Towards a Safe Development of Reconfigurable Robotic Systems

MOHAMED OUSSAMA BEN SALEM
Team Project IMAGES_ESPACE-Dev
UMR 228 Espace Dev IRD UA UM UG UR
University of Perpignan Via Domitia, Perpignan
FRANCE
bensalem.med.oussama@gmail.com

OLFA MOSBAHI
LISI Lab, INSAT
University of Carthage, Tunis
TUNISIA
olfamosbahi@gmail.com

Abstract: Developing reconfigurable robotic systems may be quite challenging. This contribution aims at proposing a new methodology to ensure the safety of such critical systems. It uses new tools and innovative concepts. To show the relevance of the said methodology, we apply the contribution to a real medical robotic system: BROS.

Key–Words: Design, verification, implementation, re-configurable systems, UML, R-TNCES, robotic systems

1 Introduction

Several approaches have been recently proposed in our community to develop robotic systems. Two of them are ORCCAD [1] and 4D/RDC [2] methodologies which consist in creating several hierarchical layers from each control architecture. Only a part of the control is ensured by each layer. This gives thus a decision-making system to each layer in the robot controller. However, the reactivity of the robot controller is affected because of such a decomposition. In fact, the time constraint of the execution of a given layer increases when the latter is low. The priority of its reaction becomes higher when the layer is up. Hybrid architectures have been proposed then to extend the hierarchical approach. These new architectures propose to combine it with a behavioral approach, improving thus the reactivity [3]. Other works propose to considerate the control architecture as a set of sub-systems controlling several specific parts of a given robotic system. This contribution is illustrated in IDEA (Intelligent Distributed Execution Architecture) agents architectures [4] and Chimera development methodology [5].

These works are relevant, however, they do not deal with systems with a self-reconfiguration function like BROS, a robotic system which treats humeral supracondylar fractures [6]. They do not feature functions to verify the system modeling nor to automatically generate codes. We expose in this work a new methodology to design, verify and implement robotic systems in ROS (Robot Operating System). The relevance of this methodology is proved by applying it on BROS. This methodology incorporates three main steps as shown in Figure 1: design, verification and implementation. The said figure explains the approach of our methodology: we start by designing the system using R-UML (Reconfigurable UML), a new UML profile proposed in [7] and which supports modeling and verification of reconfigurable control systems. R-UML permits to perform a behavioral and structural description of the system thanks to, respectively, R-StD (Reconfigurable State Diagram) and R-ClD (Reconfigurable Class Diagram). We perform then the verification of the system by extricating R-TNCES models from R-StD ones. R-TNCES are a Petri Nets extension proposed in [8]. The R-TNCEs models are then simulated by ZiZo, a software which models, simulate and verify reconfigurable control systems [9]. Mathematical properties are then applied on the R-TNCES-based models using SESA, a model-checker. Once the system is designed and verified, we move to the implementation step where the R-ClD models are used to generate ROS codes. This work also introduces BROS, the new robotic platform which ensures a safe treatment of the SCH (Supracondylar Humeral Fracture) for both surgeons and patients. The methodology is applied on the BROS system to serve two purposes: certifying the safety of BROS from a design perspective and proving the soundness and relevance of the said methodology. The latter is actually applicable on any other robotic system and permits to safely design it from the specification to the implementation steps.

In this paper, we follow this plan: the next section describes works dealing with formalism transformation and robotic development. Section 3 introduces several formalism and technologies which we use in this work. We expose, in Section 4, our case study: the robotic system BROS. Finally, we explain in details in Section 5 the different steps of the proposed methodology.

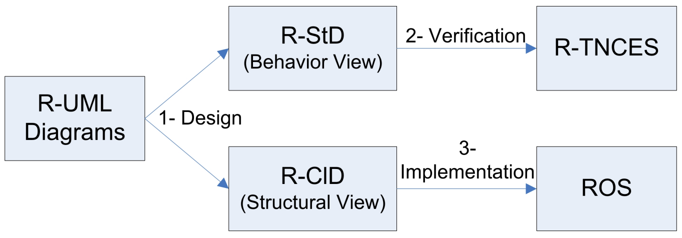


Figure 1: Approach of the Methodology.

2 State of the Art

We present, in this section, several works dealing with formalism transformation and robotic development.

2.1 Formalism Transformation

In our community, many contributions have been recently proposed to extricate performance models from UML models in an automatic way, such as queueing networks [10, 11], Petri nets [12, 13] and simulation [14]. For example, the authors in [15] propose a two-step to generate LGSPN (Labeled Generalized Stochastic Petri Nets) from UML models: First, we start by independently converting each UML statechart which describes the behavior of the software architecture into the corresponding LGSPN; secondly, we join the different obtained LGSPNs according to the data contained in the UML use case and sequence diagrams. However, this solution disregards the hardware limits and supposes that we dispose of infinite resources.

The authors in [16] propose another approach using LGSPN, which was lately extended in [13]. The proposition creates an intermediate model from a performance metamodel, named Core Scenario Model (CSM). The latter uses deployment diagrams to represent the software architecture structure. UML behavioral diagrams (sequence diagrams and/or activity diagrams) are used to describe the performance specifications related to the software architecture behavior. They are endowed with special tags and stereotypes. The CSM can be translated into different types of performance models, like Petri nets, queueing networks and simulative models.

The work in [17] uses an algorithm based on XML algebra to extricate XML format files and the corresponding layered queueing network from UML models. This work extends the results of [18] which propose to adopt the layered queueing networks as the aimed performance model. Two sequential steps are required to perform the UML transformation: First, an UML deployment diagram described the top-level representation of the software architecture is translated to a layered queueing network structure; secondly, an UML interaction or activity diagram is created using the parameters collected from the first step, combined with performance information. A similar and interesting approach is proposed in [11]. It uses an architectural approach based on MOF (Metaobject Facility) metamodeling [19]. Deployment and activity diagrams are, thus, generated.

The work in [14] use case diagrams, annotated with activity diagrams, to describe the software architecture behavior. A UML performance is proposed to transform UML diagrams into a discrete-event simulation model.

2.2 New Trends in Robotic Software Development

Many works in our community have recently proposed new concepts to improve maintainability, robustness, interoperability and reusability in robotics to face the latter's growing complexity. They proposed means of model-driven software development and component-based architectures. Thus, they gave rise to the creation of robotic frameworks and architectures such as Orocos [20], SmartSoft [21] and ROS [22]. Recent activities focus on composition in order to be able to configure at run-time both parameters and component's life-cycle and reuse them as black boxes.

Among the works promoting the component-based development, we cite the BRICS component model (BCM) [23] which uses the concept of composition with components which are grouped together. They form a new reusable component by using a life-cycle coordinator. rFSM (Restricted finite state machines) [24] were lately developed. They are state diagrams' minimal variant and are integrated into OROCOS/RTT, a robotic framework. This new concept focuses on component coordination for robotics. rFSM and BCM certainly make valuable contributions to achieve reuse and composition of components. Nevertheless, they cannot handle reconfiguration, an important feature in new robotic systems [25]. They control the component's life-cycle rather than its skills at task level.

Many tools and packages released with ROS have been rapidly developed during the last few years. For example, ROS developers can now use Matlab's Simulink [26] using the Robotics System Toolbox [27]. The latter allows them to model ROS workspaces, generate executable code, while also taking advantage of Matlab's large suite of simulation and analysis tools. Another infrastructure, OPRoS [28], couples existing component models and suites with ROS by using model transformations and integration tools. The developed OPRoS applications can thus communicate with ROS ones.

As far as we know, no one in our community proposed a complete methodology to design, verify and implement in ROS reconfigurable robotic systems. This is what we aim to do in this paper.

