

BROMETH: Methodology to Develop Safe Reconfigurable Medical Robotic Systems

Application on Pediatric
Supracondylar Humeral Fracture

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

&

University of Carthage (Tunisia)

von

Mohamed Oussama BEN SALEM

Saarbrücken

2016

Tag des Kolloquiums:	20.02.2017
Dekan:	Univ.-Prof. Dr. Guido Kickelbick
Vorsitz:	Univ.-Prof. Dr.-Ing. Michael Vielhaber
Berichterstatter:	Univ.-Prof. Dr.-Ing. Georg Frey Prof. Luis Gomes Prof. Valeriy Vyatkin
Akademischer Beisitzer:	Dr. Tilman Sauerwald

Abstract

In this thesis project, we propose a global approach called BROMETH for the development of safe reconfigurable medical robotic systems. The reconfiguration is a useful feature for the modification of software and hardware components to dynamically adapt their behaviors to the execution environment at runtime. We define R-TNCES-based patterns enriched with PCP to model reconfigurable systems with adaptive shared resources. A software tool, ZiZo, is also proposed to model, simulate and verify such R-TNCES models. We introduce, then, a new UML profile, baptized R-UML (Reconfigurable UML), to model such reconfigurable systems. The profile is enriched with a PCP-based solution for the management of resource sharing. An automatic translation of R-UML into R-TNCES is also proposed to support model checking. A new approach is also suggested to generate ROS codes from verified R-UML models. To prove the relevance of BROMETH, the latter is applied to the BROS project.

Keywords : Reconfiguration, Adaptive shared resource, R-TNCES, Petri nets, UML, ROS, E-health, Supracondylar humeral fracture, Modeling, Simulation, Model checking.

Abstrakt

Diese Arbeit schlt fr die Entwicklung sicherer rekonfigurierbarer medizinischer Robotersysteme den umfassenden Ansatz BROMETH vor. Die Rekonfiguration ist eine ntzliche Eigenschaft von Software- und Hardware-Komponenten, durch die sich deren Verhalten dynamisch zur Laufzeit an die Ausfhrungsumgebung anpassen lsst. In der Arbeit werden R-TNCES-basierte, mit PCP erweiterte Muster definiert, um rekonfigurierbare Systeme mit adaptiven geteilten Ressourcen zu modellieren. Auerdem wird ein Software-Werkzeug namens ZiZo vorgeschlagen, um derartige R-TNCES-Modelle zu erstellen, zu simulieren und zu verifizieren. Zur Modellierung solcher rekonfigurierbaren Systeme erfolgt anschlieend die Einfhrung eines neuen UML-Profiles, hier R-UML (Reconfigurable UML) genannt. Das Profil wird zum Management der Ressourcenteilung um eine PCP-basierte Lsung erweitert. Des Weiteren wird eine automatische bersetzung von R-UML nach R-TNCES vorgeschlagen; dies untersttzt das Model-Checking. Schlielich schlt diese Arbeit einen neuen Ansatz vor zur Generierung von ROS-Codes aus verifizierten R-UML-Modellen. Um die Bedeutung des BROMETH-Ansatzes zu belegen, wird dieser auf das BROS-Projekt angewendet.

List of Figures

1.1	BROMETH methodology steps.	6
1.2	BROS's class diagram.	7
1.3	Behaviour of B-BROS1 and B-BROS2 in AM.	11
1.4	Behavior of the surgeon, B-BROS1 and B-BROS2 in SAM.	11
2.1	Gartland's classification of supracondylar fractures of the humerus.	26
2.2	Lagrange's classification of supracondylar fractures of the humerus.	26
2.3	Smida's classification of supracondylar fractures of the humerus.	27
2.4	Fracture's radiographs.	28
2.5	Injured elbow installed under the fluoroscopic image intensifier.	28
2.6	Elbow immobilization after obtaining fracture reduction.	28
2.7	Lateral percutaneous pinning.	28
3.1	Behavior of A and B before a reconfiguration scenario	34
3.2	Behavior of A and B after a reconfiguration scenario	34
3.3	Illustrative example's R-TNCES	35
3.4	Shared resource's modeling.	37
3.5	Shared resource's reconfiguration TNCES.	38
3.6	State TNCES.	39
3.7	Ceiling TNCES.	39
3.8	The resource Q's modeling	40
3.9	A task's modeling.	41
3.10	Task's reconfiguration TNCES.	41
3.11	Task's state R-TNCES.	42
3.12	A and B's modeling.	43
3.13	A and B behaviors after using PCP.	43
3.14	Screenshot from SESA.	44
3.15	Screenshot from SESA.	44
3.16	Specification step.	46
3.17	BROS modeling using ZiZo.	48
3.18	Certification step.	50
3.19	BROS's simulation report.	51
4.1	Behavior of A and B before a reconfiguration scenario.	56
4.2	Behavior of A and B after a reconfiguration scenario.	57
4.3	A and B behaviors after using PCP.	57
4.4	Task class.	59
4.5	Resource class.	60

4.6	Running example's object diagram.	60
4.7	Shared resource's R-StD.	63
4.8	Task's R-StD.	64
4.9	Task A's R-StD.	65
4.10	Resource R's R-StD.	66
4.11	Illustrative example's R-TNCES.	70
4.12	Screenshot from SESA.	71
4.13	Tasks A and B's modeling using PCP.	72
4.14	Screenshot from SESA.	73
4.15	Screenshot from SESA.	73
4.16	ZiZo's different modules.	80
4.17	R-StD of BROS.	81
4.18	R-TNCES of BROS.	82
4.19	BROS's simulation report.	82
5.1	Implementation step.	87
5.2	Middleware's class diagram.	89
5.3	Sequence diagram of communication with CU.	90
5.4	Coordinate system axes.	91
5.5	Image matching applied on two fractured elbow images.	92
5.6	Contour comparison performed by MW.	92
5.7	Different coordinate systems.	93
5.8	Reduction of a type II fracture.	94
5.9	Reduction of a type III fracture.	94
5.10	Operating room definition.	95
5.11	Blocker 1 and Blocker 2's 3D modeling.	96
5.12	Pinning's 3D modeling.	96
5.13	Limb's 3D modeling.	96
5.14	Blocking the patient's limb.	97
5.15	Robotized fracture reduction.	97
5.16	Blocking the fractured limb at the forearm.	97
5.17	Orientation of "Pinning" in the case of a type II fracture.	98
5.18	Orientation of "Pinning" in the case of a type III fracture.	98
5.19	Error rates per SCH fracture type.	101

List of Tables

1.1	Complementary Strengths of Robots and Humans.	3
1.2	BROS's operating modes.	10
2.1	Gartland's classification of supracondylar fractures of the humerus.	25
2.2	Lagrange's classification of supracondylar fractures of the humerus.	26
2.3	Smida's classification of supracondylar fractures of the humerus.	27
4.1	Correspondence table for R-StD translation into R-TNCES. .	67
4.2	Correspondence table for R-UML translation into ROS configuration file.	78
5.1	Synchronization logic signals.	99
5.2	Sorting 42 SCH fractures' radiographies according to Lagrange's classification.	100

Nomenclature

α	A function which maps a stereotype to an attribute
β	A function which maps an attribute to a class
δ	A function which maps a guard to a transition
ϵ	A function which maps an action to a transition
η	A function controlling tasks
γ	A function which maps an event to a transition
ι	A function defining whether a task is using a resource in the triggered mode
Ω	A function mapping a reconfigurable state diagram to a class
Ψ	An input/output structure of TNCES module
$\sum M$	A set of Judgment Matrices
$\sum R$	A set of n_2 shared resources
$\sum R - TNCES$	A set of n_1 R-TNCES
θ	A function controlling resources
ϖ	A virtual coordinator handling $\sum M$
ζ	A function which maps a transition to a pair of states
A	A finite set of attributes of classes
Ac	A finite set of actions
B	The behavior module that is a union of multi StD
B	The distance separating the two wires
Bc	A set of TNCES module input condition arcs
Be	A set of TNCES module input event arcs
C	A finite set of classes in a class diagram
C^{in}	A finite set of TNCES module condition input signals
C^{out}	A finite set of TNCES module condition output signals

Cl	A ceiling of a resource
$CLDs$	A finite set of class diagrams
CM	A camera matrix
CN	A finite set of condition arcs
CN	A superset of condition signals
Cs	A set of TNCES module output condition arcs
D	The humeral palette's width
D_0	The initial set of the clocks associated with the places
DC	A superset of time constraints on output arcs
DR	The set of minimum times that the token should spend at particular place before the transition can fire
Dt	A set of TNCES module output event arcs
Dt	The final set of limitation time that defines maximum time that the place may hold a token
E^{in}	A finite set of TCNES module event input signals
E^{out}	A finite set of TCNES module event output signals
EN	A finite set of event arcs
EN	A superset of event signals
Ev	A finite set of events in transitions
F	A finite set of flow arcs between places and transitions
F	A superset of flow arcs
G	A finite set of guards
M	A finite set of methods of the classes
m_0	The initial marking
O	A finite set of objects
P	A finite set of transitions
P	A superset of places
PCP	Priority Ceiling Protocol

R	The control module consisting of a set of reconfiguration functions $R=\{r_1, \dots, r_n\}$
$R - StDs$	A finite set of reconfigurable state diagrams
$R - TNCES$	Reconfigurable Timed Net Condition/Event Systems
$RDCS$	Reconfigurable Distributed Control Systems
Rec	A parameter which indicates whether a resource/task is added to the systems
RV	A rotation vector
S	A finite set of stereotypes
S	The stability threshold
S	The state of a resource/task
St	A finite set of states in a state diagram
T	A finite set of transitions
T	A superset of transitions
Tr	A finite state of transitions in a state diagram
V	A function which maps an event-processing mode for every transition
W	A function which maps a weight to a flow arc
Z	The initial clock position

Contents

1	Introduction	1
1.1	Introduction	2
1.2	BROS: Robotic System for the Treatment of Supracondylar Humeral Fracture	7
1.2.1	Architecture	7
1.2.2	Reconfiguration Modes	9
1.2.3	Shared Resources and Issues	10
1.3	Outline	12
1.4	Originality	12
1.5	Publications and Awards	13
1.6	Collaboration	14
2	State of the Art	15
2.1	Introduction	16
2.2	IT State of the Art	16
2.2.1	Net Condition/Event Systems	16
2.2.2	Timed Net Condition/Event System	17
2.2.3	Reconfigurable Timed Net Condition/Event System	18
2.2.4	Tools Modeling Petri Nets	19
2.2.5	Model Checking	19
2.2.6	Temporal Logic	20
2.2.7	Priority Ceiling Protocol	22
2.2.8	Image Processing	22
2.2.9	UML Profiles	22
2.2.10	Formalism Transformation	23
2.2.11	Robot Operating System	24
2.3	Medical State of the Art	25
2.3.1	Supracondylar Humeral Fracture’s Classifications	25
2.3.2	Supracondylar Humeral Fracture Treatment	27
2.4	Summary	28
3	Formal Modeling and Verification	31
3.1	Introduction	32
3.2	Contribution 1: PCP-based Solution for Resource Sharing in R-TNCES	32
3.2.1	Motivations	32
3.2.2	Case Study	33
3.2.3	Formalization	36
3.2.4	Modeling	37

3.2.5	Verification	43
3.3	Contribution 2: New Environment ZiZo	44
3.3.1	Motivation	44
3.3.2	Features	45
3.4	Application: Specification and Certification of BROS	45
3.4.1	Specification Step	45
3.4.2	Certification Step	49
3.5	Summary	52
4	New UML Profile: R-UML and Automatic Transformation to ROS	53
4.1	Introduction	54
4.2	Motivations	54
4.3	Running Example	56
4.4	R-UML Profile	57
4.4.1	Structure Modeling	58
4.4.2	Behavior Modeling	61
4.5	Transformation from R-UML to R-TNCES	66
4.5.1	R-Std Translation into R-TNCES	66
4.5.2	Verification	68
4.5.3	Implementation	73
4.6	Transformation from R-UML to ROS	74
4.6.1	State of the Art	74
4.6.2	Transformation Rules from R-UML to ROS	74
4.7	Application to BROS	79
4.8	Summary	82
5	Implementation of BROS	85
5.1	Introduction	86
5.2	Platform and Environment	88
5.3	Middleware	89
5.3.1	Architecture	89
5.3.2	Image Processing	91
5.4	Control Unit	94
5.4.1	Station Definition	95
5.4.2	B-BROS1 Module	96
5.4.3	B-BROS2 Module	96
5.4.4	P-BROS Module	97
5.4.5	Synchronization Module	98
5.4.6	CU-MW Communication Module	99
5.4.7	Surgeon-Robot Interface	99
5.4.8	Simulation and Test of BROS	100
5.5	Summary	101

6 Conclusion and Perspectives	103
6.1 Conclusion	104
6.2 Perspectives	104
Bibliography	105

Introduction

Contents

1.1	Introduction	2
1.2	BROS: Robotic System for the Treatment of Supra- condylar Humeral Fracture	7
1.2.1	Architecture	7
1.2.2	Reconfiguration Modes	9
1.2.3	Shared Resources and Issues	10
1.3	Outline	12
1.4	Originality	12
1.5	Publications and Awards	13
1.6	Collaboration	14

1.1 Introduction

The field of robotics is expanding day after day. The ability of robots to replace, supplement or transcend human performance has had a profound influence on many fields of our society, spanning fields such as agriculture, military and especially medicine. Patients demand greater precision, less and minimally invasive procedures, and faster recovery times. The increasing life expectancy associated with a need for reducing costs and increasing efficiency have opened the door for new and innovative solutions in the medical robotic industry. The field of computer-assisted surgery is relatively new since the first clinical application of a robot was performed to a neurosurgery in 1985 [Kwoh 1988]. Since then, many research centers around the world have developed a multitude of robotic surgical products to tackle new areas such as ophthalmology, radiology, urology, cardiothoracic and orthopedics [Cleary 2001].

The field of computer-assisted surgery is relatively new. The first clinical application of a robot was performed to a neurosurgery in 1985 [Kwoh 1988]. Since then, many research centers around the world have developed a multitude of robotic and computer-assisted surgical products, tackling new areas such as orthopedics [Martelli 2000], radiology [Bertaud 2008], urology [Llobet 2007], cardiothoracic [Costa 2012] and ophthalmology [Cleary 2001, Wang 2006, Gomes 2011].

