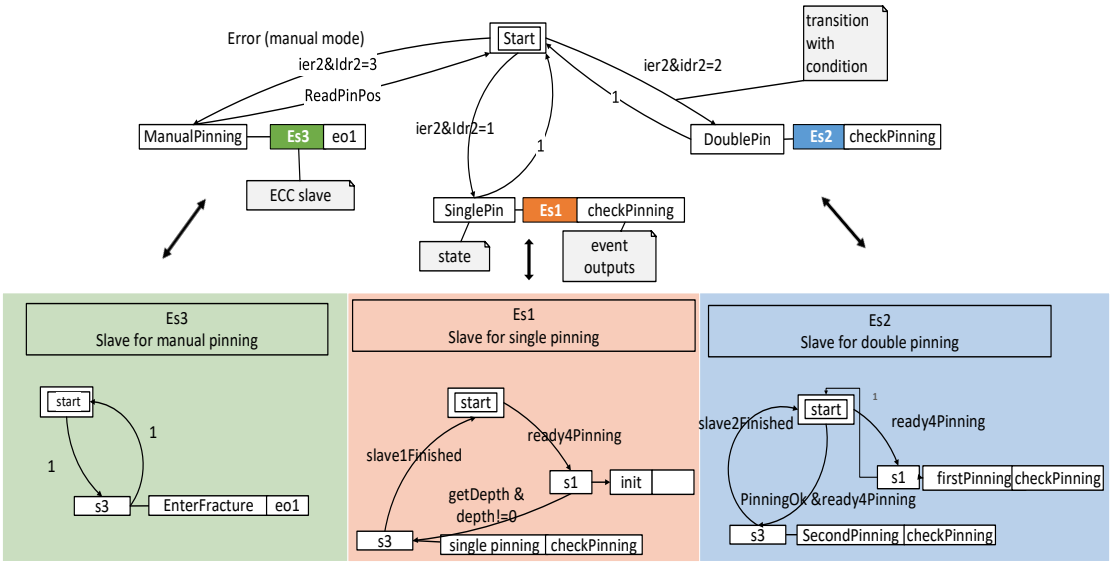


Figure 3.6 The internal architecture of $PBROS$ RFB.

Table 3.3 Reconfiguration Matrix of $Recognition$ RFB.

	Events			Data		Reconfiguration	
Rules	ier_1	$ie1$	$error$	$mode$	FT	oer_1	odr_1
1	0	1	0	Auto	IIA	1	2
2	0	1	0	Auto	IIB	1	1
3	0	1	0	Auto	IIC	1	1
4	1	1	0	Auto	III	1	2
5	0	1	1	Manual	-	1	3

The application model can be deployed in several devices interconnected via a network. $Recognition$ RFB can be deployed in a control processing unit CPU, and $RobotizedArm$ and $PBROS$ in the PBROS arm. Publisher/Subscriber SIFBs are required to send and

receive data via a communication network such as the fracture type and the pinning position. The *subscriber* FB will receive the information published by the *publisher*. In this example, the *publisher* in the CPU sends the $PinPos$ to the $subscriber$ in PBROS that is supposed to read this information and send it to the RFB $RobotizedArm$.

Using RFB components, we do not need two PBROS FB instances in the model and four *subscribers/publishers* block to read the pinning coordinates. As a result, the model based RFBs reduces the number of function Blocks and system complexity.

3.6 Dynamic RFB reconfiguration

After code compiling and system deployment, several changes can be required and must be done online, i.e. without shutting down machines. Such changes incorporate adding new system mode or functionality, deleting a product from the process, or changing production level.

To meet this need, we propose to reconfigure MSECC that can be remotely and automatically configurable. A reconfiguration scenario can be added to (updated, activated, deactivated or deleted from, respectively.) the MSECC using a remote configuration option. An event of addition (deletion, respectively) is executed to add a new (delete an existing, respectively) scenario. The system designer can go to the remote configuration module, design the scenario, assign it to an RFB, verify its feasibility in the application, and then deploy it online in the controller device. Three event types are defined as depicted in Fig. 3.7:

1. An event called “AddSc” allows to add a new reconfiguration scenario, i.e. ECC slave, in an existing RFB;
2. An event called “deleteSc” to delete a not useful reconfiguration scenario (ECC slave) from RFB;
3. And an event “UpdateSc” to change an existing scenario (algorithm, variable limits, etc.).

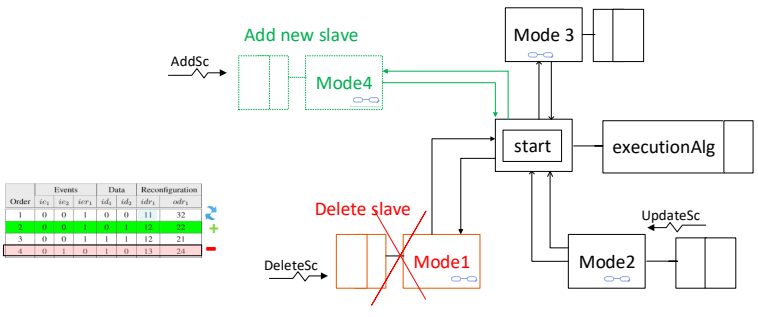


Figure 3.7 Reconfiguration scenarios Management.

The dynamic reconfiguration can be easily done thanks to the reconfiguration matrix that is a table in a local database including rules. The rules can be optimised automatically.

3.6.1 Adding a New Reconfiguration Scenario

Adding a new scenario to an RFB in the application is not time-critical. In order to add a new reconfiguration scenario, a tuple is defined $addSc$ = $\{slave_{id}, events, events_{source}, data, data_{source}, condition2Master\}$. If RFB receives an "AddSc" event request:

1. The ECC master waits for the RFB to be in the initial state,
2. Then, if it is in the initial state, all coming events must be stored in the buffer and wait for the end of reconfiguration. The master adds events and data to RFB interface,
3. Connect events and data to their sources (add if it is a necessary event function block like $ESPLIT$, E_{merge} between RFB and FB source),
4. Define the ECC slave $slave_{id}$,
5. Add the new index to the reconfiguration matrix,
6. Add a new line to the reconfiguration matrix in the current RFB containing events and a slave index,
7. Connect slave to ECC master and define fulfilled condition $condition2Master$,
8. Change the related RFBs: add new lines to reconfiguration matrix if necessary, new slaves, etc.
9. Relaunch RFB.

3.6.2 Removing an Existing Reconfiguration Scenario

Removing an unwanted scenario will speed up the execution time on run-time environment. However, it is time-critical. If the master receives "DeleteSc" event request, $delSc$ = $\{RFB_{id}, slave_{id}\}$, the ECC master must:

- Disconnect ECC slave $slave_{id}$ from the master of RFB_{id},
- Remove extra events and data which are no more needed in the system,
- Delete slave index from the reconfiguration matrix. The fact of removing the dir and dor indexes disables all related slaves automatically in other RFBs.

3.6.3 Modifying an Existing Reconfiguration Scenario

To change an existing scenario $slave_{id}$ in RFB_{id}, the designer can modify events, data, conditions, and algorithms. He can even add or remove states to a slave. An event "UpdateSc" is sent to stop the RFB in the idle state, store events in the buffer and change the selected slave. $UpdateSc(RFB_{id}, slave_{id})$ that calls the required function *AddState*($slave_{id}$), *DeleteState*($state_{id}$, $slave_{id}$), *addAction*($state_{id}$, $slave_{id}$), *UpdateAlgorithm*(Alg_{id}, $slave_{id}$).

3.6.3.1 Example of reconfiguring an RFB

In the case of discovering new fracture type in BROS system that requires three pins, the developer just needs to configure the reconfiguration matrix of $Recognition$ and $PBROS$ by adding a new row containing the new fracture type, $idr = 4$ for a new slave 4.

3.7 Comparison of RFB with BFB

Let assume that the main conveyor can distribute the workpieces to conveyor A or conveyor B as depicted in Fig. 3.8. If the coming workpiece is "A", then it should be moved to the conveyor A. Otherwise, if the coming workpiece is "B", then it should be moved to the conveyor B.

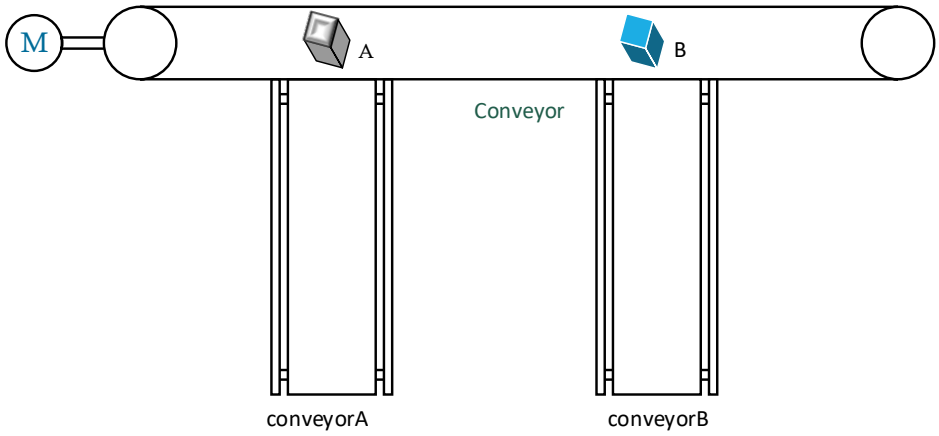


Figure 3.8 Conveyor Model using RFBs.

RFBMode includes two modes: "Move2ConveyorA" and "Move2ConveyorB". It switches fastly from a mode to another. Using basic FBs, we need three function blocks. A function block for mode selection, another function block to move to the conveyor A, and the third one to move to the conveyor B.

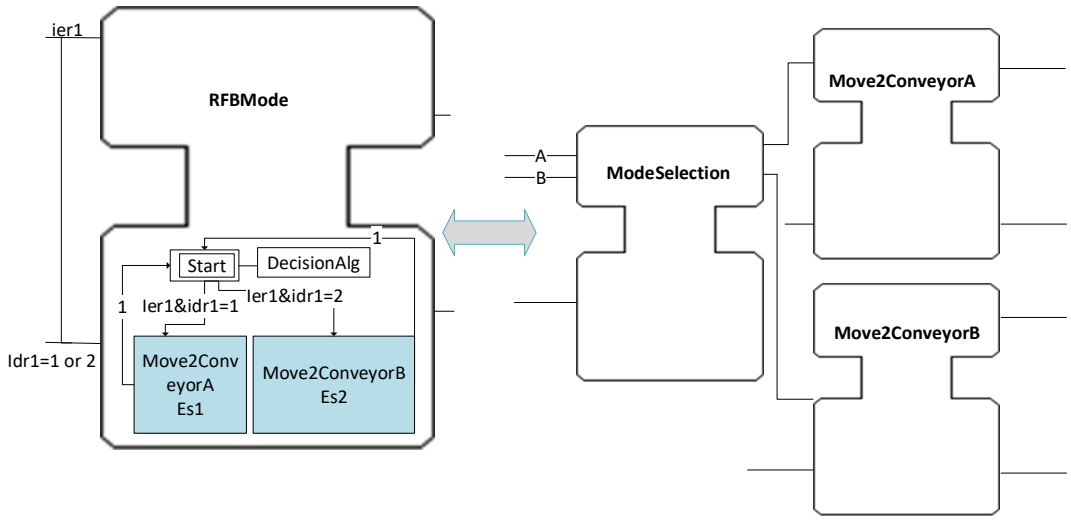


Figure 3.9 Comparison Between RFB and BFB Networks.

In comparison with the network of basic function blocks, the number of states, transitions, events and data in the RFB model is lower than the number in function blocks network. Moreover, the RFB execution time is also lower than the function blocks network's time, as shown in Fig. 3.10. It is equal to the sum of decision time T_{DAlg} and the time of the executed slave. While, the time of the function blocks network T_{BFBN} is equal to the sum of $T_{ModeSelection}$, the maximum time to execute function block $Move2ConveyorA$ or $Move2ConveyorA$, and the time of communication between function blocks. The time of scheduling is nearly the same for both the RFB and BFB blocks.

$$T_{RFBMode} \simeq T_{DAlg} + Max(T_{slave1} + T_{slave2})$$

$$T_{BFBN} = T_{ModeSelection} + Max(T_{Move2ConveyorA} + T_{Move2ConveyorB}) + T_{communication}$$

Using RFB, we can avoid the time of switching between RFBs and the time of decision.

	Components	states	Transitions	Events	Data
BFBs network	3	23	18	5	5
RFB model	1	20	15	2	2

Table 3.4 Comparison between the RFB model and the BFBs network model.

Adding a new reconfiguration scenario (adding a new mode, a product, or new conveyor) to the conveyor model is more manageable than adding new function block type and connecting them to the FB network. We just need a new slave, connect it with ECC master and finally add a reconfiguration matrix row.

For example, the designer need to add a new product type "C" that should be moved to the conveyor A. In this case, thanks to the remote configuration of the reconfiguration matrix, an "$addSc$" request is sent to add a new line in the reconfiguration matrix of

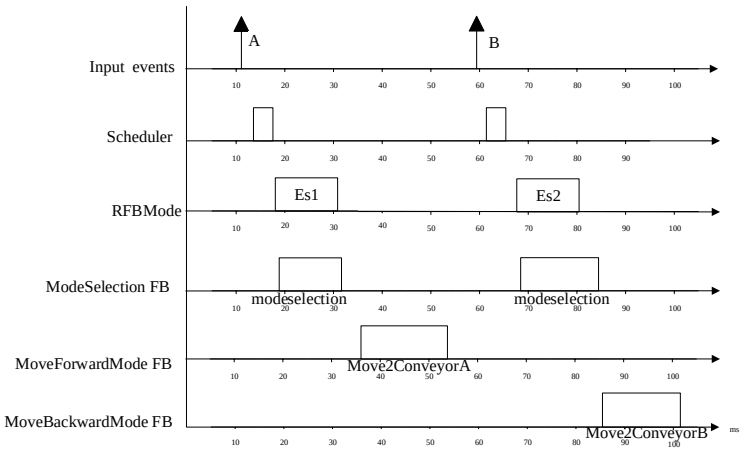


Figure 3.10 Execution Scheduling Comparison.

$RFBMode$. Thus, a new rule indicating that workpiece "C" corresponds to slave 1 is added.

3.8 Discussion and Originalities

As we can see in the related works, none of the mentioned researchers have proposed a way for modelling reconfiguration scenarios within the function block unit. In this work, a reconfiguration scenario is assumed to be any run-time automatic operation that modifies the behaviour of the system. The activation of a scenario depends on new types of events and data named event/data of reconfiguration as well as a set of rules stored in the reconfiguration matrix. It allows to describe reconfiguration scenarios in slaves, and switch fast from one slave to another thanks to a Master-slave execution control chart and a decision algorithm. The reconfigurable function block has multiple advantages compared to a composite function block and basic function block. CFB is a static component that encapsulates many FBs, while BFB has also a static execution control chart combining the control with reconfiguration level. An RFB allows several ways of functioning thanks to reconfiguration. Hence, it reduces the number of FBs used in the design as well as its complexity. Additionally, it minimises the number of events, data, states and transitions. An RFB facilitates the design of reconfigurable distributed automation systems. Moreover, the rules in the reconfiguration matrix can be changed online. To optimize the network of FBs and make it more adjustable to external changes, we propose an RFB network that provides a simple model of reconfigurable systems. An RFB can be related to BFB standard. Clearly, the time of a path execution is lower than the time using BFBs.

3.9 Conclusion

There is now considerable concern about reconfiguration in IEC 61499. Hence, a new reconfigurable function block is proposed. It is different from BFB and CFB as it puts forward a master-slave execution control chart architecture and reconfiguration events. This function block, based on a decision algorithm and the occurred events, is responsible for executing the adequate reconfiguration scenario. It guarantees a fast switching from one scenario to another while reducing the number of function blocks, states, events, and transitions.

Chapter 4

Formal Modelling of Reconfigurable Distributed Control Systems based on RFBs

4.1 Introduction

RFBA methodology seeks to design the system mainly with RFBs. In order to verify the proposed pattern with model checking technique and mitigate the state space explosion problem, we propose to translate the RFB model automatically into a generalized model of reconfigurable timed net condition/event systems (GR-TNCESs). Therefore, a set of transformation rules is defined in this chapter. Furthermore, a new software environment called RFBTool is developed to support the RFBA approach from design to verification.

4.2 Motivation

IEC 61499 standard cannot ensure correctness and coherence of states such as simultaneous transition and deadlocks. The simulation technique supported in the existing IEC 61499 compliant software might provide wrong results and affect negatively the target application code. Moreover, analysing qualitative and quantitative properties to verify IEC 61499 model is a crucial need for companies and engineers. At the best of our knowledge, none focuses on combining qualitative and quantitative verification in IEC 61499 model except the work of [Bhatti et al., 2017]. The authors in this work propose to convert directly IEC 61499 model into a specific PRISM model, and then inject some faults in the model to verify safety properties using the model checker PRISM. However, the direct translation from IEC 61499 to a Markov chain aggravates the state explosion problem for large scale systems. Indeed, the total verification time increases exponentially with the total number of possible states in the system. For this reason, a modular verification is recommended for complex systems.

Additionally, modelling and verification are well-known approaches in the distributed control systems. Petri nets are extensively used in the modelling and analysis of discrete event control systems [Ivanova-Vasileva et al., 2008, Pang and Vyatkin, 2008, Zhang et al., 2013, Zhang et al., 2018a, Khlifi et al., 2019]. Compared with relevant studies in Petri nets, GR-TNCES formalism [Khlifi et al., 2015] models the dynamic behaviour of a reconfigurable distributed event control system with less number of places and transitions. The main reasons for selecting GR-TNCES from several other formalism can be summarised in four points. The first point is the modularisation where the formalism is based on control components. The second point is the reconfiguration. In GR-TNCES, it is possible to switch from one configuration to another by enabling or disabling components, updating token marking, and changing places/transitions on run-time. The third point is the probabilistic aspect attached to the transition arcs which is needed for performance estimation. Each TNCES in GR-TNCES can be characterised by a probability of occurrence assigned on the arc, while in RFB formalisation, a reconfiguration event can be attached with a probability. The last point is the possibility to check memory resources and energy before executing a reconfiguration scenario which allows to guarantee system safety. In the light of the above mentioned points, GR-TNCES fits perfectly with the reconfigurable function block system. Accordingly, we are dealing with an automatic transformation from an IEC 61499 model based on RFBs to a GR-TNCES model based on RFBA methodology.

4.3 RFBA Approach

We propose an RFBA approach for the design, modelling and verification of reconfigurable distributed control system following the new extension to IEC 61499. This approach is detailed in the flow diagram in Fig. 4.1. It recapitulates how the system is firstly specified, designed with RFBs, then transformed into a GR-TNCES model, after that verified using PRISM model checker.

In the specification step, the designer captures all the possible reconfiguration scenarios that can happen in the system and defines them. After that, RFBA permits to design the captured scenarios with RFBs pattern. Additionally, for verification purposes, RFBA converts automatically the designed RFB model to a generalised model of reconfigurable timed net condition/event system GR-TNCES that makes it possible for the designer to add probabilities to several events.

We characterise every input event of reconfiguration with a probability of occurrence since most reconfiguration events are unpredictable and stochastic. They can be caused by errors, weather, external environment changes, insufficient energy or resources, etc. Probabilistic estimation leads to an improvement in the design and enhancement in the system performance. We assume that they can be a Poisson mean arrival $1/\lambda$.

Finally, a combined verification approach based on qualitative and quantitative analyses is presented. It helps the designer and developer to refine the system model and estimate how much the model satisfies the functional specifications. It is based on a probabilistic model checking ensured by the probabilistic symbolic model checker PRISM that provides and analyses such results. In this step, the generated GR-TNCES model is converted into a Discrete-Time Markov Chain (DTMC) analysed by PRISM. The designer can specify several properties using computation tree logic for qualitative analysis and probabilistic computation tree logic for quantitative analysis. If all the defined properties are satisfied, then the verification is successful and the software code can be deployed. Otherwise, the designer should refine the initial model.

The approach is supported by a toolchain composed of a new environment called RFBTool [Guellouz et al., 2016a], ZiZo v2 [Khlifi et al., 2015] and PRISM [Kwiatkowska et al., 2002]. RFBTool is an environment developed to create and edit reconfigurable function blocks model and transform it automatically to a GR-TNCES model. It implements also the RFB execution model. ZiZo v2 reads the generated GR-TNCES model, simulates and converts it to a DTMC model.