3 Background

We describe in this section several formalisms and technologies relevant to our work.

3.1 Timed Net Condition/Event System

A TNCES is defined as a tuple as follows:

$$TNCES = (P, T, F, m_0, \Psi, CN, EN, DC) \quad (1)$$

where:

- $P = \{p_1, p_2, ..., p_n\}$ is a finite set of places;
- $T = \{t_1, t_2, ..., t_m\}$ is a finite set of transitions;
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of flow arcs linking places and transitions;
- m_0 is initial marking;
- $CN \subseteq (P \times T)$ is a finite set of condition arcs;
- $EN \subseteq (T \times T)$ is a finite set of event arcs.

Ψ is an input/output structure of TNCES module which is represented by the following tuple:

$$\Psi = (C^{in}, E^{in}, C^{out}, E^{out}, Bc, Be, Cs, Dt) \quad (2)$$

where:

- C^{in} defines a finite set of TNCES module condition input signals;
- E^{in} defines a finite set of TCNES module event input signals;
- C^{out} defines a finite set of TNCES module condition output signals;
- E^{out} defines a finite set of TCNES module event output signals;
- $Bc \subseteq C^{in} \times T$ is a set of TNCES module input condition arcs;
- $Be \subseteq En \times T$ is a set of TNCES module input event arcs;
- $Cs \subseteq P \times C^{out}$ is TNCES module output condition arcs;
- $Dt \subseteq T \times E^{out}$ is a set of TNCES module output event arcs.

We assign time intervals to the pre-transition flow arcs $F \subseteq P \times T$, which impose temporal constrains to the firing of the transition:

$$DC = (DR, DL, D_0) \quad (3)$$

where:

- *DR* is the set of minimum elapsed times during which a token remains at particular place before the transition firing;
- *DL* represents the final set of limitation time defining the maximum time during which the place may keep a token (transition firing's other conditions have to be met before);
- D_0 denotes the initial set of the clocks which are associated with the places.

3.2 Reconfigurable Timed Net Condition/Event System

A Reconfigurable Timed Net Condition/Event System (R-TNCES) is an extension of the formalism TNCES with a specific function of self-reconfiguration [8]. It is defined as a structure *RTN =(B, R)*, where *R* denotes the control module composed of a set of reconfiguration functions *R* $=\{r_1,...,r_n\}$ and *B* is the behavior module which is a union of multi TNCESs. This is represented by a tuple:

$$B = (P, T, F, W, CN, EN, DC, V, Z) \quad (4)$$

where:

- *P* (respectively, *T*) is a superset of places (respectively, transitions);
- $F \subseteq (P \times T) \cup (T \times P)$ is a superset of flow arcs;
- 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 $x, y \in P \cup T$;
- $CN \subseteq (P \times T)$ (respectively, $EN \subseteq (T \times T)$) is a superset of condition signals (respectively, event signals);

- $DC : F \cap (P \times T) \rightarrow \{[l_1, h_1], ..., [l_{|F\cap(P\times T)|}, h_{|F\cap(P\times T)|}\}$ is a superset of time constraints on output arcs, where $i \in [1, |F \cap (P \times T)|], l_i, h_i \in N$, and $l_i < h_i$;
- $V : T \rightarrow \{\vee, \wedge\}$ maps an event-processing mode (AND or OR) for every transition;
- Z_0 = (M_0, D_0), where $M_0 : P \rightarrow \{0, 1\}$ is the initial marking and $D_0 : P \rightarrow \{0\}$ is the initial clock position.

3.3 UML Profiles

Many works in our community have already treated the idea of verification and validation of UML-based models in a more or less automatic and systematic way. Thus, several UML profiles are proposed for RT (real-time) and embedded systems. The authors in [29] propose a profile which helps to specify, design and verify embedded RT systems. It takes advantage of the advantages of UML, RT UML, co-design of functional-architecture and design based on platform. Nevertheless, this profile features implementation, communication and concurrency issues. The profile proposed in [30], UML-RT, is a complete language to model complex and event-driven RT systems. Nonetheless, it does not handle time constraints modeling [31] and has limited capabilities when it comes to modeling architecture and performance [32]. SPT (UML Profile for Schedulability Performance and Time) is another UML profile which permits to model and handle time, performance and scheduling of embedded RT systems [33]. However, it is less useful when it comes to express power, flexibility and reconfiguration [34]. SPT was replaced, then, with MARTE [35]. This new UML profile is helps tp model and analyze RT and embedded systems [36]. It permits to cover hardware and software resources and the aspects of time, extending thus UML 2.0. UML MARTE can not be used, however, to model dynamic or reconfigurable composition as it only provides static and predefined sets of reconfiguration [37].

3.4 Robot Operating System

Robot Operating System (ROS) is a software development framework which is open-source and dedicated to robots. It provides several services like low-level device control and hardware abstraction. Besides, it helps to implement commonly-used functionality and to handle message-passing between processes. It also manages packages. Robotic Operating System features tools to develop distributed robot applications. The latter are composed of nodes which are a set of processes networked using ROS communication infrastructure. Robot Operating System OS has become a standard in robotics which is well-know and extensively used by robot experts and developers. ROS is known for having a wide repository of software components such as sensor drivers, visualization tools, navigation systems, arm manipulation systems or artificial vision algorithms [38]. Besides, ROS is actively promoted and supported by the Open Source Robotics Foundation [39]. Given what has been said, we select ROS to be used in the methodology proposed in this paper.

4 Case Study: BROS

In this section, we introduce the architecture and reconfiguration modes of the EU-funded project, BROS (Browser and Reconfigurable Orthopedic Surgery). We expose, thereafter, the constraints which have to be followed while implementing this robotized platform.

4.1 Architecture of BROS

BROS is a robotic platform which is developed to treat to the supracondylar fracture of humerus. BROS can perform different tasks, such as fracture reduction , arm blocking and fixing the fragments of a fractured elbow bone. This last task is performed by pinning, that is BROS also offers a navigation function which helps to follow the pins as they progress into the treated elbow. As shown in Figure 2, BROS is composed of a control unit (UC), a browser (BW), a middleware (MW), a pinning robotic arm (P-BROS) and two blocking and reducing arms (B-BROS1 and B-BROS2).

The control unit is responsible of the functional safety and the smooth running of the surgery. It requests from MW the type of the humeral supracondylar fracture. The latter is necessary to help CU compute different coordinates defining the robotic arms' behaviors when reducing a fracture, blocking a fractured limb or performing pinning. CU also features a human-machine interface to help surgeon monitor the intervention progress.

BW is a Medtronics's product which is called FluoroNav. It combines software for image guidance and specialized surgical hardware. These features enable a surgeon to continuously track and update a surgical instrument's position. This virtual navigation presents many advantages over conventional fluoroscopic navigation. It simultaneously offers multiple fluoroscopic views simultaneously of the member to be treated. It also allows to remove the C-Arm (the medical imaging device which uses X-ray technology) from the operative field during navigation. This significantly reduces radiation exposure to the patient and medical staff.

MW, which is a software installed on BW, acts as a mediator between the control unit and the browser. This intelligent component features decision making and real-time monitoring. The middleware contains Several modules are encompassed in MW: a controller, an image processing module and a communication module with the control unit.

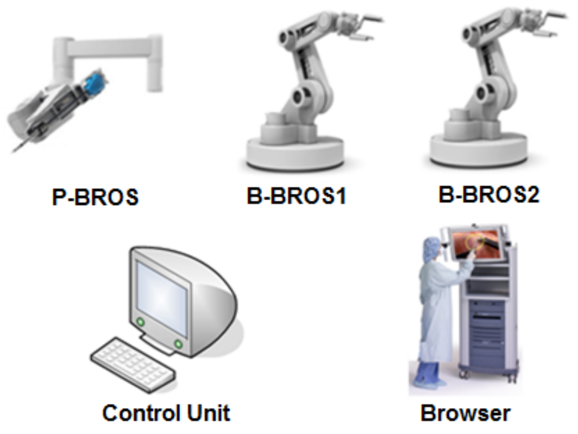


Figure 2: Architecture of BROS.