At the beginning, the products consisted mainly of modifying industrial robots for a given procedure, but the safety of such mechanisms was called into question [Xu 2015, Garg 2013]. The current trend in surgical robotics is that systems will no longer be adaptations of industrial robots, but increasingly systems will be dedicated to a single task, with some autonomous capabilities and an extreme level of safety [Riviere 2006]. Medical robots are more and more accepted thanks to their ability to significantly improve surgeon's technical capability [Kooijmans 2007, Martelli 2003]. They also take advantage of the complementary strengths between humans and robotic devices as described in Table 1.1 [Kutz 2003, Vendruscolo 2001]. Their function is far from replacing the surgeon, but supporting him with ameliorated dexterity, visual feedback, and information integration [Marcus 2013, Hoeckelman 2015]. Furthermore, the actual systems still lack of flexibility concerning their capacity to be autonomous and reconfigurable [Haux 2006].

Table 1.1: Complementary Strengths of Robots and Humans.

Strengths	Limitations
<p>Humans</p> <ul style="list-style-type: none"> - Excellent judgment - Excellent hand-eye coordination - Excellent dexterity (at natural human scale) - Able to integrate and act on multiple information sources - Easily trained - Versatile and able to improvise 	<ul style="list-style-type: none"> - Prone to fatigue and inattention - Tremor limits fine motion - Limited manipulation and dexterity outside natural scale - Cannot see through tissue - Bulky end-effectors - Limited geometric accuracy - Hard to keep sterile - Affected by radiation, infection
<p>Robots</p> <ul style="list-style-type: none"> - Excellent geometric accuracy - Untiring and stable - Immune to ionizing radiation - Operate at many different scales of motion and payload - Integrate multiple sources of numerical and sensor data - Possibility of less or minimally invasive surgery (MIS) 	<ul style="list-style-type: none"> - Poor judgment - Hard to adapt to new situations - Limited dexterity - Limited hand-eye coordination - Limited haptic sensing - Limited ability to integrate and interpret complex information

One of the most common injuries faced by pediatric orthopedic surgery is the supracondylar fracture of the humerus (or SCH). It accounts for 18% of all pediatric fractures and 75% of all elbow fractures [Landin 1986]. It mainly occurs during the first decade of life and are more common among boys [Landin 1983]. The current treatment of SCH may lead to many complications. The neurological ones consists in damages caused to the median nerve during the reduction of the fracture or during the open procedure. The study in [Gosens 2003] also reports some vascular complications, mostly consisting in the disruption of the brachial artery. All those complications are principally caused by the "blind" pinning the surgeons perform [Flynn 1974]. Even though they are usually using an image intensifier, the medical staff can't guess in advance the trajectory the pin will follow. Images are actually taken once the pin is inserted, which may cause the previously mentioned complications. Other inconvenient of the current treatment technique is the recurrent medical staff exposure to radiations when using the fluoroscopic C-arm [Clein 1954]. These X-ray Radiations are harmful, and fluoroscopic examinations usually involve higher radiation doses than simple radiography. For example, a work in [Rampersaud 2000a] showed that, for spine surgeons, radiation exposures may approach or exceed guidelines for cumulative exposure. Another research in [Haque 2006] showed that the fluoroscopically assisted placement of pedicle screws in adolescent idiopathic scoliosis may expose the spine surgeon to radiation levels that exceed established lifetime dose equivalent limits.

Considering these constraints and issues, a new national project, baptized BROS (Browser-based Reconfigurable Orthopedic Surgery), has been launched to remedy these problems. BROS is a new reconfigurable robotized platform dedicated to the treatment of supracondylar humeral fractures in children. It is capable of running under several operating modes to meet the surgeons requirements and well-defined constraints. Thus, it can automatically perform the whole surgery or bequeath some tasks to the surgeon.

Given the said issues faced when designing a reconfigurable system with adaptive resources, we aim to extricate a new methodology to be followed each time we want to validate and certify a medical robotized platform. And this is what we are proposing in this thesis. This methodology will be baptized BROMETH and composed of four main steps as shown in Figure 1.1: specification, certification, implementation and deployment. The first step, specification, consists in defining the expectations from the system to design and express them in a modeling formalism. During the certification step, we apply mathematical properties on the obtained models to check some properties. Once the system is specified and certified, we move to the implementation step where the checked models are translated into programming codes which are simulated, then, on specific software tools. The last step, deployment, comes to integrate the validated codes on the corresponding hardware. This work introduces BROS, the new robotic platform which

ensures a safer treatment of the SCH for both surgeons and patients. The whole BROMETH 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 reconfigurable distributed architecture and permits to safely design it from the specification to the deployment steps.

Several methodologies have been recently proposed in our community to design safe medical robotic systems. The work in [Kouskoulas 2013] applies quantified differential-dynamic logic (QdL) to analyze a control algorithm designed to provide directional force feedback for a surgical robot. Several problems were revealed in the algorithm, proving that it was in general unsafe, and described exactly what could go wrong. The work then applied QdL to guide the development of a new algorithm that provides safe operation along with directional force feedback.

The authors in [Rahimi 1991] explains a design framework for the software safety module for industrial robotic operations. The module proposed includes preconditions and post-conditions for each robot action based on a static model of the robot physical configurations and the environment. A generic software safety verification and encoding is presented for safety-critical actions of the robot. Another work is introduced in [Sun 2007] and uses RT-PROMELA to construct models which were checked in RT-SPIN. In order to reduce the state explosion problem, a model is decomposed into multiple sub-models, each with a smaller state space that can be checked individually, and then the proofs checked for noninterference. Cooperation among multiple clock variables due to their lack of notion of urgency and their asynchronous interactions, are also addressed.

Another work cited in [Dumitru 2015] examined the potential risks for the design and manufacture products for medical robotics. It researched all foreseeable potential defects and their causes and effects for a robotic arm. Through a qualitative assessment of the effect of a defect, the probability of occurrence and probability of detection were discovered risks and set appropriate corrective measures.

These works are relevant, however, they don't deal with systems with a self-reconfiguration function like BROS. They don't simulate either the models nor the codes after being corrected.

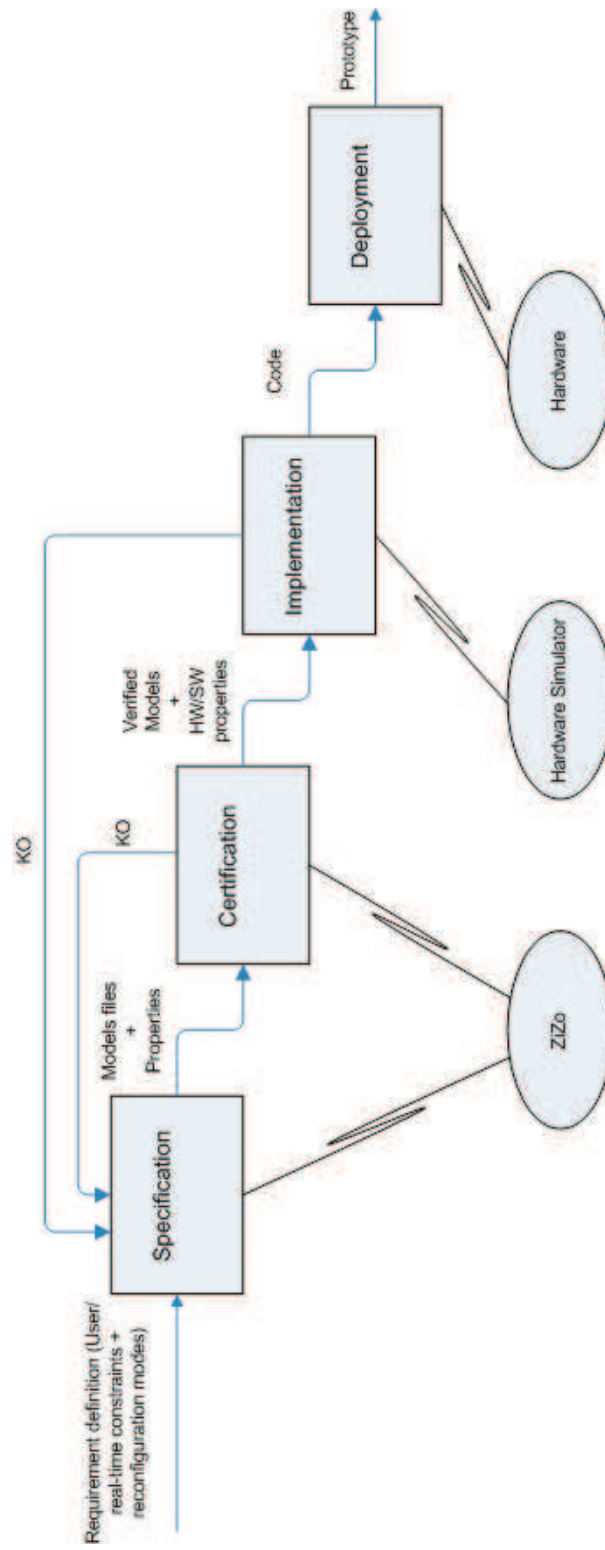


Figure 1.1: BROMETH methodology steps.

1.2 BROS: Robotic System for the Treatment of Supracondylar Humeral Fracture

Considering these constraints and issues, a new national project, baptized BROS (Browser-based Reconfigurable Orthopedic Surgery), has been launched to remedy these problems. BROS is a new reconfigurable robotized platform dedicated to the treatment of supracondylar humeral fractures in children. It is capable of running under several operating modes to meet the surgeons requirements and well-defined constraints. Thus, it can automatically perform the whole surgery or bequeath some tasks to the surgeon.

1.2.1 Architecture

BROS is a robotic platform dedicated to humeral supracondylar fracture treatment. It is able to reduce fractures, block the arm and fix the elbow bone's fragments by pinning. It also offers a navigation function to follow the pins' progression into the fractured elbow. BROS is, as shown in the class diagram in Figure 1.2, composed of a browser (BW), a control unit (UC), a middleware (MW), a pinning robotic arm (P-BROS) and 2 blocking and reducing arms (B-BROS1 and B-BROS2). The said components are detailed hereafter.

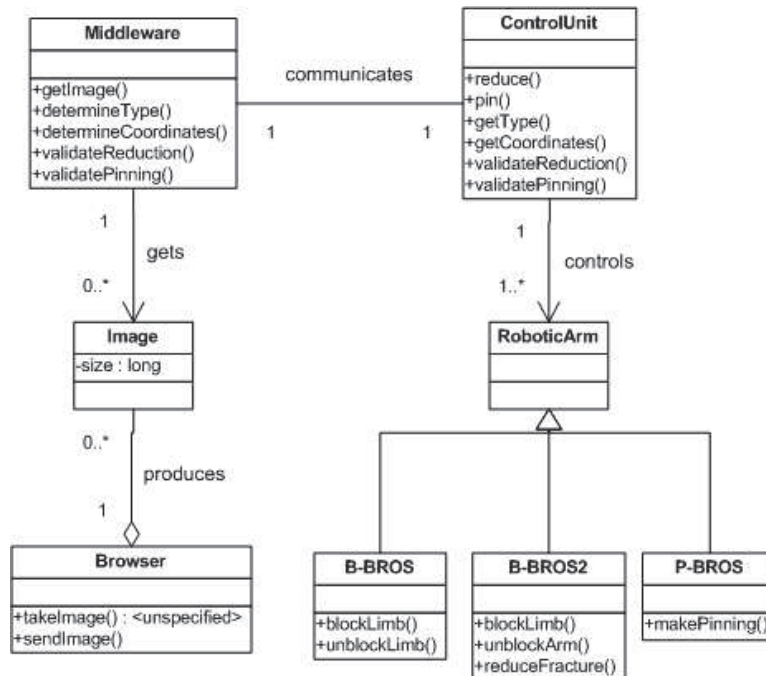


Figure 1.2: BROS's class diagram.

Browser

The browser, which is a Medtronic's product and called FluoroNav, is a combination of specialized surgical hardware and image guidance software designed for use with a StealthStation Treatment Guidance System. Together, these products enable a surgeon to track the position of a surgical instrument in the operating room and continuously update this position within one or more still-frame fluoroscopic images acquired from a C-arm. The advantages of this virtual navigation over conventional fluoroscopic navigation include:

- the ability to navigate using multiple fluoroscopic views simultaneously;
- the ability to remove the C-arm from the operative field during navigation;
- significant reduction in radiation exposure to the patient and staff.

In addition, the FluoroNav System allows the surgeon to:

- simulate and measure instrument progression or regression along a surgical trajectory;
- save instrument trajectories, and display the angle between two saved trajectories or between a saved trajectory and the current instrument trajectory;
- measure the distance between any two points in the camera's field of view;
- measure the angle and distance between a surgical instrument and a plane passing through the surgical field (such as the patient midplane).

Primary hardware components in the FluoroNav System include the FluoroNav Software, a C-arm Calibration Target, a reference frame, connection cables, and specialized surgical instruments.

Control Unit

The CU ensures the smooth running of the surgery and its functional safety. It asks the supracondylar fracture's type to the middleware, and then computes, according to it, the different coordinates necessary to specify the robotic arms' behaviors concerning the fracture's reduction, blocking the arm and performing pinning. The surgeon monitors the intervention progress thanks to a dashboard installed on the CU.

Middleware

The middleware is a software installed on the browser which acts as a mediator between the CU and the BW. It is an intelligent component that provides several features of real-time monitoring and decision making. The middleware contains several modules: (i) an image processing module, (ii) a controller, (iii) a communication module with the CU.

Pinning Robotic Arm

The pinning robotic arm, P-BROS, inserts two parallel Kirschner wires according to Judet technique [Judet 1953] to fix the fractured elbow's fragments. To insure an optimal postoperative stability, BROS respects the formula:

$$S = B/D > 0.22 \tag{1.1}$$

where S is the stability threshold, B the distance separating the two wires and D the humeral palette's width [Smida 2007].

Blocking and Reducing Robotic Arms

B-BROS1 blocks the arm at the humerus to prepare it to the fracture reduction. B-BROS2 performs then a closed reduction to the fractured elbow before blocking it once the reduction is properly completed.

1.2.2 Reconfiguration Modes

Reconfiguration is an important feature of BROS and makes the latter an original medical robotic system. It is designed to be able to operate in different modes to be as flexible as possible and adapted to the characteristics and requirements of the fracture under treatment. The surgeon can actually decide to take over and manually perform a task if BROS does not succeed to automatically perform it. The said task can be fracture reduction, blocking the arm or pinning the elbow. Thus, five different operating modes are designed, detailed hereafter and summarized in Table 1.2.

