ZiZo: A Complete Tool Chain for the Modeling and Verification of Reconfigurable Function Blocks

Safa Guellouz, Adel Benzina
LISI Laboratory, INSAT and Tunisia Polytechnic School, University of Carthage, Tunis, Tunisia
Email: guellouz.safa@gmail.com, adel.benzina@isd.rnu.tn

Mohamed Khalgui
School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China
Email: khalgui.mohamed@gmail.com

Georg Frey
Chair of Automation and Energy Systems, Saarland University, Saarbrücken, Germany
Email: georg.frey@aut.uni-saarland.de

***Abstract*—Ubiquitous systems support reconfigurable hardware and software self-adapted components to external changes for a better performance. IEC 61499 is the most suitable manufacturing standard that designs distributed ubiquitous systems. All the available IEC 61499 development tools ensure the design, simulation and code generation of function block systems. There is no complete tool chain which supports design, modeling with Petri nets and automatic verification with model checking. This paper presents ZiZo v3 tool that supports the whole process for a new extension of IEC 61499 named Reconfigurable Function Block (RFB). ZiZo automates the transformation from RFB design diagrams to the Generalized Reconfigurable Timed Net Condition/ Event Systems GR-TNCES model that preserves its behavioural semantics and also exports it to the probabilistic symbolic model checker PRISM. A case study is presented to demonstrate the whole process from the design with RFB to verification.**

Keywords–IEC 61499; Reconfiguration; Reconfigurable distributed system; Automatic transformation; GR-TNCES.

I. INTRODUCTION

The flexibility and reconfigurability of manufacturing is one of the major drivers of IEC 61499 [1]. Several works address various aspects of hardware and software reconfiguration using this standard. These include works on down-timeless evolution [2] and real-time implementations [3]. Other works focus on agent-based reconfiguration [4], ontology-based reconfiguration agent [6] and even reconfiguration protocol [7]. We notice that these approaches of reconfiguration following IEC 61499 increase engineering efficiency and also the design complexity. In fact, the number of function blocks and the interconnections between them in the design system become very complex as well as their verification. In order to make the design easier, we propose an extension to the function block, named Reconfigurable Function Block (RFB) that encapsulates many scenarios of reconfiguration related to the changes in the controlled process. We aim to modify first of all the implementation of a function block by modifying the execution control chart model and the interface by adding a new event type to support reconfiguration, as well as the probabilistic aspect. Probabilistic events and scenarios are suggested to add a degree of uncertainty to events, thus it will be possible to evaluate after words the probability of occurrence of some unwanted states or scenarios like deadlocks.

On another hand, the verification and validation of automation software for reconfigurable distributed systems is an especially hard task. Many research works model manually the system with Petri nets [18] to verify it with a model checker. To handle the design and the modeling of systems with RFBs and automate their transformation to GR-TNCES [8], a class of Petri nets that preserves the behavioral semantics of RFB, we developed a complete tool ZiZo v3 as an extension to an existing tool ZiZo v2 [9] that models and simulates adaptive probabilistic discrete event control systems with GR-TNCES. Thanks to ZiZo, the verification of functional and temporal properties becomes easy: The designer can export the GR-TNCES model to PRISM model checker [10]. He/She can verify the functional correctness and safety of individual function blocks and entire control applications.

The remainder of this paper is structured as follows: We begin with the state of the art. In Section III, we present the reconfigurable function block as a new extension to IEC 61499. Then, we discuss the transformation of RFBs models to GR-TNCES model through a well-defined set of transformation rules in Section IV. We continue presenting the way that ZiZo v3 supports and automates the transformation process in Section V. This is followed by a presentation of the BROS system, which we have considered as a case study for our approach. Finally, we summarize this work.

II. STATE OF THE ART

A. IEC 61499 Modeling

Today, in industry, ubiquitous computing software must operate in conditions of radical change [5]. That is why several component-based technologies have been proposed to develop ubiquitous embedded control systems. Among all these technologies, the Industrial International Standard IEC 61499 is a component-based technology that defines Function Blocks (FBs) to model and implement distributed Industrial Process Measurement and Control Systems (IPMCSs).

A function block is an event triggered unit encapsulating some functionalities in algorithms. It contains data/event inputs and outputs to interact with the external environment. The activation of the block is ensured by events while data contain valued information. The algorithms encapsulated in the function block use data associated with incoming events to update internal and output data. The functionality of function block is defined by a state machine called execution control chart. It controls the algorithms execution and produces output events. Each state is assigned to actions (ECAction) that include

 ISBN: 978-1-61208-505-0

algorithms to be executed before sending output events. The states are connected to each other with ECtransitions that fire if the corresponding event occurs and the guard conditions are met. The conditions are based on internal variables or data inputs but not events.

Several tools have been developed in the past years to design IPMCSs following the standard: FBDK [11], IsaGRAF [12], nxtSTUDIO [13] and other IEC 61499 IDEs. Nevertheless, they do not offer verification support. The most important tool for that is VEDA [14] that mainly focuses on the modeling and verification of the function block execution control. In fact, the modeling and verification of reconfigurable manufacturing systems attract many researchers. Pang et al. [15] present a prototype model generator, which aims to automatically translate IEC 61499 function blocks into Net Condition/Event Systems following sequential execution semantics. Gerber et al. [16] present a formal model for integer-valued data types. This allows automatic model generation from arbitrary function block programs. Suender et al. [17] present a new formal validation of "on-the-fly" modification of control software in IEC 61499 automation systems. The main objective of modeling with Petri nets is to validate the system before its deployment. The above works use Petri nets to verify the correctness of the IEC61499 designs. But none of them has proposed a way for considering and modeling probabilistic reconfiguration scenarios within the function block paradigm and none has provided a complete tool that supports the design of function blocks, modeling with Petri nets and verification using model checking. In this work, a reconfigurable function block is formalised to support probabilistic reconfigurations in the current standard IEC 61499. Furthermore, a complete tool chain supports the whole process from the design of RFB to verification with model checking.