4.2 Reconfiguration Modes

One of the most important feature of BROS is reconfiguration. The robotic system should actually be able to run under different reconfiguration modes. For example, if BROS fails at performing a given task, such as reducing a fracture, blocking an arm and pinning, the surgeon is supposed to be able to decide to manually perform the said task. Thus, we decide to design five different operating modes. We detail them hereafter:

- Automatic Mode (AM): The surgeon oversees the smooth operation running whereas all the different tasks are performed by BROS;
- Semi-Automatic Mode (SAM): BROS performs the whole surgery, except of the fracture reduction which is done by the surgeon;
- Degraded Mode for Pinning (DMP): The surgeon performs the whole surgery, except of the pinning which is done by BROS;
- Degraded Mode for Blocking (DMB): The surgeon realizes the whole surgery, except of blocking the fractured arm. The latter is performed by BROS;
- Basic Mode (BM): The surgeon performs the whole surgery. Nevertheless, BROS assists him by providing real-time navigation using its middleware.

4.3 Real-Time Constraints

When AM is triggered, BROS follows the hereafter steps to treat a fracture:

A) The robotic system is launched when one of the five operating modes is triggered by the surgeon;

B) The fracture coordinates are requested by CU from MW;

C) An image of the fracture is requested by MW from BW;

D) Once the requested image is received, MW uses image processing techniques to compute the different coordinates of the fracture. This data is then sent to CU;

E) CU uses the received coordinates to give instructions to B-BROS1 which blocks the arm at the humerus;

F) The limb is blocked by B-BROS1;

G) B-BROS2 reduces the fracture at the request of CU;

H) The fracture is reduced by B-BROS2;

I) MW checks whether the reduction was successfully performed at the request of CU;

J) In order to assess the fracture reduction, a new image is requested by MW from BW. If reduction is successfully done, step *K* will be performed. Otherwise, steps from *G* to *I* are repeated;

K) B-BROS2 blocks the arm at the request of CU;

L) UC orders P-BROS to perform two pinnings;

M) At the request of CU and if pinning is successfully performed, B-BROS1 and B-BROS2 unblock the fractured arm.

5 Contribution: Development Methodology of Reconfigurable Robotic Systems

We describe in this section the development methodology steps and their application to BROS. It can be applied when designing any reconfigurable robotic system. BROS is used to prove the relevance of this methodology. The latter extends and combines many of our previous work. Figure 3 indicates that three steps are gone through in this methodology: design, verication and implementation. The rst step consists in dening the expectations from the system to design and express them in a modeling formalism, RUML in instance. During the verication step, the R-UML models are translated in R-TNCES ones [8]. The R-TNCEs models are then reduced and simulated by ZiZo, a software which models, simulates and verifies reconfigurable control systems [10]. Mathematical properties are then applied on the R-TNCES models using the model checker SESA. Once the system is designed and veried, we move to the implementation step where we generate ROS codes from the checked R-UML models.

5.1 Design Using R-UML

The design step is performed within two sub-steps as illustrated in Figure 3: constraints definition and modeling. The first sub-stem consists in defining the expectations from the system to design and the constraints set on it. The system is then modeled using R-UML formalism (detailed in the next subsection). Comes then the next step, the verification, which starts by transforming the generated models into R-TNCES ones (section 3.2). The latter are then simulated and verified thanks to formal verification. The contribution is applied on BROS.

In this section, we propose patterns and rules to set structural and behavioral models of a reconfigurable control system. We use R-UML as proposed in [40].

5.1.1 Structure Modeling

The class diagram provided by UML aims at showing a system's logical structure. It highlights the links between a system's components or modules by defining conceptual connections. The distinctive properties of each module or component are indicated by a class. UML's core semantics are extendable. Thus, we use stereotypes, a mechanism to categorize an element, in order to express new properties. On the basis of what has been said, we introduce eight stereotypes to define a class's attributes:

- $<< input >>$: a system input;
- $<< output >>$: a system output;
- $<< in >>$: a system module input;
- $<< out >>$: a system module output;
- $<< eventInput >>$: a system module event input;
- $<< eventOutput >>$: a system module event output;
- $<< integer >>$: an integer;
- $<< boolean >>$: a boolean (TRUE or FALSE).

The proposed stereotypes distinguish between *system* and *module*. The whole system under control is denoted by *System*, whereas *module* is a part of it. The stereotypes $<< in >>$ and $<< out >>$ specify the internal connections which may exist between a system's modules, whereas $<< input >>$ and $<< output >>$ specify the connections provided by a module to the controller. An event is an action which occurrence may be detected by another module in the system. It aims at interconnecting two system modules. An event should not be confused with an input/output, since the first is just a signal informing about the occurrence of an action. The $<< eventInput >>$ and $<< eventOutput >>$ stereotypes respectively represent the event inputs and outputs of a given module.

Thus, a reconfigurable class diagram (denoted by R-ClD) is represented by the following tuple:

$$R - ClD = (C, A, M, S, \alpha, \beta) \tag{5}$$

where:

- $C = \{cl_1, cl_2, ..., cl_n\}$ denotes a finite set of classes;
- $A = \{attr_1, attr_2, ..., attr_n\}$ denotes a finite set of attributes belonging to the classes;
- $M = \{setOutput, resetOutput, setlnput, resetInput, setCeiling\}$ denotes a set of methods belonging to classes;
- S represents a set of stereotypes / $S = \{<< input >>, << output >>, << in >>, << out >>, << eventInput >>, << eventOutput >>, << boolean >>, << integer >>\}$;
- $\alpha : st_i \to attr_j$ is a function mapping a stereotype st_i from *S* to an attribute $attr_j$ from A;
- $\beta : attr_i \to cl_j$ is a function mapping an attribute $attr_i$ to a class cl_j.

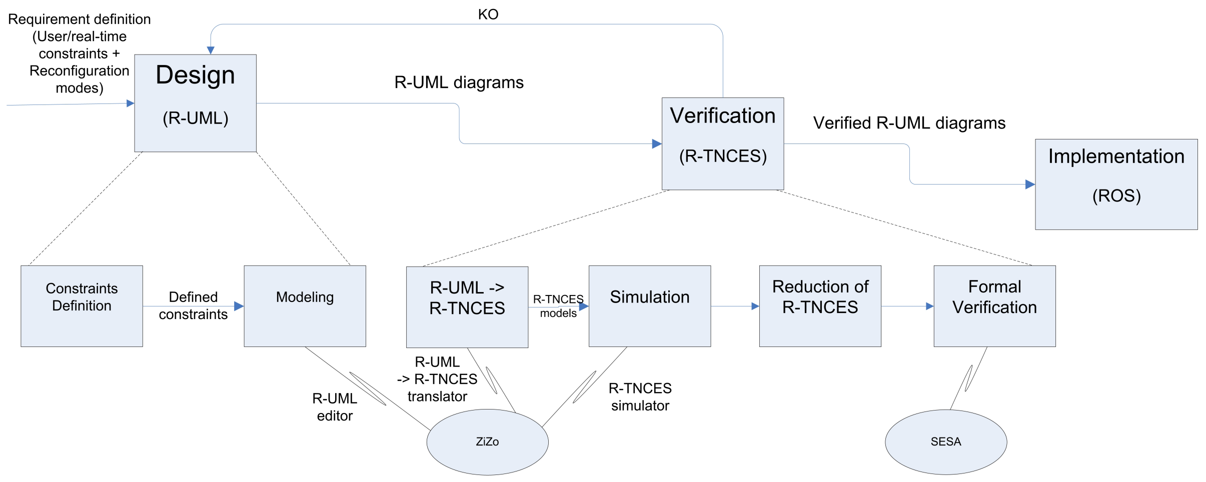


Figure 3: Methodology steps.