Automatic Mode (AM)

The whole surgery is performed by BROS. The surgeon oversees the operation running.

Semi-Automatic Mode (SAM)

The surgeon reduces the fracture. BROS performs the remaining tasks.

Degraded Mode for Pinning (DMP)

BROS only realizes the pinning. It is to the surgeon to insure the rest of the intervention.

Degraded Mode for Blocking (DMB)

BROS only blocks the fractured limb. The remaining tasks are manually done by the surgeon.

Basic Mode (BM)

The whole intervention is manually performed. BROS provides navigation function using the middleware that checks in real time the smooth running of the operation.

Table 1.2: BROS's operating modes.

	Reduction	Blocking	Pinning	Unblocking
AM	Robotized	Robotized	Robotized	Robotized
SAM	Manual	Robotized	Robotized	Robotized
DMP	Manual	Manual	Robotized	Robotized
DMB	Manual	Robotized	Manual	Robotized
BM	Manual	Manual	Manual	Manual

1.2.3 Shared Resources and Issues

BROS is a distributed system composed of several entities: the browser, the control unit, the middleware, the two blocking and reducing arms and the two pinning arms. The said entities may be represented by several sharing resource processes. The most relevant resources in our system are the browser and the patient's arm. The first is solicited by the robotic arms, the MW and the surgeon (when a manual reduction or pinning is performed) to update the image on the screen. As to the fractured arm, it may be used by whether the surgeon or the robotic arms.

Applying a reconfiguration scenario on BROS by switching from one operating mode to another may actually lead to a deadlock because of concurrent access to shared resources and which is usually unsafe. This is what happens for example when switching from AM to SAM. To illustrate this situation, we represent B-BROS1, B-BROS2 and the surgeon by three processes with increasing priorities (B-BROS1 < B-BROS2 < the surgeon). This is due to the fact that human intervention takes precedence over the

robotic one, and B-BROS2 has one more function than B-BROS1, which is the fracture reduction.

As illustrated in Figure 1.3, B-BROS1 starts by locking the patient's arm to block it and frees it once the blocking is achieved ($P(A)$ and $V(A)$ respectively stand for locking and freeing the fractured limb). B-BROS2 locks it to reduce the fracture, and then locks the browser ($P(BW)$ and $V(BW)$ respectively stand for locking and freeing the browser) time to update the image displayed on the screen. Once the blocking is done, B-BROS2 frees the browser. The two robotic arms will successively use the patient's arm to unlock it at the end of the intervention.

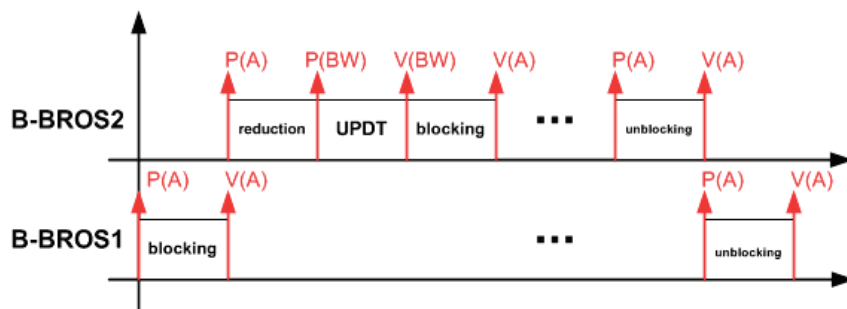


Figure 1.3: Behaviour of B-BROS1 and B-BROS2 in AM.

When the surgeons judges the fracture reduction performed by B-BROS2 as unsatisfying, he can decide to manually do it, and, thus, switches the system from AM to SAM. However, when trying to use the patient's arm, he finds it already locked by B-BROS2 and a deadlock occurs as illustrated in Figure 1.4.

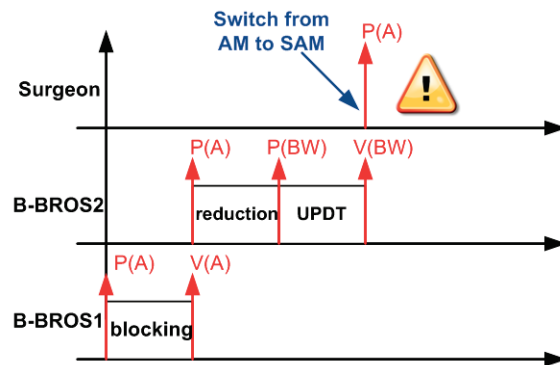


Figure 1.4: Behavior of the surgeon, B-BROS1 and B-BROS2 in SAM.

We see, according to this example, that problems may be faced in reconfigurable control systems, and in this case with BROS, when dealing with concurrent processes that share resources.

1.3 Outline

In Chapter 2, an overview of IT concepts and computer techniques is given, followed by an introduction to some medical knowledge necessary to the achievement of the project. This chapter also relates the state of the art in several areas on which we work throughout this thesis.

Chapter 3 defines new Petri Nets-based to model both tasks and resources of a reconfigurable system. A new Petri Nets-based editor and random-simulator named ZiZo is also developed to model and simulate the generated models. These new concepts and tools are applied on BROS as part of going through the specification and certification steps of BROMETH.

Chapter 4 proposes a new UML profile, baptized R-UML (Reconfigurable UML), to model flexible control systems sharing adaptive shared resources. The profile is enriched with a PCP-based solution for the management of resource sharing which is defined in Chapter 3. The chapter also presents an automatic translation of R-UML into R-TNCES to support model checking. BROS serves as an application of this new concept on a real study case. This chapter also introduces new solutions to automatize the generation of codes usable by Robot Operating System from R-UML models.

Chapter 5 discusses the implementation step of BROS. It exposes and evaluates the results of the treatment of supracondylar humeral fracture by BROS. The development of the middleware and control unit of BROS are detailed in this chapter.

In Chapter 6, the results shown are discussed and conclusions of the work presented are drawn. Future improvements that could enrich the work developed during this dissertation are proposed.

1.4 Originality

This work starts from a real medical issue to extricate a new methodology relevant to the development of various reconfigurable robotic systems.

The said methodology proposes several theoretic contributions such as PCP-based solutions for the management of adaptive shared resources in R-TNCES. A new UML profile (R-UML) is also introduced and which allows to model and verify reconfigurable systems. Likewise, automatic translation from R-UML to R-TNCES and from R-UML to ROS are introduced.

The previous contributions were integrated in a new software tool, baptized ZiZo, which permits modeling, simulating and verifying reconfigurable real-time control tasks sharing adaptive resources.

The relevance of the methodology is proved by applying it to the medical robotic system BROS. The latter is now certified to be safe and is ready to be implemented.

1.5 Publications and Awards

The outcomes of this thesis are published in the hereafter list of publications:

International Conferences (published):

- [1] **Mohamed Oussama Ben Salem**, Olfa Mosbahi, Mohamed Khalgui: *PCP-based Solution for Resource Sharing in Reconfigurable Timed Net Condition/Event Systems*. ADECS @ Petri Nets 2014: 52-67, **Class B**
- [2] **Mohamed Oussama Ben Salem**, Olfa Mosbahi, Mohamed Khalgui, Georg Frey: *ZiZo: Modeling, Simulation and Verification of Reconfigurable Real-time Control Tasks Sharing Adaptive Resources - Application to the Medical Project BROS*. HEALTHINF 2015: 20-31, **Class C**
- [3] **Mohamed Oussama Ben Salem**, Olfa Mosbahi, Mohamed Khalgui, Georg Frey: *BROS - A New Robotic Platform for the Treatment of Supracondylar Humerus Fracture*. HEALTHINF 2015: 151-163, **Class C. Selected Paper**
- [4] **Mohamed Oussama Ben Salem**, Olfa Mosbahi, Mohamed Khalgui, Georg Frey: *Transformation from R-UML to R-TNCES: New Formal Solution for Verification of Flexible Control Systems*. ICISOFT-PT 2015: 64-75, **Class B. Selected Paper**

Journals (Accepted):

- [1] **Mohamed Oussama Ben Salem**, Olfa Mosbahi, Mohamed Khalgui, Zied Jlalialia, Georg Frey, Mahmoud Smida: *BROMETH: Methodology to Design Safe Reconfigurable Medical Robotic Systems*, International Journal of Medical Robotics and Computer Assisted Surgery, **Impact factor: 1.511**

Book Chapters (published):

- [1] **Mohamed Oussama Ben Salem**, Olfa Mosbahi, Mohamed Khalgui, Georg Frey: *Towards a Safer and More Optimal Treatment of the Supracondylar Humerus Fracture*, Biomedical Engineering Systems and Technologies 2015. ISBN: 978-3-319-27706-6 (Print) 978-3-319-27707-3 (Online).
- [2] **Mohamed Oussama Ben Salem**, Zied Jlalialia, Olfa Mosbahi, Mohamed Khalgui, Mahmoud Smida: *New Robotic Platform for a Safer and More Optimal Treatment of the Supracondylar Humerus Fracture*, European Project Space on Intelligent Systems, Pattern Recognition and Biomedical Systems 2015. ISBN: 978-989-758-095-6.
- [3] **Mohamed Oussama Ben Salem**, Olfa Mosbahi, Mohamed Khalgui, Georg Frey: *R-UML: An UML Profile for Verification of Flexible Control Systems*. Communications in Computer and Information Science 2016.

ISSN: 1865-0929.

Awards :

- 1st Prize - Tunisian National Contest "My thesis in 180 seconds" (Franco-phone University Association, June 2015).
- Best Mobidoc PhD Student (Agence Nationale de Promotion de la Recherche Scientifique, November 2015).

1.6 Collaboration

This research work is carried out within a MOBIDOC PhD thesis of the PASRI program, EU-funded and administered by ANPR (Tunisia). The BROS national project is a collaboration between the orthopedic institute of Mohamed Kassab, eHTC, INSAT (LISI Laboratory), Tunisia Polytechnic School and Saarland University.

State of the Art

Contents

2.1	Introduction	16
2.2	IT State of the Art	16
2.2.1	Net Condition/Event Systems	16
2.2.2	Timed Net Condition/Event System	17
2.2.3	Reconfigurable Timed Net Condition/Event System	18
2.2.4	Tools Modeling Petri Nets	19
2.2.5	Model Checking	19
2.2.6	Temporal Logic	20
2.2.7	Priority Ceiling Protocol	22
2.2.8	Image Processing	22
2.2.9	UML Profiles	22
2.2.10	Formalism Transformation	23
2.2.11	Robot Operating System	24
2.3	Medical State of the Art	25
2.3.1	Supracondylar Humeral Fracture's Classifications	25
2.3.2	Supracondylar Humeral Fracture Treatment	27
2.4	Summary	28

2.1 Introduction

This dissertation reports formal modeling and verification of reconfigurable discrete event control systems (RDECSs). All independent innovation works relating to modeling are based on the formalisms Net Condition/Event System (NCES) and Timed Net Condition/Event System (TNCES) and Reconfigurable Timed Net Condition/Event System (R-TNCES). Model checking technologies are applied to perform the formal verification. The main contributions of this work are applied to a medical robotic platform dealing with supracondylar humeral fracture. Thus, for the better understanding of works of this dissertation, relevant elemental knowledge on Petri Nets-based formalisms, model checking technologies and other software technologies are recalled in this chapter. We present, then, the different classifications of supracondylar humeral fracture and how it is currently treated.

2.2 IT State of the Art

Several computer concepts, techniques and software used throughout this work are introduced in this section.

2.2.1 Net Condition/Event Systems

The formalism of Net Condition/Event Systems (NCES) is an extension of the well known Petri net formalism. It was introduced by Rausch and Hanisch in [Rausch 1995] and further developed through the last years, in particular in [Hanisch 1999], according to which a NCES is a place-transition net formally represented by a tuple:

$$NCES = (P, T, F, CN, EN, C^{in}, E^{in}, C^{out}, E^{out}, B_c, B_e, C_s, D_t, m_0) \quad (2.1)$$

where:

- P (resp. T) is a non-empty finite set of places (resp. transitions),
- F is a set of flow arcs, $F : (P \times T) \cup (T \times P)$,
- CN (resp. EN) is a set of condition (resp. event) arcs, $CN \subseteq (P \times T)$ (resp. $EN \subseteq (T \times T)$),
- C^{in} (resp. E^{in}) is a set of condition (resp. event) inputs,
- C^{out} (resp. E^{out}) is a set of condition (resp. event) outputs,
- B_c (resp. B_e) is a set of condition (resp. event) input arcs in a NCES module,

- $B_c \subseteq (C^{in} \times T)$ (resp. $B_e \subseteq (E^{in} \times T)$),
- C_s (resp. D_t) is a set of condition (resp. event) output arcs,
- $C_s \subseteq (P \times E^{out})$ (resp. $D_t(T \times E^{out})$),
- $m_0 : P \rightarrow 0, 1$ is the initial marking.

2.2.2 Timed Net Condition/Event System

The formalism was introduced by [Hanisch 1997]. A TNCES is a tuple:

$$TNCES = (P, T, F, m_0, \Psi, CN, EN, DC) \quad (2.2)$$

where:

- $P = \{p_1, p_2, \dots, p_n\}$ is a finite set of places;
- $T = \{t_1, t_2, \dots, t_m\}$ is a finite set of transitions;
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of flow arcs between 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 input/output structure of TNCES module which is represented by the following tuple:

$$\Psi = (C^{in}, E^{in}, C^{out}, E^{out}, B_c, B_e, C_s, D_t) \quad (2.3)$$

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;
- $B_c \subseteq C^{in} \times T$ is a set of TNCES module input condition arcs;
- $B_e \subseteq E^{in} \times T$ is a set of TNCES module input event arcs;
- $C_s \subseteq P \times C^{out}$ is TNCES module output condition arcs;
- $D_t \subseteq T \times E^{out}$ is a set of TNCES module output event arcs.

Time intervals are assigned to the pre-transition flow arcs $F \subseteq P \times T$, which impose time constraints to the firing of the transition:

$$DC = (DR, DL, D_0) \quad (2.4)$$

where:

- DR represents the set of minimum times that the token should spend at particular place before the transition can fire;
- DL is the final set of limitation time that defines maximum time that the place may hold a token (if all the other conditions for transition firing are met);
- D_0 is the initial set of the clocks associated with the places.