The detection of the unfavourable cases before deployment is a major benefit of the approach that engineers need to estimate and refine the designed model. Several properties are easily verified and estimated such as deadlock freedom, system feasibility, confluence, estimation of the reconfiguration failure, and system availability.

4.4 Formal Modelling

The system design based on RFBs pattern is converted into a GR-TNCES model to verify and validate the system on run-time. The architecture, behavioural semantics and reconfiguration of an RFB system are very similar to a GR-TNCES model. In fact, the event-driven, modularisation, reconfiguration, and probabilistic aspect of GR-TNCES comply with the RFB formalism and its semantic. This motivates us to generate automatically an optimised GR-TNCES model of the input system. Thanks to this typical Petri net, memory and energy constraints can also be verified before executing a reconfiguration.

GR-TNCES formalism is considered to be more beneficial than automata, Colored Petri Nets (CPNs) and other Petri nets since it inherits R-TNCES properties. R-TNCES is based on control components which are software units encapsulating algorithms which implement functionalities to control a physical process and interact with other processes as follows: (i) it reads data from sensors, (ii) executes the appropriate algorithm that needs these data, and (iii) upon the completion of the algorithm execution, event signals are either sent to the corresponding controlled actuators in order to activate them, or to be sent to the next control component [Zhang, 2015]. Each control component is a TNCES module with temporal constraints. As discussed in [Zhang et al., 2018b], R-TNCESs are

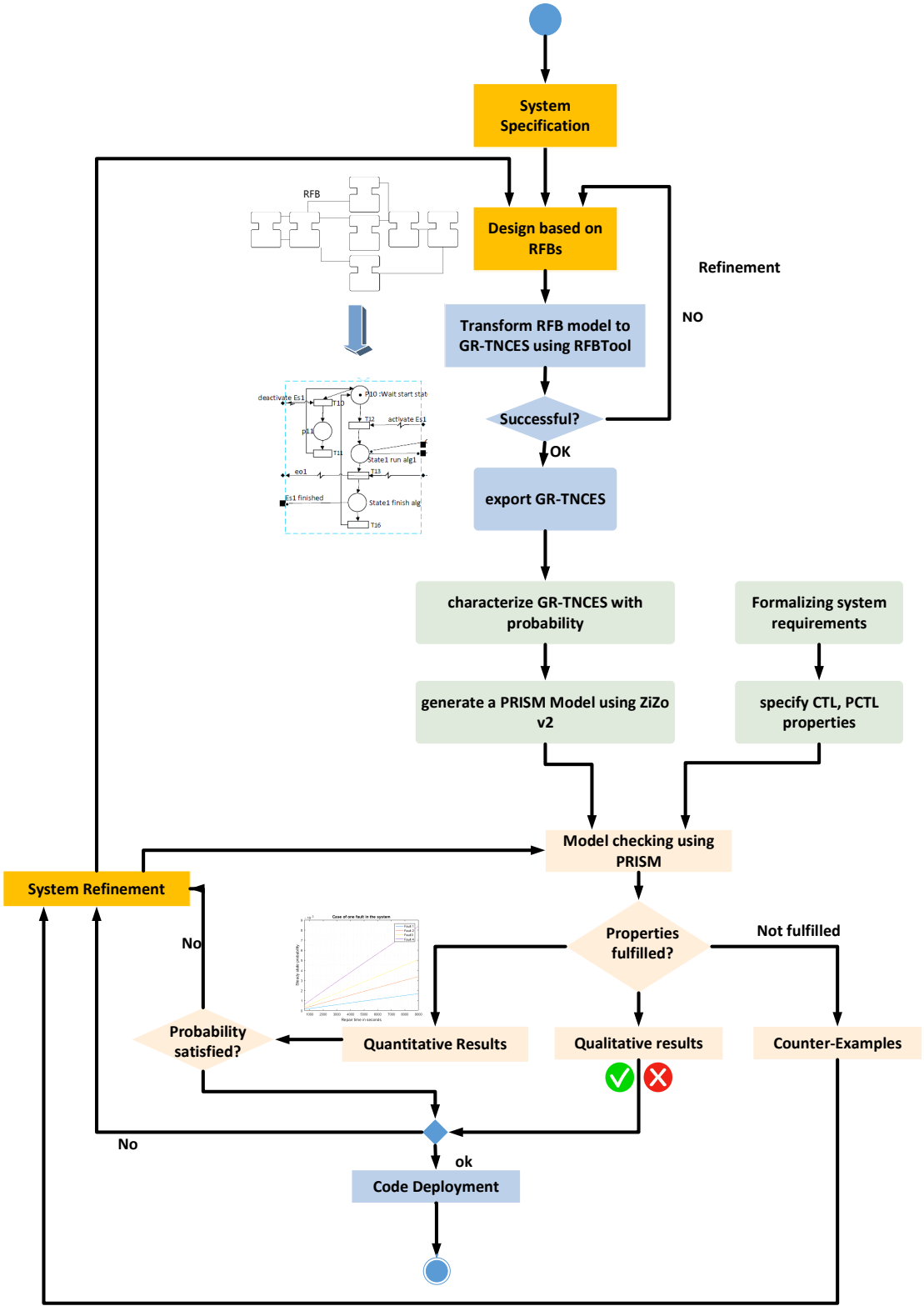


Figure 4.1 Detailed RFBA Methodology Diagram.

characterised by their dynamic reconfiguration process, their intuitive and direct structure change, and their ability to guarantee the state coherence during the design. They also offer the possibility of events concurrence such as the occurrence of standard and reconfiguration events, and the possibility of assigning an interleaving time for the reconfiguration events.

A GR-TNCES contains places modelling states, and transitions linking a state to another. A transition associated with event can fire only when this event occurs and the required conditions are fulfilled. In RFB interface, RFB data can be considered as GR-TNCES conditions, whereas RFB events can be considered as GR-TNCES events, as shown in Fig. 4.2.

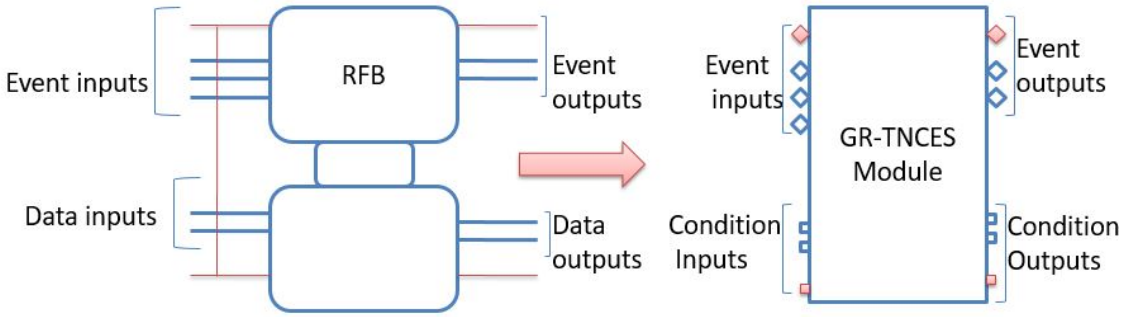


Figure 4.2 RFB interface transformed into a GR-TNCES module

A GR-TNCES, denoted by $G = \{g_1, g_2, \cdots, g_n\}$, is a set of $|G|$ R-TNCES g_i, where g_i = (B, R) with B being the behaviour module and R being the control module which is a set of reconfiguration functions. The behaviour module B is a union of multiple TNCES N_i as explained in Chapter 2 Section 2.5.4.

$$\cup N = (P \times T \times F \times W \times CN \times EN \times DC \times V)$$

$\cup N$ is a set of p feasible net structures that can be performed by an R-TNCES. Each $N_i \in \cup N$, $i \in [1 \cdots |N|]$, is a behaviour in the GR-TNCES which corresponds to an ECC slave E_s^{ij}.

We propose in the next section the transformation rules needed to translate an RFB model into a GR-TNCES model.

4.4.1 Transformation Rules

4.4.1.1 Rule 1: RFB Transformation Rule

Each RFB in the system is modelled as a GR-TNCES module T_{R_i} where T_{R_i}= $\{T_{Interface}$, $G_{Emaster}$, $T_{ESlaves}\}$ with $T_{Interface}$=$\{T_{E_i}$, T_{D_i}, T_{W_i}, $T_{IV_i}\}$ detailed in the next rules. The module T_{R_i} = (B, R) is composed of a behaviour module B and a set of reconfiguration functions R where

$$B = (P, T, F, W, CN, EN, DC, V, Z_0),$$
$$R = \{r_1, r_2, \cdots, r_{|R|}\} \text{ and } r_k = (cond, P_0, E_0, M_0, Q, L)$$

The RFB behaviour B module is the interconnection of the transformed RFB components $T_{Interface}$ and $T_{ESlaves}$ whereas the reconfiguration module R corresponds to the structure changer $G_{Emaster}$.

The RFB interface is mapped to a TNCES $T_{Interface}$ module $R_{Interface} \rightarrow T_{Interface}$ where each component is translated to a TNCES: the events $E_i \rightarrow T_{E_i}$, data $D_i \rightarrow T_{D_i}$, with associations $W_i \rightarrow T_{W_i}$ and the internal variables $IV \rightarrow T_{IV_i}$.

The structure changer of GR-TNCES $G_{Emaster} = (P, T, F, V, M_0, P_0)$ corresponds to the ECC master E_m^i of an RFB R_i where M_0 controls the memory requirements by the TNCES and P_0 is the TNCES probability. Each place $p \in P$ in the structure changer corresponds to a specific TNCES of the GR-TNCES model. The latter corresponds to a slave E_s^{ij} of the RFB R_i. Each transition in the structure-changer corresponds to a reconfiguration function. It is characterised by a probability of occurrence that corresponds to a probability of executing a slave. Each state in MSECC is translated into a corresponding place in the GR-TNCES model. We can recognise the active state by a token. This token is removed when a transition is fired and new tokens are generated for the next target places. Each initial state is transformed into a place having a single token, while each non-initial state is translated into a place without token.

4.4.1.2 Rule 2: Event Transformation Rule

Each event in an RFB R_i interface $ie \in E_i$ remains as an event in GR-TNCES model T_{R_i}, i.e. $E_i \rightarrow T_{E_i}$. An input event ie can occur to trigger an RFB or can be cleared after consuming it in the model. An input event in ie is translated into a TNCES module as illustrated in Fig. 4.3. The TNCES of the transformed input event is composed of two places, two transitions, two input events "ie_1 arrives" and "clear ie_1" and an output event "read associated data of ie_1" to read the associated data. It can be n data associated to this event. Accordingly, n event will be emitted. The behaviour of an input of reconfiguration $t_{ier1} \in T_{E_i}$ is composed of: P=$\{p1, p2\}$; T=$\{t1, t2\}$; F=$\{(p1, t1), (t1, p2), (p2, t2), (t2, p1)\}$; W=$\{\}$; CN=$\{\}$; EN=$\{ie1, ie2, oe1\}$; DC=$\{\}$; $V(t1)$=$\wedge$; $V(t2)$=$\wedge$; $Z0$=$\{T0, D0\}$ where $T0(p1)$=1, $T0(p2)$=0 and $D0(p1)$=0;

The same logic is respectively applied for all event types that are the output events, input events of reconfiguration and the output events of reconfiguration.

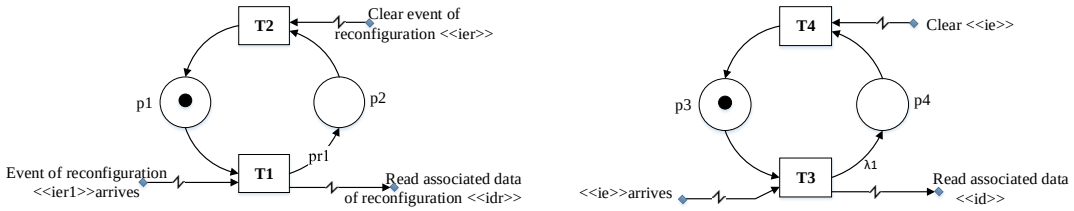


Figure 4.3 TNCES Model for Input Events Models.

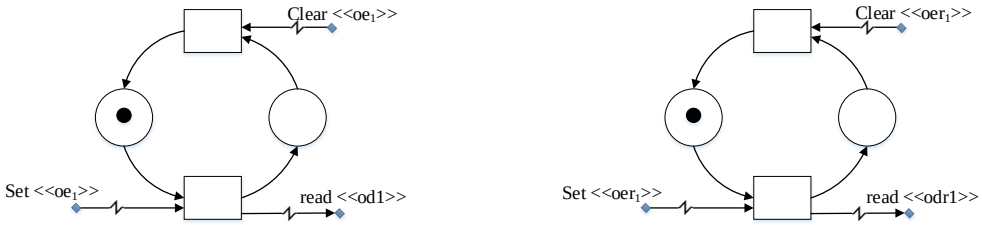


Figure 4.4 TNCES Model of Output Events.

4.4.1.3 Rule 3: Reconfiguration Data Transformation Rule

Each input data of reconfiguration is converted to a TNCES module $Dr_i \rightarrow T_{Dr_i}$, as depicted in Fig. 4.5. An input data of reconfiguration idr is associated to an input event of reconfiguration. Accordingly, when an input event of reconfiguration "read idr" occurs, the place $p15$ will take the decision and select the corresponding branch on basis of the stored data. An output condition will be issued to activate the corresponding slave module. The same rule is applied for the output data of reconfiguration.

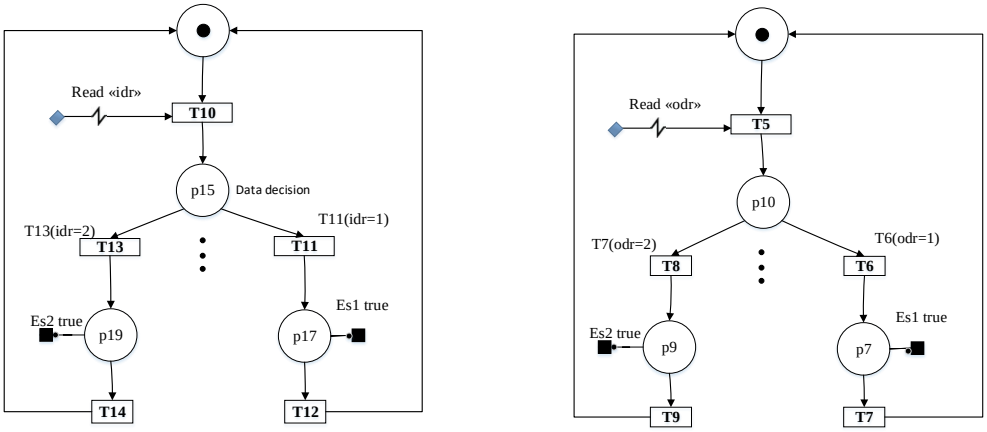
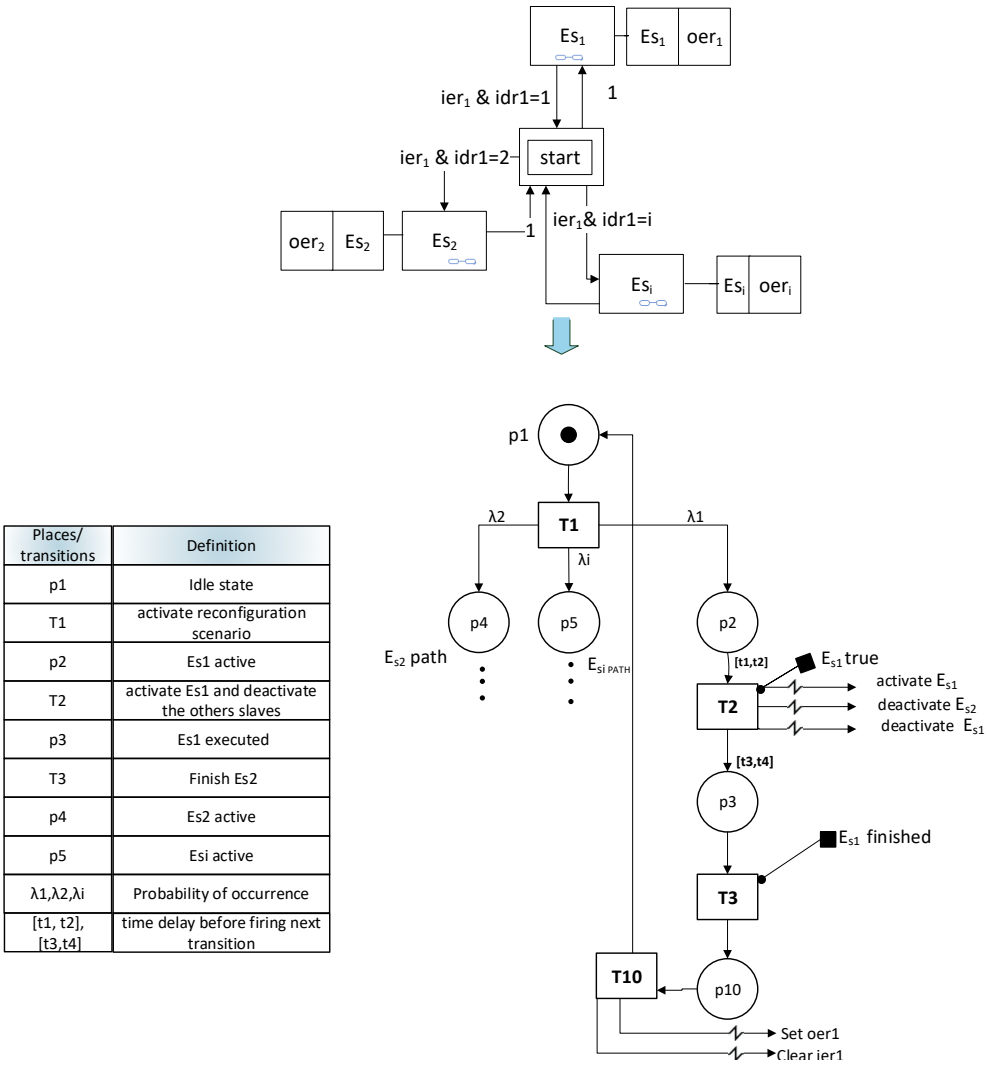


Figure 4.5 TNCES Model for Input and Output Data of Reconfiguration.

4.4.1.4 Rule 4: ECC Master Transformation Rule

An ECC master Em_i is transformed into a structure changer denoted by $G_{Emaster}$ module,i.e. $Em_i \rightarrow G_{Emaster}$, where each place corresponds to an ECC slave and each transition corresponds to a reconfiguration function. A structure changer $G_{Emaster}$, as illustrated in Fig. 4.6, is composed of:



Places/ transitions	Definition
p1	Idle state
T1	activate reconfiguration scenario
p2	Es1 active
T2	activate Es1 and deactivate the others slaves
p3	Es1 executed
T3	Finish Es2
p4	Es2 active
p5	Esi active
λ1,λ2,λi	Probability of occurrence
[t1, t2], [t3,t4]	time delay before firing next transition

Figure 4.6 GR-TNCES Model of an ECC Master.

- An initial marked place and a transition from that emerges $|Es_i|$ branches. Each one has a certain probability λ_i that corresponds to the probability of each reconfiguration scenario where $\lambda_i \in [0, 1]$ and $\sum_{n=1}^{|Es_i|} \lambda_i = 1$. In other words, we assigned a probability to each branch to indicate the chance of choosing such a path.

- Each branch is composed of three places and three transitions. The first transition is associated with an input condition "es_{ij} true" coming from the module representing the input data of reconfiguration. This indicates that slave es_{ij} will be enabled to

fire the transition $T2$. This transition emerges u output events for enabling the slave es_{ij} module and disabling the rest of slaves in RFB R_i. This ensures that a single reconfiguration scenario is active at a given time.

- The second transition is linked to an input condition "es_{ij} finished", which marks the termination of execution of the current active slave es_{ij}. This condition came from the slave module.

- Each branch terminates with a transition that emits two output events: the first one is for setting the output events of reconfiguration oer_i and the second one for clearing the input event of reconfiguration ier_i sent to $gier$ module.