5.1.2 Behavior Modeling

The UML state diagram models the behavior of the different objects. An object is the implementation of a particular class. Thus, we define a set of states which may be taken by a system or its components. We distinguish each state by its name. Transitions represent the change from a state to another. They also define the law causing that change and the consequences of it. Transitions may be fired by rules which may be expressed by events or guards which are boolean expressions firing a transition when evaluated to TRUE. Three manners exist to fire a given transition: a guard (when certain properties are assigned with particular values), an event (when a certain action takes place somewhere in the system) or combination of both. Transitions serve to interconnect different states and determine the cause and the consequences of a transition to fire. We use time events (*after (n)* where *n* is a positive integer) to specify the number of time units which should elapse before the transition firing. We can also specify events by using attributes stereotyped by $<< eventInput >>$ or $<< eventOutput >>$. Actions may be activated when a transition is fired. They serve to set the system's some properties by calling methods (which are defined in the classes) and modify attributes' values, such as *setOutput*, *resetOutput*, *setlnput*, *resetInput* and *setCeiling*.

A state diagram is formally represented by the following tuple:

$$StD = (St, Tr, Ev, G, Ac, \gamma, \delta, \epsilon, \zeta) \qquad (6)$$

where:

- $St = \{st_1, st_2, ..., st_n\}$ is a finite set of states in an *StD*;
- $Tr = \{tr_1, tr_2, ..., tr_m\}$ is a finite state of transitions in an *StD*;
- *Ev* is a finite set of events in transitions of *StD*;
- *G* is a finite set of guards in *StD*;
- *Ac* is a final set of actions;
- $\gamma : ev_i \rightarrow tr_j$ is a function mapping an event ev_i of *Ev* to a transition tr_j of *Tr*;
- $\delta : gr_k \rightarrow tr_j$ is a function mapping a guard gr_k of *Gr* to a transition tr_j of *Tr*;
- $\epsilon : act_l \rightarrow tr_j$ is a function mapping an action act_l of *Ac* to a transition tr_j of *Tr*;
- $\zeta : trj \rightarrow \{st_b, ste\}$ is a function mapping a transition tr_j of *Tr* to a pair of states st_b and st_e, where st_b is the state from which the transition is taken and st_e is the next state if tr_j fires.

In order to model the reconfiguration feature expected from the system, we define a reconfigurable state diagram (denoted R-StD) as a structure:

$$R - StD = (B, R) \qquad (7)$$

where:

- *B* is the union of multi StD and represents the behavior module;
- *R* is a set of reconfiguration functions *R*=$\{r_1,...,r_n\}$ and represents the control module.

Once a reconfiguration scenario is triggered, a reconfiguration function r_i applies to the system the required changes. The function *r* is defined as follows:

$$r = (\eta, \theta, \kappa) \tag{8}$$

where:

- $\eta : t_i \rightarrow \{0,1\}$ is a function controlling tasks, $\eta(t_i) = 1$ if the task t_i is added to the system, $\eta(t_i) = 0$ otherwise;
- $\theta : res_j \rightarrow \{0,1\}$ is a function controlling resources, $\theta(res_j) = 1$ if the resource res_i is added to the system and $\theta(res_j) = 0$ otherwise;
- $\kappa : (res_j, t_i) \rightarrow \{0,1\}$ is a function where $\kappa(res_j, t_i) = 1$ if res_j is used by t_i in this triggered reconfiguration scenario, $\kappa(res_j, t_i) = 0$ otherwise.

5.1.3 Application to BROS

In order to automate and evaluate the proposed transformation from R-UML to R-TNCES, we decide to develop two new modules in an R-TNCES editor, simulator and model-checker named ZiZo [41]. Thus, the latter will be able, first, to edit R-UML models and, secondly, translate them into R-TNCES ones during the verification step. The tool's different modules as indicated in Figure 4. The user can, lately, simulate the generated models and apply CTL formulas on them to check different properties.

In this step, we edit an R-StD diagram describing the behavior of BROS. The obtained diagram is illustrated in Figure 5.

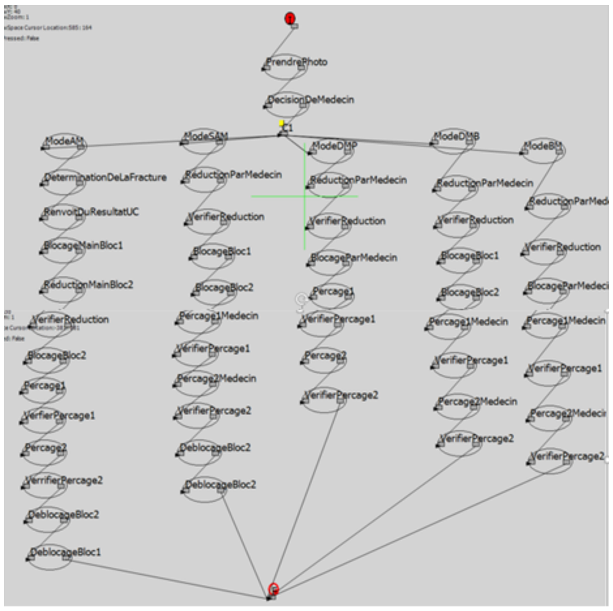


Figure 5: R-StD of BROS.

5.2 Verification

In order to perform a formal verification of our system, we start by transforming the R-UML diagrams generated during the design step into R-TNCES models. The latter are then reduced to optimize and simplify the formal verification. These contributions are then applied on BROS case study.

5.2.1 Transformation Rules from R-UML to R-TNCES

Having basic elements of R-UML used for system modeling defined by the equations 5 and 7, R-UML project defining a reconfigurable system with adaptive shared resources model can be represented via 4-tuple as follows:

$$R - UML_{system} = (R - ClDs, R - StDs, O, \Omega) \tag{9}$$

where:

- $R - ClDs = \{R - ClD_1, R - CID_2, ..., R - CID_n\}$ is a finite set of class diagrams, where each ClD_i element is defined by (5);
- $R - StDs = \{R - StD_1, R - StD_2, ..., R - StD_l\}$ is a finite set of reconfigurable state diagrams, where each $R - StD_j$ element is defined by (7);
- O is a finite set of objects, where each one is an instance of ClD_i and has its corresponding $R - StD_j$;
- $\Omega : R - StD_a \rightarrow cl_b$ is a function that maps the reconfigurable state diagram $R - StD_a$ to the class cl_b of C (5).