2.2.3 Reconfigurable Timed Net Condition/Event System

An R-TNCES is an extension of the formalism TNCES with a specific function of self-reconfiguration [Zhang 2013, Zhang 2015, Wang 2015]. It is defined as a structure $R\text{-TNCES} = (B, R)$, where R is the control module consisting of a set of reconfiguration functions $R = r_1, \dots, r_n$ and B is the behavior module that is a union of multi TNCESs, represented as

$$B = (P, T, F, W, CN, EN, DC, V, Z) \quad (2.5)$$

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),
- $DC: F \cap (P \times T) \rightarrow \{[l_1, h_1], \dots, [l_{|F \cap (P \times T)|}, h_{|F \cap (P \times T)|}]\}$ 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, (vii) $Z_0 = (M_0, D_0)$, where $M_0: P \rightarrow \{0, 1\}$ is the initial marking and $D_0: P \rightarrow \{0\}$ is the initial clock position.

2.2.4 Tools Modeling Petri Nets

Several tools already exist to model and/or simulate Petri nets and their extensions. For example, CPN tools is a software for editing, simulating and analyzing Colored Petri Nets. It features a fast simulator that efficiently handles both untimed and timed nets. Full and partial state spaces can be generated and analyzed, and a standard state space report contains information such as boundedness properties and liveness properties [Ratzer 2003]. Petri.NET is another tool which allows modeling, simulation and real-time implementation of static and dynamic Petri nets. The results of a Petri net model simulation are presented to the user in the form of a token game and in the graphical form showing diagrams of a state vector [Genter 2007]. Nevertheless, neither CPN tools nor Petri.NET can support R-TNCES with their condition and event signals.

The VisualVerification (ViVe) toolset is a tool chain for automatic verification of distributed control systems. It allows creation and modification of model components in modelling language of Net Condition/Event Systems (NCES) [Suender 2011]. But, it does not deal with the notion of time in NCES and the reconfiguration feature they may have. The TNCES-Editor, developed at the Martin Luther University Halle-Wittenberg, allows the graphical modeling of all NCES based subtypes, including R-TNCES [Dubinin 2006]. To support interpretation and reachable state analysis, the TNCES-Editor offers an optional labeling of transitions. The whole net structure including the labels will be stored in a special file format (*.pnt) which can be used as an import file for the model-checker SESA. However, TNCES-Editor doesn't feature the simulation of a built R-TNCES, nor highlights the reconfiguration aspect of a DRCS.

2.2.5 Model Checking

Model-checking is a technique for automatically verifying the correctness properties of finite-state systems [Rouff 2012]. It is a general verification approach that is applicable to a wide range of applications such as embedded systems, software engineering, and hardware design. It also supports partial verification, i.e., properties can be checked individually, thus allowing focus on the essential properties first. Model-checking is a potential "push-button" technology; the use of model checking requires neither a high degree of user interaction nor a high degree of expertise. It can be also easily integrated in existing development cycles since its learning curve is not very steep, and empirical studies indicate that it may lead to shorter development times [Baier 2008].

Model checking for TNCES and R-TNCES is based on their reachability graphs. SESA [Starke 2002] is an effective software environment for the analysis of TNCES, which computes the set of reachable states exactly.

Typical properties which can be verified are boundedness of places, liveness of transitions, and reachability of states. In addition, temporal/functional properties based on Computation Tree Logic (CTL) specified by users can be checked manually. Nevertheless, R-TNCES lacks of an environment to edit, simulate and verify its models. It cannot model systems with adaptive shared resources neither.

2.2.6 Temporal Logic

The Computation Tree Logic (CTL) offers facilities for the specification of properties to fulfill by the system behavior [Roch 2000a, Roch 2000b]. In this section, we present this logic and two of its extensions: Extended Computation Tree Logic (denoted by ECTL) and the Timed Computation Tree Logic (denoted by TCTL).

Computation Tree Logic

In CTL, all formulae specify behaviors of the system starting from an assigned state in which the formula is evaluated by taking paths (e.g. sequence of states) into account. The semantics of formulae is dened with respect to a reachability graph where states and paths are used for the evaluation. A reachability graph M consists of all global states that the system can reach from a given initial state. It is formally dened as a tuple $M = [Z, E]$ where

- Z is a finite set of states,
- E is a finite set of transitions between states, e.g. a set of edges (z, z') , such that $z, z' \in Z$ and z' is reachable from z .

In CTL, paths play a key role in the denition and evaluation of formulae. A path denoted by (z_i) starting from the state z_0 is a sequence of states, $(z_i) = z_0 z_1 \dots$ such that $\forall j \in \mathbb{N}$, there is an edge $(z_j, z_{j+1}) \in E$. The truth value of a CTL formula is evaluated with respect to a certain state of the reachability graph. Let $z_0 \in Z$ be a state of the reachability graph and φ be a CTL formula. The relation $z_0 \models \varphi$ means that the CTL formula φ is satisfied in the state z_0 . Then the relation \models for a CTL formula is defined as follows:

- $z_0 \models EF\varphi$, if there is a path (z_i) and $j > 0$ such that $z_j \models \varphi$,
- $z_0 \models AF\varphi$, if for all paths (z_i) , there exists $j > 0$ such that $z_j \models \varphi$,
- $z_0 \models AG\varphi$, if for all paths (z_i) and for all $j > 0$, it holds $z_j \models \varphi$.

Extended Computation Tree Logic

In CTL, it is rather complicated to refer to information contained in certain transitions between states of a reachability graph. A solution is given in [Roch 2000a, Roch 2000b] for this problem by proposing an extension of CTL called Extended Computation Tree Logic ECTL. A transition formula is introduced in ECTL to refer to a transition information contained in the edges of the reachability graph. Since it is wanted to refer not only to state information but also to steps between states, the structure of the reachability graph $M = [Z, E]$ is changed as follows:

- Z is a finite set of states,
- E is a finite set of transitions between states, e.g. a set of labeled edges (z, s, z') , such that $z, z' \in Z$ and z' is reachable from z by executing the step s .

Let $z_0 \in Z$ be a state of the reachability graph, τ a transition formula and φ an ECTL formula. The relation \models for ECTL formulae is defined inductively:

- $z_0 \models E\tau X\varphi$: iff there exists a successor state z_1 such that there is an edge $(z_0, s, z_1) \in E$ where $(z_0, s, z_1) \models \tau$ and $z_1 \models \varphi$ holds,
- $z_0 \models A\tau X\varphi$: iff $z_1 \models \varphi$ holds for all successors states z_1 with an edge $(z_0, s, z_1) \in E$ such that $(z_0, s, z_1) \models \tau$ holds.

Timed Computation Tree Logic

TCTL is an extension of CTL to model qualitative temporal assertions together with time constraints. The extension essentially consists in attaching a time bound to the modalities and we note that a good survey can be found in [Alur 1991]. For a reachability graph $M = [Z, E]$, the state delay D is defined as a mapping $D : Z \rightarrow N_0$ and for any state $z = [m, u]$ the number $D(z)$ is the number of time units which have to elapse at z before firing any transition from this state. For any path (z_i) and any state $z \in Z$ we put:

- $D[(z_i, z)] = 0$, if $z_0 = z$,
- $D[(z_i, z)] = D(z_0) + D(z_1) + \dots + D(z_{k-1})$, if $z_k = z$ and $z_0, \dots, z_{k-1} \neq z$.

In other words, $D[(z_i, z)]$ is the number of time units after which the state z on the path (z_i) is reached the first time, e.g. the minimal time distance from z_0 .

Let $z_0 \in Z$ be a state of the reachability graph and φ a TCTL formula. The relation \models for TCTL is defined as follows:

- $z_0 \models EF[l, h]\varphi$, iff there is a path (z_i) and a $j > 0$ such that $z_j \models \varphi$ and $l \leq D((z_i), z_j) \leq h$,
- $z_0 \models AF[l, h]\varphi$, iff for all paths (z_i) there is a $j > 0$ such that $z_j \models \varphi$ and $l \leq D((z_i), z_j) \leq h$.

2.2.7 Priority Ceiling Protocol

The Priority Ceiling Protocol (PCP) [Goodenough 1988] in real-time computing is a synchronization protocol for shared resources to avoid unbounded priority inversion and mutual deadlock due to wrong nesting of critical sections. In this protocol, each resource R is assigned a priority ceiling $Cl(R)$, which is equal to the highest priority of the tasks that may lock it. A task can acquire a resource only if the resource is free and has a higher priority than the priority ceiling of the rest resources in lock by other tasks.

Let us assume a system to be composed of the tasks T_1, T_2, T_3 and T_4 (having respectively the increasing priorities 1, 2, 3 and 4) and two resources R and Q : R can be used by T_1 and T_2 and Q by T_1 and T_4 . Then, $Cl(R)=2$ and $Cl(Q)=4$. Thus, T_2 is blocked if it tries to block R which is free when Q is locked.

2.2.8 Image Processing

One of image processing techniques is image matching which consists in comparing images in order to obtain a measure of their similarity. It extracts invariant local features for all images, and then uses voting to rank the database images in similarity with the query image [Grauman 2005a]. Another technique is contour comparison. It consists in detecting an image contour by quantifying the presence of a boundary at a given image location through local measurements [Arbelaez 2011]. The said two techniques are used by the middleware of our reconfigurable system, BROS.

For image processing, a C++ library named OpenCV is used. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics [Bradski 2008]. OpenCV has already used in medical image processing and it gained the spurs of the community working on this field [Intel 2012, Nolden 2013, Membarth 2012].

2.2.9 UML Profiles

The idea of being able to, more or less automatically and systematically, verify and validate UML-based models has been around for a while, so there is a rather large body of literature on the topic. Thus, a number of UML

profiles are proposed for embedded and real-time systems. The authors in [Martin 2001] propose a profile for specification, design and verification of embedded real-time systems. It takes advantage of the best concepts of UML, Real-Time UML, functional-architecture co-design and platform based design. Nevertheless, the profile features concurrency, communication and implementation issues. The profile proposed in [Selic 1998], UML-RT, is a complete modeling language which allows to model complex and event-driven real-time systems. However, it does not support time and time constraints modeling [Gherbi 2006] and has limited modeling capabilities for performance and architecture [Staines 2007]. Another UML standard profile, SPT, permits to model scheduling, performance and time of embedded real-time systems [Group 2005], but it has shortcomings in expressive power, flexibility and reconfiguration [Gérard 2010]. SPT was replaced, then, with another UML standard profile, MARTE [Shousha 2012], which is used for modeling and analysis of real-time and embedded systems [Mallet 2009]. It extends UML 2.0 to cover the aspects of time and hardware and software resources. UML MARTE can not be used, however, for modeling of dynamic composition as it only provides the static and predefined set of configuration [Hamid 2010].

2.2.10 Formalism Transformation

Several works have been recently proposed to automatically map UML models into performance models, such as queueing networks [Smith 2005, D'Ambrogio 2005], Petri nets [Bernardi 2007, Petriu 2007] and simulation [Marzolla 2004].

For example, the authors in [Merseguer 2002] propose a two-step correspondence between UML models and labeled generalized stochastic Petri nets (LGSPN): First, each UML state machine diagram describing the software architecture behavioral is independently converted into the corresponding LGSPN; secondly, the different obtained LGSPNs are joined according to the information 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 [Woodside 2005] propose another approach using LGSPN, which was lately extended in [Petriu 2007]. The proposition specifies a performance metamodel, named Core Scenario Model (CSM), as an intermediate model. The latter uses deployment diagrams to represent the software architecture structure. UML behavioral diagrams (activity diagrams and/or sequence diagrams) are used to describe the software architecture behavior and the related performance specifications, endowed with special tags and stereotypes. The CSM can be mapped into several kinds of performance models, such as Petri nets, queueing network and simulative models.

The work in [Gu 2005] uses an algorithm based on XML algebra to

transform UML models in XML format and, then, to the corresponding layered queueing network. This work extends the results of [Petriu 2002] which propose adopting the layered queueing networks as the target 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, annotated with performance information.

A similar and valuable approach is proposed in [D'Ambrogio 2005]. It uses the the Metaobject Facility (MOF) [ISO 2005] metamodeling driven architecture approach. Deployment and activity diagrams are, thus, generated. The work in [Marzolla 2004] use case diagrams, detailed through activity diagrams, to describe the software architecture behavior. A UML performance is proposed to transform UML diagrams into a discrete-event simulation model.

2.2.11 Robot Operating System

Robot Operating System (ROS) is a meta-operating system framework that facilitates robotic system development. It is widely used in various fields of robotics including industrial robotics, UAV swarms and low-power image processing devices [Quigley 2009].

It provides services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes and package management. Robotic Operating System features tools to develop distributed robot applications composed of with a network of processes called nodes that are coupled using the ROS communication infrastructure. ROS has become a de-facto standard in robotics. Due to its extensive use by robot developers, ROS has a wide repository of software components such as sensor drivers, visualization tools, navigation systems, arm manipulation systems or artificial vision algorithms [Koubâa 2016]. Besides, ROS is being actively supported by the Open Source Robotics Foundation [sit a]. Based on all those features, ROS was selected for the developments presented in this paper.

ROS enables the deployment of a network of interacting ROS nodes, that communicate using the ROS middleware infrastructure. A ROS application is a packaged set of ROS nodes that communicate through a ROS Master, where a ROS Master is a single discovery and communications broker that facilitates node-node flow setup [Kumar 2015].

ROS is not an operating system in the traditional sense of process management and scheduling; rather, it provides a structured communications layer above the host operating systems of a heterogenous compute cluster. The philosophical goals of ROS can be summarized as: peer-to-peer, tools-

based, multi-lingual, thin, free and open-source [Quigley 2009]. ROS enables software developers to create robot applications by providing libraries for hardware abstraction, device drivers, visualizers, and many more [Baumgartl 2013].

Robot Operating System (ROS) is an open-source software development framework dedicated to robots.

2.3 Medical State of the Art

This section exposes three different classifications of supracondylar humeral fracture and justifies the adopted one. We give then a report on the conduct of a conventional treatment of the SCH fracture after attending a real surgical intervention.

2.3.1 Supracondylar Humeral Fracture's Classifications

Many classifications of the supracondylar humeral fractures were established. They are based on both the direction and the degree of displacement of the distal fragment [Barton 2001]. The Gartland's classification system and the Lagrange's are the most widely used. In the English literature, the first is the most commonly used: the Gartland's classification is based on the lateral radiography and fractures are classified, as illustrated in Figure 2.1, according to a simple three-type system (Table 2.1) [Pirone 1988]. Lagrange's classification is the most widely used in the French literature. It divides these fractures into four types on the basis of antero-posterior and lateral radiography [Lagrange 1962] as illustrated in Figure 2.2.