4.4.1.5 Rule 5: ECC Slave Transformation Rule

Each slave es_{ij} is transformed into a TNCES module denoted by $Ts_{ij} \in T_{ESlaves}$ module, i.e. $es_{ij} \rightarrow Ts_{ij}$. A detailed slave is illustrated in Fig. 4.7. The slave es_{ij}, as we aforementioned, is a standard execution control chart so it contains states, transitions and actions. The initial state is transformed into an initial marked place linked to a transition that is fired with the arrival of an input event for enabling es_{ij} module. Each state of slave es_{ij} is transformed into two places "state run alg" and "state finish alg" as well as a transition between them. Each action is modelled with: An initial place "wait", an initial transition that is fired when the input condition "start algorithm" is *true* and M places linked to M transitions for running the algorithm Alg_t, where M is the number of algorithms within the action and $t \in [1, M]$. When all the algorithms in the different actions finish their execution, the slave es_{ij} generates an output condition "Es_1 finished" indicating the end of the slave. To summarise, switching from one reconfiguration scenario to another in the transformed RFB model correlates with enabling the corresponding TNCES and disabling the others in the GR-TNCES model G. It affects the other modules in the system which results in connecting, disconnecting modules, places, transitions.

Adding a new reconfiguration scenario to an RFB in an existing IEC 61499 application corresponds also to adding a new slave module T_{sij} to GR-TNCES system. The GR-TNCES slave module is connected to the GR-TNCES master module via transitions, events and condition signals.

Places/ transitions	Definition
p10	Idle state
T10	Deactivate scenario encapsulated in Es1
p11	Slave2 not active
T12	Deactivate event/data
T11	Activate scenario Es1
p12	Scenario active
T13	Send Events
p15	Schedule algorithm
p15	Agorithm executing
T16	Finish algorithm execution
p16	ES1 finished
T17	finishAlg1
T13 P14	Active Data/event Activate Data/Event

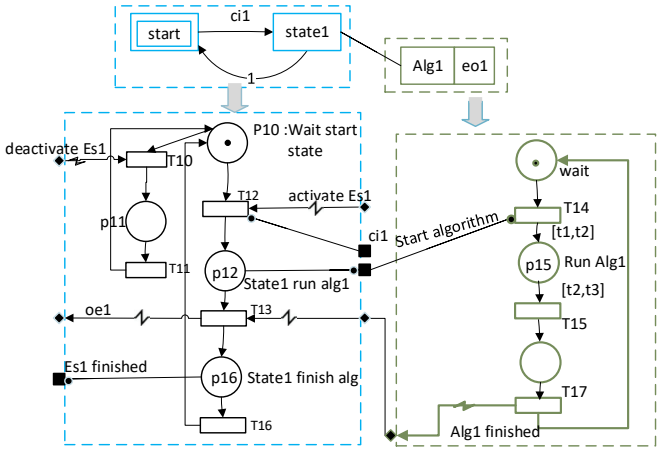


Figure 4.7 GR-TNCES Model of an ECC Slave.

4.4.1.6 Rule 6: FB Transformation Rule

Each basic function block is a TNCES module tf_i having events, places, transitions and conditions. The execution control chart is transformed into a TNCES. An FB event is an event in TNCES, a FB state corresponds to a place in TNCES, an FB transition is a transition, a condition is a condition arc, an action is a state with a transition.

4.4.1.7 Rule 7: System Transformation Rule

The transformed system $TSys$ is a set of R-TNCES formalism, defined as a tuple $TSys$ = $\{TF, TL\}$ where TF is the transformed function block module and TL is a set of linking transitions which relate the $tf_i \in TF$ modules together. A tf_i can be an RFB or a basic function block. A link can be an event, condition, or a transition arc.

4.4.1.8 Rule 8: Resources model

A resource model in IEC 61499 is responsible for scheduling algorithms and communicating with devices. A device model in IEC 61499 is responsible for mapping parameters and also controlling energy. We propose a GR-TNCES module controlling the memory and energy resources to avoid their violation during the reconfiguration process. Before executing the highest probabilistic scenario, the scenario has to guarantee that: (i) the needed energy and memory are available, (ii) time constraints are respected, and (iii) the events and conditions occur at the firing time. The reconfiguration cannot start if the memory reserve is lower than the chosen path's consumption. The reconfiguration should be synchronised in all devices.

The control module R has a memory reserve Mem and an energy reserve Eng. The executed reconfiguration scenario should not violate the global memory and energy, i.e. $E <= Eng$ and $M <= Mem$, and also the time $t \in [t_1, t_2]$. If the reconfiguration is performed, the tokens reserved in the slave place will be consumed from the global memory. At the end of reconfiguration, the memory tokens are added back to the model reserve.

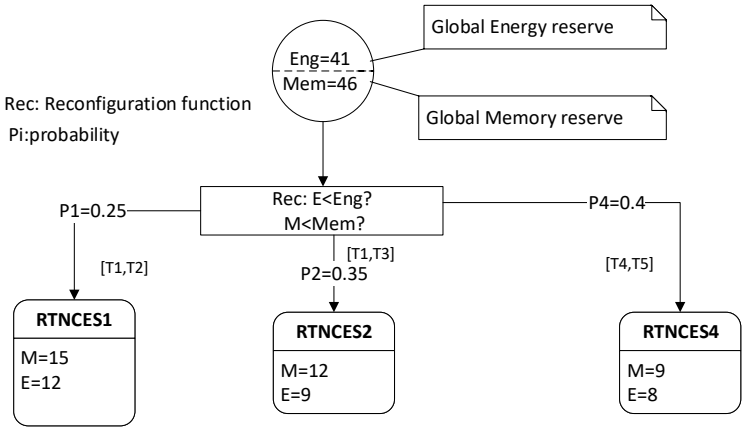


Figure 4.8 A GR-TNCES module controlling memory and energy resources.

4.4.2 Formal Example

Reconfiguration in GR-TNCES is supported by enabling/disabling control components and modifying condition/event signals among them. In this respect, it is plausible to consider an IEC 61499 input system based on six basic function blocks and one RFB RFB_1 as shown in fig 4.9. In the system three feasible paths can be performed thanks to the dynamic reconfiguration executed by $RFB1$:

1. $path_1$: $FB1$-$RFB1$-$FB3$-$FB4$-$FB5$
2. $path_2$: $FB1$-$RFB1$-$FB2$-$FB4$-$FB5$
3. $path_3$: $FB1$-$RFB1$-$FB6$-$FB5$

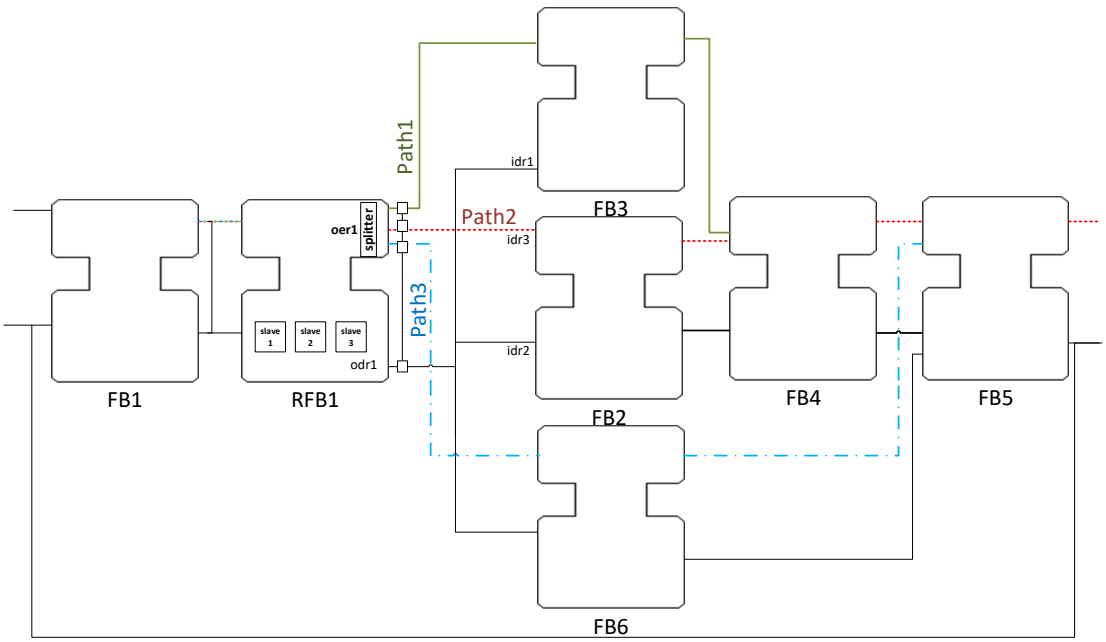


Figure 4.9 A system modelled with RFB and FBs.

On applying the transformation rules on the RFB system, the $RFB1$ is transformed into a GR-TNCES module and each basic function block is transformed into a TNCES module marked in blue. The transformed RFB model, as illustrated in Fig. 4.10, includes an ECC master module and three slaves module configured by the master. In addition, the same paths in RFB model are preserved in the transformed system.

4.5 Formal Verification

In order to mitigate the state space explosion problem, we propose in RFBA a design based on modular RFB where each ECC slave is considered as an ECC master state and is independently verified. Furthermore, the proposed master–slave based execution control chart reduces the states and transitions number in the whole system as shown in the last chapter. Finally, the verification module by module using a modular Petri net GR-TNCES minimises the size of the state space that is divided into M modules. Accordingly, the sum of verification time for each module is clearly less than the verification time of the whole system without modularisation.

Modular verification in RFBA is considered for such complex reconfigurable distributed systems. It permits to minimise the risk of the state explosion problem and the verification time since it does not check the modules that are not changed by the reconfiguration. It is not required to check receptively the common parts in GR-TNCES that are not changed by the reconfiguration and they have been checked to be correct in the previous step. As a result, the total verification time of a target system is the sum of verification time of only the changed modules after executing a reconfiguration scenario.

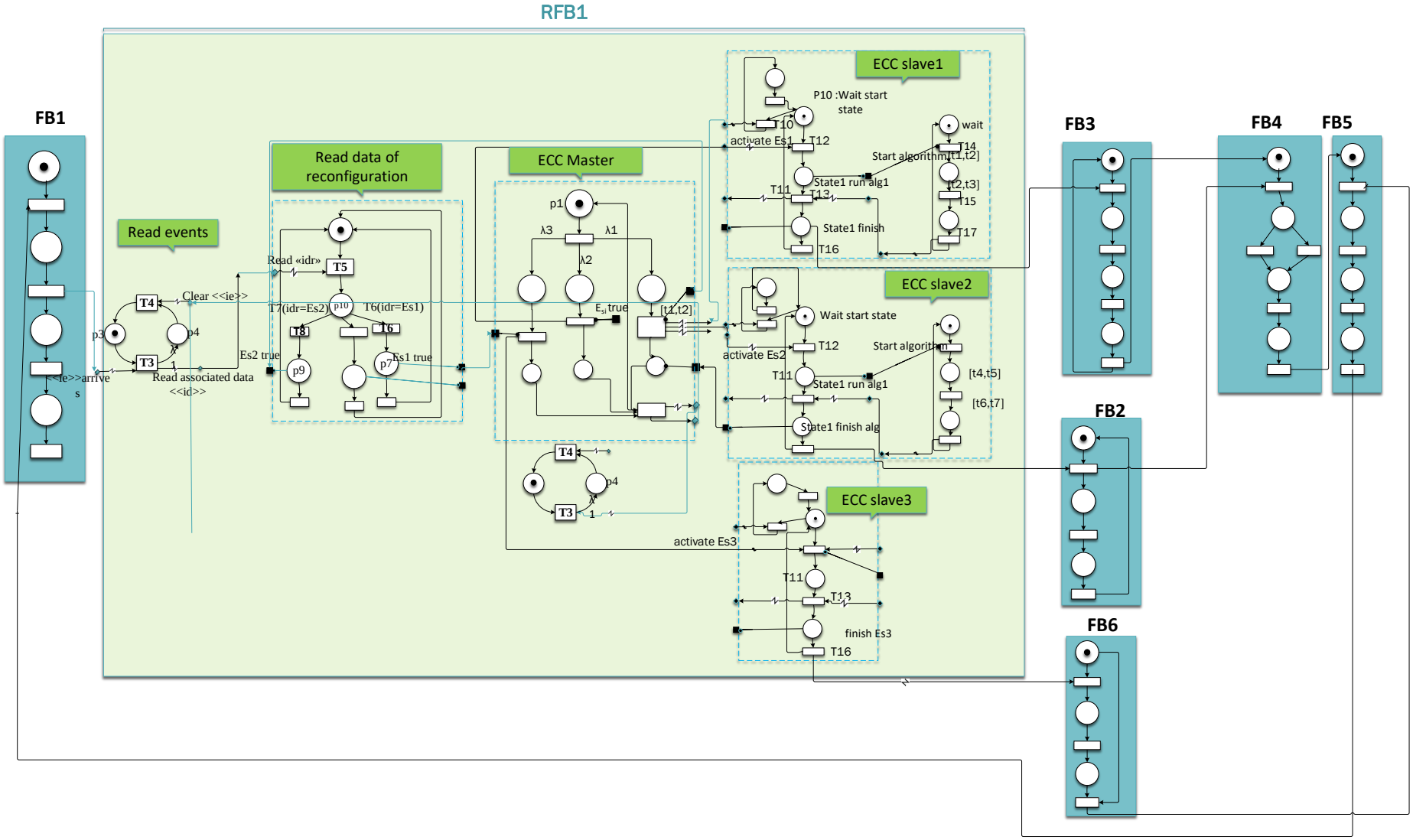


Figure 4.10 GR-TNCES Model for the Formal System.

The complexity of the proposed verification is $O(n^2)$ where n is the number of modules.

In the formal example, a modular verification is applied. The static part of the GR-TNCES model is verified separately from the dynamic part. We verify first of all FB_1 TNCES model then the RFB_1 model, after that FB_3, FB_2 and FB_6 simultaneously. At the end we verify FB_4 and FB_5. As result, the verification time of the whole system $T_{verification}$ is equal to:

$$T_{verification} = T_{FB1} + T_{RFB} + max(T_{FB2}, T_{FB3}, T_{FB6}) + T_{FB4} + T_{FB5}$$

Therefore, it is clear that the verification layer by layer of such system takes less time and resources since only one FB ($FB1$, $FB2$, $FB3$) is enabled in each path. Moreover, the static part of the system is verified only one time if it is not changed by the reconfiguration. The repetitive verification of the unchanged parts is avoided during the next steps of the verification process.

A reconfiguration process can violate system safety. The activation/deactivation, addition or removal of a reconfiguration scenario during the execution of an old one can lead to a deadlock within the system. For these reasons, deadlock-free, liveness and safety properties should be proved using model checking.

Let's consider (G, M_0) the transformed Petri net G where M_0 is its initial marking.

1. The deadlock is detected in the system (G, M_0) using the CTL formula:

   $$E[F\text{“}deadlock\text{”}].$$

   The system is deadlock free if the property is false, i.e. every reachable marking enables at least one transition (its successor). Otherwise, a deadlock exists and one or more reachable markings are dead. Hence, the system model should be rectified.

2. (G, M_0) is live if every transition can always fire again.

3. (G, M_0) is bounded if for every place s there is a number $b \geq 0$ such that $M(s) \leq b$ for every reachable marking M.

Moreover, in order to analyse and verify the correctness of the automatically generated model, confluence and termination properties are also checked. They are the most important criteria that prove the correctness of the model transformation. The termination property guarantees the existence of a target model, i.e. that the transformation execution is done for any well-formed transformation specification [Ulitin et al., 2019]. The termination property is guaranteed since the generated model is a set of finite state machines.

The confluence property, i.e. determinism, ensures the uniqueness of the target model for a given source model and transformation specification [Ulitin et al., 2019]. It ensures

that transformations always produce the same result. This property is considered up to the interactions with the environment or the users [Amrani et al., 2012]. The confluence is also guaranteed when no more than one scenario is executed at the same time inside each reconfigurable function block modelled with GR-TNCES. Within the framework of these criteria, checking the correctness of the constructed models generated in RFBA approach become attainable. However, the properties should be specified by the designer in a computation tree logic language.

Thanks to the probabilistic aspect in the GR-TNCES, the designer can estimate also system performance and quantify properties specified using probabilistic computation tree logic and PRISM model checker. In order to support RFBA methodology and transform an RFB model into a GR-TNCES model, an RFBtool environment is developed and integrated with ZiZo v2.

4.6 RFBTool

RFBTool is a new environment developed to support RFBA approach from design to formal modelling with a GR-TNCES formalism. It is integrated with ZiZo second version to transform GR-TNCES into a discrete-time Markov chain for verifying it with the model checker PRISM.

Interestingly, RFBTool is an RFB editor that designs any reconfigurable control system using RFB formalism. It facilitates modelling different reconfiguration scenarios using the hierarchical ECC master-slave, and then connecting RFB with other FB types through standard event and data links. The complete designed system is saved as an XML file validated against Document Type Definitions DTD. Holobloc has defined a DTD structure in the IEC Standard 61499-2 to check the compliance of XML files generated by several tools. It specifies the structure, syntax, legal elements and attributes following IEC 61499-2 [HOLOBLOC, 2019a]. In order to validate the generated XML file, we propose a new DTD File defining the new types described in RFB model such as events/data of reconfiguration and the master-slave hierarchy. An example of a generated XML code is illustrated in Fig. 4.11.

Fig.4.12 presents the RFBTool class diagram that describes the RFBTool components, its architecture, and the way in which the classes are linked together. The application is composed of BFBs, CFBs, SIFBs, or RFBs executed in a resource container. Each function block has standard events and data flow. However, RFB has extra events of reconfiguration and types of data. It also has an ECC master and at least one ECC slave.

The designer can create or load a project in RFBTool, after that he can create new RFB or other function blocks types, instantiate them and link them using standard or reconfiguration events and data. As shown in Fig. 4.13, the user can add the following