Table 1 represents the correspondence between R-StD and R-TNCES. The numbers in parentheses denote the references to formulas giving details about the used syntax. Hereafter are the detailed seven translation rules:

- **Rule 1:** Each state *St* from an R-StD is translated into a place *P* in an R-TNCES;
- **Rule 2:** Each transition *Tr* in an R-StD is translated to a transition (*T*) in an R-TNCES;
- **Rule 3:** A transition *tr* from an R-StD is mapped to the pair of states, st_b and st_e. st_b is the state from which *tr* is taken and st_e is the next state firing *tr*. The two places (p_{out}, p_{to}) and the corresponding transition (*t*) are created using, respectively, Rule 1 and Rule 2. Rule 3 creates 2 flow

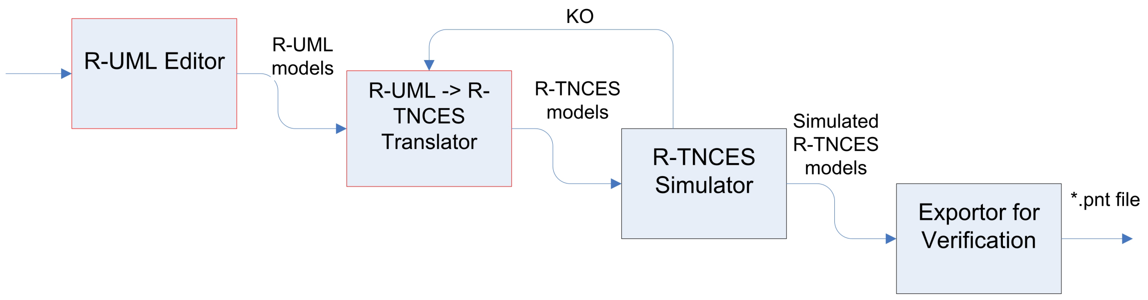


Figure 4: ZiZo's different modules.

arcs in the R-TNCES: a flow arc, fa_1, linking p_{out} to *t*, and a second, fa_2, linking *t* to p_{to};

- **Rule 4:** In an R-StD, guards are translated into some transitions. A guard *gr* is translated into a condition arc *ca* in an R-TNCES. A condition output signal *co* is added to the place from which *ca* is leaving and a condition input signal *ci* is added to the place which is pointed by *ca*;

- **Rule 5:** In an R-StD, actions are translated into some transitions. An action *ac* is translated into an event arc, *ea*, in an R-TNCES. An event output signal *eo* is added to the place from which *ea* is leaving and a event input signal *ei* is added to the place which is pointed by *ea*;

- **Rule 6:** Each event of the stereotype <<eventInput>> from an R-StD (denoted by *ev*) is translated into an event arc in the R-TNCES (denoted by *ea*). We add an event output signal *eo* and an event input signal *ei* to, respectively, the place from which *ea* is leaving and the place which is pointed by *ea*;

- **Rule 7:** We may find after(n)-typed events (where $n \in \mathbb{N}^*$) in an R-StD. If applicable, we add *n* to the set of minimum times during which a token should remain at a particular place before the transition fires (denoted by *DR*). We also add ∞ to the set of limitation time defining maximum time during which a place may keep a token (denoted by *DL*). This is due to the fact that the place from which the after(n)-typed event is leaving may indefinitely hold the token.

Algorithm 1 is a general algorithm for translating R-UML to R-TNCES. The numbers given in parentheses show the reference to the formula that gives details about the used syntax.

5.2.2 Simulation

Simulation is realized using the ZiZo software. The latter is an R-TNCES's editor, simulator and model checker which was introduced in [42]. Simulation can be tracked by selection of a token game. Once it is finished, a report is displayed at the debug window. This step aims at proving that the system is deadlock-free. If a deadlock occurs, we go back to the specification step to treat the deadlock problem.

5.2.3 Reduction of R-TNCES

In this section, we propose a reduction algorithm to solve the redundancy problem and validate the different R-TNCES models.

Petri net reduction is a technique which that translates Petri nets into simplified and reduced nets. This procedure naturally preserves some desirable properties of the original nets. The reachable state space is reduced thanks to Petri nets reduction. We can consequently get sufficient information from the simplified nets to understand the original ones. Nevertheless, the reduced models successfully permit to verify and validate the modeled system [43]. Petri reduction alleviates, then, the state-space explosion problem [44].

Figure 6 presents a set of reduction rules for R-TNCES. As we can observe, the said rules preserve properties such as bound of places and liveness. We hereafter explain the six proposed rules:

- **Rule 1:** This rule represents a *macroplace rule*'s particular case [45].

- **Rule 2:** It is a *transition fusion rule*'s particular case [46].

- **Rules 3 and 5:** These rules are *implicit place rule*'s particular cases [47]. They preserve properties such as liveness, the bound of places, and reversibility.

- **Rules 4 and 6:** They are, respectively, particular cases of *identical* and *identity transition rules* [46].

Table 1: Transoformation Rules from R-StD to R-TNCES.

Rules	R-StD	R-TNCES
Rule 1	St (6)	P (3.2)
Rule 2	Tr (6)	T (3.2)
Rule 3	$\{st_b, st_e\} := \zeta(tr)$ (6)	$\{p_{out}, p_{to}\} \subseteq P$; $\{fa_1, fa_2\} \subseteq F$ (3.2)
Rule 4	$gr := \delta^{-1}(tr)$ (6)	$ci \in C^{in}$ (2) ; $co \in C^{out}$ (2) ; $ca \in CN$ (1)
Rule 5	$ac := \epsilon^{-1}(tr)$ (6)	$ei \in E^{in}$ (2) ; $eo \in E^{out}$ (2) ; $ea \in EN$ (1)
Rule 6	$ev := \zeta^{-1}(tr)$ (6) AND $<< eventInput >>:= \alpha^{-1}(ev)$ (5)	$ei \in E^{in}$ (2) ; $eo \in E^{out}$ (2) ; $ea \in EN$ (1)
Rule 7	$ev := \zeta^{-1}(tr)$ (6) AND ev is an after(n) event	$n \in DR$; $\infty \in DL$ (1) (2) (3)

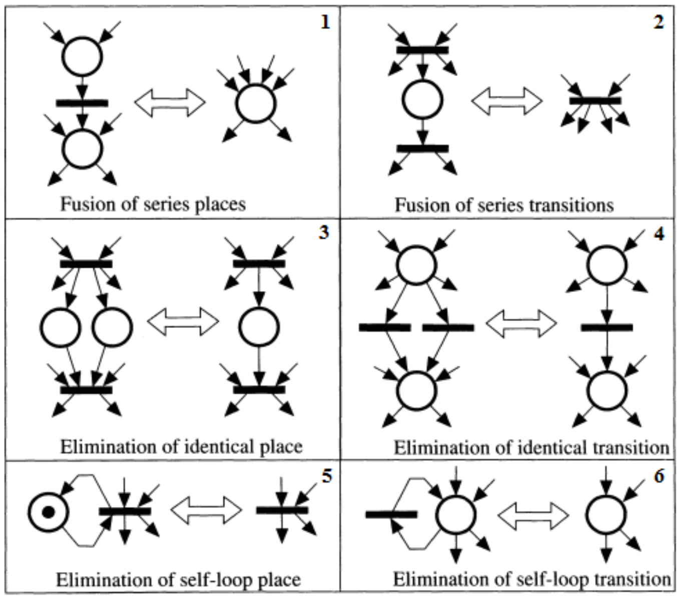
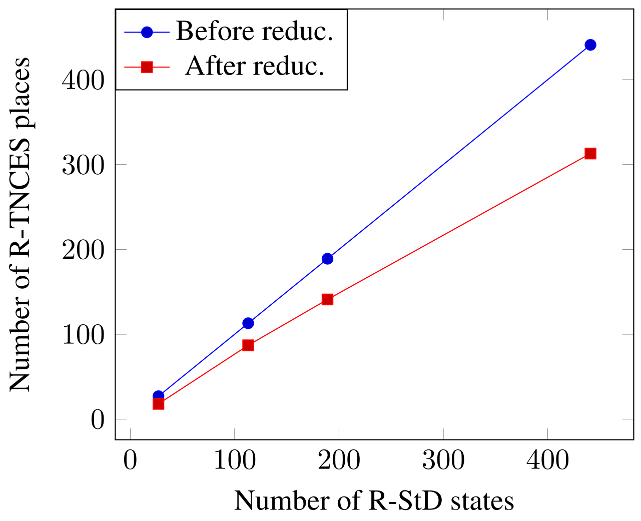
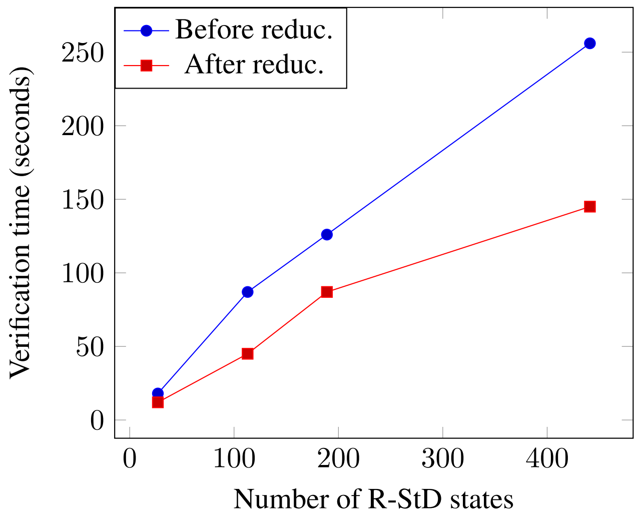