These two classifications do not consider the rotary displacement, an important feature to assess the SCH fracture [Barton 2001, Kuo 2004]. As a consequence, the majority of authors overlook this important component of the displacement. Thus, a new classification, named Smida's classification, is proposed in [Douira-Khomsy 2012] to remedy this deficiency. This classification sorts the SCH fractures according to five types as illustrated in Figure 2.3 and detailed in Table 2.3. We adopt this classification in this paper.

Table 2.1: Gartland's classification of supracondylar fractures of the humerus.

Type	Radiologic characteristics
I	Undisplaced fractures
II	Displaced fracture with intact posterior hinge
III	Completely displaced fractures with no contact between the fragments

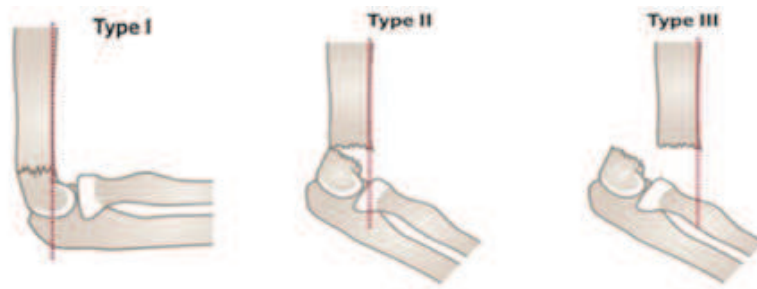


Figure 2.1: Gartland's classification of supracondylar fractures of the humerus.

Table 2.2: Lagrange's classification of supracondylar fractures of the humerus.

Type	Radiologic characteristics
I	Undisplaced fractures
II	Unidirectional displacement
III	Multidirectional displacement including posterior tilt, translation, rotation and coronal angulation. Contact between bone fragments is maintained
IV	Fracture with complete displacement without contact between the fragments



Figure 2.2: Lagrange's classification of supracondylar fractures of the humerus.

Table 2.3: Smida's classification of supracondylar fractures of the humerus.

Type	Radiologic characteristics
I	Undisplaced fracture with cracks in the anterior cortex of the two columns
IIA	Greenstick fracture with total anterior discontinuity
IIB	Total discontinuity in the lateral column
IIC	Total discontinuity in the medial column
III	Total displacement and loss of contact



Figure 2.3: Smida's classification of supracondylar fractures of the humerus.

2.3.2 Supracondylar Humeral Fracture Treatment

In this section, we expose the treatment which was performed on a true case of a patient presenting a supracondylar humeral fracture who came to the Children Hospital of Béchir Hamza (Tunis). The patient who is a ten-year-old girl fell on her outstretched right hand on November 12th 2013. After clinical examination and radiological diagnosis, the patient's elbow was immobilized in a plaster splint and the patient was admitted in the pediatric orthopedics department and operated on the same day. Radiographies showed a type III fracture according to Smida's classification as shown in Figure 2.4.

During the surgical intervention we attended, closed reduction of fracture and lateral percutaneous pinning were performed under general anesthesia and fluoroscopic control. The injured elbow was, then, placed under the fluoroscopic image intensifier (Figure 2.5). The fracture was reduced by external maneuvers: pulling gentle, longitudinal traction and correcting frontal displacement, flexing the elbow and pushing anteriorly on the olecranon, hyperflexing the elbow and confirming maintenance of coronal alignment. Reduction was controlled by the image intensifier and a total of

9 radioscopic images were taken. The elbow was immobilized once a satisfying reduction was achieved (Figure 2.6). As illustrated in Figure 2.7, two lateral and parallel smooth pins were then percutaneously inserted from the lateral condyle through the opposite cortical bone to stabilize the fracture. After the placement of the two pins, the second pin had to be removed and reinserted since it did not straightaway follow the right trajectory. In this step, 15 fluoroscopic images were taken. After placement, the pins were bent over and cut off outside the skin. A long arm cast was then applied at the elbow in approximately 90° of flexion.

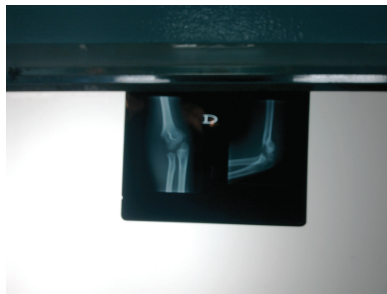


Figure 2.4: Fracture's radiographs. Figure 2.5: Injured elbow installed under the fluoroscopic image intensifier.



Figure 2.6: Elbow immobilization after obtaining fracture reduction. Figure 2.7: Lateral percutaneous pinning.

During this surgery, a total of 24 fluoroscopic images were taken, which involves high doses of radiation to the medical staff, especially since such interventions are performed 2 times per day on average.

2.4 Summary

For the better understanding of works of this dissertation, an IT state of the art introducing relevant elemental knowledge on Petri Nets-based for-

malisms, model checking technologies and other software technologies was recalled in this chapter. We presented, then, the different classifications of supracondylar humeral fracture and how it is currently treated.

Formal Modeling and Verification

Contents

3.1	Introduction	32
3.2	Contribution 1: PCP-based Solution for Resource Sharing in R-TNCES	32
3.2.1	Motivations	32
3.2.2	Case Study	33
3.2.3	Formalization	36
3.2.4	Modeling	37
3.2.5	Verification	43
3.3	Contribution 2: New Environment ZiZo	44
3.3.1	Motivation	44
3.3.2	Features	45
3.4	Application: Specification and Certification of BROS	45
3.4.1	Specification Step	45
3.4.2	Certification Step	49
3.5	Summary	52

3.1 Introduction

In this chapter, we propose to enrich the formalism Reconfigurable Timed Net Condition/Event Systems (R-TNCES) with the well-known protocol Priority Ceiling Protocol (PCP) in order to model and verify safety re-configuration scenarios. The contribution is applied first to a case study illustrating our contribution, and then to BROS to specify and certify the latter.

3.2 Contribution 1: PCP-based Solution for Resource Sharing in R-TNCES

Since R-TNCES is a useful formalism to model reconfigurable systems, we aim in this section to propose a new PCP-based extension to this formalism in order to check the system's safety after any reconfiguration scenario dealing with the addition or removal of resources.

3.2.1 Motivations

Reconfiguration is the qualitative change in the structure, functionality, and algorithms of the control systems. This is due to qualitative changes of goals of control, the controlled system or of the environment the system behaves within. Partial failures, breakdowns, or even human intervention may cause such changes. Thus, the development of Reconfigurable Distributed Control Systems (RDCS) is not an easy activity to perform since they should be adapted to their environment under functional and temporal constraints. These systems are distributed on networked devices which execute control tasks with shared resources. The RDCS's devices may actually share physical or logical resources (robots, material movement systems, machining centers, etc.) which may be added or removed according to user's requirements with specific evolution in the environment.

A reconfiguration scenario is assumed, in [Khalgui 2011], to be any addition, removal or up-date of control tasks to insure the required flexibility of the system and according to well-defined user's strategies. Nevertheless, this definition is incomplete since it does not consider the reconfiguration of resources. We propose, then, a new definition of reconfiguration to deal with the addition and removal of resources shared by tasks. We note that no one in our community proposed a same. To model and verify reconfigurable systems, the authors in [Zhang 2013] propose a new formalism named Reconfigurable Timed Net Condition/Event Systems (R-TNCES) extending the formalism TNCES [Hanisch 1997] and Petri nets to model the different behaviors of the system to be executed after different reconfiguration scenarios. Even though this formalism is original and useful, the authors

do not consider the case of shared resources to be adapted at run time. Nowadays, several research works are proposed to manage the problems of shared resources [Koh 1991, Zhou 1991]. Two well-known protocols have been selected in our community to encode the management of resources: Priority Inheritance Protocol (PIP) [Sha 1990] and Priority Ceiling Protocol (PCP) [Goodenough 1988]. The second protocol is better than the first since it avoids deadlock problems. Today, several research works are based on these two protocols, but no one consider them for the case of reconfigurable resources in DRCS. Since it is an expressive formalism for adaptive systems, we propose in this paper to enrich R-TNCES with the protocol PCP in order to model both tasks and resources. Our goal is to check that any reconfiguration of resources or tasks does not bring the whole architecture to a blocking situation.

3.2.2 Case Study

Let us assume a reconfigurable discrete event system to be composed of two tasks A and B . We suppose that these two tasks share initially the resources Q and R (as shown in Figure 3.1) before applying a reconfiguration scenario which will add a new resource S (to be used by both A and B). This case was not treated in any related work and forms a new problem dealing with reconfigurable resources. We suppose that B has the highest priority ($B > A$). We suppose that the system is safe before the reconfiguration scenarios. However, once the reconfiguration is applied, a deadlock certainly occurs according to Figure 3.2. In fact, A starts by using R before being interrupted by B due to the latter's higher priority. B is then blocked because it tries to lock R ($P(R)$) which is hold by A . A continues progressing until it frees R ($V(R)$) and B interrupts it. When B asks for Q ($P(Q)$), it is interrupted because Q is hold by A . A is lately blocked when it asks for S (hold by B). A deadlock occurs thus, because A is waiting for S while B for Q . Regarding to situation, we affirm that reconfiguration scenarios can bring the system to blocking situations. The latter can be met if we apply R-TNCES according to [Zhang 2013] to model the tasks A and B as illustrated in Figure 3.3. This is proven by applying the CTL formula $AG EX TRUE$ which turned out to be false .

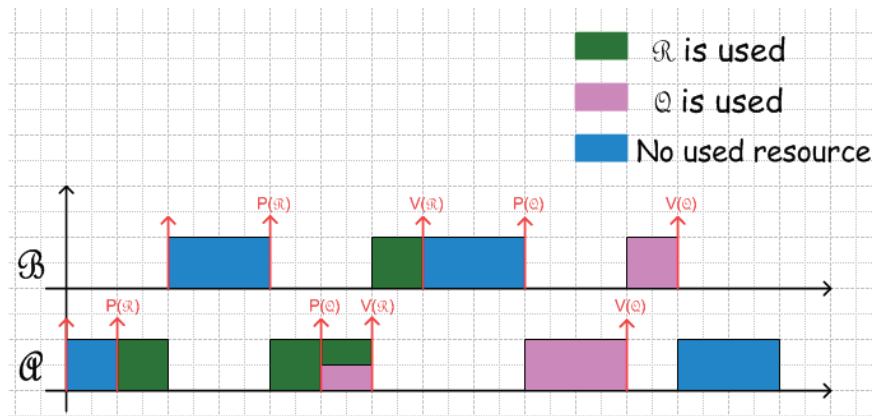


Figure 3.1: Behavior of A and B before a reconfiguration scenario

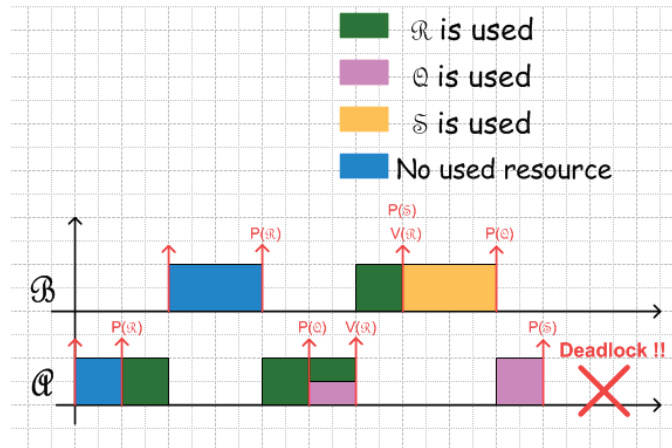


Figure 3.2: Behavior of A and B after a reconfiguration scenario

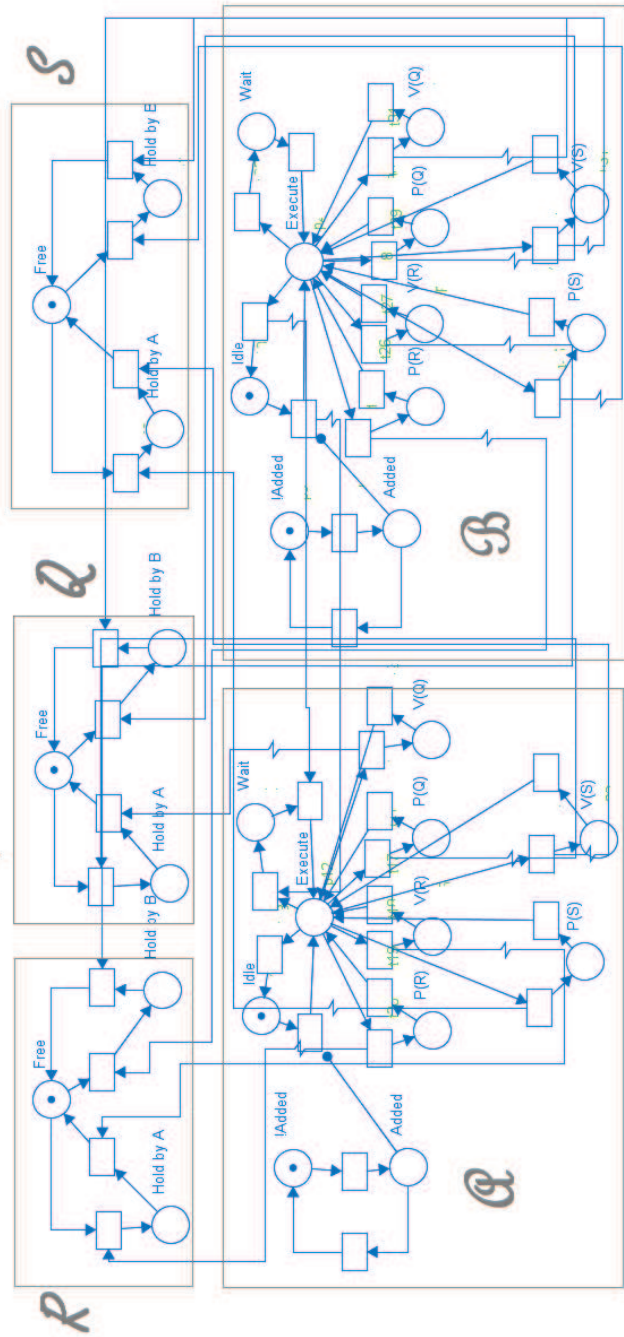


Figure 3.3: Illustrative example's R-TNCES

This is why we aim to check the safety of each scenario by enriching R-TNCES with the PCP protocol. We propose in the following sections new patterns to model reconfigurable discrete event systems according to R-TNCES by using PCP. This contribution is original since R-TNCES is an original formalism for reconfigurable systems, but lacks of useful mechanisms to manage shared resources. Let us remember the reader that in [Zhang 2013], the authors do not consider the eventual addition or removal of resources in the system.