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE FBType SYSTEM "RFBLibraryElement.dtd">
- <FBType Comment="Reconfigurable Function Block Type" Name="rfb1">
  - <InterfaceList>
    - <EventInputs>
      - <ReconfigurationEvent Comment="input event of reconfiguration" Name="ier1">
          <With Var="idr1"/>
        </ReconfigurationEvent>
      - <StandardEvent Comment="Initialization Request" Name="INIT">
          <With Var="MODE"/>
        </StandardEvent>
      - <StandardEvent Comment="Update Request" Name="REQ">
          <With Var="MODE"/>
        </StandardEvent>
      </EventInputs>
    - <EventOutputs>
      - <ReconfigurationEvent Comment="output event of reconfiguration" Name="oer1">
          <With Var="odr1"/>
        </ReconfigurationEvent>
        <StandardEvent Name="oe1"/>
      </EventOutputs>
    - <InputVars>
        <ReconfigurationData Comment="1=slave1,2=slave2" Name="idr1" Type="UINT"/>
      </InputVars>
    - <OutputVars>
        <ReconfigurationData Comment="1=slave1,2=slave2" Name="odr1" Type="UINT"/>
      </OutputVars>
  </InterfaceList>
  - <RFB>
    - <ECC_MASTER>
      - <MSTAE Name="start" initial="true">
          <ECAction output="oer1" slave="DecisionAlg"/>
        </MSTAE>
      - <MSTATE Name="slave1" y="102" x="1368">
          <ECAction output="oe1" slave="SLAVE1"/>
        </MSTATE>
      - <MSTATE Name="slave2" y="102" x="1368">
          <ECAction output="oe1" slave="SLAVE2"/>
        </MSTATE>
        <MTRANSITION Condition="ier1 & idr1=1" Destination="slave1" Source="start"/>
        <MTRANSITION Condition="ier1 & idr1=2" Destination="slave2" Source="start"/>
      - <ECC_SLAVE NAME="SLAVE1">
          <ECState Name="WAIT" y="1022" x="1444"/>
        - <ECState Name="INITst" y="102" x="1368">
            <ECAction Output="oe1" Algorithm="INITAlg"/>
          </ECState>
        - <ECState Name="PIN" y="511" x="2061">
            <ECAction Output="oe1" Algorithm="ALG1"/>
          </ECState>
          <ECTransition y="637" x="1486" Condition="INIT" Destination="INITst" Source="WAIT"> </ECTransition>
          <ECTransition y="698" x="1741" Condition="REQ&(MODE=0)" Destination="PIN" Source="WAIT"/>
          <ECTransition y="1217" x="1820" Condition="REQ&(MODE=1)" Destination="WAIT" Source="PIN"> </ECTransition>
        </ECC_SLAVE>
      - <ECC_SLAVE NAME="SLAVE2">
          <ECState Name="WAIT" y="1022" x="100"/>
        - <ECState Name="STATE1" y="511" x="2061">
            <ECAction Output="OE1" Algorithm="ALG1"/>
          </ECState>
          <ECTransition y="243" x="100" Condition="INIT" Destination="STATE1" Source="WAIT"> </ECTransition>
          <ECTransition y="250" x="120" Condition="1" Destination="WAIT" Source="STATE1"> </ECTransition>
        </ECC_SLAVE>
      </ECC_MASTER>
    - <Algorithm Name="INITAlg">
        <ST Text="D0:=(MODE>2); "/>
      </Algorithm>
    - <Algorithm Name="ALG1">
        <ST Text="IF D0!=0 THEN MODE=3; "/>
      </Algorithm>
  </RFB>
</FBType>
```

Figure 4.11 A generated XML Code for an RFB Example.

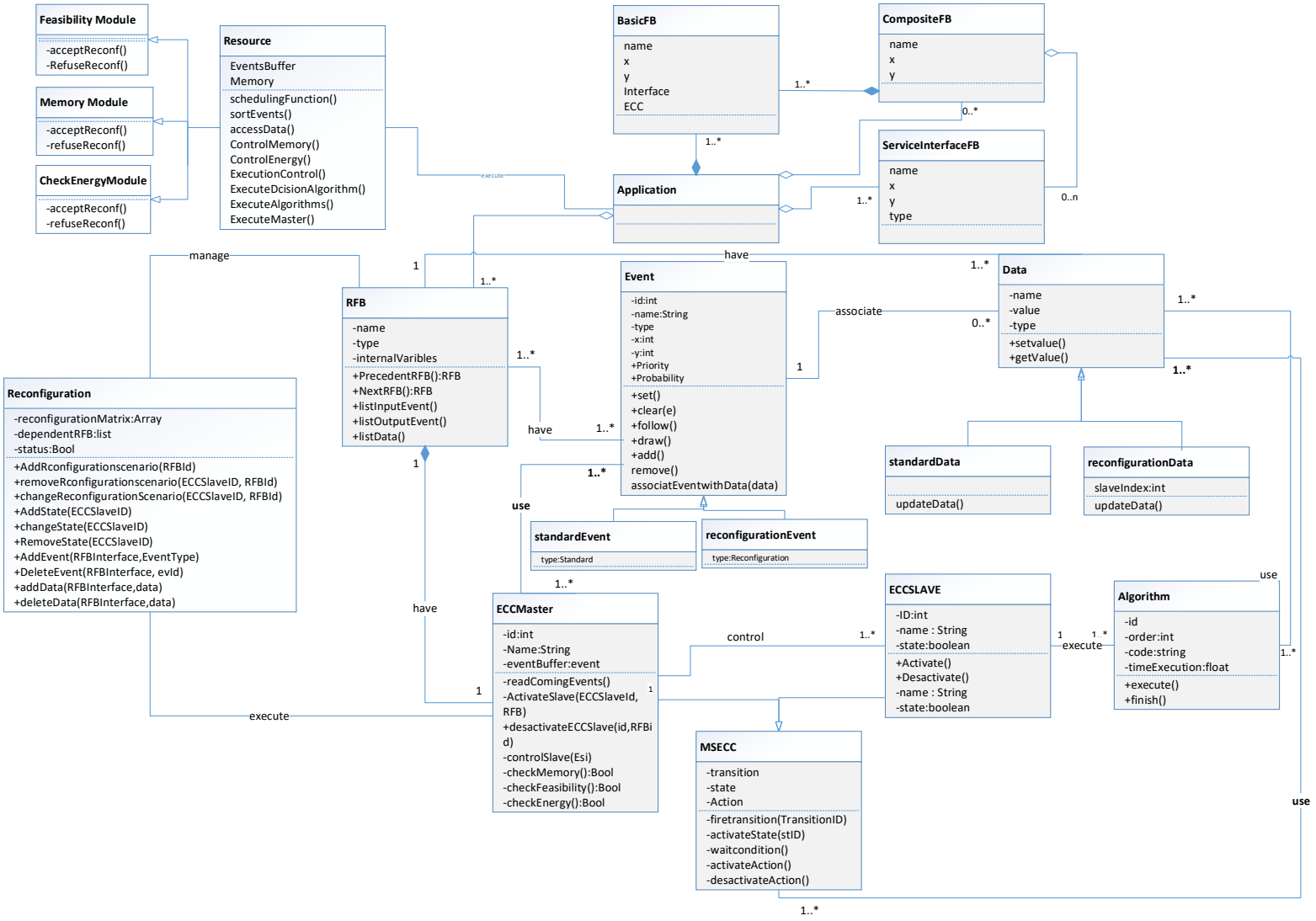


Figure 4.12 Class Diagram showing the Architecture of RFBTool.

components to an RFB: ECC master, ECC slave, standard events or events of reconfiguration, standard data input/outputs or data of reconfiguration and internal variables. The designer can add freely the aforementioned components to create the needed RFB. Finally, the designer must link the components of the RFB to the function block network through event links, data links and with association links.

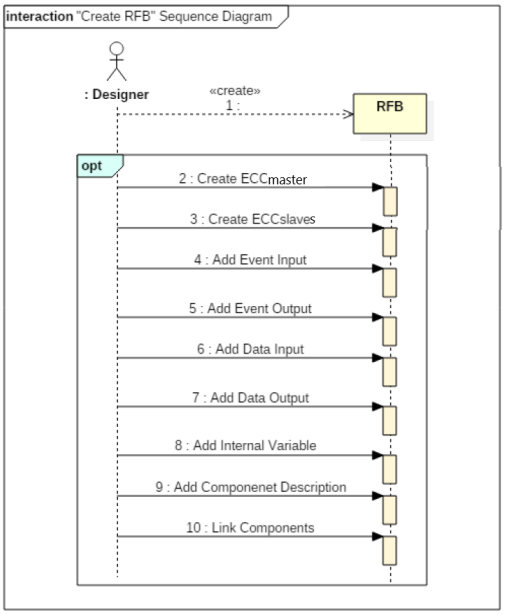


Figure 4.13 "Create RFB" Sequence diagram.

RFBTool proves to be an effective tool to create a new project and save it as an XML file. It provides a "drag and drop" area for the system designer as well as allows to load and edit an existing project from an XML file. Two RFBs are related together with events and data are shown in Fig. 4.15. On the left, a hierarchy of the created system is represented by a tree view. Furthermore, the tool offers an export feature that automatically converts the created system to a GR-TNCES model saved in ".zz" file that is based on the transformation rules mentioned in the previous section.

RFBTool also stems its originality from the fact that it is not just an editor but it includes also a run-time environment module. It is developed to implement the proposed execution semantic detailed in section 3.3.4 (Execution Semantic and Event Consumption Policy). It executes the ECC master in a thread and each slave in another thread both controlled by the resource as shown in Fig. 4.14. It is allowed to execute only one slave in parallel with the master. RFBtool implies a model of invocation which is implemented with an optimised queue considering the arriving time, event type and priority. Furthermore, RFBTool guarantees that no more than one input event is emitted at any given time. It allows the preemption but with some constraints. The execution of a reconfiguration event may preempt the execution of other events since it has the highest priority. The execution model is a preemptive multithreaded resource (PMTR) under special constraints.

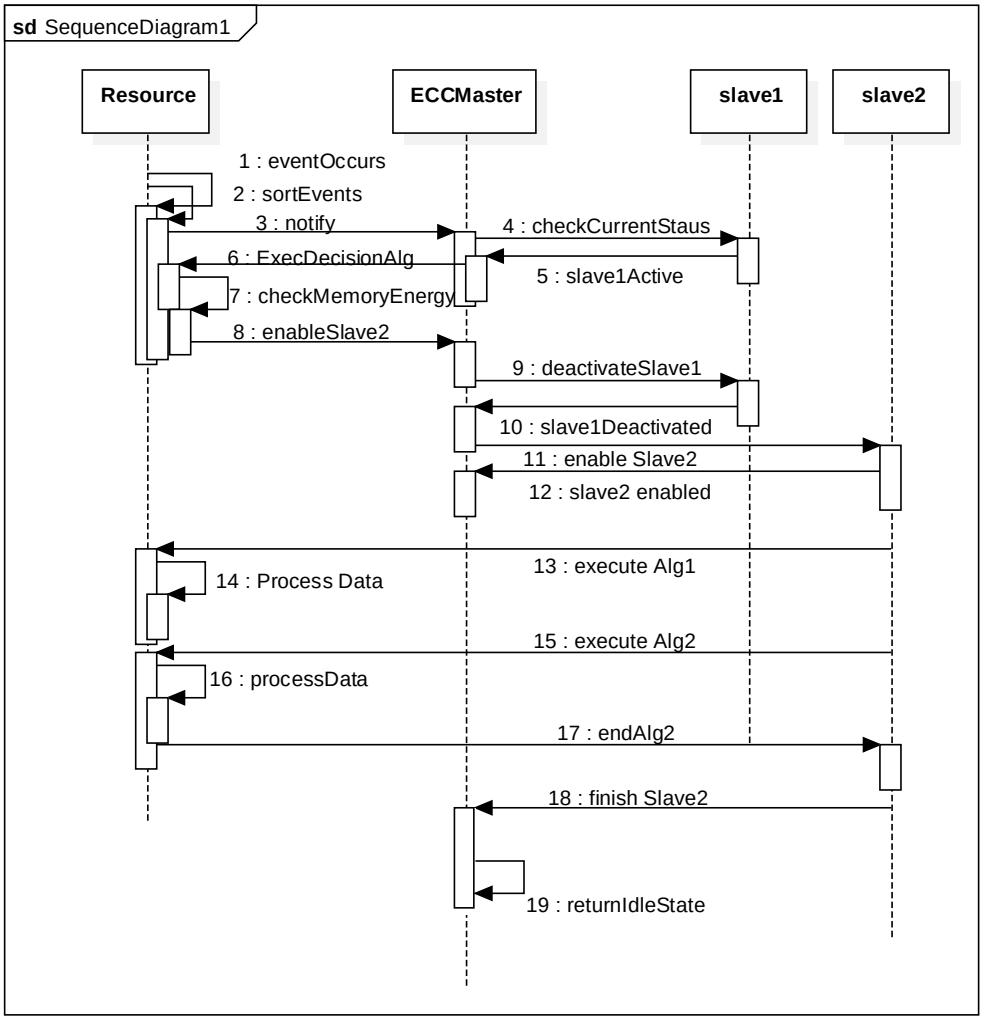


Figure 4.14 Sequence diagram showing the RFB execution semantic.

RFBTool generates from the RFB model a C++ code for deployment. There is a thread controlling the ECC master, and for each slave, a thread is implemented but only one can be in process. RFBtool is integrated with the second version of ZiZo in order to verify the GR-TNCES system.

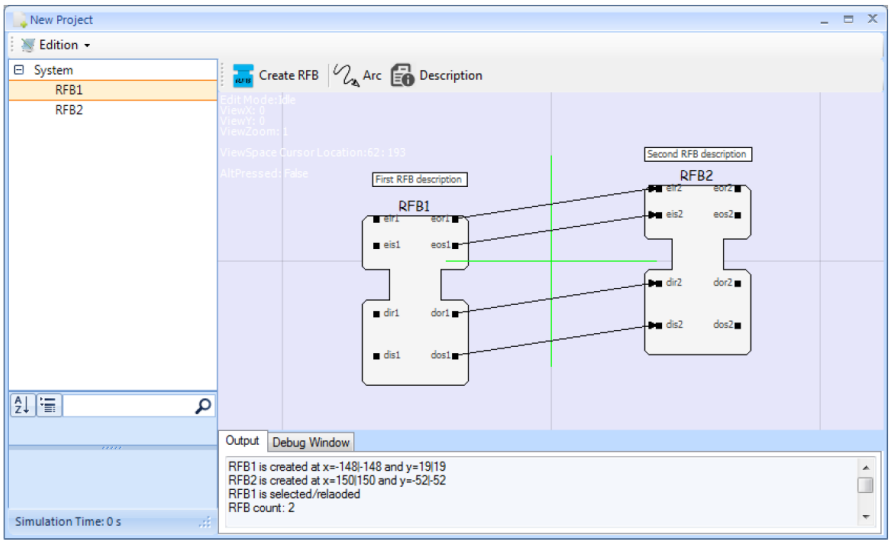


Figure 4.15 RFBs Network as shown in RFBTool.

4.7 ZiZo V2

ZiZo v2 [Khlifi et al., 2015] is the unique tool that models GR-TNCES formalism. It is developed in Saarland University Germany in collaboration with LISI Laboratory of INSAT Institute Tunisia. It creates, edits, simulates and checks adaptive systems modelled with GR-TNCES.

ZiZo v2 provides a graphical GR-TNCES module for each component in IEC 61499 as depicted in Fig. 4.16. It can also simulate the highest or lowest probabilistic scenario to be executed in the project according to the user choice. The user should enter the memory and energy reserves to verify resources by simulation. However, this feature is not treated in the context of this thesis since we focus on the probabilistic and the reconfiguration aspect of GR-TNCES and the model checking. Indeed, the designer can inject in this step probability values on the GR-TNCES model to estimate certain performance obtained from PRISM. The latter is a probabilistic model checker that is an effective software tool used to verify functional, temporal and probabilistic properties.

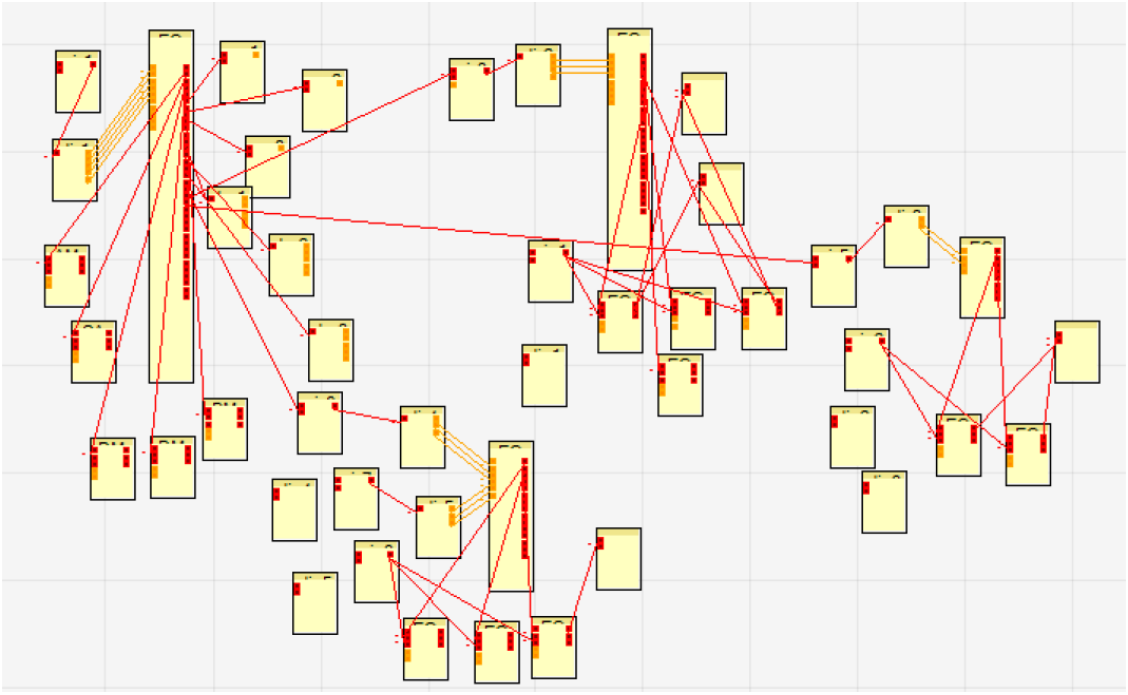


Figure 4.16 The Generated GR-TNCES model as shown in ZiZo v2 after exporting the RFB model from RFBTool.

Furthermore, ZiZo v2 software allows to export the generated GR-TNCES model into a specific PRISM model. It permits to convert a ".zz" file to a ".pm" file to check functional and real-time properties. The generated PRISM model presents a discrete-time Markov chain defined as a set of modules including places and probabilities. BRBTool and the second version of Zizo construct toolchain.

4.8 Discussion and Originalities

In order to simplify and promote the design and verification of reconfigurable control systems, we have proposed a new methodology called RFBA that is based on RFB pattern. RFBA compared to other approaches has several benefits. It supports design based RFBs, automatic formal modelling and verification. It reduces the verification time of reconfigurable systems thanks to the modular verification. Only the changed modules by reconfiguration will be checked during the reconfiguration. Thus, the state explosion problem is mitigated. We introduce, in the GR-TNCES model, the probability corresponding to each reconfiguration scenario. This will allow the designer to estimate probabilistic properties.

The design of reconfigurable systems following the IEC 61499 extension using the existing IEC 61499 development tools is complex and difficult. Moreover, only simulation technique is supported. Accordingly, we developed an RFBTool environment to support:

(i) design with RFB pattern,

(ii) automatic transformation from an RFB model to a GR-TNCES model by implementing the defined set of transformation rules.

(iii) verification of the generated model with probabilistic model checking by integrating RFBTool with ZiZo second version.

As a result, our approach is very helpful in the development process of reconfigurable distributed automation systems. Qualitative and quantitative properties can be both proved and estimated. The proposed toolchain is a very effective tool that insures the whole process from the design with RFBs to verification.

4.9 Conclusion

This chapter highlights the importance of formal modelling and verification in checking the system correctness. In RFBA methodology, IEC 61499 model based on RFBs is transformed automatically into a GR-TNCES model thanks to a set of transformation rules. This transformation is proposed due to the existence of several similarities between both models. In order to support RFBA methodology, a new RFBTool environment is implemented and integrated together with ZiZo v2. The whole approach will be applied on a distributed power system in the next chapter.

Chapter 5

Case Study and Performance Evaluation

5.1 Introduction

In order to illustrate the whole contributions, and prove the efficiency of RFBA approach for design, modelling and verification of reconfigurable distributed control system, RFBTool and the model checker are applied on a distributed power system. The "fault, location, isolation, and supply restoration" (FLISR) case study part is published in IEEE Transactions on Automation Science and Engineering (TASE) journal [Guellouz et al., 2019].

5.2 Case Study Presentation

A distributed intelligent power system is presented in this chapter as illustrated in Fig. 5.1. It allows to distribute smartly the energy to different consumers according to their demands. Photo-voltaic generators are installed to generate electricity in addition to the utility grid.

The power system depends generally on uncontrolled events and unpredictable disturbances like faults, meteorological factors, excess generation during low load periods, interconnection issues, or an alteration of the grid topology. We characterise each unpredictable event by its probability of occurrence based on a knowledge database located on the data acquisition and management platform (DAMP).

The system is based on "fault, location, isolation, and supply restoration" (FLISR) mechanism [Guellouz et al., 2019]. It detects automatically the faults. It identifies the faulty section, isolates the fault and restores power to consumers by automatically switching the faulty section to the nearest non-faulty section. Consequently, most consumers will be supplied without any manual intervention. Each occurred fault causes a reconfiguration that is the automatic switching from a faulty section into a non-faulty one of

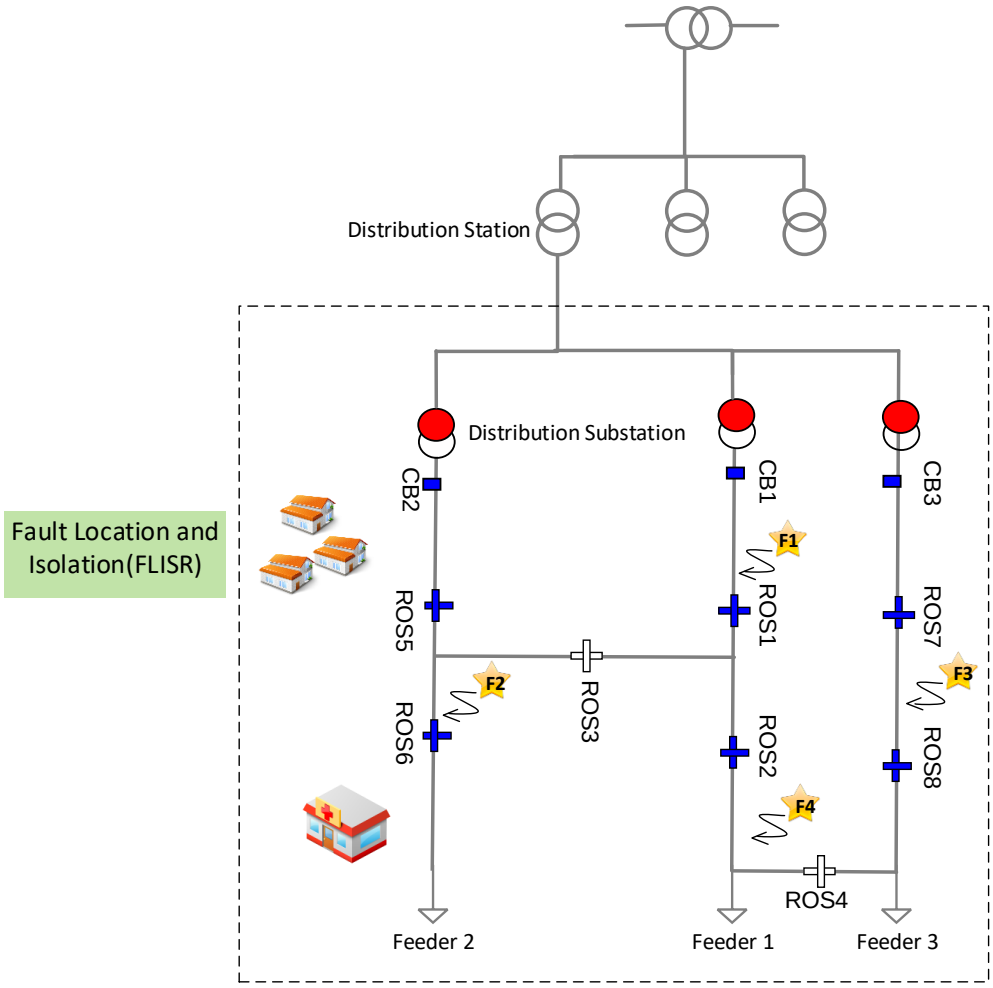


Figure 5.1 Case Study Topology.