1
Fusion of series places
2
Fusion of series transitions
3
Elimination of identical place
4
Elimination of identical transition
5
Elimination of self-loop place
6
Elimination of self-loop transition

Figure 6: Reduction rules of R-TNCES.

Plot 1: Number of Generated R-TNCES Places



Plot 2: Time Consumed by Verification



To check the relevance of the said reduction rules, we propose to compare, first, the number of generated R-TNCES places from different R-StD diagrams, before and after the reduction. The results shown in Plot 1 prove the relevance of these rules. Secondly, we compare the time consumed by the formal verification of the CTL formula *AG EX TRUE*, again before and after the reduction. The said formula checks the deadlock freeness of a given R-TNCES. The results shown in Plot 2 proves that verification time of an R-TNCES sensitively reduces after its reduction.

5.2.4 Formal Verification

Once the simulation is finished without detecting any deadlock, we start the formal verification. The latter consists in defining CTL, eCTL and TCTL formulas, based on the specification step. These properties are formally verified using the model checker SESA [48]

Algorithm 1 R-UML translation into R-TNCES

```
Input: $R-UML_{system}$
Output: $R-TNCES_{system}$
for $obj_j \in O$ (9); $j \in [0, |O|]$ do
    Initialize $R-TNCES_k$, each element is $\emptyset$ (4)
    Define class of the object $Cl := \Omega(obj_j)$ (9)
    for $attr_l \in Cl(l \in [0, |A_{Cl}|]$ do
        if $\alpha^{-1}(attr_l) = << input >>$ (5) then
            Add condition input $ci$ to $C^{in}$ of $R-TNCES_k$ (2)
            Add condition arc $ca$ to $CN$ of $R-TNCES_k$ (1)
        end if
        if $\alpha^{-1}(attr_l) = << output >>$ (5) then
            Add condition output $co$ to $C^{out}$ of $R-TNCES_k$ (2)
            Add condition arc $ca$ to $CN$ of $R-TNCES_k$ (1)
        end if
        if $\alpha^{-1}(attr_l) = << in >>$ (5) then
            Add condition input $ci$ to $C^{in}$ of $R-TNCES_k$ (2)
        end if
        if $\alpha^{-1}(attr_l) = << out >>$ (5) then
            Add condition input $co$ to $C^{out}$ of $R-TNCES_k$ (2)
        end if
        if $\alpha^{-1}(attr_l) = << eventInput >>$ (5) then
            Add event input $ei$ to $E^{in}$ of $R-TNCES_k$ (2)
        end if
        if $\alpha^{-1}(attr_l) = << eventOutput >>$ (5) then
            Add event input $eo$ to $E^{out}$ of $R-TNCES_k$ (2)
        end if
        if $\alpha^{-1}(attr_l) = << integer >>$ and $Attr_l == x$ (5) then
            Add $x$ to $DR$ of $DC$(3)
        end if
        if $\alpha^{-1}(attr_l) = << boolean >>$ and $Attr_l == y$ (5) then
            Add $y$ to $G$ of $StD$ (6)
        end if
    end for
    Define a state diagram for each object $obj_j$ : $R-StD := \Omega^{-1}(Cl)$ (9)
    for $tr \in Tr$ of $StD$ (6) do
        Add transition $t$ to $T$ of $R-TNCES_k$ (4)
        Get outgoing $st_{out}$ and incoming $st_{to}$ states for transition: $st_{out}, st_{to} := \zeta(tr)$
        for $st \in \zeta(tr)$ do
            place $p_{out}$ and $p_{to}$ to $P$ of $R-TNCES_k$ (4)
            place flow arc $fa_1$ to $F$ of $R-TNCES_k$: $(p_{out}, t)$
            place flow arc $fa_2$ to $F$ of $R-TNCES_k$: $(t, p_{to})$
        end for
        Define guard for transition $tr$:$gr := \delta^{-1}(tr)$ (6)
        Define action for transition $tr$:$ac := \epsilon^{-1}(tr)$ (6)
        Define event for transition $tr$:$ev := \gamma^{-1}(tr)$ (6)
        for $< operand > (P_{guard}) \in gr$ do
            Add condition input $ci$ to $C^{in}$ of $R-TNCES_k$ (4)
            Add condition arc $ca$ to $CN$ of $R-TNCES_k$ (1)
        end for
        for $< action > (P_{action}) \in ac$ do
            Add event input $ei$ to $E^{in}$ of $R-TNCES_k$ (4)
            Add event arc $ea$ to $EN$ of $R-TNCES_k$ (1)
        end for
        if $ev$ is of $<< eventInput >>$ stereotype then
            Add event input $ei$ to $E^{in}$ of $R-TNCES_k$ (4)
            Add event arc $ea$ to $EN$ of $R-TNCES_k$ (1)
        end if
        if $ev$ is $after(n)$ event then
            Add $n$ to $DR$ of $DC$ for $fa_1$ (3)
            Add $\infty$ to $DL$ of $DC$ for $fa_1$ (3)
        end if
    end for
end for
```

which takes as input a .pnt file exported from ZiZo. If all the checked formulas meet the users' expectations, we obtain verified models as output. Otherwise, we go back to the specification step.

5.2.5 Application to BROS

This step consists in translating the R-UML model into an R-TNCES one. The latter is shown in Figure 7. Upon definition of the model, we simulate it using ZiZo. The obtained report displayed in Figure 8 proves that, after exploring 3057 places by ZiZo, our system is deadlock-free.

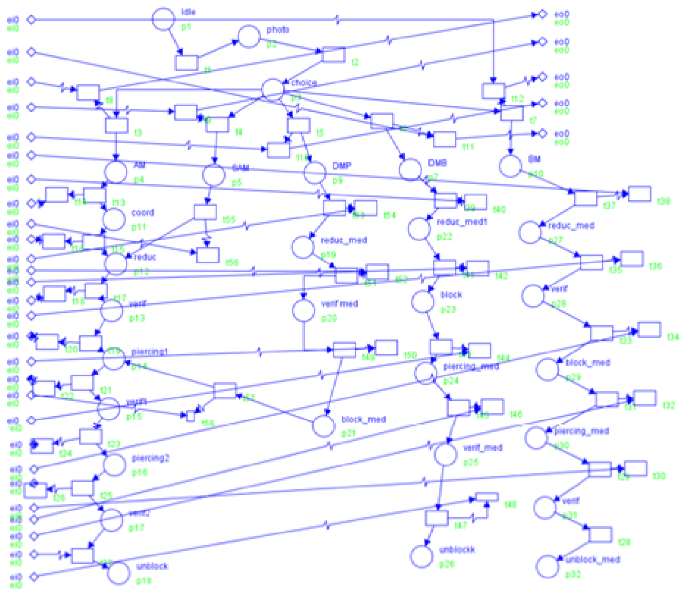
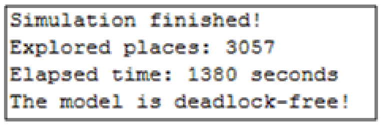


Figure 7: R-TNCES of BROS.



```
Simulation finished!
Explored places: 3057
Elapsed time: 1380 seconds
The model is deadlock-free!
```

Figure 8: BROS's simulation report.