B. GR-TNCES

Petri nets [18] are widely used for the modeling of distributed control systems and support formal analysis such as model checking. GR-TNCES is a class of Petri nets considering reconfiguration and probability of occurrence of events. A GR-TNCES, as defined in [8], is an extension of the formalism Reconfigurable Timed Net Condition/Event Systems (R-TNCES) [19], which models unpredictable systems under memory and energy constraints and has a real-time probabilistic reconfigurable supervised control architecture. It is defined as a structure $G = \{\sum R - TNCES\}$. $R - TNCES = (B, R)$, where R is the control module consisting of a set of reconfiguration functions. B is the behavior module that is a union of multi TNCES, represented as follows:
B = (P, T, F, W, CN, EN, DC, V, $Z0$) where: (i) P (respectively, T) is a set of places (respectively, transitions), (ii) F is a set of flow arcs $F \subseteq (P \times T) \cup (T \times P)$, (iii) 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, where $x, y \in P \cup T$, (iv) CN (respectively, EN) is a set of condition (respectively, event) signals with $CN \subseteq (P \times T)$ (respectively, $EN \subseteq (T \times T)$), (v) DC: $F(P \times T) \rightarrow \{[l, h]\}$ is a super-set of time constraints on output arcs, (vi) V: $T \rightarrow \{\vee, \wedge\}$ maps an event-processing mode (AND or OR) to each transition, (vii) Z0= $(T0, D0)$ where $T0$: $P \rightarrow \{0, 1\}$ is the initial marking and $D0$: $P \rightarrow 0$ is the initial clock position.

A reconfiguration function $r \in R$ is a structure $r = (Cond, P, E, M, S, X)$ where: (a) $Cond \rightarrow \{true, false\}$: The precondition of r, (b) P: $F \rightarrow [0..1]$ is the TNCES probability, (c) E: $P \rightarrow [0..max]$: controls the energy requirements, (d) M: $P \rightarrow [0..max]$ controls the memory requirements, (e) S: $TN(\bullet r) \rightarrow TN(r\bullet)$ is the structure modification instruction for reconfiguration scenario, (f) X: $laststate(\bullet r) \rightarrow initialstate(r\bullet)$ is the state processing function.

We note, finally, that ZiZo v2 is the only tool that models GR-TNCES models. It is an extension to ZiZo v1 [20], developed in LISI Laboratory of INSAT Institute in collaboration with Saarland University in Germany. It edits, simulates and checks adaptive systems modeled with R-TNCES formalism. Furthermore, Zizo v2 allows designers to edit, simulate and export GR-TNCES models to the probabilistic model checker PRISM [10]. PRISM is a useful tool to verify functional and temporal properties of GR-TNCES. We apply it in our work to verify adaptive probabilistic discrete event control systems.

C. Originality

The design of reconfigurable systems following the IEC 61499 standard using the available tools is complex and difficult. To optimize the network of FBs and make it more adjustable to external changes, we propose a new function block named RFB that provides a simple model of reconfigurable systems. 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. Consequently, we extend the Petri nets editor ZiZo v2 to support: (i) design with RFB, (ii) automatic transformation from RFB to GR-TNCES model, and (iii) verification with model checking. ZiZo v3 is the only tool that supports the whole process from the design with RFB to verification.

III. RECONFIGURABLE FUNCTION BLOCKS

An RFB is a new extension of IEC 61499 that includes reconfiguration and probability aspects. It encapsulates several reconfiguration scenarios within a single FB and incorporates the probability of triggering each scenario. It is able to self-adapt the behavior of the system when changes occur in the environment. An RFB, as illustrated in Fig. 1, has new events and data of reconfiguration. It has also a specific architecture of the known execution control chart ECC: an $ECC_{controller}$ for the supervision of the elementary ECCs named ECC_{slaves} such that each one executes a scenario of reconfiguration. When an input event of reconfiguration occurs, $ECC_{controller}$ becomes active and reads the associated data of reconfiguration to select which scenario of reconfiguration will be active, and therefore, which ECC_{slave} will be executed. If the guard condition of the transition in $ECC_{controller}$ is met, then the corresponding ECC_{slave} waits for the occurrence of an input event to run the suitable algorithm, updates data and sends output events. At the end of all algorithms execution, $ECC_{controller}$ receives an event from the active ECC_{slave}. It updates the data of reconfiguration and generates output events of reconfiguration to communicate with the next RFBs.

The formalization of an RFB is defined as a tuple:
RFB=($Interface$, $ECC_{controller}$, ECC_{slave}), where:

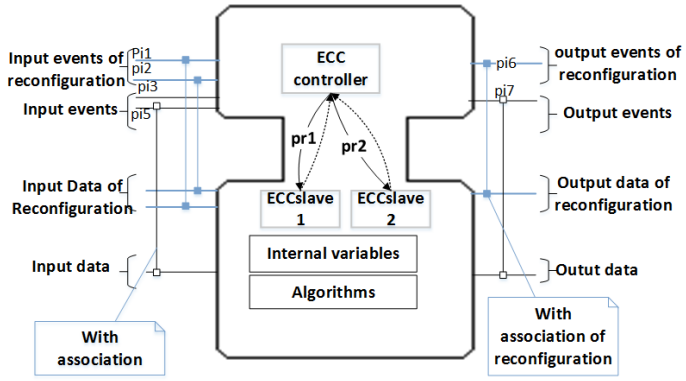
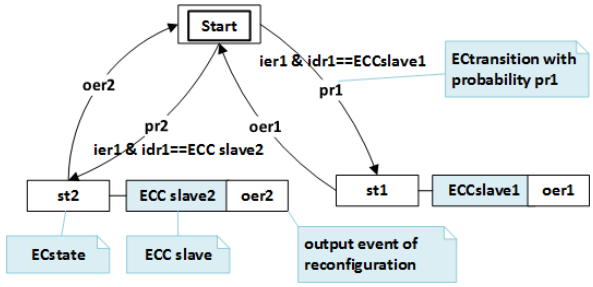


Figure 1. RFB interface.