3.2.3 Formalization

We present in this section the formalization of Distributed Reconfigurable Control Systems sharing resources.

DRCS

We assume a DRCS D to be composed of n_1 networked reconfigurable subsystems sharing n_2 resources. They extend the formalization of DRCS in [Zhang 2013] by adding the new set of resources as follows:

$$D = (\sum R - TNCES, \varpi, \sum M, \sum R) \quad (3.1)$$

where:

- $\sum R - TNCES$ is a set of n_1 R-TNCES,
- ϖ a virtual coordinator handling $\sum M$, a set of Judgment Matrices,
- $\sum R$, a set of n_2 shared resources.

Shared Resources

On the basis of PCP's definition and the flexibility expected from the DRCS, a resource R is defined as follows :

$$R = (Rec, S, Cl) \quad (3.2)$$

where:

- Rec (Reconfiguration) indicates whether R is added to the system / $Rec \in \{added, !added\}$,
- S indicates the state of R / $S \in \{free, hold_by_a_task_i\}$,
- Cl is used for the ceiling of R .

Tasks

Based on the expected reconfiguration of the system, we define a task T by:

$$T = (Rec, S) \tag{3.3}$$

where:

- Rec (Reconfiguration) indicates whether T is added to the system / $R \in \{added, !added\}$,
- S indicates the state of T / $S \in \{idle, execute, wait, P(R_i), V(R_i)\}$ and $P(R_i)$ means locking R and $V(R_i)$ unlocking it.

3.2.4 Modeling

We are interested in this research work in the reconfiguration of both tasks and shared resources. We propose, then, new solutions to introduce PCP in R-TNCES to avoid any blocking problem after reconfiguration scenarios. We propose an R-TNCES model for each resource of $\sum R$ and task of $\sum R - TNCES$.

Shared Resource

We model each shared resource by a R-TNCES as shown in Figure 3.4. The latter is composed of three TNCES modeling the resource’s reconfiguration (Rec), state (S) and ceiling (Cl). Here is the modeling of a resource R :

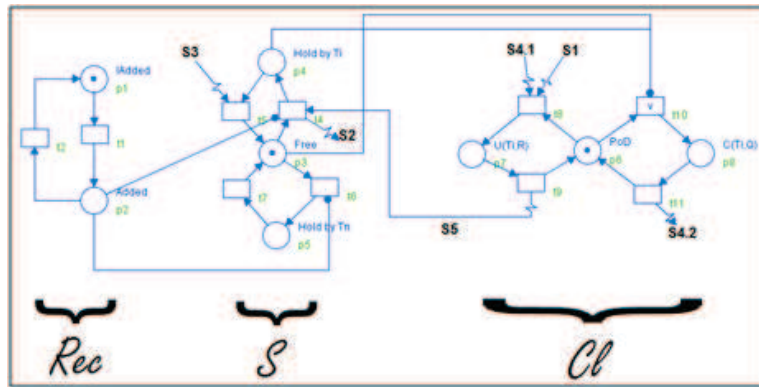


Figure 3.4: Shared resource’s modeling.

Let’s, now, dissect this R-TNCES into 3 TNCESs to analyze the structure of each one and understand the role of each arc and place. The first TNCES, named Rec and represented in Figure 3.5, models the reconfiguration of the resource R . Actually, a DRCS, as its name suggests, is reconfigurable and may work under different modes. It may need, depending on

the activated mode, the use of a set of resources. Thus, a resource may be used under some modes but not others. We suggest, therefore, that a resource may be added to a system or removed from it as required from the activated mode. That explains the fact that we have two places in the reconfigurable TNCES: *Added* and *!Added* (not added) respectively represented by $p1$ and $p2$. Two conditions leave the place "Added" to join transitions in the TNCES S (State). This is explained by the fact that a resource can not change status if it is not added to the system.

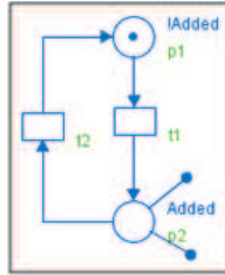


Figure 3.5: Shared resource's reconfiguration TNCES.

The second TNCES (illustrated in Figure 3.6), S , models the state of the resource R . A resource may be free or hold by a task T_i . Thus, we suggested the places *Free*, *Hold by T_i, \dots* and *Hold by T_n* (respectively represented by $p3$, $p4$ and $p5$) where R may be exclusively hold by a task from a set of n different tasks (n is an integer $\in [1, +\infty[$). The conditions leaving the different places are due to the fact that, according to PCP, a task T may hold a given resource if the latter is free and the resources hold by other tasks have a ceiling lower than T 's dynamic priority. We will see where those conditions are entering in the next TNCES. The different signals indicated on the figure above stand for:

- $S2$: a signal going to T_i and confirming the lock of R by it,
- $S3$: a signal from T_i asking to unlock R ,
- $S5$: A signal confirming that all the conditions required to lock R are met (based on PCP).

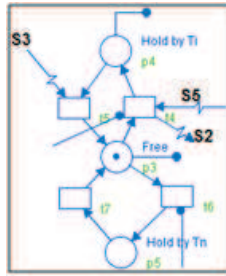


Figure 3.6: State TNCES.

The last TNCES (represented in Figure 3.7), Cl , models R 's ceiling. It contains a place named PoD (Point of Decision) and several places type $U(task, resource)$ and type $C(task, resource)$. The two last types stand for:

- $U(T_i, R)$: this place will allow the use of the resource R by the task T_i . To switch from PoD to $U(T_i, R)$, we need the two events $S1$ and $S4.1$. $S1$ is an event coming from the task T_i and asking to lock the resource R . $S4.1$ is actually the $S4.2$ leaving the transition following $C(T_i, R)$. The latter place will be drawn on the R-TNCES of the resource Q . $S5$ is the signal which we explained in the previous TNCES.
- $C(T_i, Q)$: this place controls the use of Q by T_i by checking the PCP's requirements. To reach the place $C(T_i, Q)$, we need, according to PCP, one of the two following conditions: the other resources, whose ceiling are higher than T_i 's priority, are free or hold by T_i (hence the OR sign in the transition preceding $C(T_i, R)$). $S4.2$ on the opposite figure is the $S4.1$ which will enter the transition preceding $U(T_i, Q)$ (drawn of the R-TNCES of the resource Q).

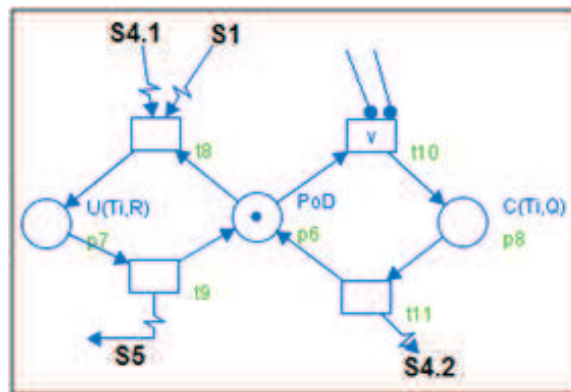


Figure 3.7: Ceiling TNCES.

Running example 1 in case study

In our case study (Section 3.2.2), the different resources have the same modeling since they have the same ceiling and are used by the same tasks. We choose to model the resource Q as shown in Figure 3.8.

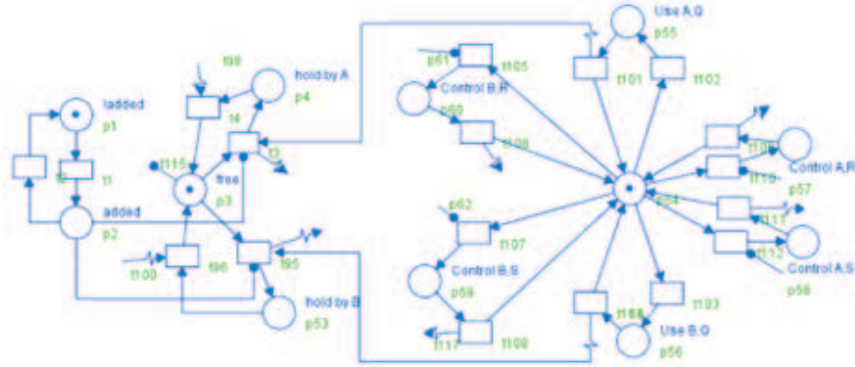


Figure 3.8: The resource Q 's modeling

The conditions entering the transition preceding $C(Task, Resource)$ are used to guarantee that, when a task T tries to lock a resource, all the other resources -whose ceilings are not lower than the task's priority- are free or hold by T . Thus, we avoid any eventual deadlock and see the relevance of the PCP.

Control Tasks

Each task T is modeled by a R-TNCES to be composed of two TNCESs as shown in Figure 3.9: the first one is illustrating its reconfiguration (Rec), the second its state (S).

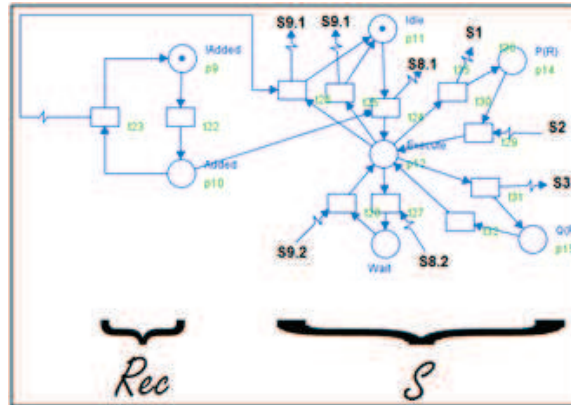


Figure 3.9: A task's modeling.

We unpack the R-TNCES in Figure 3.9 and analyze the functioning of its two TNCES. Due to the reconfigurable aspect of DRCS, a task may be added or removed to the system according to the activated mode. That's why the T 's reconfiguration TNCES (Rec), represented in Figure 3.10, contains two places: $Added$ and $!Added$ (respectively the places $p9$ and $p10$). Once T is removed, the event leaving the transition following $Added$ will force the task T to switch from the state $Execute$ to the state $Idle$. A condition leaving the place $Added$ will allow the switching from $Idle$ to $Execute$.

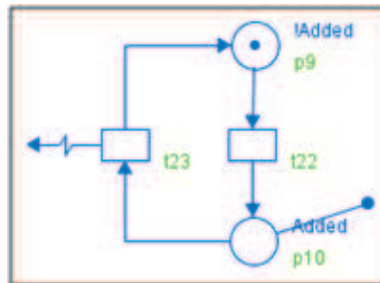


Figure 3.10: Task's reconfiguration TNCES.

The second TNCES (Figure 3.11) models the state of the task T . It contains the following places:

- $Idle$: as its names suggests, the task is idle,
- $Execute$: the task is running,
- $Wait$: the task was interrupted by another one, so it is waiting,
- $P(R)$: the task T is asking to lock a resource R ,
- $Q(R)$: the task T is unlocking the resource R .

The R-TNCES of a task T should include as many $P(R)$ and $Q(R)$ as the resources it may lock, but, in this illustrative example, we decided that T will use just one resource (R). The different signals indicated on the figure above stand for:

- $S8.1$: when T switches from *Idle* to *Execute*, the event $S8.1$ will force the running tasks with lower priorities to switch from *Execute* to *Wait*. T 's $S8.1$ is actually the $S8.2$ of tasks with lower priorities,
- $S9.1$: when T switches from *Execute* to *Idle*, the events $S9.1$ will force the waiting tasks with lower priorities to switch from *Wait* to *Execute*. T 's $S9.1$ are actually the $S9.2$ of tasks with lower priorities,
- $S1$, $S2$ and $S3$: refer to the last paragraph.

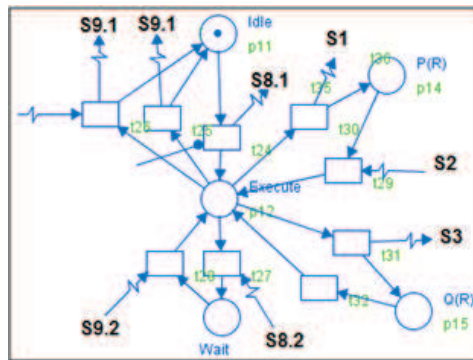


Figure 3.11: Task's state R-TNCES.

Running example 2 in case study

We continue our case study by modeling, here, the case study using R-TNCES and tasks and shared resources' modeling we previously proposed. Let's remember that B has a higher priority than A ($\text{priority}(A)=1$ and $\text{priority}(B)=2$), and Q , R and S are all shared by the two tasks ($\text{Cl}(Q) = \text{Cl}(R) = \text{Cl}(S) = \text{priority}(B) = 2$). We start by modeling the two tasks as following in Figure 3.12:

ties which can be verified are boundedness of places, liveness of transitions, and reachability of states. In addition, temporal/functional properties based on Computation Tree Logic (CTL) specified by users can be checked manually. The following e-CTL formula is applied:

$$AG \ EX \ true \quad (3.4)$$

This formula is proven to be true by SESA as shown in the screenshot in Figure 3.14, so there no deadlocks in our R-TNCES.

```
Parsing formula:
AG EX TRUE
The formula is TRUE.
```

Figure 3.14: Screenshot from SESA.

We also check the safety property by checking if a given resource may be simultaneously locked by two different tasks. The following CTL formula is checked:

$$EF \ p22 \ AND \ p23 \quad (3.5)$$

where: (i) $p22$ is the place translating that the resource R is locked by the task A , (ii) $p23$ means that B locks R . This formula is proven to be false as illustrated in Figure 3.15.

```
Parsing formula:
(EF p22 AND p23)
Current model checking options are:
  write a proof
  but only witnesses and counterexamples
  but only for the top level formula
  to the session file
.....Reset options? Y/N N
States:          53
The formula is FALSE.
```

Figure 3.15: Screenshot from SESA.