After proving, by simulation, the non-existence of problems related to concurrent access on BROS's reconfigurable shared resources, we do an exhaustive CTL-based verification to check the potential existence of several problems that may be faced at BROS's runtime. Thus, we apply several CTL formulas on the model of the whole BROS system, built using ZiZo and then exported to SESA.

Simultaneous Blocking and Pinning: Pinning in the patient's arm while moving it by unblocking it may lead to dramatic consequences. We check, then, whether this two actions may be simultaneously performed by applying the following formula to BROS's model:

$$EF\ p23\ AND\ p43 \tag{10}$$

where: (i) p23 translates unblocking the arm, (ii) p43 pinning it. The formula is found to be false.

Timeout Issue: We check that the whole surgical intervention does not last more than a given definite time. We apply, then, formula 11 as follows:

$$EF\ [0, 301]\ p23 \tag{11}$$

This formula is also proven to be false.

Intervention Sequence: We have to be sure that BROS complies with the specified logic by performing in order the following actions: reduction, blocking, pinning 1, pinning 2 and unblocking. We apply, therefore, the following CTL formula:

$$AGA\ t18\ X\ AFE\ t25\ X\ AFE\ t40\ X\ AFE\ t74\ X\ AFE\ t111\ X\ TRUE \tag{12}$$

where t18, t25, t40, t74 and t111 are respectively the transitions leading to the places translating reduction, blocking, pinning 1, pinning 2 and unblocking. The formula is proved to be true.

5.3 Implementation: R-UML translation into ROS

In this section, we propose a technique to automatically generate ROS code from R-UML diagrams.

5.3.1 *Dynamic_reconfigure* Package

The *dynamic_reconfigure* ROS package is an extension of ROS parameter server. It permits easy adjustment of filtering parameters on-the-fly (during execution). We can dynamically change the parameters of ROS nodes using this tool. A hierarchical structure in the server is used to store the different parameters. Nevertheless, some relevant parameters are locally copied in each of the nodes for performance reasons. When a parameter is changed on the server, service calls are used to notify the appropriate nodes. The entire parameter hierarchy are loaded and saved thanks to an existing and appropriate functionality. In each robot, a well-defined configuration file is used to get the default parameter values on system startup [49]. This feature is very relevant to our contribution, since configuration parameters are dynamically changed.

5.3.2 R-UML Translation into ROS Configuration File

The paper introduces Table 2 which highlights the correspondence between R-UML class diagrams (Formula 5) and ROS configuration file. The three translation rules are summarized in Table 2 and explained hereafter:

- Rule 1: Each object from the R-ClD is translated into a node in ROS;
- Rule 2: The different attributes belonging to an object are equivalent to parameters in ROS nodes;
- Rule 3: Some attributes of a given class may not be of a predefined type. Therefor, we create stereotypes and enumerated types in, respectively, R-ClD and ROS.

Table 2: Correspondence table for R-UML translation into ROS configuration file.

Rules	R-ClD	ROS
Rule 1	Object	Node
Rule 2	Attribute	Parameter
Rule 3	Stereotype	Enumerated type

5.3.3 Configuration File

We explain in this section how to create a basic configuration file which will be used by $dynamic\_reconfigure$ package.

This first lines aims at initializing ROS and creating a generator using the command *gen = ParameterGenerator()*. The different parameters can be defined once a generator is created. A given parameter can be added to the list using the *add* function. The latter needs the following arguments:

- **name** - a string specifying the parameter's name;
- **type** - it defines the stored parameter's type. It can be whether a *int_t*, a *double_t*, a *str_t* or a *bool_t*;
- **level** - This is a bitmask which will be used later when calling a dynamic reconfigure back. When this happens, an OR function is applied to the level values of parameters that have been changed. The value resulting from this operation is passed to the callback;

- **description** - this argument describes the parameter;
- **default** - this argument defines the parameter's default value;
- **min** - This an optional argument which specifies the minimal value of the parameter. It is not used when defining strings and boolean parameters;
- **max** - This an optional argument which specifies the maximal value of the parameter. It is also not used when defining strings and boolean parameters.

5.3.4 Application to BROS

We start by generating a configuration file for each component of BROS as illustrated in Figure 2. After several discussions with our medical and industrial partners, we decided to use ABB's IRB 120 manipulators. The latter's features, such as payload (3 Kg), accuracy ($+-0.01$ mm) and freedom degree (6°), meet our expectations from BROS robotic arms [50]. Icing on the cake, ROS features packages to handle communication with ABB industrial robot controllers and manipulators [51]. The generated files are then simulated using ABB RobotStudio [52]. Figures 9 and 10 show the simulation of BROS when performing, respectively, the reduction of fracture and the pinning.

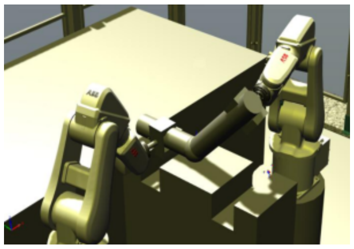


Figure 9: Reduction of fracture.

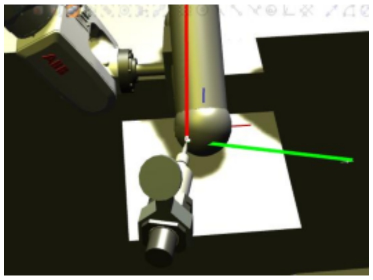


Figure 10: Pinning.

6 Conclusion

The research work presents a new methodology to develop reconfigurable robotic systems and insure their safety from design to implementation. This methodology is original since it uses new technologies like the R-TNCES, R-UML and a new tool, ZiZo. The methodology is applied to a medical robotic system, but it can besides be used with systems belonging to other fields and presenting issues of reconfiguration scenarios and concurrent access to shared resources. Thanks to this methodology, we are able to guarantee the safety of the medical project BROS. The results of the experiments performed on real SCH fracture radiographies were quite satisfactory. Clinical experiments can then be performed after deploying the system on real hardware. This is going to be our future work's subject.

References:

[1] J.-J. Borrelly, É. Coste-Maniere, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon, and N. Turro, "The orccad architecture," *The International Journal of Robotics Research*, vol. 17, no. 4, pp. 338–359, 1998.

[2] J. Albus, H.-M. Huang, A. Lacaze, M. Schneier, M. Juberts, H. Scott, S. Balakirsky, P. W. Shackleford, T. Hong, J. Michaloski *et al.*, "4d/rcs: A reference model architecture for unmanned vehicle systems version 2.0," 2002.

[3] R. C. Arkin and T. Balch, "Aura: Principles and practice in review," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 175–189, 1997.

[4] N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt, "Idea: Planning at the core of autonomous reactive agents," 2002.

[5] D. B. Stewart and P. K. Khosla, "The chimera methodology: Designing dynamically reconfigurable and reusable real-time software using port-based objects," *International Journal of Software Engineering and Knowledge Engineering*, vol. 6, no. 02, pp. 249–277, 1996.

[6] M. O. B. Salem, O. Mosbahi, M. Khalgui, and G. Frey, "Bros-a new robotic platform for the treatment of supracondylar humerus fracture." in *HEALTHINF*, 2015, pp. 151–163.