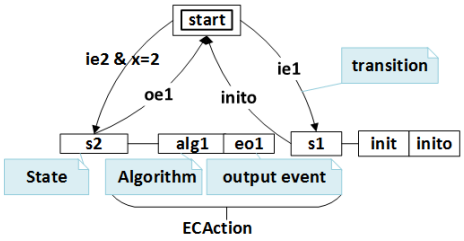
1) Interface: The interface of an RFB is determined by the tuple $Interface$=(IE, OE, ID, OD, IW, OW, IER, OER, IDR, ODR, IWR, OWR), where (i) IE (respectively OE) is a set of input (respectively output) events; (ii) ID (respectively OD) is a set of data inputs (respectively outputs); (iii) IW (respectively OW) is a set of WITH-associations for inputs (respectively outputs); (iv) IER (respectively OER) is a set of input (respectively output) events of reconfiguration, each $ier \in IER$ has a probability $p \in [0,1]$; (v) IDR (respectively ODR) is a set of input (respectively output) data which corresponds to all distinct ECC_{slaves} in this RFB (respectively in the next RFBs); (vii) $IWR \subseteq IER \times IDR$ is a set of WITH-associations for inputs of reconfiguration. Each input event of reconfiguration is associated with the corresponding $ECC_{slave} \in IDR$ that will be activated; (viii) $OWR \subseteq OER \times ODR$ is a set of WITH-associations for output events of reconfiguration. Each output event of reconfiguration is associated with the corresponding $ECC_{slave} \in nextRFB$ that will be activated. (ix) Each event in the interface has a probability of occurrence $pi_j \in [0,1]$ / $j \in \mathbb{N}$ and each ECC_{slave} has a probability of activation $pr \in [0,1]$.

2) $ECC_{controller}$: Is the main component in an RFB. It controls the activation of ECC_{slaves}, as illustrated in Fig. 2a. It is defined as a tuple: $ECC_{controller}$ = ($State_{controller}$, $Transition_{controller}$, $Condition_{controller}$, $Action_{controller}$), where: (i) $State_{controller}$ is a set of states where $s0 \in State_{controller}$ is the initial state; each state can have zero or more $Action_{Controller}$; (ii) $Transition_{Controller} \subseteq State_{controller} \times State_{controller}$ is a set of arcs representing transitions, a transition may have a probability $pr \in [0,1]$, which corresponds to the probability of activation of an ECC_{slave}. The sum of probabilities of transitions issued from the same state must be equal to one $\sum pr_i = 1$. The probabilities in the interface are the same as those on the $ECC_{controller}$ transitions; (iii) $Condition_{Controller}$ is a guard condition defined on an input event and data of reconfiguration; (iv) $Action_{controller} : State_{controller} \setminus \{s0\} \longrightarrow ECA$ where $ECA = ECC_{slave} \times OER$ is a set of actions. Each action should select one ECC_{slave} and one output event of reconfiguration.

3) ECC_{slave}: Let us consider n $\text{ECC}_{\text{slave}_i} \in RFB$ where $i \leq n$. Each $\text{ECC}_{\text{slave}_i}$ is controlled by $ECC_{controller}$, it encapsulates all algorithms to execute. According to the received data of reconfiguration, $ECC_{controller}$ selects the corresponding ECC_{slave}. As illustrated in Fig. 2b, an $\text{ECC}_{\text{slave}_i}$ =(S, Tr, C, A), where: (i) S is a set of states; (ii) $Tr \subseteq S \times S$ is a set of arcs representing transitions from one state to another; (iii) C is a set of guard conditions defined over input, internal and output variables of RFB; and (iv) A 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.



(a) $ECC_{controller}$.



(b) $ECC_{Slave}1$.

Figure 2. RFB components.

As a result, a reconfigurable distributed system is modeled as a network of interconnected RFBs linked by data and event flows. At the end of each scenario of reconfiguration encapsulated in an ECC_{slave}, each output data of reconfiguration contains the next ECC_{slave}, which will be active in the next RFB. When its value is updated, the associated output event of reconfiguration occurs to trigger the suitable RFBs.

IV. TRANSFORMATION RULES

The main objective of this transformation is to validate the system before its deployment. Each functional component of the RFB is modeled by a GR-TNCES [8] module. RFB = $\{G_{Interface}, G_{ECCController}, G_{ECCSlave}\}$ where $G_{Interface}$ =$\{$ G_{IE}, G_{OE}, G_{ID}, G_{OD}, G_{IER}, G_{OER}, G_{IDR}, G_{ODR}, G_{IW}, G_{OW}, G_{IWR}, $G_{OWR}\}$ where G_X models the X components of the RFB. $G_X = \sum \text{G}_{\text{X}_i}$. We have G_{X_i}=(B,R) where B=(P, T, F, W, CN, EN, DC, V, Z_0) and $R = \sum r_i$; r_i=($cond$, $P0$, $E0$, $M0$, S, X) a reconfiguration function.