the system. The latter must have enough energy to supply the faulty section. When several faults occur simultaneously, the software reconfiguration can resolve the problem if it gets a sufficient supply from other feeders. The FLISR system in Fig. 5.1 contains three feeders, each of which is consists of a circuit breaker (CB) and two sectionalising remotely operated switches (ROS) that divide the feeder into sections and by default are closed ($ROS1$, $ROS2$ in $feeder1$, $ROS5$, $ROS6$ in $feeder2$, $ROS7$, and $ROS8$ in $feeder3$). The feeders are interconnected together via tie switches, i.e. $ROS3$ and $ROS4$, that are open by default and may be closed if a successful reconfiguration is executed or by a maintenance crew. If a permanent fault occurs, then the system tries to locate an alternative supply from the nearest neighbour feeder. If it succeeds to supply the faulty section, then the reconfiguration is successful. Otherwise, it waits for manual repair [Guellouz et al., 2019].

We apply in the next sections the proposed approach RFBA on the system. We start by system specification, then design and after that formal verification. An estimation of the system performance such as availability and probability of failed reconfiguration will

be provided.

5.3 System Specification

We begin by capturing the system specification, determining the reconfiguration scenarios within RFBs. Reconfiguration can be applied in case of faults occurrence: when a fault occurs, the automatic switching from the faulty section into the nearest non-faulty one in the system having enough energy is required. Accordingly, fault location, isolation and supply restoration should be provided and modelled. Protection and control of the switches in the system are also required.

5.4 RFBs-based Architecture

The control application is designed based on reconfigurable function blocks. The case study is an extension to the [Zhabelova and Vyatkin, 2012] work where each logical node presented in their study is modelled by an RFB where:

- $PIOC$ is a logical node (LN) responsible for the protection of over-current relay in the circuit breakers and switches.
- $CSWI$ presents the switch control of a sectionalising ROS.
- $XCBR$ is the primary equipment representing the circuit breaker CB.
- $XSWI$ is the primary equipment representing a switch logical node,
- $TCTR$ is the current transformer logical node in the circuit breaker CB.

As shown in Fig. 5.2, the model includes $TCTR_i$, $PIOC_i$, $CSWI_i$ and $XCBR_i$ RFB instances. Each $PIOC$ RFB type has in its interface an input event of reconfiguration reserved for modelling unpredictable faults $Fault$, an ECC master em_1 and two ECC slaves es_{11} and es_{12} which represent the reconfiguration scenarios of this unit, as depicted in Fig. 5.3. The first slave es_{11} is the $FaultOperation$ mode that is enabled by the master when a fault occurs, while the second slave es_{12} is the $NormalOperation$ mode that is enabled when the fault is restored or a request about its status is encountered. All $PIOC_i$ in the model are instances of the $PIOC$ RFB type where $i \in [0, 3]$.

$CSWI3$ and $CSWI4$ RFBs correspond respectively to the tie switches $ROS3$ and $ROS4$. The control switches logical nodes $CSWI1$, $CSWI2$, $CSWI5$, $CSWI6$, $CSWI7$ and $CSWI8$ corresponds respectively to the sectionalising remotely operated switches $ROS1$, $ROS2$, $ROS5$, $ROS6$, $ROS7$, and $ROS8$.

The interaction between RFBs is as follows: $TCTR1$ is the current transformer of $CB1$ which reads and compares continuously the electrical current value to a maximum

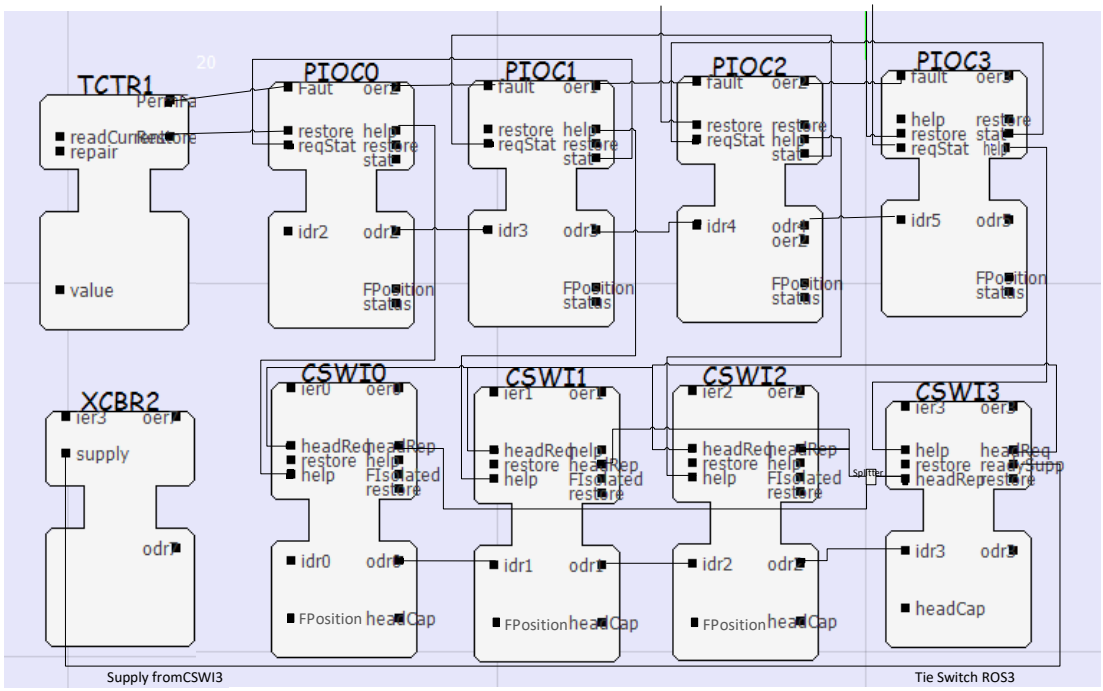


Figure 5.2 Subsystem B modelled with an RFB Network.

limit. If it detects a fault $f1$ in $feeder1$, it informs the corresponding PIOCs in the feeder about their fault state. The input event of reconfiguration $fault$ received from $TCTR1$ indicates that a permanent fault happens and a reconfiguration should be launched. Once the event has been received, the upstream PIOC circulates the event to downstream PIOCs. All PIOCs on the faulty section enable the $FaultOperation$ slave to locate exactly the fault position. The ECC master $em1$ disables the $NormalOperation$ scenario and switches directly to the $FaultOperation$ scenario. In the $FaultOperation$ reconfiguration scenario, PIOC executes the fault location algorithm and sends a request status to the neighbours $stat$, as illustrated in Fig. 5.4. It can also receive a status request from other PIOC and propagate the fault status. If the status of the next downstream PIOC is false, then the fault is identified in this section of the feeder. Accordingly, the RFB $PIOC1$ of the remote switch $ROS1$ sends the output data $FPosition$ to $CSWI1$ to indicate that the fault is located there.

$CSWI1$ receives $FPosition$ true associated with ier_1, enables the slave *FaultOperation* to isolate the fault and emits $help$ output events to $CSWI2$, $CSWI3$, and $CSWI4$ in order to determine an alternative supply, as illustrated in Fig. 5.2.

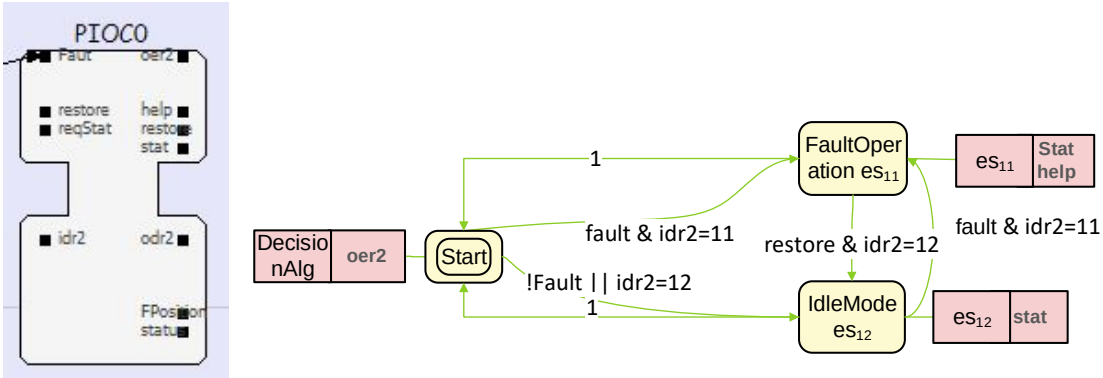


Figure 5.3 $PIOC0$ Interface and its ECC master.

Each CSWI of a tie switch ($CSWI3$ and $CSWI4$) receives $help$ and sends $HeadReq$ to the CSWI of the sectionalising switches ($CSWI5$ and $CSWI6$ in $feeder2$) and ($CSWI7$ and $CSWI8$ in $feeder3$) to determine the current headroom capacity. It replies to the tie switch with the available headroom capacity $HeadCap$. The available headroom capacity is the difference between the current energy and the needed load. The output data $HeadCap$ is associated with the output event headroom reply $HeadRep$. For clarity reasons in Fig. 5.2, the association links are not mentioned.

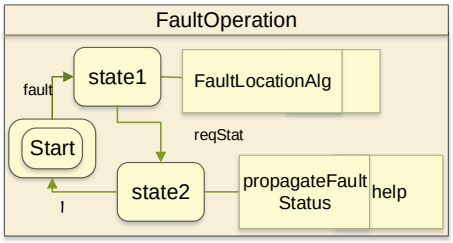
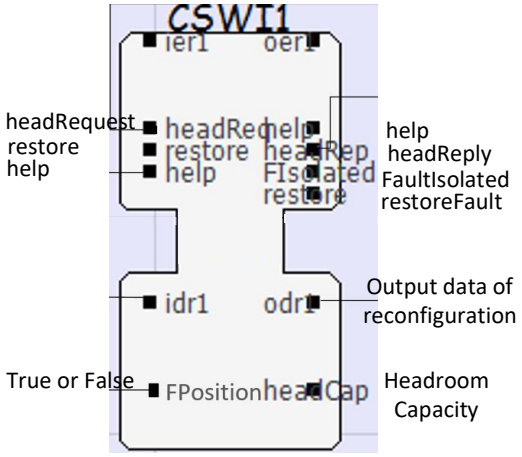
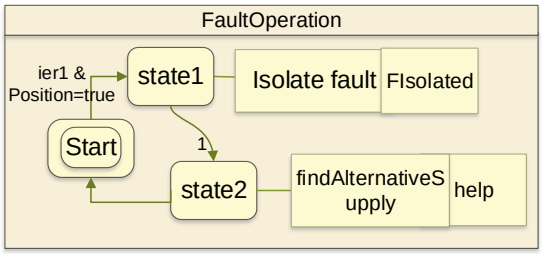


Figure 5.4 ECC slave es_{11} $FaultOperation$.

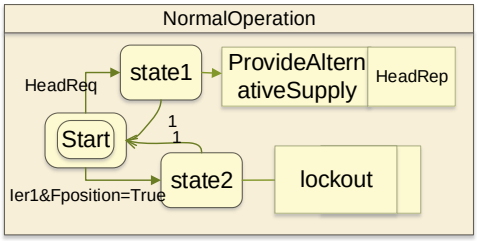
The $CSWI3$ RFB (respectively $CSWI4$) of the tie switch $ROS3$ (respectively, $ROS4$) is able to decide if the capacity of the alternative supply satisfies the current needed load on the faulty section or not. If it has an excess in $feeder3$, then $CSWI3$ sends a ready to give supply event $readySupp$ to close the switch $XSWI2$ (respectively, $XSWI3$). Consequently, the supply is restored on faulty switches and the corresponding CSWIs will return to their $normalOperation$ slave. The latter is responsible for receiving the headroom request, calculating the headroom capacity and providing the alternative supply.



(a) $CSWI$ Interface



(b) $FaultOperation$ slave



(c) $NormalOperation$ slave

Figure 5.5 $CSWI$ Interface and its slaves.

5.5 Automatic Transformation into a GR-TNCES Model

After designing the system with RFBs, the XML file is transformed automatically into a GR-TNCES model using the RFBTool environment. The latter generates a ".zz" file compatible with ZiZo v2 tool.

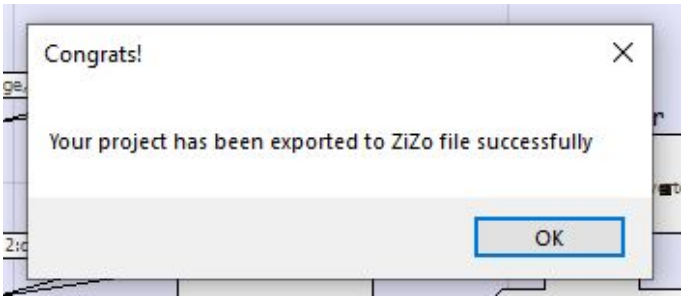


Figure 5.6 Successful Export to ZiZo v2 tool.

GR-TNCES is used in formal modelling to guarantee system correctness and robustness. Thanks to ZiZo-v2 tool, we simulated the GR-TNCES model of FLISR system. The obtained report proves that the system design based RFBs is deadlock-free after exploring 1053 places in 486 seconds, where ZiZo-v2 software is running on a PC with

an Intel Core (TM) i5-3230M CPU @2.60GHz processor, Windows 10 operating system and 4GB RAM. Additionally, ZiZo-V2 displays the generated GR-TNCES model and gives us an overview about the FLISR system such as the number of transitions, places, events and conditions, connections, and modules number. ZiZo shows that the whole FLISR system is composed of 66 modules, 1053 places and 788 transitions, as presented in Table 5.1.

Table 5.1 Components Size in the Generated GR-TNCES.

Places	Transitions	Events	Conditions	Connections	Modules
1053	788	186	198	144	123

We inject in this step the probability of occurrence for each fault event in the PIOC modules. The reconfiguration event $fault$ is characterised by its probability of arrival λ_i. We can also attach the probability of each branch in the structure changer to simulate the highest probabilistic scenario path. We inject four faults in the system model in different positions as follows:

(i) The fault f_1 in $feeder1$ is between $CB1$ and $ROS1$ where its probability of occurrence is ones per month $\lambda 1 = \frac{1}{(1*30*24*60*60)}$;

(ii) The fault f_2 in $feeder2$ is between $ROS2$ and $ROS4$ where its probability of occurrence is twice per month $\lambda 2 = \frac{2}{(1*30*24*60*60)}$;

(iii) The fault f_3 is triggered between $ROS5$ and $ROS6$ in $feeder3$. Its probability of occurrence is three times per month $\lambda 3 = \frac{3}{(30*24*60*60)}$;

(iv) And the fault f_4 in $feeder2$ is triggered when the link between $ROS2$ and $ROS4$ is cut. Its probability is five times per month $\lambda 4 = \frac{5}{(30*24*60*60)}$;

We also assume that every period T_r in seconds a repair agent checks the system and repairs all the faulty sections. The expected repair rate is supposed to be equal to $\mu_n = \frac{(n+1)}{T_r}$, where n is the fault number, and T_r is the repair time.

The main GR-TNCES modules modelling $PIOC0$ RFB are its ECC master and its slaves. The transformed GR-TNCES model of the ECC master em_1 in $PIOC0$ is depicted in Fig. 5.7 which corresponds to the ECC master shown in Fig. 5.3. In each arc in the structure changer model, the probability of each scenario can be assigned. From the transition $t2$ to $p3$ we assigned a probability of the slave es_{11} equal to λ_i. The master of $PIOC0$ activates the transition that presents the corresponding scenario to be executed according to the coming input condition. It issues an output event to deactivate the remained slave. After executing the encapsulated algorithms, the master module receives

an input condition. At the end of the slave, the fault occurred event is cleared and a reconfiguration output event is issued to communicate with the next PIOC modules.

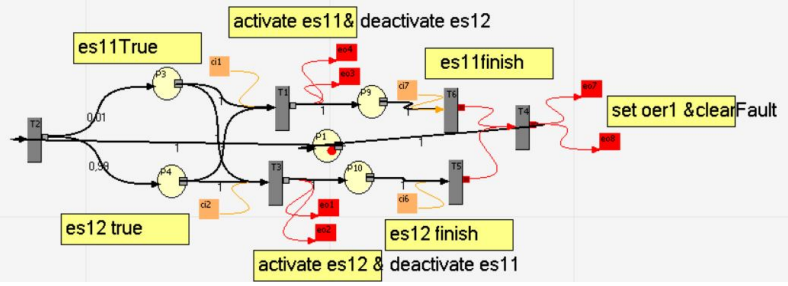


Figure 5.7 GR-TNCES Model of the $PIOC0$ ECC master.

Since the model is deadlock free, we can move to the probabilistic verification by model checking. ZiZo-v2 tool facilitates the verification process by converting the ".zz" file to a discrete-time Markov chain DTMC model that is compatible with PRISM model checker.

5.6 Formal Verification

RFBA methodology seeks to verify qualitative and quantitative properties using the PRISM model checker as well as the specified CTL and PCTL properties. PRISM supports model checking for run-time verification of adaptive systems and analyses systems that exhibit random or probabilistic behaviours. The goal of unified verification is to guarantee system safety and evaluate in the worst-case scenarios some requirements such as system availability. The generated GR-TNCES model describes in a PRISM language the sections control, the fault location, the fault occurrence, the repair process, and the reconfiguration in modules. Each module is connected to others. A fault reconfiguration module is presented in Fig. 5.8. It includes several states that depend on fault occurrence, the real-time needed load in the faulty feeder, the real-time available headroom capacity, and the repair rate.

We illustrate in the next sections the feasibility of the system model starting by the qualitative analysis and then exhibiting the quantitative analysis. We have verified the functional correctness and the safety of individual RFBs as well as the entire system using CTL and PCTL properties.