[7] ——, "Transformation from r-uml to r-tnces: New formal solution for verification of flexible

control systems," in *Software Technologies (ICSOFT), 2015 10th International Joint Conference on*, vol. 2. IEEE, 2015, pp. 1–12.

[8] 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.

[9] 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, pp. 20–31.

[10] C. U. Smith, C. M. Lladó, V. Cortellessa, A. D. Marco, and L. G. Williams, "From uml models to software performance results: an spe process based on xml interchange formats," in *Proceedings of the 5th international workshop on Software and performance*. ACM, 2005, pp. 87–98.

[11] A. D'Ambrogio, "A model transformation framework for the automated building of performance models from uml models," in *Proceedings of the 5th international workshop on Software and performance*. ACM, 2005, pp. 75–86.

[12] S. Bernardi and J. Merseguer, "Performance evaluation of uml design with stochastic well-formed nets," *Journal of Systems and Software*, vol. 80, no. 11, pp. 1843–1865, 2007.

[13] D. B. Petriu and M. Woodside, "An intermediate metamodel with scenarios and resources for generating performance models from uml designs," *Software & Systems Modeling*, vol. 6, no. 2, pp. 163–184, 2007.

[14] M. Marzolla and S. Balsamo, "Uml-psi: the uml performance simulator," in *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*. IEEE, 2004, pp. 340–341.

[15] J. Merseguer, J. Campos, S. Bernardi, and S. Donatelli, "A compositional semantics for UML state machines aimed at performance evaluation," in *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*. IEEE, 2002, pp. 295–302.

[16] M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, and J. Merseguer, "Performance by unified model analysis (puma)," in *Proceedings of the 5th international workshop on Software and performance*. ACM, 2005, pp. 1–12.

[17] G. P. Gu and D. C. Petriu, "From uml to lqn by xml algebra-based model transformations," in *Proceedings of the 5th international workshop on Software and performance*. ACM, 2005, pp. 99–110.

[18] D. C. Petriu and H. Shen, "Applying the uml performance profile: Graph grammar-based derivation of lqn models from uml specifications," in *Computer Performance Evaluation: Modelling Techniques and Tools*. Springer, 2002, pp. 159–177.

[19] ISO, "Iso/iec 19502:2005 information technology—meta object facility (mof)," 2005. [Online]. Available: http://www.iso.org

[20] H. Bruyninckx, "Open robot control software: the orocos project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3. IEEE, 2001, pp. 2523–2528.

[21] C. Schlegel, "Navigation and execution for mobile robots in dynamic environments: an integrated approach." Ph.D. dissertation, University of Ulm, 2004.

[22] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[23] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The brics component model: a model-based development paradigm for complex robotics software systems," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1758–1764.

[24] M. Klotzbücher and H. Bruyninckx, "Coordinating robotic tasks and systems with rfsm statecharts," *JOSER: Journal of Software Engineering for Robotics*, vol. 3, no. 1, pp. 28–56, 2012.

[25] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-tran: Self-reconfigurable modular robotic system," *IEEE/ASME transactions on mechatronics*, vol. 7, no. 4, pp. 431–441, 2002.

[26] "Simulink." [Online]. Available: http://www.mathworks.com/products/simulink/

[27] "Robotics system toolbox." [Online]. Available: http://www.mathworks.com/help/robotics/index.html

[28] C. Jang, B. Song, S. Jung, and S. Kim, "A heterogeneous coupling scheme of opros component framework with ros," in *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2012.

[29] G. Martin, L. Lavagno, and J. Louis-Guerin, "Embedded uml: a merger of real-time uml and co-design," in *Proceedings of the ninth international symposium on Hardware/software codesign*. ACM, 2001, pp. 23–28.

[30] B. Selic, "Using uml for modeling complex real-time systems," in *Languages, Compilers, and Tools for Embedded Systems*. Springer, 1998, pp. 250–260.

[31] A. Gherbi and F. Khendek, "Uml profiles for real-time systems and their applications." *Journal of Object Technology*, vol. 5, no. 4, pp. 149–169, 2006.

[32] A. S. Staines, "A comparison of software analysis and design methods for real time systems," in *Proceedings of World Academy of Science, Engineering and Technology*, vol. 21. Citeseer, 2007.

[33] O. Group *et al.*, "Uml profile for schedulability, perfomance and time specification," *Version 1.1, formal/05-01*, vol. 2, 2005.

[34] S. Gérard, H. Espinoza, F. Terrier, and B. Selic, "6 modeling languages for real-time and embedded systems," in *Model-Based Engineering of Embedded Real-Time Systems*. Springer, 2010, pp. 129–154.

[35] M. Shousha, L. Briand, and Y. Labiche, "A uml/marte model analysis method for uncovering scenarios leading to starvation and deadlocks in concurrent systems," *Software Engineering, IEEE Transactions on*, vol. 38, no. 2, pp. 354–374, 2012.

[36] F. Mallet and C. André, "On the semantics of uml/marte clock constraints," in *Object/Component/Service-Oriented Real-Time Distributed Computing, 2009. ISORC'09. IEEE International Symposium on*. IEEE, 2009, pp. 305–312.

[37] B. Hamid and F. Krichen, "Model-based engineering for dynamic reconfiguration in drtes," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. ACM, 2010, pp. 269–276.

[38] A. Koubâa, *Robot Operating System (ROS): The Complete Reference*. Springer, 2016, vol. 1.

[39] "The Open Source Robotics Foundation," http://www.osrfoundation.org/, accessed: 2016-09-14.

[40] M. O. B. Salem, O. Mosbahi, M. Khalgui, and G. Frey, "R-uml: An uml profile for verification of flexible control systems," in *International Conference on Software Technologies*. Springer, 2015, pp. 118–136.

[41] ——, "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 February, 2015*, 2015, pp. 20–31.

[42] M. O. Ben Salem, O. Mosbahi, M. Khalgui, Z. Jlalia, G. Frey, and M. Smida, "Brometh: Methodology to design safe reconfigurable medical robotic systems," *The International Journal of Medical Robotics and Computer Assisted Surgery*, 2016.

[43] M. H. T. Hack, "Analysis of production schemata by petri nets," DTIC Document, Tech. Rep., 1972.

[44] A. Valmari, "Stubborn sets for reduced state space generation," in *International Conference on Application and Theory of Petri Nets*. Springer, 1989, pp. 491–515.

[45] M. Silva, "Sur le concept de macroplace et son utilisation pour l'analyse des reseaux de petri," *RAIRO-Systems Analysis and Control*, vol. 15, no. 4, pp. 57–67, 1981.

[46] G. Berthelot, "Transformations and decompositions of nets," in *Petri Nets: Central models and their properties*. Springer, 1987, pp. 359–376.

[47] M. Silva and J. M. Colom, "On the computation of structural synchronic invariants in p/t nets," in *European Workshop on Applications and Theory in Petri Nets*. Springer, 1987, pp. 386–417.

[48] P. H. Starke and S. Roch, *Analysing signal-net systems*. Professoren des Inst. für Informatik, 2002.

[49] "Dynamic reconfigure," http://wiki.ros.org/dynamic_reconfigure/, accessed: 2018-06-16.

[50] P. Mikaelsson and M. Curtis, "Portrait-robot d'un petit prodige: Abb présente son nouveau robot irb 120 et son armoire de commande irc5 compact," *Revue ABB*, no. 4, pp. 39–41, 2009.

[51] "Ros-industrial support for abb manipulators (metapackage)." http://wiki.ros.org/abb, accessed: 2017-02-10.

[52] C. Connolly, "Technology and applications of abb robotstudio," *Industrial Robot: An International Journal*, vol. 36, no. 6, pp. 540–545, 2009.