All the input and output events have the same structure. The graphical models presented in Fig. 3 and Fig. 4 explain the transformation rules for all the components. As an example, let us detail the transformation rule for an input event:

Rule1: $ie \in IE \longrightarrow \text{G}_{\text{IE}_i}$

The objective of this rule is to transform each input event of RFB to a G_{IE_i} module. The graphical model is presented in Fig. 3a. The behavior of G_{IE_i} is composed of: P=$\{p1,p2\}$;T=$\{t1,t2\}$; F=$\{(p_1,t_1),(t_1,p_2),(p_2,t_2),(t_2,p_1)\}$; W=$\{\}$;CN=$\{\}$; EN=$\{ie1, ie2, oe1\}$; DC=$\{\}$; $V(t1)$=$\wedge$;

$V(t2)$=∧; Z_0={$T0$,$D0$} where $T0(p_1)$=1, $T0(p_2)$=0 and $D0(p_1)$=0;

Rule2: $IW \in Interface$ (respectively OW) is translated to G_{IW}:$G_{IE} \rightarrow G_{ID}$ (respectively G_{IW}:$G_{OE} \rightarrow G_{OD}$) where an output event $oe \in G_{IE}$ (respectively $eo \in G_{OE}$) is linked to an input event $ie \in G_{ID}$ (respectively $ie \in G_{OD}$.
$IWR \in Interface$ (respectively OWR) translated to G_{IWR}:$G_{IER} \rightarrow G_{IDR}$ (respectively G_{IWR}:$G_{OER} \rightarrow G_{ODR}$) where an output event $oe \in G_{IER}$ (respectively $oe \in G_{OER}$) is linked to an input event $ie \in G_{IDR}$ (respectively $ie \in G_{ODR}$).

The $ECC_{Controller}$ transformation rule is different from Rule1, it is modeled in Fig. 5 as follows:

Rule3: $ECC_{Controller} \longrightarrow G_{ECCcontroller}$
An $ECC_{Controller}$ is transformed to a $G_{ECCcontroller}$ module. $G_{ECCcontroller}$ is composed of: An initial marked place and a transition from which emerge n branches such that each one has a certain probability (probability of each reconfiguration scenario) where n is the number of ECC_{slave} in the current RFB. Each branch is composed of: two places and two transitions. The first place is linked to an input condition "ECC_{slave_i} true" ($i \in [1..n]$), which indicates that the ECC_{slave_i} must be activated. This place is linked to a transition T2 from which emerge n output events for activating the ECC_{slave_i} and deactivating the rest of the ECC_{slave}. This ensures that a single scenario of reconfiguration is active at a given time. The second place is linked to an input condition "ECC_{slave_i} finished", which marks the end of execution of the current active ECC_{slave_i}. Each branch finishes with a transition that sends 2 output events: the first one is for setting the output events of reconfiguration $oer \in OER$ and the second one for unsetting the input event of reconfiguration $ier \in IER$. Let us consider that there are n $ECC_{slave}s$ and m output events of reconfiguration in an RFB, the model of the $G_{ECCcontroller}$ has then n reconfiguration function r_i. B will be composed of: $P = \{p_1..p_{2n+1}\}$; $T = \{t_1..t_{n+2}\}$; F= {(p_1,t_1),(t_1,p_2),(t_1,p_3)..(t_1,p_{n+1}), (p_2,t_2),(p_3,t_3)..(p_n,t_n), (t_2,p_{n+2}),(t_3,p_{n+3})..(t_n,p_{2n})..(p_{2n+1},t_{n+2}),(t_{n+2},p_1)};
EN={oe_1..oe_{2n+1+m}}; CN= {ci_1..ci_{2n}} (for each slave an input condition ECC_{slave} true and an input condition ECC_{slave}finshed); W={}; DC={}; $V(t_i)$=∧; Z_0={$T0$,$D0$} where $T0(p_1)$=1, $T0(p_i)$=0 and $D0(p_1)$=0;

Rule4: $ECC_{slave_i} \in ECC_{Slave} \longrightarrow G_{ECC_{slave_i}}$
An ECC_{slave_i} is transformed to a $G_{ECC_{slave_i}}$ module. ECC_{Slave}, as we aforementioned, is a standard execution control chart so it contains states, transitions and actions.

The initial state is transformed to an initial marked place linked to a transition that is fired with the arrival of an input event for activating ECC_{slave_i}. As shown in Fig. 6, each state is transformed to two places "state run alg" and "state finish alg" as well as a transition between them. Each action is modeled with: an initial place "wait", an initial transition T14 that is fired when the input condition "start algorithm" is *true* and M places linked to M transitions for running $Algo_j$ (where M is the number of algorithms within the action) and $j \in [1..M]$). When all the algorithms in the different actions finish their execution, the ECC_{slave} generates an output condition indicating the end of the slave. Let us consider k states and k actions in $ECCslave_i$, j guard conditions on transition, each action contains an algorithm and an output event of reconfiguration then $B(G_{ECCslave_i})$ is composed of: P={ $p_1...p_{5k+2}$}; T={$t_1...t_{4k+4}$}; $F \subseteq (P \times T) U (T \times P)$; EN={ie_1, ie_2,oe_1,...,oe_k}; CN={ci_1, ci_2, ci_{j+k}} W={};DC={}; $V(t_i)$=∧; Z_0={$T0$,$D0$} where $T0(p_1)$=1, $T0(p_i)$=0 and $D0(p_1)$=0;

V. ZIZO V3

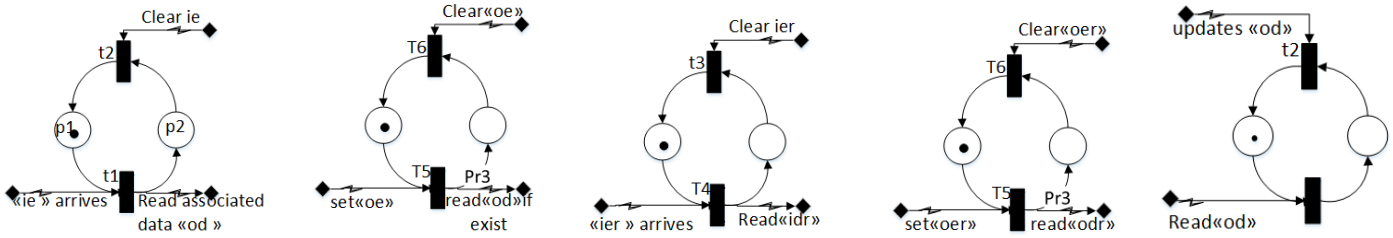
The third version of ZiZo allows to model any reconfigurable distributed system with RFB formalism by modeling the different reconfiguration scenarios and save it as ".nrfb" file. It supports new probabilistic events named "input and output event of reconfiguration" and new data types named "input and output data of reconfiguration" that present the ECC_{slaves}. As shown in Fig. 7, ZiZo v3 is more than a basic RFB editor, but it allows the system designer to transform automatically the RFB system into GR-TNCES that supports random reconfigurations with real-time constraints. Each component of RFB corresponds to a GR-TNCES module. This transformation is from ".nrfb" file to ".zz" file and it helps to verify functional and real-time properties after exporting it to PRISM model checker. During the exportation, a ".pm" file is generated to contain the places and the probability between them. This process is not sufficient to prove the correctness of the RFB system as it lacks the formal properties that is described in computation tree logic (CTL) [19] and Probabilistic Computation Tree Logic (PCTL) [21] language.