```
ctmc
const double lambda_1 =1/(24*60*60);
const double lambda_2 = 2/(30*24*60*60);
const double lambda_3 = 3/(30*24*60*60);
const double lambda_4 = 5/(30*24*60*60);
const double Tr;//in seconds
const int available_HC_f1;
const int available_HC_f2;
const int available_HC_f3;
const int neededLoadf1;
const int neededLoadf2;
const int neededLoadf3;
const double rate1;
const double rate2;
const double rate3;
const double mu1=2/Tr;
const double mu2=3/Tr;
const double mu3=4/Tr;
const double mu4=5/Tr;

module ReconfiguarationModule

s : [0..7] init 0;
fault:[0..4] init 0;//0: no faults,1: fault1 in feeder1, 2: fault1 in feeder2, 3: fault1 in feeder3,
reconfigurationFrom:[0..3] init 3;// 0: no reconfiguration, 2: reconfiguration from feeder2, 3: reco
repair:bool init false;//false=no repair, true= repaire

[] s=0 -> lambda_1:(s'=1)&(fault'= 1) +lambda_2:(s'=2) & (fault'= 2)+ lambda_3:(s'=3) &(fault'= 3)
+ lambda_4:(s'=4)&(fault'=4) + (1- lambda_1 - lambda_2 -lambda_3 - lambda_4): (s'=0)&(repair'=false)
[] s=1 & fault= 1 &  (available_HC_f3 >= neededLoadf1) -> rate1:(s'=5) & (reconfigurationFrom'=3)
 & (available_HC_f3'=(available_HC_f3 - neededLoadf1)) +(1-rate1):(s'=1);
[] s=1 & fault= 1 &  (available_HC_f2 >= neededLoadf1) -> 1:(s'=6) & (reconfigurationFrom'=2)&
(available_HC_f2'=available_HC_f2 - neededLoadf1)+  (1-rate2):(s'=1);
[] s=1 & fault= 1 &  ((available_HC_f2 < neededLoadf1 ) & (available_HC_f3 < neededLoadf1 ))
 -> 1:(s'=7) & (reconfigurationFrom'=0)+ (1-rate3):(s'=1);
[]s=2 & fault= 2-> mu1:(s'=0)&(repair'=true)+ (1-mu1):(s'=2);
[]s=3 & fault= 3-> mu1:(s'=0)&(repair'=true)+ (1-mu1):(s'=3);
```

Figure 5.8 PRISM Model Reconfiguration Module.

5.6.1 Qualitative Verification

We begin by verifying the deadlock problem in the whole system using the CTL formula: $E[F\text{“}deadlock\text{”}]$, as shown in Fig. 5.9. The formula was proven to be false. Moreover, the termination property is guaranteed since finite state machines are generated in ZiZo-v2 and PRISM model checker.

In the order to verify the confluence property, we have checked that no more than one scenario is executed at the same time in each RFB. The PCTL formula $P =?[F(p = 11 \ \& \ p = 12)]$, is proven to be false for $PIOC1$ RFB, where $p = 11$ is the state of the PIOC module corresponding the ECC slave es_{11}, and $p = 12$ is the state corresponding the ECC slave es_{12} in $PIOC1$. Thus, the ECC master of $PIOC1$ never executes two ECC slaves at the same time (respectively, all the other instances are proven) and only one is in active status.

Additionally, it is proven that the $CSWI1$ RFB executes only one scenario. We have verified this property using the PCTL formula: $P =?[F(p = 2 \ \& \ p = 3)]$, where $p = 2$ indicates the ECC slave es_{21} corresponding to the $FaultOperation$ and $p = 3$ indicates the ECC slave es_{22} corresponding to $NormalOperation$ in $CSWI1$. PRISM certifies

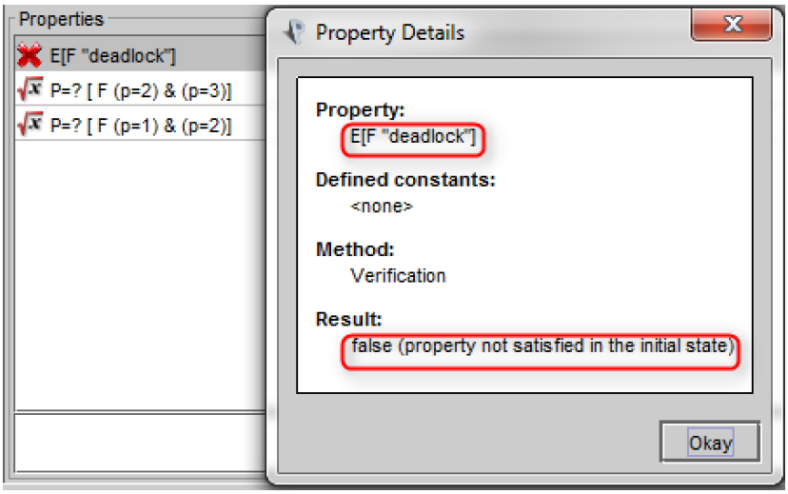


Figure 5.9 Property of deadlock freedom using PRISM 4.6

that the probability of reaching ECC slave es_{21} and es_{22} simultaneously is zero, i.e. false, as illustrated in Fig. 5.10.

We have also verified the concurrency property which is the ability to perform several behaviours at the same time. We checked the simultaneous occurrence of $reqStat$ event with the $fault$ reconfiguration event in $PIOC1$ RFB. In this case, concurrency is not allowed since the priority is for the reconfiguration events. In the RFB concept, reconfiguration events are more prioritised compared to standard events. The CTL formula $E[F(s = 1 \ \& \ p = 1)]$ is proven to be false.

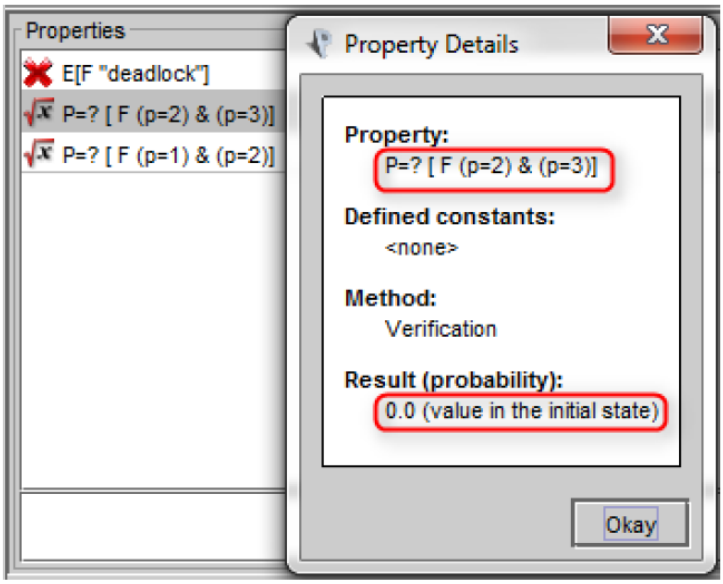


Figure 5.10 Simultaneous Scenarios Execution Property.

5.6.2 Quantitative Verification

RFBA methodology aims to gain an insight into the design of efficient reconfigurable distributed control systems. Using the PRISM model checker, we can not only analyse the system behaviour, but also estimate system performances by computing probabilities using PCTL and CSL in order to specify quantitative properties.

We assume that fault $f2$ between $ROS5$ and $ROS6$ occurs in $feeder2$ before that fault $f1$ in $feeder1$ has been repaired. To restore the fault $f2$ in $PIOC6$, the system should be informed by the results of reconfiguration. The reconfiguration depends mainly on the real-time available headroom capacity in non-faulty feeders and the current needed load. When $f1$ occurs, the reconfiguration can fail or succeed. In the case of:

(1) no available supply, the reconfiguration is failed. $CSWI5$ sends $help$ to $CSWI4$ via $CSWI3$ and $CSWI2$. The control switch $CSWI4$ has already known the available headroom capacity in $feeder3$. It compares it with the needed load in the faulty section and then decides.

(2) successful reconfiguration where there is

(a) a supply from $feeder2$: $CSWI5$ sends to $CSWI3$ new event $HeadRep$ associated with the new headroom capacity. The $ROS2$ RFB is notified by the change and searches another supply from $feeder3$.

(b) or a supply from $feeder3$. $CSWI5$ sends the needed load to $CSWI4$ that calculates the current available headroom capacity and then decides if satisfies the loads o not.

The reconfiguration module, depicted in Fig. 5.8, presents the fault states. The initial state ($s = 0$) is the idle state of the system. If one fault f_1 occurs in $feeder1$, then the system goes to state ($s = 1$). From this state, several cases are possible:

(a) If the needed load in the faulty section is less than or equal to the available headroom request in $feeder3$, then state ($s = 5$) is reached to get alternative supply from $feeder3$, and consequently the reconfiguration succeeds,

(b) If the needed load in $feeder1$ is less than or equal to the available headroom request in $feeder2$, then state ($s = 6$) is reached to get supply from $feeder2$ and accordingly the reconfiguration succeeds,

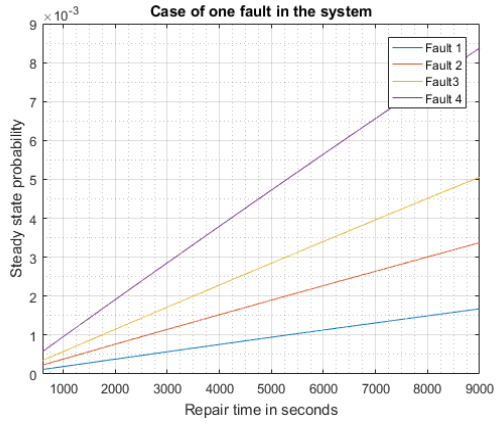
(c) If the needed load in $feeder1$ is greater than the available headroom request in $feeder2$ and $feeder3$, then state ($s = 7$) is reached and the reconfiguration fails in $feeder1$.

Table 5.2 Steady-state properties.

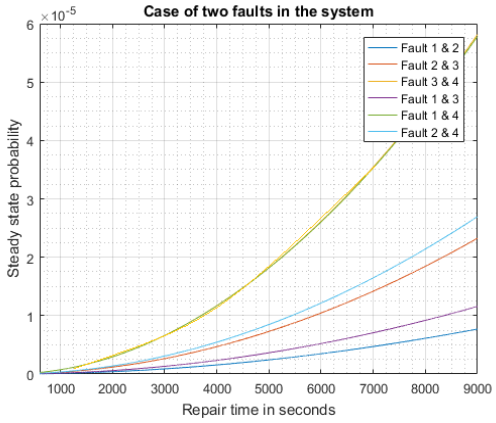
Formula	Results
$S=?[s=1]$ $S=?[s=2]$ $S=?[s=3]$ $S=?[s=4]$	Fig. 5.11(a)
$S=?[s=1 \ \& \ s=2]$ $S=?[s=1 \ \& \ s=3]$ $S=?[s=1 \ \& \ s=4]$ $S=?[s=2 \ \& \ s=3]$ $S=?[s=2 \ \& \ s=4]$ $S=?[s=3 \ \& \ s=4]$	Fig. 5.11(b)
$S=?[s=1 \ \& \ s=2 \ \& \ s=3]$ $S=?[s=1 \ \& \ s=3 \ \& \ s=4]$ $S=?[s=2 \ \& \ s=3 \ \& \ s=4]$ $S=?[s=1 \ \& \ s=2 \ \& \ s=4]$	Fig. 5.11(c)
$S=?[s=1 \ \& \ s=2 \ \& \ s=3 \ \& \ s=4]$	Fig.5.11(d)

If a repair agent repairs the fault f_1, the corresponding $PIOC1$ of the faulty section will returns to its $normalOperation$ scenario with a repair rate equal to $\frac{2}{Tr}$. The same logic is applied to other faults.

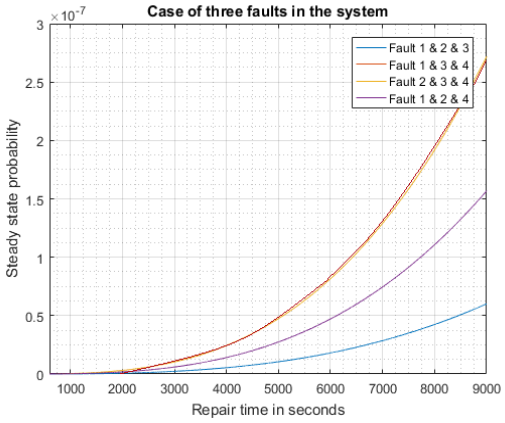
Using PRISM, we can evaluate also the steady state probability that is the long-run probability to reach a state st. The operator S in PRISM is applied to provide the long-run probability that one or several faults occur. We can estimate the steady state probability when (i) one fault appears, (ii) two faults appear in the system where the first fault occurs before the previous one is fixed, (iii) three faults appears successively without manual repair intervention and finally, (iv) four faults occur successively. The results are shown in Fig. 5.11. The latter reveals the steady state probability of one to four faults in the system when T_r varies between 10 and 150 minutes (i.e. 600 and 9000s). The curves (a, b, c and d) are obtained on applying the PCTL formula included in Table 5.2 where $s=1$ (respectively $s=2$, $s=3$, $s=4$) is the state that denotes the occurrence of fault $f1$ (respectively $f2$, $f3$, and $f4$). We deduce that the steady state probability grows in accordance with the repair time parameter. If the repair time T_r is equal to 150 minutes, the probability of successful or failed reconfiguration in the all the possible cases with different order can be determined.



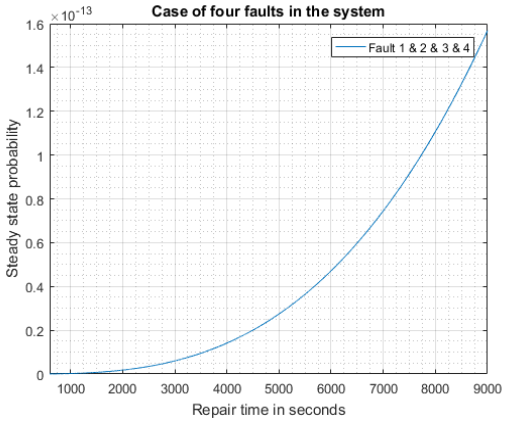
(a) Case of single fault



(b) Case of two simultaneous faults



(c) Case of three successive Faults



(d) Case of four successive faults

Figure 5.11 Steady state probability in the case of one, two, three and four fault/s.

Furthermore, the probability of reconfiguration failure p_{fr} denoting the case of no supply from the alternative feeders can be estimated thanks to PRISM and the following PCTL formula shown in Table 5.3. We assume that $deadline$ variable is equal to $364 * 24 * 3600$ seconds.

(i) In the case of fault occurrence f_1 and the case that the headroom capacity exceeds the required load, p_{fr} is equal to 0 and the alternative supply always satisfies the needed load, using the formula (A) where $s = 7$ denotes the case of no available supply for $feeder1$ within one year, and $s = 1$ corresponds to the occurred fault $f1$;

(ii) In the case of two successive faults f_1 & f_3 without repair, p_{fr} is equal to $1.09 * 10^{-5}$ using the formula (B);

(iii) In the case of two successive faults f_3 & f_4 without repair, the probability of failed reconfiguration p_{fr} is equal to $5.1 * 10^{-5}$ using the formula (C);

(iv) And finally, in the case of three successive faults f_1 & f_2 & f_3 without repair intervention, the probability of failed reconfiguration p_{fr} is equal to 1 using the formula (D) where $s = 55$ is the state that denotes no reconfiguration in the whole system. This result is obtained since $feeder3$ supplies $feeder1$ and $feeder2$ when its available headroom capacity exceeds the required load in $feeder1$ and $feeder2$.

Table 5.3 The probability of failed reconfiguration properties.

Notation	PRISM Formula
A	$P =?[trueU <= (deadline)(s = 7)\ \&\ (s = 1)]$
B	$P =?[trueU <= (deadline)(s = 14)\ \&\ (s = 5 \vert s = 6 \vert s = 7)]$
C	$P =?[trueU <= (deadline)(s = 24)\ \&\ (s = 17)]$
D	$P =?[trueU <= (deadline)(s = 55)]$

Finally, we can expect in our case study that the gain is the probability of satisfying all loads in a period without recoursing to a maintenance agent intervention. We assume that the state "allsupplied" is reached when all sectionalising switches are closed and $(p = 35|p = 36|p = 37|p = 38)$ are manual repair states. Using the same assumption, we have verified that the maximum probability of not using manual maintenance within a deadline of 30 days is equal to $8,75 * 10^{-3}$ using the following formula:

$$Pmax =?[F(!(p = 35|p = 36|p = 37|p = 38)\ \&\ \text{“}allsupplied\text{”})Ut >= deadline]$$

5.6.3 PRISM Simulation

Using PRISM, we can also analyse the system using the simulation interface. The simulation results in the case of three successive faults without repair intervention (f_1, then f_2, after that f_3) are illustrated in Fig. 5.12 where the repair time is six hours, the needed load $neededLoadf1$ in $feeder1$ can range between 300 and 390 (Watt) at the instant when $f1$ occurs. The needed load $neededLoadf2$ in $feeder2$ can range between 50 and 150 (Watt) when $f2$ occurs. The available headroom capacity in $feeder2$ can range between 100 and 200 $availableHCf2 \in [100, 200]$. It can range in $feeder3$ between 400 and 521 $availableHCf3 \in [400, 521]$.

In the simulation scenario, we suppose that the fault f_2 appears before the repair of f_1, and after that, another fault f_3 occurs before the repair of f_2. The real-time needed load in the faulty feeder and the real-time available headroom capacity are both defined as time series in the simulation. As a result, it is clear that the reconfiguration succeeds twice and then fails. The simulation is detailed as follows:

(i) At the instant $t = 0.48$, a fault f_1 occurs in $feeder1$, the available headroom capacity in $feeder3$ which is equal to 400W is higher than the actually required

load in $feeder1$ 300W, and then the reconfiguration will be held and the reached state is $s = 5$;

(ii) At $t = 3.52$, a fault f_2 appears in $feeder2$, the ROS switches in $feeder2$ ask its neighbours to get supply, the current needed load is equal to $100W$ and the available Headroom capacity in $feeder3$ is $100W$. Then the supply is ensured by $feeder3$ and the reconfiguration succeeds (state $s = 9$);

(iii) At $t = 29.54$, a fault f_3 occurs in $feeder3$ that engenders no supply for $feeder1$ and $feeder2$. Therefore, the system reaches the state $s = 10$ that indicates a failed reconfiguration.

(iv) At $t = 29.64$, the system returns to its idle state $s = 0$ after a manual repair ($repair = true$).

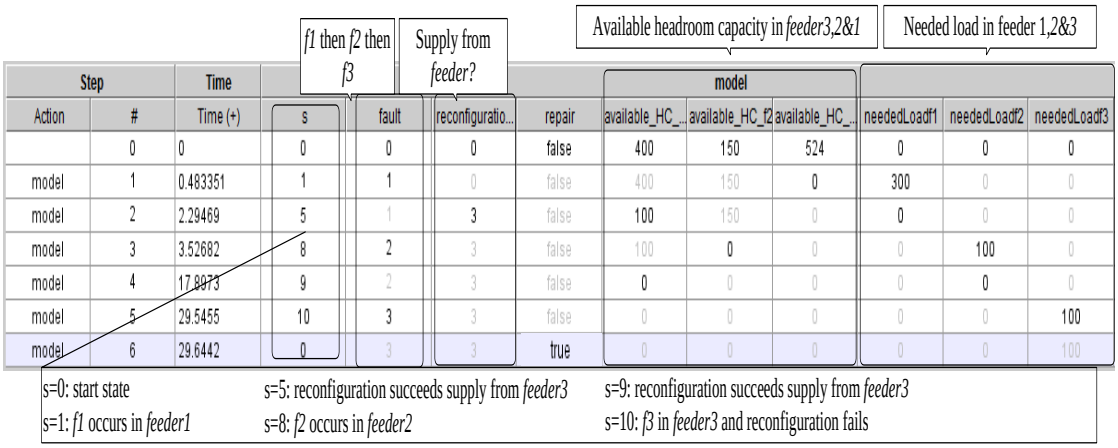


Figure 5.12 Fault simulation using PRISM.

5.6.4 System Availability

The system availability is the probability that the system is operating at a satisfying level and all the customers are supplied. It can be ranged from 0%, i.e. never available, to 100% always available. It is then equal to the probability that the system is in its idle state or in one of the possible states that succeeds to get a supply from a power source. In case of error at instant t, or no supply, the system becomes not available.

The same parameters in the simulation section are assumed. We vary here the repair time parameter during a period of eight days. As a result, the system availability decreases when the repair time increases as shown in Fig. 5.13. In order to obtain a system availability more than 99.5%, the time to detect and repair the faults should be less than $0.5 * 10^5$ seconds, i.e. thirteen hours approximately. The designer and engineers

can improve the system design and reduce the repair cost by adding another source in $feeder1$.

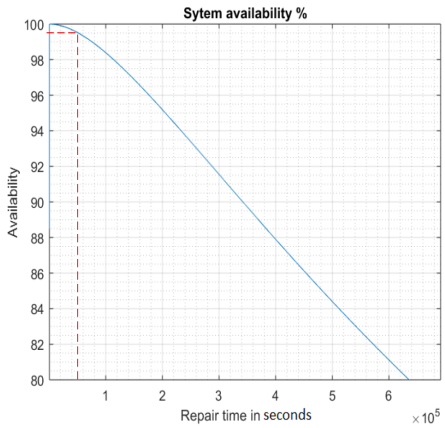


Figure 5.13 The availability of the FLISR system during 8 days.

5.7 Evaluation of Performance

We compared RFBA approach with the approaches proposed in [Sinha et al., 2015], [Bhatti et al., 2017], [Zhabelova et al., 2015], and [Zhabelova and Vyatkin, 2012] as presented in Table 5.4. The authors in [Higgins et al., 2011] model every section of transmission wires with basic function blocks. [Zhabelova and Vyatkin, 2012] model each logical node with a complex composite function block. In the proposed RFBA approach, each logical node in the FLISR system is modelled with an RFB considering the case of one or multiple faults. RFBA splits the execution control chart of $CSWI$ function block, as shown in Fig. 9 in [Zhabelova et al., 2015], into an ECC master and two ECC slaves. Accordingly, a slave for the normal operation and the other for executing the fault operation. This separation aims to switch smoothly from a scenario to another depending on the coming events and data. In the case of an occurred fault, all the reconfigurable function blocks presented the faulty section switch directly from the ECC slave $normalOperation$ to the ECC slave $FaultOperation$ by sending an output event of reconfiguration split in the model.

In comparing the proposed approach with [Bhatti et al., 2017] approach, an estimation of faults is provided in both studies. However, modelling directly IEC 61499 to a Markov chain model is not suitable for reconfigurable systems since it can increase the explosion state problem as explained in the previous chapter.

Compared with the study of [Sinha et al., 2015] based on hierarchical and concurrent execution control chart HCECC, the reconfiguration feature and the easy switch from a level to another are not supported. Only concurrency is ensured that is not needed inside a reconfigurable function block.

Table 5.4 Comparative study.

references	ECC hierarchy	Execution strategy	Validation	Features
[Zhabelova et al., 2015],[Zhabelova and Vyatkin, 2012]	Standard ECC	Synchronous execution	The authors use Co-simulation approach to detect a single fault	Modelling with basic function blocks. Deliberative and reactive layers in Logical Node in IEC 61850. Normal and faulty mode are in the same ECC as depicted in Fig. 6 and 9 in [Zhabelova et al., 2015].
[Strasser et al., 2014, Zoitl and Strasser, 2017]	function blocks management	The authors propose reconfiguration commands. However, the commands are executed by the user and the function block type definitions must exist in the target execution environment.	System inconsistencies can appear after commands execution which disturbs the system. Simulation and model checking using SESA are used.	A reconfiguration command is able to create/update/delete FBs on run-time.
[Sinha et al., 2015]	Hierarchical concurrent ECC (HCECC)	Concurrent and parallel execution, No inter-level communication, Static strategy	No verification	HCECCs are purely syntactic sugar. Transformation of the parallel HCECC into CFB.
Current proposed work	Master-slave ECC (MSECC)	The execution of the suitable scenario according to reconfiguration event, Inter-level transitions, Dynamic strategy: activation and deactivation of reconfiguration scenarios,	Model checking using PRISM for the verification of deterministic and probabilistic properties. Verification and simulation process are possible. The authors consider the case of one or multiple faults and their influence on the system availability.	Separation between Reconfiguration and control model within the RFB for readability reason. Formal Modelling using GR-TNCES model. Qualitative and quantitative verification of the system. Estimation of the system availability.
[Bhatti et al., 2017]	Basic function blocks	synchronous execution semantics [Yoong et al., 2014] based on a time-triggered way of execution	Unified functional safety assessment of industrial automation systems using IEC 61499 and a safety standard IEC 61508	Qualitative and quantitative verification using PRISM. Direct translation from IEC 61499 model into a Markov chain.

In the comparative study about formal verification presented in Table 5.5, most studies have interested on qualitative analysis using model checkers such as SESA. Despite the benefits of probabilistic model checker, only our work and [Kwiatkowska et al., 2002] use PRISM to get quantitative analysis. GR-TNCES compared with other Petri nets is reconfigurable and probabilistic. While reconfiguration reduces notably the time of model rebuilding [Zhang, 2015] since it checks only the changed states on real-time verification, probability gives the designer an insight into system performance.

Table 5.5 Comparative study about verification.

References	**Model**	**Model checker**	**Features**	**Verification strategy**
[Vyatkin and Hanisch, 2000a, Pang and Vyatkin, 2008]	NCES	Vive/SESA	Qualitative analysis	High modelling effort.
[Stanica and Guéguen, 2004]	timed automata		Qualitative analysis	High modelling effort.
[Pang and Vyatkin, 2008]	TNCES	SESA	Qualitative analysis	Time constraints, functional and safety properties are considered. Automatic transformation of IEC 61499 model.
The proposed work[Guellouz et al., 2019]	GR-TNCES	PRISM model checker	Qualitative and quantitative analyses	Automatic modelling transformation. Probabilistic events, modular and dynamic verification is considered, verification of time, memory, and energy constraints.