3.3 Contribution 2: New Environment ZiZo

We present in this section the new tool ZiZo and its usefulness in certifying distributed reconfigurable control systems. BROS is the case study.

3.3.1 Motivation

ZiZo is an R-TNCES modeling and random-simulating tool written in C# programming language for the Windows platform and developed in LISI laboratory of INSAT and eHTC (Tunisia). Its originality consists in featuring the simulation of a built R-TNCES and highlighting the reconfiguration

aspect of a DRCS. These features which are not offered in any other Petri Nets editor. The main window of ZiZo GUI comprises five dockable frames: Menu Bar, Model Arborecence, Place Properties, the Document Explorer and the Debug Window. Further details are provided in the official webpage of ZiZo [sit b].

3.3.2 Features

The originality of ZiZo consists in featuring the edition of an R-TNCES and highlighting the reconfiguration aspect of a distributed reconfigurable control system, which are not offered in any other Petri Nets editor. ZiZo is capable of

- creating several modules within the same model,
- interconnecting modules by input/output condition and event signals,
- randomly simulating the created model to detect any eventual dead-lock,(iv) storing the created model in a special file format (*.pnt),
- loading a created model to edit it and/or simulate it,
- exporting the model to the model-checker SESA [Starke 2002].

3.4 Application: Specification and Certification of BROS

We expose in this section the BROMETH's first two steps and their application on BROS. BROMETH can be applied when designing any medical robotic platform. BROS is used to prove the relevance of this methodology.

3.4.1 Specification Step

We detail hereafter how to perform the specification step when following BROMETH. The step in question is realized within two sub-steps as illustrated in Figure 3.16: constraints definition and modeling.

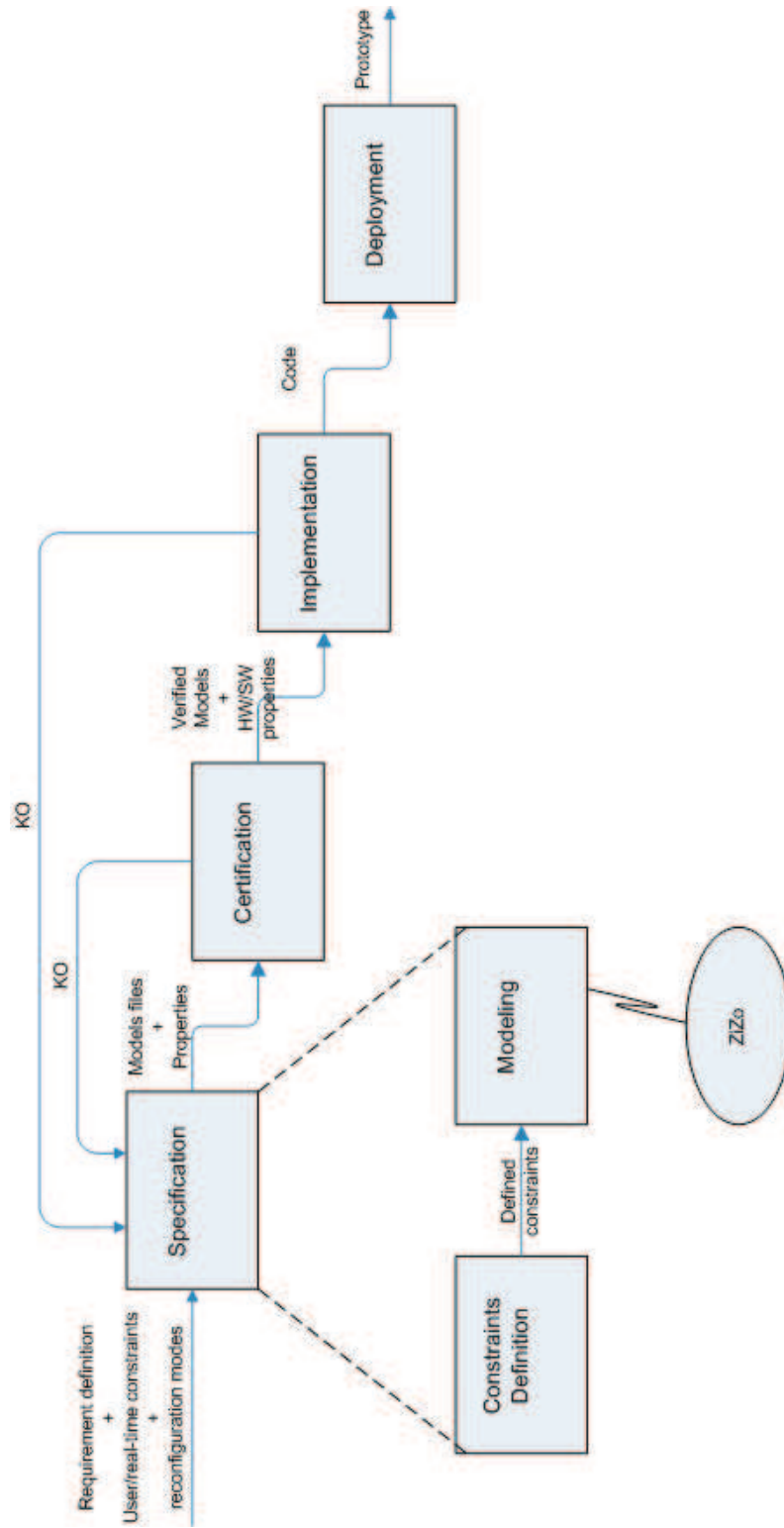


Figure 3.16: Specification step.

Constraints Definition

Several discussions between the medical and technical staffs defined the constraints controlling the functioning of BROS. And thus, to treat a humeral supracondylar fracture using BROS, the following steps will be performed in the automatic mode:

1. the surgeon launches the system and chooses one of the five operating modes;
2. CU asks MW about the fracture coordinates;
3. MW requests an image from BW and the latter sends it;
4. MW determines the different coordinates by image processing and sends them to CU;
5. based on the received coordinates, CU orders B-BROS1 to block the arm at the humerus;
6. B-BROS1 blocks the limb;
7. CU asks B-BROS2 to reduce the fracture based on the latter's line;
8. B-BROS2 reduces the fracture;
9. CU asks MW to ensure that the reduction was successful;
10. MW requests a new image from BW and checks the fracture reduction result. If it is satisfactory, BROS moves to step 11. Steps from 7 to 9 are repeated otherwise;
11. CU orders B-BROS2 to block the arm;
12. under the request of UC, P-BROS performs the first and the second pinning;
13. once the pinning is successful, CU asks B-BROS1 and B-BROS2 to unblock the limb.

The previously described steps delineate how BROS treats a SCH fracture when the automatic mode is triggered. As for the other modes, some steps will be manually performed as indicated in Section 1.2.2.

Modeling

Using ZiZo, we model the whole architecture of BROS with its different modules (UC, MW, BW, B-BROS1, B-BROS2, P-BROS and the surgeon) and shared resources. We obtain an R-TNCES model of 186 places and 283 transitions as shown in Figure 3.17.

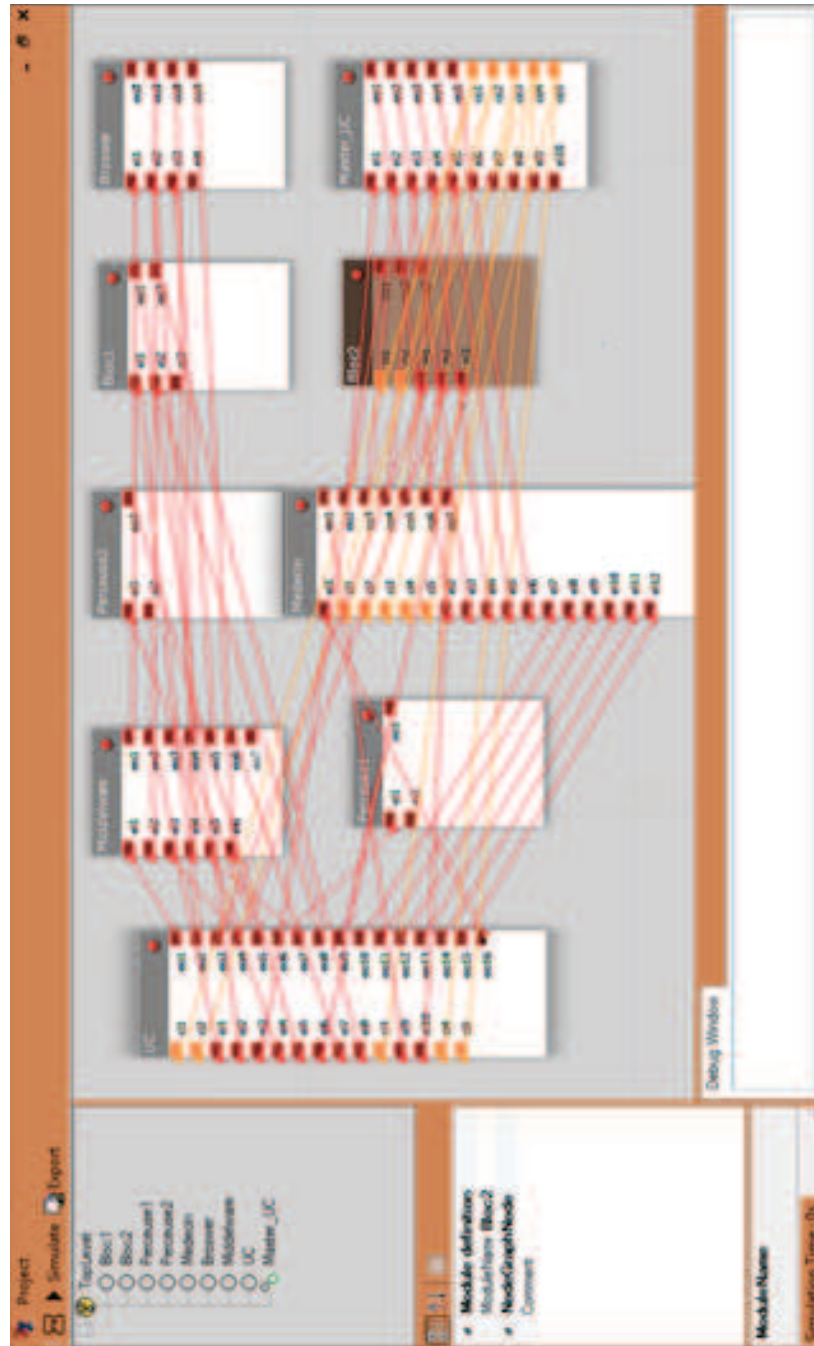


Figure 3.17: BROS modeling using ZiZo.

3.4.2 Certification Step

BROMETH's second step, the certification, takes as input model files (Specification's output) and properties written by the researcher based on the constraints defined during the specification step. Figure 3.18 shows that the certification is performed thanks to two sub-steps, simulation and formal verification. These are detailed hereafter. The purpose from the certification step is to obtain verified models.

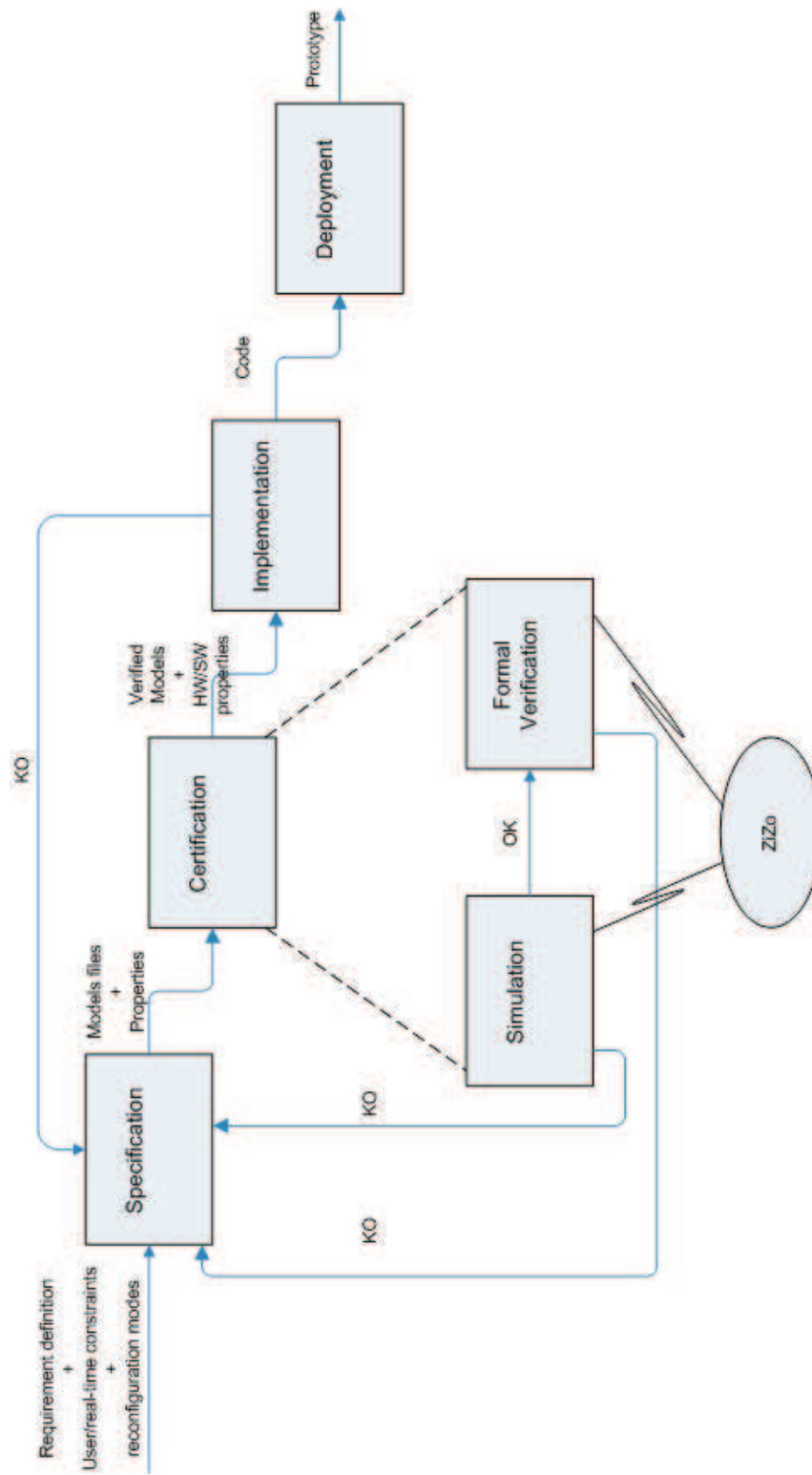
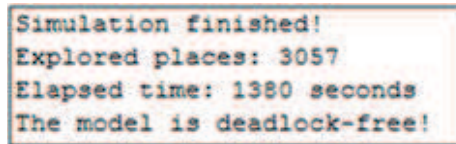


Figure 3.18: Certification step.