VI. CASE STUDY

The presented case study consists of a surgical robotized platform *BROS* [22] that is able to change its behavior in an unpredictable way during run-time processes. BROS [23] is a new automated intelligent platform developed for an optimal orthopaedic surgery to treat supracondylar elbow fractures in children. It is a complex system that contains the following interactive components: An intelligent robotic arm P-BROS to place the pins, two intelligent manipulator arms (B-BROS1 and B-BROS2) for the blocking and the reduction of the fracture respectively, a system browser (BW), a control unit (CU), which orchestrates the various components and finally a middleware between the CU and the BW. The surgeon triggers the system and selects the operating mode. Based on the fracture coordinates determined by BW, the CU calculates the new coordinates to reduce the fracture by B-BROS2 (move the patient arm to the calculated position). CU asks MW to check the correctness of the reduction. If the reduction is successful, then the CU orders B-BROS1 to block the patient's arm. Else, the system repeats the reduction. Under the request of CU and according to the identified fracture type, P-BROS performs the single or double pinning in the elbow. Once the pinning is successful, CU asks B-BROS1 to unblock the limb.

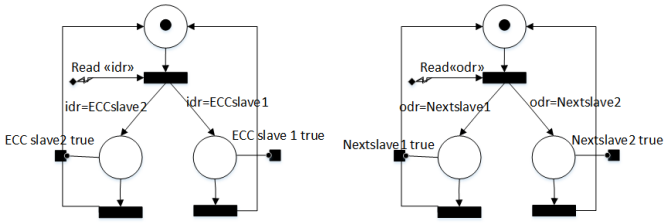
Using "ZiZo v3", we begin with modeling the three robotic arms in BROS with RFB formalism. The whole architecture, as shown in Fig. 8, is composed of four interconnected RFBs (A: RFBmode, B: BBROS1, C: BBROS2 and D: PBROS). RFBMode has five ECC_{slaves} each corresponds to an operating mode. BBROS1 has four ECC_{slaves}: (i) ECC_B offering the automatic blocking; (ii) $ECC_{ManualB}$ deactivating the blocking if $datarec$ is equal to manual; (iii) ECC_U activating the automatic unblocking; (iv) $ECC_{ManualU}$ deactivating the robotized unblocking.

B-BROS2 has two ECC_{slaves}: ECC_R to activate reduction and ECC_{notR} to deactivate the robotized reduction;



(a) GR-TNCES model of an input event "ie". (b) GR-TNCES model of an output event "oe". (c) GR-TNCES model of an input event of reconfiguration "ier". (d) GR-TNCES model of an output event of reconfiguration "oer". (e) GR-TNCES model of an output data "od".

Figure 3. GR-TNCES model of events.



(a) GR-TNCES model of an input data of reconfiguration "idr". (b) GR-TNCES model of an output data of reconfiguration "odr".

Figure 4. GR-TNCES model of reconfiguration data.

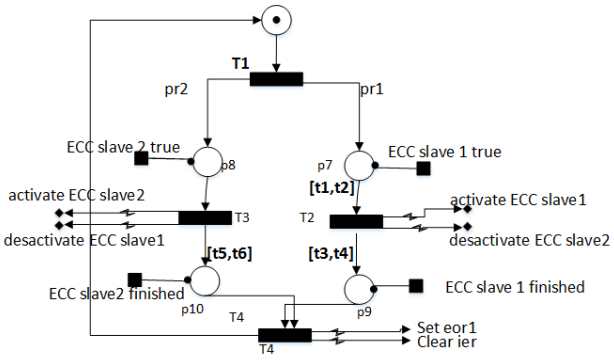


Figure 5. GR-TNCES model of an $ECC_{controller}$.

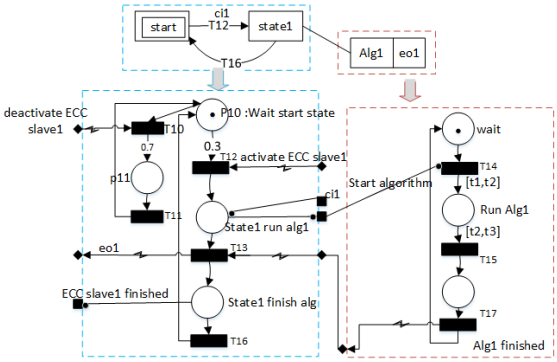


Figure 6. GR-TNCES model of ECC_{slave1}.