To overview, RFBA approach has important advantages since the designer can describe the reconfiguration scenarios which cannot be easily designed using basic function blocks. The proposed master-slave execution control chart allows to promote the readability and maintainability. The proposed methodology analyses the system functionalities and estimates the system performance before deployment. The system designer can verify qualitatively the functional correctness, and assess its availability that depends on repair time in our case study and refine the system model. It permits to determine a real estimation of the system availability and the probability of failed reconfiguration. Several qualitative and quantitative properties are easily checked before deployment thanks to ZIZO toolchain and PRISM such as: (i) deadlock detection, (ii) confluence properties, (iii) system feasibility before and after reconfiguration, (iv) estimation of the reconfiguration failure, (v) system availability, (vi) best repair time estimation.

The detection of the worst cases before deployment is the major strength of the approach that the system designer needs to evaluate and estimate in his attempt to ameliorate

the current model. Accordingly, the designer can improve the system design and refine the model depending on the results. For example, he can modify the hardware of the system topology by adding other energy sources and determine the optimal repair time and other parameters before deployment in the FLISR case study.

5.8 Conclusion

The goal of this chapter is to show that design based on RFBs, formal modelling and verification can guarantee system correctness and can offer an insight into performance. RFBA approach is applied on a smart grid system. We begin by system specification and then the design based on RFBs, i.e. the new extension of IEC 61499. Formal modelling and model checking are used. The developed RFBTool environment transforms automatically the RFB model into a GR-TNCES model, and ZIZO v2 converts the generated model into a deterministic Markov chain to check the designed model correctness and evaluate system performance. In the FLISR case study, several faults have been injected into the model to estimate system availability. The fault and repair events are characterised by probabilistic information needed to get a better insight into the system performance thanks to the probabilistic model checker PRISM. In order to obtain an optimised system, several properties must be satisfied such as deadlock freedom, confluence, system feasibility before and after reconfiguration. Additionally, the probability of a failed reconfiguration, system availability, and estimation of best repair time can be estimated.

Chapter 6

General Conclusion

6.1 Context and Challenges

Software reconfiguration is increasingly becoming a crucial need in industrial automation. The market fluctuations and the introduction of new strategies have led to several changes in the system design, implementation and verification. Our research brings us to the conclusion that IEC 61499 is the best standard which allows the design and development of platform-independent applications for industrial processes, measurement and control systems. It offers portability, interoperability and configurability features. Clearly, IEC 61499 is based on function blocks that are software components representing hardware or service elements in an application. However, the design and development of reconfiguration scenarios in reconfigurable distributed control systems are still not easy. The static definition of a basic function block control logic, i.e. the execution control chart ECC, imposes manual adjustment for every change in the environment or user requirements. With the ongoing changes, a smart execution control chart is recommended. Using basic function block ECC, the system designer needs several states, transitions, and actions to develop all reconfiguration scenarios. This results in a cumbersomeness in the design, low readability and maintainability as well as an increase in design complexity. Hence, a hierarchical ECC structure is required to provide a dynamic logic to the function block and a readable design. A separation between control and reconfiguration models proves effective in decreasing design complexity. While the control model defines the behaviour, the reconfiguration model supervises each change to modify the system behaviour appropriately.

IEC 61499 standard faces another limitation which is the verification of reconfigurable control systems. Although verification is crucial for ensuring system feasibility and safety, it is not supported by any compliant tool. In fact, the reconfiguration process changes the system behaviour and execution model. Consequently, the guarantee of the system feasibility cannot be provided after executing a reconfiguration process. Additionally, most IEC 61499 compliant development tools offer simulation techniques that

do not prove system correctness, deadlocks, and concurrent tasks.

Another important conclusion drawn from this work is that the analysis of the QoS is of great importance for engineers and practitioners. However, only deterministic properties are checked in the existing research works. A quantitative analysis is necessary. It offers the possibility to quantify and analyze some probabilistic properties such as system availability, probability of successful reconfiguration or failed reconfiguration. This analysis also grants an efficient insight into the system while triggering the need for system refinement.

6.2 Contributions and Originalities

We propose in this thesis a reconfigurable function block as an extension to IEC 61499 standard to support reconfiguration inside the function block. It is characterised by an extra reconfiguration event associated with a reconfiguration data in its interface. Moreover, a dynamic master-slave execution control chart MSECC is proposed to separate the reconfiguration model from the control model. While the control model defines the behaviour, the reconfiguration model, i.e. the ECC master, supervises the changes and executes the best control model,i.e. the ECC slave. Thanks to a decision algorithm and a reconfiguration matrix that includes several rules, the RFB selects and executes dynamically the most adequate reconfiguration scenario.

In this thesis, we have worked on a new methodology based on RFBs called (RFBA) to deal with design, modelling and verification of reconfigurable distributed control systems in industrial automation. It is addressed to simplify the design and verify reconfiguration feasibility. In order to verify a designed system with RFBs, formal modelling is used. We transform automatically the RFB model into a Generalised Reconfigurable Timed Net Condition/Event system (GR-TNCES) model which is a reconfigurable Petri net class preserving RFB features and semantic. In GR-TNCES, switching from a configuration to another is made possible by enabling/disabling components. This fits perfectly with the proposed master-slave execution control chart inside for modelling unpredictable reconfiguration events. We integrate probability in the generated formal model to provide a quantitative analysis using a probabilistic model checker PRISM. This interest is mainly due to the fact that checking and evaluating system performance as well as estimating risks after reconfiguration have a major virtue for engineers.

Furthermore, the proposed RFBA approach allows a modular verification that mitigates the explosion state problem by verifying the model module by module. In order to analyse qualitative and quantitative properties specified in computation tree logic, we attempt to transform the GR-TNCES model into a discrete Markov chain model interpreted by PRISM. ZiZo v2 tool is used to analyse the GR-TNCES models and export them to PRISM models.

The RFBA methodology is supported by a toolchain that includes RFBtool, ZiZo v2 and PRISM model checker. RFBtool is a new environment for RFB, it creates and edits reconfigurable function blocks models and converts them automatically into GR-TNCES models. If the verification is successful, a C++ code is generated and can be deployed in any devices.

Several properties are easily checked and estimated using CTL and PCTL properties such as system feasibility before and after reconfiguration, deadlock detection, confluence, estimation of the reconfiguration failure, system availability, and best repair time that cannot be proved using simulation. The detection of the worst cases before deployment is also a major value of the approach that practitioners need to estimate. Accordingly, they can refine and enhance the designed system.

The proposed framework also stems its originality from the fact that it can be applied in any control system requiring reconfiguration, verification and performance estimation. In order to show RFB feasibility, we adopt our contribution to a distributed power system as a case study. As a result, we proved confluence properties, deadlocks freedom, system availability, probability of successful reconfiguration. In addition, we compared the results of our approach with the related works. It seems that in using RFBs the number of application components becomes lower than the components of an application based on BFBs, i.e. number of events, states and transitions. Moreover, the verification time is reduced thanks to a modular verification.

6.3 Perspectives

As perspectives to our work, we will:

- extend RFBA approach to support reconfiguration in cloud for internet of things (IoT) systems using web services to manage several reconfiguration matrices. The reconfiguration model can be supervised by web services deployed in the cloud to improve monitoring and online reconfiguration. Thus, the rules in the reconfiguration matrix can be adjusted automatically by a web service.

- deal with the security of distributed RFBs in future work since a reconfiguration event can be an attack that disturbs the predicted behaviours. In this case, securing any reconfiguration of the execution control chart master is required for guaranteeing correct RFBs behaviour. Several security policies can be automatically applied in order to restrict control access.

- improve our RFBtool interface and extend it to deploy RFBs in several independent platforms and operating systems such as FreeRTOS. The current generated code is only deployable on POSIX. Moreover, the existing IEC 61499 benchmarks are very

basic. Therefore, we will develop reconfigurable and complicated benchmarks to help engineers to use the proposed framework.

– work also on the automatic model rebuilding based on artificial intelligence in order to refine automatically the existing models after a not satisfied verification. The refinement is based on the results of qualitative and quantitative analyses as well as the defined limits to obtain optimised system satisfying the needs of companies and engineers.

Bibliography

R. L. A. Zoitl. *Modelling control systems using IEC 61499. Applying function blocks to distributed systems, IET Control Engineering Series 95. 2nd Edition*. London, United Kingdom, 2014.

Y. Alsafi and V. Vyatkin. Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing. *Robotics and Computer-Integrated Manufacturing*, 26(4):381–391, 2010.

F. Ameri, C. Urbanovsky, and C. McArthur. A systematic approach to developing ontologies for manufacturing service modeling. In *Proceedings of the workshop on ontology and semantic web for manufacturing*, volume 14. Citeseer, 2012.

M. Amrani, L. Lucio, G. Selim, B. Combemale, J. Dingel, H. Vangheluwe, Y. Le Traon, and J. R. Cordy. A tridimensional approach for studying the formal verification of model transformations. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 921–928. IEEE, 2012.

C. Angelov, K. Sierszecki, and N. Marian. Design models for reusable and reconfigurable state machines. In *Proceedings of the Embedded and Ubiquitous Computing – EUC 2005*, volume 3824, pages 152–163, December 2005. doi: 10.1007/11596356_18.

M. Arenas, W. Fan, and L. Libkin. On verifying consistency of XML specifications. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 259–270, 2002.

S. Arora and M. P. Rao. Probabilistic model checking of incomplete models. In *Proceedings of the International Symposium on Leveraging Applications of Formal Methods*, pages 62–76. Springer, 2016.

I. Batchkova, Y. Belev, and D. Tzakova. IEC 61499 based control of cyber-physical systems. *Industry 4.0*, 5(1):10–13, 2020.

M. O. Ben Salem, O. Mosbahi, M. Khalgui, Z. Jlalia, G. Frey, and M. Smida. Brometh: Methodology to design safe reconfigurable medical robotic systems. *International Journal of Medical Robotics and Computer Assisted Surgery*, 15, 2016. doi:10.1002/rcs.1786.

Z. E. Bhatti, P. S. Roop, and R. Sinha. Unified functional safety assessment of industrial automation systems. *IEEE Transactions on Industrial Informatics*, 13(1):17–26, Feb 2017. ISSN 1941-0050.

G. Black and V. Vyatkin. Intelligent component-based automation of baggage handling systems with IEC 61499. *IEEE Transactions on Automation Science and Engineering*, 7 (2):337–351, 2010.

R. Brennan, M. Fletcher, and D. Norrie. A holonic approach to reconfiguring real-time distributed control systems. In *Proceedings of the Multi-Agent Systems and Applications II*, pages 323–335, 07 2002a. doi: 10.1007/3-540-45982-0_21.

R. Brennan, P. Vrba, P. Tichý, A. Zoitl, C. Sünder, T. Strasser, and V. Marík. Developments in dynamic and intelligent reconfiguration of industrial automation. *Computers in Industry*, 59:533–547, August 2008. doi: 10.1016/j.compind.2008.02.001.

R. W. Brennan, X. Zhang, Y. Xu, and D. H. Norrie. A reconfigurable concurrent function block model and its implementation in real-time java. *Integrated Computer-Aided Engineering*, 9(3):263–279, 2002b.

G. Cengic and K. Akesson. Definition of the execution model used in the fuber IEC 61499 runtime environment. In *Proceedings of the 2008 6th IEEE International Conference on Industrial Informatics*, pages 301–306. IEEE, 2008.

J. Christensen, T. Strasser, A. Valentini, V. Vyatkin, and A. Zoitl. The IEC 61499 function block standard: Software tools and runtime platforms. *ISA Automation Week*, 12, January 2012a.

J. H. Christensen, T. Strasser, A. Valentini, V. Vyatkin, and A. Zoitl. The IEC 61499 function block standard: Overview of the second edition. *ISA autom. Week*, 6:6–7, 2012b.

E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

E. M. Clarke, E. A. Emerson, and J. Sifakis. Model checking: algorithmic verification and debugging. *Communications of the ACM*, 52(11):74–84, 2009.

W. Dai and V. Vyatkin. Redesign distributed PLC control systems using IEC 61499 function blocks. *IEEE Transactions on Automation Science and Engineering*, 9(2):390–401, 2012.

W. Dai, V. Vyatkin, C. Chen, and X. Guan. Modeling distributed automation systems in cyber-physical view. In *Proceedings of the 2015 IEEE 10th Conference on Industrial*

Electronics and Applications (ICIEA), pages 984–989, 2015. doi: 10.1109/ICIEA.2015.7334251.

W. Dai, V. Vyatkin, J. H. Christensen, and V. N. Dubinin. Bridging service-oriented architecture and IEC 61499 for flexibility and interoperability. *IEEE Transactions on Industrial Informatics*, 11(3):771–781, 2015.

W. W. Dai and V. Vyatkin. A case study on migration from IEC 61131 PLC to IEC 61499 function block control. In *Proceedings of the 2009 7th IEEE International Conference on Industrial Informatics*, pages 79–84. IEEE, 2009.

E. Demin, S. Patil, V. Dubinin, and V. Vyatkin. IEC 61499 distributed control enhanced with cloud-based web-services. In *Proceedings of the 2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, pages 972–977. IEEE, 2015.

D. Drozdov, S. Patil, and V. Vyatkin. Formal modelling of distributed automation cps with cp-agnostic software. In *Proceedings of the International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*, pages 35–46. Springer, 2016.

V. Dubinin and V. Vyatkin. Towards a formal semantic model of IEC 61499 function blocks. In *Proceedings of the 2006 4th IEEE International Conference on Industrial Informatics*, pages 6–11. IEEE, 2006.

V. Dubinin, V. Vyatkin, and H.-M. Hanisch. Modelling and verification of IEC 61499 applications using prolog. In *Proceedings of the 2006 IEEE Conference on Emerging Technologies and Factory Automation*, pages 774–781. IEEE, 2006.

F. Duhem, F. Muller, and P. Lorenzini. Reconfiguration time overhead on field programmable gate arrays: reduction and cost model. *IET Computers and Digital Techniques*, 6(2):105–113, 2012.

Eclipse. 4diac IDE – IEC 61499 Compliant Development Environment, 2020. URL `https://www.eclipse.org/4diac/en_ide.php`. [Accessed on 2020-12-29].

H. K. Flora and S. V. Chande. A systematic study on agile software development methodologies and practices. *International Journal of Computer Science and Information Technologies*, 5(3):3626–3637, 2014.

V. Forejt, M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma. Incremental runtime verification of probabilistic systems. In *Proceedings of the International Conference on Runtime Verification*, pages 314–319. Springer, 2012.

S. Guellouz, A. Benzina, M. Khalgui, and G. Frey. Zizo: A complete tool chain for the modeling and verification of reconfigurable function blocks. In *Proceedings of the*

10th Int. Conf. Mobile Ubiquitous Comput., Syst., Services Technol.(UBICOMM), pages 144–151, 2016a.

S. Guellouz, A. Benzina, M. Khalgui, and G. Frey. Reconfigurable function blocks: Extension to the standard IEC 61499. In *Proceedings of the 13th International Conference of Computer Systems and Applications IEEE/ACS (AICCSA) 2016*, pages 1–8, Nov 2016b. doi: 10.1109/AICCSA.2016.7945784.

S. Guellouz, A. Benzina, M. Khalgui, G. Frey, Z. Li, and V. Vyatkin. Designing efficient reconfigurable control systems using IEC 61499 and symbolic model checking. *IEEE Transactions on Automation Science and Engineering*, 16(3):1110–1124, July 2019. ISSN 1558-3783. doi: 10.1109/TASE.2018.2868897.

H. Hanisch and A. Lüder. Modular modelling of closed-loop systems. In *Proceedings of the Colloquium on Petri Net Technologies for Modelling Communication Based Systems, Berlin, Germany*, pages 103–126, 1999.

H.-M. Hanisch, J. Thieme, A. Luder, and O. Wienhold. Modeling of PLC behavior by means of timed net condition/event systems. In *Proceedings of the 1997 IEEE 6th International Conference on Emerging Technologies and Factory Automation Proceedings, EFTA'97*, pages 391–396. IEEE, 1997.

H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

N. Higgins, V. Vyatkin, N. C. Nair, and K. Schwarz. Distributed power system automation with iec 61850, iec 61499, and intelligent control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(1):81–92, 2011. doi: 10.1109/TSMCC.2010.2046322.

HOLOBLOC. XML Document Type Definitions (DTDs), 2019a. URL `http://ftp.holobloc.com/xml/index.htm`. [accessed on 2020-01-04].

HOLOBLOC. Resources for the New Generation of Automation and Control Software, 2019b. URL `https://www.holobloc.com/doc/fb/rt/net/pubsub.htm`. [accessed on 2020-01-04].