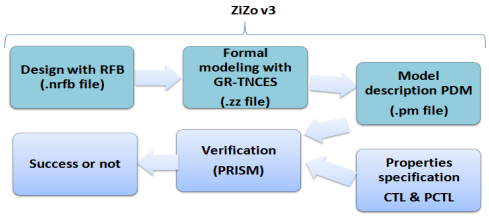


Figure 7. Flow diagram of the verification process using ZiZo and PRISM.

finally, PBROS not only depends on the operation mode but also on the fracture type that determines the pins number. It has $ECC_{ManualP}$, ECC_{1Pin} (respectively ECC_{2Pin}) encapsulating the single (respectively double) pinning algorithms that require the current P-BROS position, P-BROS orientation and the patient arm position. In the case of automatic reduction, blocking or unblocking the CU checks each functionality and returns *reductionOK*, *blockingOK* and *PinningOK* when it is successful. In the other case (manual reduction, manual blocking or manual unblocking), the system waits for these events from the surgeon to continue the operation. When exporting RFB model to GR-TNCES model, a file ".zz" is generated. The transformed model of BROS, according to the transformation rules in Section IV, is composed of four parts, as shown in Fig. 9, where: (i) A= $\sum GR-TNCES$ modules of *RFBmode*; (ii) B=$\sum GR-TNCES$ modules of *B-BROS1*; (iii) C=$\sum GR-TNCES$ modules of *B-BROS2*; (iv) and D= $\sum GR-TNCES$ modules of *P-BROS*.

Let us detail *RFBmode* presented in Fig. 10 that permits to activate one operating mode between five (AM, SAM, DMP, DMB, BM). The $ECC_{controller1}$ of "RFBmode" (Fig. 11a), reads input data of reconfiguration $idr1$. If the guard condition $ier1\&idr1 = AM$ is met (an event of reconfiguration $ier1$ arrives and $idr1$ equal to "AM") then it activates the ECC_{slave} "AM" detailed in Fig. 11b.

Fig. 12 presents the transformed GR-TNCES model of *RFBMode*, which is composed of GR-TNCES modules colored in yellow. Each module is composed of places, transitions, events and conditions that preserve the behavioral semantic of an RFB component. *ier1* module models the input event of reconfiguration $ier1 \in RFBMode$, which is associated to *idr1* that can be equal to one of the ECC_{slaves} (AM, SAM, DMP, DMB or BM). $idr6$ contains the fracture type, which

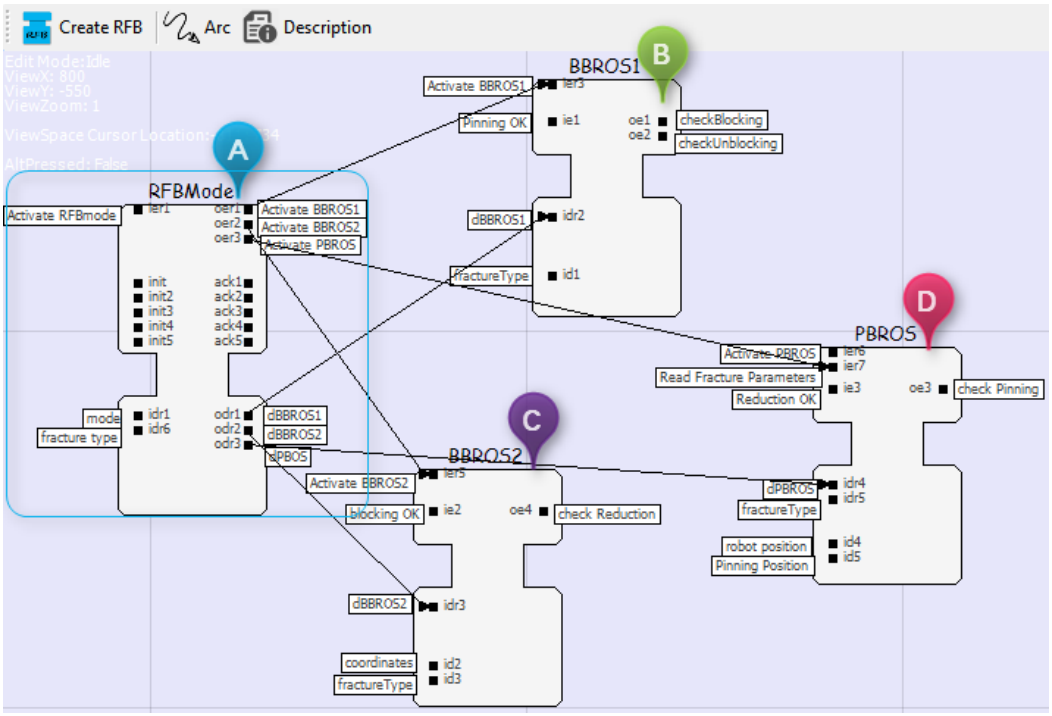


Figure 8. BROS model with RFB.

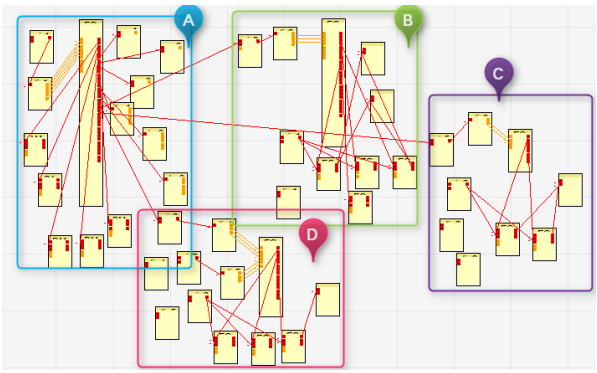


Figure 9. GR-TNCES model of the whole system.

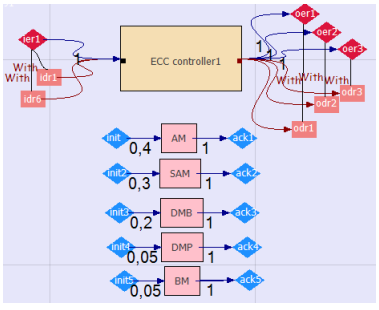
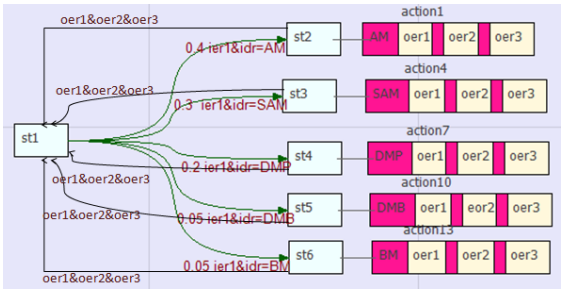
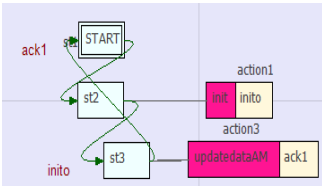


Figure 10. "RFBMode" internal components.



(a) $ECC_{controller}$ of "RFBMode"



(b) ECC_{slave}AM

(c) Update data AM algorithm

Figure 11. RFBMode details.

can be equal to IIB or IIC that involves single pin, IIA or III that involves double pins. If the pinning is robotized (the operation mode must be AM, SAM or DMP as detailed in [22]) then BROS must activate ECC_{1Pin} or ECC_{2Pin} according to the fracture type, else $ECC_{ManualP}$ will be executed. The module $idr1$ has five output conditions, only one is true, which corresponds to the selected ECC_{slave}. $ECC_{controller}$, shown in Fig. 13a, activates the scenario to execute and deactivates the others. At the end of the algorithm execution (init0 and UpdatedataAM, UpdatedataSAM, UpdatedataDMP, UpdatedataDMB or UpdatedataBM), the slave depicted in Fig. 13b returns an output condition to the $ECC_{controller}$ and an output event ack_i where $i \in \{1..5\}$. $ECC_{controller1}$ generates three output events of reconfiguration $oer1$, $oer2$

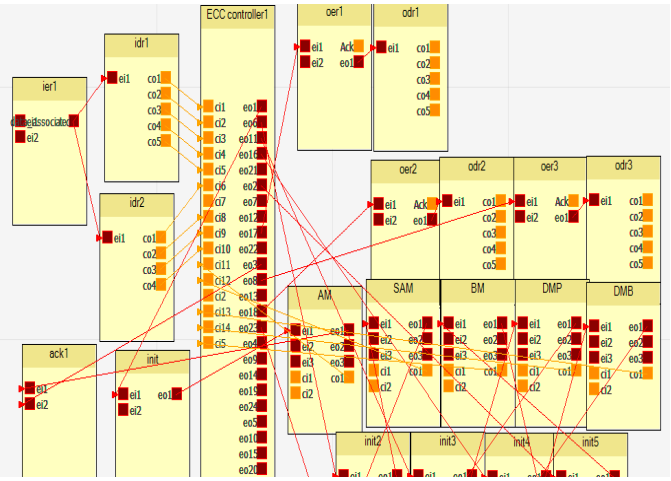
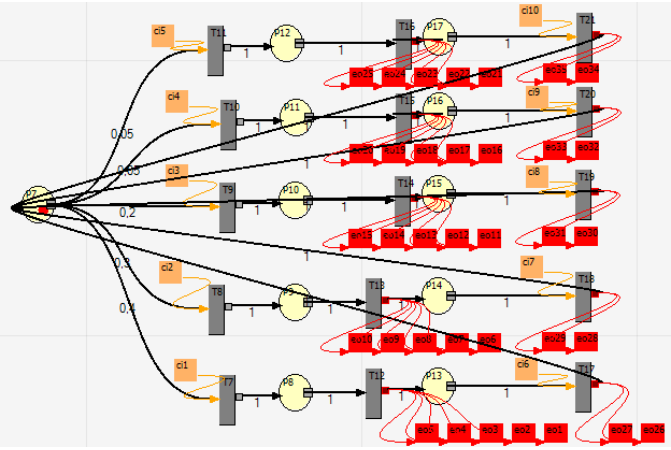
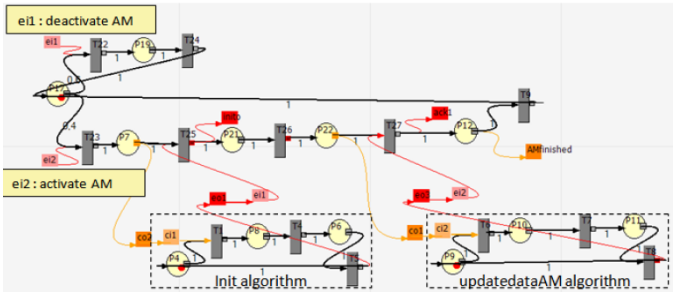


Figure 12. GR-TNCES model of RFBMode.



(a) $ECC_{controller}$.



(b) ECC_{slave} AM.

Figure 13. Some GR-TNCES modules of RFBMode.

and $oer3$ that activate respectively the controller of BBROS1, BBROS2 and PBROS. Each output event of reconfiguration is associated with an output data of reconfiguration: $odr1$, $odr2$ and $odr3$ that contain ECC_{slave} in the next RFB. Therefore, $odr1$ module activates ECC_B, ECC_U, $ECC_{ManualB}$ or $ECC_{ManualU}$ module of $BBROS1$ and vice versa.

After transforming the RFB model automatically to GR-TNCES, we have verified the functional correctness and safety of individual RFBs as well as the entire system using PRISM. It is used as a model checker that offers a probabilistic model checking for run-time verification of adaptive systems and analyses systems that exhibit random or probabilistic behaviors.

ZiZo facilitates the verification process by converting the GR-TNCES model to MDP model readable by PRISM. A ".pm" file is generated that contains model descriptions written in the PRISM language. CTL and PCTL are used to check the following safety properties: (i) The deadlock problem in each RFB and in the RFBs system, using the formula: "E[F"deadlock"]", which was proven to be false; (ii) BROS never executes pinning and unblocking simultaneously thanks to the following formula "P=?[F p=1& p=2]", where "p=1" is the unblocking and "p=2" is the pinning, which returns zero. (iii) Let us attach a probability 0 to the event "reductionOK" in *BBROS1*. What is the probability to change the mode of pinning in *PBROS* from automatic to manual? Using the formula "$P =?[Fp = 11\&(p = 12|p = 13|p = 14|p = 15)]$", where p=11 (AM), p=12 (SAM), p=13 (DMP), p=14 (DMB) and p=15 (BM), we proved that the probability of switching from automatic pinning to manual is 0.9.