HOLOBLOC. FBDK 8.2 - The Function Block Development Kit, 2020. URL `https://www.holobloc.com/fbdk8/index.htm`. [accessed on 2020-01-04].

A. Homay, A. Zoitl, M. d. Sousa, and M. Wollschlaeger. A survey: Microservices architecture in advanced manufacturing systems. In *Proceedings of the 2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 1165–1168, 2019.

A. Hopsu, U. D. Atmojo, and V. Vyatkin. On portability of IEC 61499 compliant structures and systems. In *Proceedings of the 2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, pages 1306–1311, 06 2019. doi: 10.1109/ISIE.2019.8781290.

S. J. Hu. Evolving paradigms of manufacturing: From mass production to mass customization and personalization. *Procedia CIRP*, 7:3 – 8, 2013. ISSN 2212-8271. doi: https://doi.org/10.1016/j.procir.2013.05.002. Forty Sixth CIRP Conference on Manufacturing Systems 2013.

C. C. Insaurralde. Modeling standard for distributed control systems: IEC 61499 from industrial automation to aerospace. In *Proceedings of the 35th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, pages 1–8, 2016. doi: 10.1109/DASC.2016.7777980.

I. Ivanova-Vasileva, C. Gerber, and H.-M. Hanisch. Basics of modelling IEC 61499 function blocks with integer-valued data types. volume 41, pages 169–174, Szczecin, Poland, 2008. Elsevier.

Z. Jakovljevic, S. Mitrovic, and M. Pajic. Cyber physical production systems—an IEC 61499 perspective. In *Proceedings of the 5th International Conference on Advanced Manufacturing Engineering and Technologies*, pages 27–39, 04 2017. ISBN 978-3-319-56429-6. doi: 10.1007/978-3-319-56430-2_3.

P. Jhunjhunwala, U. D. Atmojo, and V. Vyatkin. Towards implementation of interoperable smart sensor services in iec 61499 for process automation. In *Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1409–1412, 2020. doi: 10.1109/ETFA46521.2020.9211925.

A. R. Kan. *Machine scheduling problems: classification, complexity and computations*. Springer Science and Business Media, 2012.

M. Khalgui, O. Mosbahi, Z. Li, and H.-M. Hanisch. Reconfiguration of distributed embedded-control systems. *IEEE/ASME Transactions on Mechatronics*, 16(4):684–694, 2010.

O. Khlifi, O. Mosbahi, M. Khalgui, and G. Frey. GR-TNCES: New extensions of R-TNCES for modelling and verification of flexible systems under energy and memory constraints. In *Proceedings of the 2015 10th International Joint Conference on Software Technologies (ICSOFT)*, volume 1, pages 1–8. IEEE, 2015.

O. Khlifi, C. Siegwart, O. Mosbahi, M. Khalgui, and G. Frey. Modeling and simulation of an energy efficient skid conveyor using zizo. In *Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics ICINCO*, 2016.

O. Khlifi, O. Mosbahi, M. Khalgui, G. Frey, and Z. Li. Modeling, simulation and verification of probabilistic reconfigurable discrete-event systems under energy and memory constraints. *Iranian Journal of Science and Technology, Transactions of Electrical Engineering*, 43:229–243, 2019.

W. Kim, S. Cha, and K. Sok. On the formal model for IEC 61499 composite function blocks. *CoRR*, abs/1805.08984, 2018. URL `http://arxiv.org/abs/1805.08984`.

Y. Koren, S. J. Hu, and T. W. Weber. Impact of manufacturing system configuration on performance. *CIRP Annals*, 47(1):369 – 372, 1998. ISSN 0007-8506. doi: https://doi.org/10.1016/S0007-8506(07)62853-4. URL `http://www.sciencedirect.com/science/article/pii/S0007850607628534`.

Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, A. Ulsoy, and H. Brussel. Reconfigurable manufacturing systems. *CIRP Annals-Manufacturing Technology*, 48: 527–540, 08 1999. doi: 10.1007/978-3-642-55776-7_19.

Y. Koren, X. Gu, and W. Guo. Reconfigurable manufacturing systems: Principles, design, and future trends. *Frontiers of Mechanical Engineering*, 13(2):121–136, 2018.

G. Koumoutsos and K. Thramboulidis. A knowledge-based framework for complex, proactive and service-oriented e-negotiation systems. *Electronic Commerce Research*, 9 (4):317, 2009.

K. Kruger and A. Basson. Multi-agent systems vs IEC 61499 for holonic resource control in reconfigurable systems. *Procedia CIRP*, 7:503–508, 12 2013. doi: 10.1016/j.procir.2013.06.023.

M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. In *Proceedings of the International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 200–204. Springer, 2002.

M. Kwiatkowska, G. Norman, and D. Parker. Quantitative analysis with the probabilistic model checker PRISM. *Electronic Notes in Theoretical Computer Science*, 153(2):5–31, 2006.

P. Leitão, V. Mařík, and P. Vrba. Past, present, and future of industrial agent applications. *IEEE Transactions on Industrial Informatics*, 9(4):2360–2372, 2012.

W. Lepuschitz, A. Zoitl, M. Vallée, and M. Merdan. Toward self-reconfiguration of manufacturing systems using automation agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(1):52–69, 2011.

J. Liu. *Intelligent control design and Matlab simulation*. Springer, 2018.

G. Lyu and R. W. Brennan. Towards IEC 61499 based distributed intelligent automation: A literature review. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2020. doi: 10.1109/TII.2020.3016990.

A. Malik, P. S. Roop, N. Allen, and T. Steger. Emulation of cyber-physical systems using IEC-61499. *IEEE Transactions on Industrial Informatics*, 14(1):380–389, 2017.

V. Mironovich, M. Buzdalov, and V. Vyatkin. Automatic generation of function block applications using evolutionary algorithms: Initial explorations. In *Proceedings of the 2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 700–705, July 2017. doi: 10.1109/INDIN.2017.8104858.

D. Missal, M. Hirsch, and H. Hanisch. Hierarchical distributed controllers - design and verification. In *Proceedings of the 2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pages 657–664, 2007. doi: 10.1109/EFTA.2007.4416832.

D. Mourtzis and M. Doukas. Decentralized manufacturing systems review: challenges and outlook. *Logistics Research*, 5(3):113–121, Nov 2012. ISSN 1865-0368. doi: 10.1007/s12159-012-0085-x. URL `https://doi.org/10.1007/s12159-012-0085-x`.

A. Mousavi and V. Vyatkin. Energy efficient agent function block: A semantic agent approach to IEC 61499 function blocks in energy efficient building automation systems. *Automation in Construction*, 54:127 – 142, 2015. ISSN 0926-5805. doi: https://doi.org/10.1016/j.autcon.2015.03.007. URL `http://www.sciencedirect.com/science/article/pii/S0926580515000412`.

nxtControl GmbH. Nxtcontrol- NXT Technology IDE, 2021. URL `https://www.nxtcontrol.com/en/engineering`. [Accessed on 2020-12-29].

U. of Oxford. PRISM Manual version 4.6, 2020. URL `http://prismmodelchecker.org/manual/Main/AllOnOnePage`. [Accessed on 2020-12-29].

C. Pang and V. Vyatkin. Automatic model generation of IEC 61499 function block using net condition/event systems. In *Proceedings of the 6th IEEE International Conference on Industrial Informatics*, pages 1133–1138, Daejeon, South Korea, 2008.

C. Pang, S. Patil, C.-W. Yang, V. Vyatkin, and A. Shalyto. A portability study of IEC 61499: Semantics and tools. In *Proceedings of the 2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 440–445. IEEE, 2014a.

C. Pang, V. Vyatkin, and W. Dai. Iec 61499 based model-driven process control engineering. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–8. IEEE, 2014b.

S. Patil, V. Vyatkin, and C. Pang. Counterexample-guided simulation framework for formal verification of flexible automation systems. In *Proceedings of the 2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1192–1197. IEEE, 2015.

C. A. Petri. *Kommunikation mit automaten*. PhD thesis, University of Bonn, 1962.

L. I. Pinto, C. D. Vasconcellos, R. S. U. Rosso, and G. H. Negri. Icaru-fb: An iec 61499 compliant multiplatform software infrastructure. *IEEE Transactions on Industrial Informatics*, 12(3):1074–1083, 2016.

L. I. Pinto, A. B. Leal, and R. S. U. Rosso. Safe dynamic reconfiguration through supervisory control in IEC 61499 compliant systems. In *Proceedings of the 2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 753–758, July 2017. doi: 10.1109/INDIN.2017.8104866.

L. Prenzel, A. Zoitl, and J. Provost. IEC 61499 runtime environments: A state of the art comparison. In *Proceedings of the International Conference on Computer Aided Systems Theory*, pages 453–460. Springer, 2019.

M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

M. Ramdani, L. Kahloul, M. Khalgui, and Y. Hafidi. R-TNCES Rebuilding: A New Method of CTL Model Update for Reconfigurable Systems. In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering ENASE*, pages 159–168, 2019.

M. Ramdani, L. Kahloul, M. Khalgui, Z. Li, and M. Zhou. Rctl: New temporal logic for improved formal verification of reconfigurable discrete-event systems. *IEEE Transactions on Automation Science and Engineering*, pages 1–14, 2020. doi: 10.1109/TASE.2020.3006435.

M. Rausch and H.-M. Hanisch. Net condition/event systems with multiple condition outputs. In *Proceedings of the 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation. ETFA'95*, volume 1, pages 592–600. IEEE, 1995.

M. Sadeghi. Developing IEC61499 in industrial processes, measurement and control systems (ipmcs). *WSEAS Transactions on Systems and Control*, 5, 04 2010.

M. Salem, O. Mosbahi, M. Khalgui, and G. Frey. Transformation from R-UML to R-TNCES: New formal solution for verification of flexible control systems. *2015 10th International Joint Conference on Software Technologies (ICSOFT)*, 2:1–12, 2015.

M. O. B. Salem, O. Mosbahi, and M. Khalgui. Pcp-based solution for resource sharing in reconfigurable timed net condition/event systems. In *Proceedings of the ADECS@ Petri Nets*, pages 52–67. Citeseer, 2014.

R. Sinha, K. Johnson, and R. Calinescu. A scalable approach for re-configuring evolving industrial control systems. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–8. IEEE, 2014.

R. Sinha, P. S. Roop, G. Shaw, Z. Salcic, and M. M. Kuo. Hierarchical and concurrent eccs for IEC 61499 function blocks. *IEEE Transactions on Industrial Informatics*, 12(1): 59–68, 2015.

M. Sorouri, V. Vyatkin, S. Xie, and Z. Salcic. Plug-and-play design and distributed logic control of medical devices using IEC 61499 function blocks. *International Journal of Biomechatronics and Biomedical Robotics*, 2(2-4):102–110, 2013.

M. Stanica and H. Guéguen. Using timed automata for the verification of IEC 61499 applications. *IFAC Proceedings Volumes*, 37(18):375–380, 2004.

P. H. Starke and S. Roch. *Analysing signal-net systems*, volume 162. Informatik-Bericht, Humboldt-Universitat zu Berlin, Germany, 2002.

T. Strasser and R. Froschauer. Autonomous application recovery in distributed intelligent automation and control systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1054–1070, 2012. doi: 10.1109/TSMCC.2012.2185928.

T. Strasser, A. Zoitl, and M. Rooker. Zero-downtime reconfiguration of distributed control logic in industrial automation and control. In *Reconfigurable embedded control systems: Applications for flexibility and agility*, pages 55–81. IGI Global, 2011.

T. Strasser, M. Rooker, G. Ebenhofer, and A. Zoitl. Standardized dynamic reconfiguration of control applications in industrial systems. *International Journal of Applied Industrial Engineering*, 2:57–73, 01 2014. doi: 10.4018/ijaie.2014010104.

C. Sünder, V. Vyatkin, and A. Zoitl. Formal verification of downtimeless system evolution in embedded automation controllers. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(1):1–17, 2013.

TC 65/SC 65B. *IEC 61499: Function blocks, Part 1 - Part 4*. International Electrotechnical Commission, Geneva, 2nd ed. edition, 2012a.

TC 65/SC 65B. *IEC 61499-1:2012 Function blocks - Part 1: Architecture*. International Electrotechnical Commission, Geneva, 2nd ed. edition, 2012b.

TC 65/SC 65B. *IEC 61499-2:2012 Function blocks - Part 2: Software tool requirements*. International Electrotechnical Commission, Geneva, 2nd ed. edition, 2012c.

TC 65/SC 65B. *IEC 61499-4: 2013 Function blocks - Part 4: Rules for compliance profiles*. International Electrotechnical Commission, Geneva, 2nd ed. edition, 2013.

T. Terzimehic, M. Wenger, A. Zoitl, A. Bayha, K. Becker, T. Müller, and H. Schauerte. Towards an industry 4.0 compliant control software architecture using IEC 61499 and opc ua. In *Proceedings of the 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE, 2017.

G. Theocharis, M. Kuhrmann, J. Münch, and P. Diebold. Is water-scrum-fall reality? on the use of agile and traditional development practices. In *Proceedings of the Product-Focused Software Process Improvement PROFES*, volume 9459, pages 149–166, 12 2015. doi: 10.1007/978-3-319-26844-6_11.

J. Thieme and H. . M. Hanisch. Model-based generation of modular PLC code using iec61131 function blocks. In *Proceedings of the Industrial Electronics, 2002 IEEE International Symposium on*, volume 1, pages 199–204 vol.1, 2002. doi: 10.1109/ISIE.2002.1026065.

K. Thramboulidis and G. Frey. Towards a model-driven IEC 61131-based development process in industrial automation. *Journal of Software Engineering and Applications*, 4 (04):217, 2011.

B. Ulitin, E. Babkin, T. Babkina, and A. Vizgunov. Automated formal verification of model transformations using the invariants mechanism. In M. Pańkowska and K. Sandkuhl, editors, *Proceedings of the Perspectives in Business Informatics Research*, pages 59–73, Cham, 2019. Springer International Publishing. ISBN 978-3-030-31143-8.

V. Vyatkin. Practice of modeling and verification of distributed controllers using signal net systems. 08 2000.

V. Vyatkin. Event-driven traversal of logic circuits for re-evaluation of boolean functions in reactive systems. In M. Broy and A. V. Zamulin, editors, *Proceedings of the Perspectives of System Informatics*, pages 319–328, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39866-0.

V. Vyatkin. Software engineering in industrial automation: State-of-the-art review. *Industrial Informatics, IEEE Transactions on*, 9:1234–1249, 08 2013. doi: 10.1109/TII.2013.2258165.

V. Vyatkin. *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design, third edition*. ISA International Society of Automation, 2016. ISBN 9781941546727.

V. Vyatkin and J. Chouinard. On comparisons of the isagraf implementation of IEC 61499 with fbdk and other implementations. In *Proceedings of the 2008 6th IEEE International Conference on Industrial Informatics*, pages 289–294. IEEE, 2008.

V. Vyatkin and H.-M. Hanisch. A modeling approach for verification of iec1499 function blocks using net condition/event systems. In *Proceedings of the 1999 7th IEEE International Conference on Emerging Technologies and Factory Automation. Proceedings ETFA'99 (Cat. No. 99TH8467)*, volume 1, pages 261–270. IEEE, 1999.

V. Vyatkin and H.-M. Hanisch. Modelling of IEC 61499 function blocks as a clue to their verification. In *Proceedings of the XI Workshop on Supervising and Diagnostics of Machining System*, pages 59–68, Poland, 03 2000a.

V. Vyatkin and H.-M. Hanisch. Modeling of IEC 61499 function blocks–a clue to their verification. In *Proceedings of the XI Workshop on supervising and diagnostics of Machining Systems*, volume 76, pages 59–68, 2000b.

V. Vyatkin and H.-M. Hanisch. Verification of distributed control systems in intelligent manufacturing. *Journal of Intelligent Manufacturing*, 14:123–136, 02 2003. doi: 10.1023/A:1022295414523.

V. Vyatkin and J. M. Lastra. Architectural foundations for reconfigurable manufacturing system. In *Proceedings of the 3rd International Symposium on Open Control Systems SoftSympo*, volume 3, 2003.

X. Wang and X. Xu. Icms: A cloud-based manufacturing system. *Cloud Manufacturing: Distributed Computing Technologies for Global and Sustainable Manufacturing*, pages 1–22, 03 2013. doi: 10.1007/978-1-4471-4935-4_1.

E. Wozniak, T. A. Cherfia, and C. Prehofer. Application of IEC 61499 to develop apps for open platforms. In *Proceedings of the 2018 IEEE International Conference on Industrial Technology (ICIT)*, pages 1574–1579, 2018.

J. Yan and V. V. Vyatkin. Distributed execution and cyber-physical design of baggage handling automation with IEC 61499. In *Proceedings of the 2011 9th IEEE International Conference on Industrial Informatics*, pages 573–578. IEEE, 2011.

L. H. Yoong, P. S. Roop, and Z. Salcic. Implementing constrained cyber-physical systems with IEC 61499. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(4): 1–22, 2013.

L. H. Yoong, P. S. Roop, Z. E. Bhatti, and M. M. Kuo. *Model-driven design using IEC 61499: a synchronous approach for embedded and automation systems*. Springer, 2014.

G. Zhabelova and V. Vyatkin. Multiagent smart grid automation architecture based on iec 61850/61499 intelligent logical nodes. *IEEE Transactions on Industrial Electronics*, 59(5):2351–2362, 2012. doi: 10.1109/TIE.2011.2167891.

G. Zhabelova and V. Vyatkin. Towards software metrics for evaluating quality of IEC 61499 automation software. In *Proceedings of the 2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, 2015.

G. Zhabelova, V. Vyatkin, and V. N. Dubinin. Toward industrially usable agent technology for smart grid automation. *IEEE Transactions on Industrial Electronics*, 62(4): 2629–2641, 2015.

J. Zhang. *Modeling and verification of reconfigurable discrete event control systems*. PhD thesis, 2015.

J. Zhang, M. Khalgui, Z. Li, O. Mosbahi, and A. M. Al-Ahmari. R-TNCES: A Novel Formalism for Reconfigurable Discrete Event Control Systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(4):757–772, July 2013. ISSN 2168-2232. doi: 10.1109/TSMCA.2012.2217321.

J. Zhang, M. Khalgui, Z. Li, G. Frey, O. Mosbahi, and H. B. Salah. Reconfigurable coordination of distributed discrete event control systems. *IEEE Transactions on Control Systems Technology*, 23(1):323–330, Jan 2015a. ISSN 2374-0159. doi: 10.1109/TCST.2014.2313352.

J. Zhang, M. Khalgui, Z. Li, G. Frey, O. Mosbahi, and H. B. Salah. Reconfigurable coordination of distributed discrete event control systems. *IEEE Transactions on Control Systems Technology*, 23(1):323–330, 2015b. doi: 10.1109/TCST.2014.2313352.

J. Zhang, H. Li, G. Frey, and Z. Li. Shortest legal firing sequence of net condition/event systems using integer linear programming. In *Proceedings of the 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1556–1561, 08 2018a. doi: 10.1109/COASE.2018.8560459.

J. Zhang, Z. Li, and G. Frey. Simulation and analysis of reconfigurable assembly systems based on R-TNCES. *Journal of the Chinese Institute of Engineers*, 41(6):494–502, 2018b.

A. Zoitl. *Real-Time Execution for IEC 61499*. ISA, 2009. ISBN 1934394270.

A. Zoitl and T. Strasser. *Distributed control applications: guidelines, design patterns, and application examples with the IEC 61499*. CRC Press, 2017.

A. Zoitl, C. Sünder, and I. Terzic. Dynamic reconfiguration of distributed control applications with reconfiguration services based on IEC 61499. volume 2006, pages 109 – 114, 07 2006. ISBN 0-7695-2589-X. doi: 10.1109/DIS.2006.28.

R. Zurawski. Achieving reconfigurability of automation systems by using the new international standard iec 61499. In *The Industrial Information Technology Handbook*, pages 1069–1088. CRC Press, 2004.