VII. CONCLUSION AND FUTURE WORK

In order to enhance and simplify the system design and the verification process of reconfigurable systems, we have defined a new extension to IEC 61499 standard named RFB. We introduce in our RFB the probability associated to each reconfiguration scenario. We defined then a set of transformation rules for the RFB design to GR-TNCES model, implemented by ZiZo v3 that automates the transformation and the verification with PRISM. We have found our approach very helpful in the development process of reconfigurable distributed automation systems. In our future work, we will focus on the code generation of RFB to deploy it in manufacturing. We are also working on translating R-UML requirements to RFB design to promote reusability.

References

[1] V. Vyatkin, "IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review," Industrial Informatics, IEEE Transactions on, vol. 7, no. 4, pp. 768–781, 2011.

[2] M. N. Rooker et al., "Zero downtime reconfiguration of distributed automation systems: The epsiloncedac approach." in HoloMAS. Springer, pp. 326–337, 2007.

[3] A. Zoitl, G. Grabmair, F. Auinger, and C. Sunder, "Executing real-time constrained control applications modelled in IEC 61499 with respect to dynamic reconfiguration," in Industrial Informatics, 2005. INDIN'05. 2005 3rd IEEE International Conference on. IEEE, pp. 62–67, 2005.

[4] R. W. Brennan, M. Fletcher, and D. H. Norrie, "A holonic approach to reconfiguring real-time distributed control systems," in Multi-Agent systems and applications II. Springer, pp. 323–335, 2001.

[5] T. Kindberg, and A. Fox, System software for ubiquitous computing. IEEE pervasive computing, vol. 1, no. 1, pp.70-81, 2002.

[6] Y. Al-Safi and V. Vyatkin, "An ontology-based reconfiguration agent for intelligent mechatronic systems," in Holonic and Multi-Agent Systems for Manufacturing. Springer, pp. 114–126, 2007.

[7] M. Khalgui, O. Mosbahi, Z. Li, and H.-M. Hanisch, "Reconfiguration of distributed embedded-control systems," Mechatronics, IEEE/ASME Transactions on, vol. 16, no. 4, pp. 684–694, 2011.

[8] 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 ICSOFT-EA 2015 - Proceedings of the 10th International Conference on Software Engineering and Applications, Colmar, Alsace, France, 20-22 July, 2015., pp. 373–380, 2015.

[9] O. Khlifi, “ZiZo 2015,” http://www.aut.unisaarland.de/forschung/forschung-zizo-tool-khlifi/, [accessed: Jun,2016].

[10] M. Kwiatkowska, G. Norman, and D. Parker, “Prism: Probabilistic symbolic model checker,” in Computer performance evaluation: modelling techniques and tools. Springer, pp. 200–204, 2002.

[11] Holobloc Inc., FBDK 2.5 - The Function Block Development Kit, http://www.holobloc.com/fbdk2/, [accessed: September,2016].

[12] ICS Triplex ISaGRAF., ISaGRAF Workbench, http://www.isagraf.com, [accessed: September,2016].

[13] NxtControl., nxtSTUDIO - Engineering software for all tasks, http://www.nxtcontrol.com/en/engineering/, [accessed: September,2016].

[14] V. Vyatkin and H.-M. Hanisch, “Verification of distributed control systems in intelligent manufacturing,” Journal of Intelligent Manufacturing, vol. 14, no. 1, pp. 123–136, 2003.

[15] C. Pang and V. Vyatkin, “Automatic model generation of IEC 61499 function block using net condition/event systems,” Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on, pp. 1133–1138, 2008.

[16] C. Gerber, I. Ivanova-Vasileva, and H.-M. Hanisch, “Formal modelling of IEC 61499 function blocks with integer-valued data types,” Control and cybernetics, vol. 39, no. 1, pp. 197–231, 2010.

[17] C. Suender, V. Vyatkin, and A. Zoitl, “Formal validation of downtimeless system evolution in embedded automation controllers,” ACM Transactions on Embedded Control Systems, vol. 12, no. 17, pp. 17:1–17:17, 2013.

[18] J. Ezpeleta, J. M. Colom, and J. Martinez, “A petri net based deadlock prevention policy for flexible manufacturing systems,” IEEE transactions on robotics and automation, vol. 11, no. 2, pp. 173–184, 1995.

[19] J. Zhang, M. Khalgui, Z. Li, O. Mosbahi, and A. M. Al-Ahmari, “R-tnces: a novel formalism for reconfigurable discrete event control systems,” Systems, Man, and Cybernetics: Systems, IEEE Transactions on, vol. 43, no. 4, pp. 757–772, 2013.

[20] M. O. B. Salem, “ZiZo: a tool to model, simulate and verify reconfigurable real time control systems,” http://www.aut.unisaarland.de/forschung/forschung-zizo-tool-bensalem/, [accessed: Jun, 2016].

[21] V. Forejt, M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma, “Incremental runtime verification of probabilistic systems,” in Runtime verification. Springer, pp. 314–319, 2012.

[22] M. O. B. Salem, O. Mosbahi, M. Khalgui, and G. Frey, “Zizo: Modeling, simulation and verification of reconfigurable real-time control tasks sharing adaptive resources - application to the medical project BROS,” in HEALTHINF 2015 - Proceedings of the International Conference on Health Informatics, Lisbon, Portugal, 12-15 January, 2015, pp. 20–31, 2015.

[23] M. O. B. Salem, “BROS: a robotic platform for the treatment of the supracondylar humerus fracture,” http://www.aut.unisaarland.de/forschung/forschung-bros-platform/, [accessed: Jun, 2016].

 ISBN: 978-1-61208-505-0