Simulation

Simulation is realized using ZiZo. Upon definition of the model, ZiZo simulates it. Simulation can be tracked by selection of a token game. Once simulation is finished, a report is displayed at the debug window. If a deadlock is detected, we go back to specification to do the modeling sub-step again.

Using ZiZo, we simulate the BROS modeling performed during the specification step. The obtained report displayed in Figure 3.19 proves that, after exploring 3057 places by ZiZo, our system is deadlock-free. Hence, we can move to the formal verification sub-step.



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

Figure 3.19: BROS’s simulation report.

Simulation is actually applied prior to model checking to get a first impression about the system’s behavior and early find out the eventual design errors. If the latter exists, formal verification may be time-consuming when performed on a large state-space.

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 [Starke 2002] 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.

We do in this section an exhaustive CTL-based verification to check the 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.

Blocking and Pinning

Pinning in the patient’s arm while moving it by unblocking it may lead to several postoperative complications. We check whether this two actions may be simultaneously performed by applying the following formula to BROS’s model:

$$EF p23 \text{ AND } p43 \quad (3.6)$$

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

Timeout Issue

We check that the whole surgical intervention does not last more than a given definite time. The medical partners agreed that it shouldn't last more than 301 seconds.

$$EF [0, 301] p23 \quad (3.7)$$

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:

$$\begin{aligned} AGA t18 X AFE t25 X AFE t40 X AFE t74 X AFE \\ t111 X TRUE \end{aligned} \quad (3.8)$$

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.

3.5 Summary

This chapter deals with the management of adaptive shared resources in reconfigurable systems: it is a new challenge in industry. We proposed a solution to this issue by providing a PCP-based R-TNCES models, an extension of Petri Nets. We define formal patterns allowing reconfigurations of a DRCS, where the first module deals with tasks and the second with resources. We apply a model checking for the verification of CTL-based functional properties in order to guarantee a safe behavior of this reconfigurable architecture. A new software tool, ZiZo, is also introduced in this chapter. We finally apply the contribution to BROS in order to perform the specification and certification steps of BROMETH.

New UML Profile: R-UML and Automatic Transformation to ROS

Contents

4.1	Introduction	54
4.2	Motivations	54
4.3	Running Example	56
4.4	R-UML Profile	57
4.4.1	Structure Modeling	58
4.4.2	Behavior Modeling	61
4.5	Transformation from R-UML to R-TNCES	66
4.5.1	R-Std Translation into R-TNCES	66
4.5.2	Verification	68
4.5.3	Implementation	73
4.6	Transformation from R-UML to ROS	74
4.6.1	State of the Art	74
4.6.2	Transformation Rules from R-UML to ROS	74
4.7	Application to BROS	79
4.8	Summary	82

4.1 Introduction

Unified Modeling Language (UML) is currently accepted as the standard for modeling software and control systems since it allows to concentrate on different aspects of the system under design. However, UML lacks formal semantics and, hence, it is not possible to apply, directly, mathematical techniques on UML models to verify them. UML does not feature explicit semantics to model flexible control systems sharing adaptive shared resources either. Thus, this chapter proposes a new UML profile, baptized R-UML (Reconfigurable UML), to model such reconfigurable systems. The profile is enriched with a PCP-based solution for the management of resource sharing. This chapter also presents an automatic translation of R-UML into R-TNCES, a Petri Net-based formalism, to support model checking.

4.2 Motivations

The Unified Modeling Language (UML) is a semi formal language developed by the Object Management Group to specify, visualize and document models of both software and non-software systems. Driven by software engineering industries, it became well developed and supported with dozens of tools [Bahill 2003]. UML provides two types of diagrams to create a specific profile for a given system: structural and behavioral. The first is designed to visualize and document the static aspects of systems, while the second aims at visualizing the dynamic aspects [Warmer 1998]. UML has unquestionable advantages as a technique for visual modeling, nevertheless, it does not guarantee that the generated models are correct. Actually, no step of system development, including the modeling one, is spared from human errors. Consequently, the cost to detect and remove such defects considerably increases through the system development [Fenton 1999].

The idea of being able to, more or less automatically and systematically, verify and validate UML-based models has been around for a while, so there is a rather large body of literature on the topic. For example, the authors in [Lilius 1999] use statecharts and sequence diagrams in a combined manner to check temporal logic formulas over a statechart-based description of the system, and the model checker produces, then, counterexamples through sequence diagrams. Another approach is described in [Cardoso 2001] where sequence diagrams are formally translated into Petri nets, based on the UML collaborations package metamodel. The authors check the correctness of the sequence diagrams through the resulting Petri nets. A work described in [Cortellessa 2000] uses the sequence diagram in conjunction with use cases and deployment diagrams to obtain queuing network models for performance evaluation. An execution graph from the sequence diagram is later obtained thanks to a given algorithm. Another work reported

in [Mikk 1998] translated Statecharts into PROMELA, the input language of SPIN verification system, whereas [Lam 2007] formally analyzed activity diagrams using NuSMV model checker to determine the correctness of activity diagrams. The authors in [King 1999] produce Petri net models starting from UML diagrams, however, they only describe the methodology at an intuitive level, through an example and no translation procedure is described. The work described in [Bondavalli 1999] proposed new UML stereotypes to enrich UML diagrams with dependability aspects. The purpose is to exploit the latter to build generally distributed stochastic Petri net models. The authors didn't focus on an automatic translation, but rather on detecting the dependability aspects from the UML diagrams.

We see in the previous related works that no one of our community was interested in modeling the reconfiguration aspect which is featured by many control systems and their shared resources. Nevertheless, reconfiguration has become, nowadays, a crucial feature to consider when designing new embedded systems. It is actually the ability to dynamically improve the latter's performance and quality of service at run-time, according to well defined conditions [Salem 2015b]. Increasing safety constraints and growing expected flexibility pushed developers to focus on designing systems that are able to fit their environment and shifting user requirements under functional and temporal constraints [Salem 2015a]. In this work, a reconfiguration scenario is assumed to be any run-time automatic operation that modifies the system's structure by adding or removing tasks or resources according to user requirements in order to adapt the whole architecture to its environment as mentioned in Section 3.2.1.

Hence, we propose, in this work, a new UML profile, baptized R-UML (Reconfigurable UML), endowed with a formal semantics enabling UML to model flexible control systems sharing adaptive shared resources. R-UML relies on UML's extensibility mechanisms to enhance class and statecharts diagrams, respectively called R-CD and R-StD henceforth. The latter are extended to support Priority Ceiling Protocol (PCP). It was proved in Chapter 3 the relevance of this protocol to solve the issue of concurrent access to adaptive shared resources in reconfigurable control systems. We propose then a new solution to translate R-UML into Reconfigurable Timed Net Condition/Event Systems (R-TNCES). An application of formal verification is, then, performed and aims to (dis)prove certain properties of the system using a formal model. This contribution is original since R-TNCES is a new and original formalism for reconfigurable systems, and no one in our community worked on the translation of UML into R-TNCES to combine their respective assets, i.e. the easiness and relevance of UML for visual modeling and the formal semantics of R-TNCES to verify and validate models.

4.3 Running Example

Let us assume a reconfigurable discrete event system to be composed of two tasks A and B . We suppose that these two tasks share initially the resources Q and R (as shown in Figure 4.1) before applying a reconfiguration scenario which will add a new resource S (to be used by both A and B). This case was not treated in any related work and forms a new problem dealing with reconfigurable resources. We suppose that B has the highest priority ($B > A$). We suppose that the system is safe before the reconfiguration scenarios. But, once the reconfiguration is applied, a deadlock certainly occurs according to Figure 4.2. In fact, A starts by using R and then S before being interrupted by B due to the latter's higher priority. B is then blocked because it tries to lock R ($P(R)$) which is still hold by A . A continues progressing until it frees R ($V(R)$) and B interrupts it. When B asks for S ($P(S)$), it is interrupted because S is hold by A . A is in its turn blocked because it is asking for Q which is hold by B . A deadlock occurs thus, because A is waiting for Q while B for S . Regarding to this situation, we apply the PCP on this running example which solves the deadlock issue as illustrated in Figure 4.3.

This running example features two tasks sharing three adaptive resources like in [Salem 2014], however, the tasks' behavior and the reconfiguration scenario are different. Besides, in [Salem 2014], a deadlock occurs because A is waiting for S while B for Q , whereas, in this work, it occurs because A is waiting for Q while B for S .

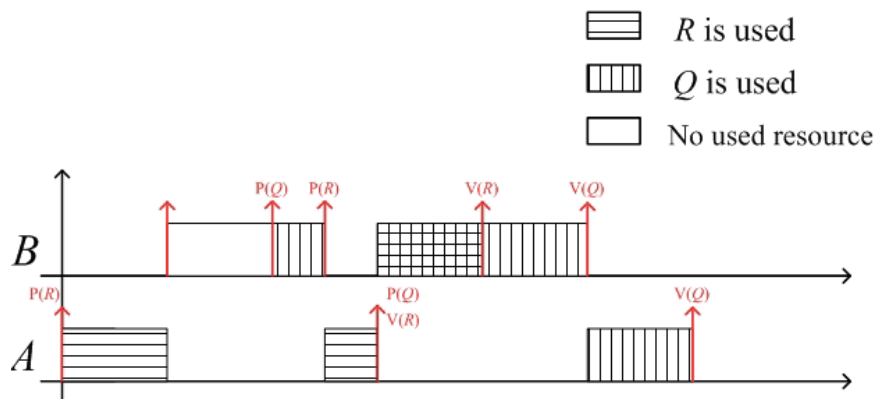


Figure 4.1: Behavior of A and B before a reconfiguration scenario.

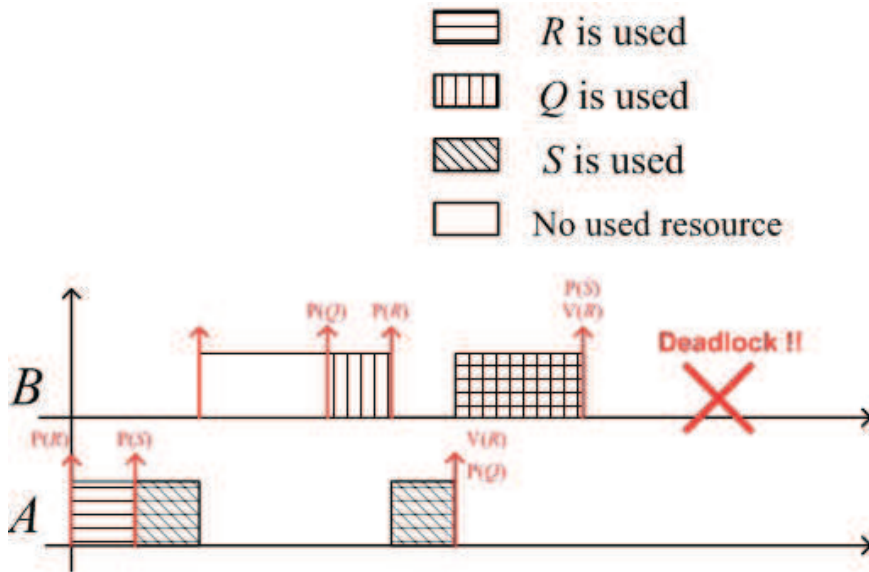


Figure 4.2: Behavior of A and B after a reconfiguration scenario.

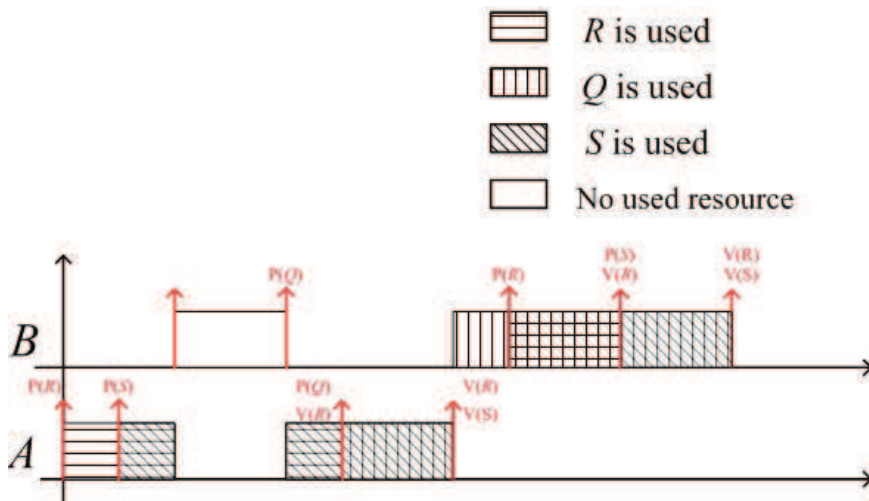


Figure 4.3: A and B behaviors after using PCP.

4.4 R-UML Profile

In this section, we define how to model the structure and the behavior of a flexible control system using R-UML. The contribution is applied on the running example of Section 4.3.

4.4.1 Structure Modeling

UML provides the class diagram to show the logical structure of a system. This diagram highlights conceptual connections showing the relations between the system's modules or components, each of which having its distinctive properties defined by a class. It is possible to extend the core semantics of UML and express new properties by using stereotypes. The latter is a mechanism to categorize an element. Thus, we extend the contribution proposed in [Lobov 2005] and define the following eight stereotypes of the class's attribute:

- `<< input >>`: the given attribute is a system input;
- `<< output >>`: the given attribute is a system output;
- `<< in >>`: the given attribute is a system module input;
- `<< out >>`: the given attribute is a system module output;
- `<< eventInput >>`: the given attribute is a system module event input;
- `<< eventOutput >>`: the given attribute is a system module event output;
- `<< integer >>`: the given attribute is an integer;
- `<< boolean >>`: the given attribute is a boolean attributed which can be evaluated to TRUE or FALSE.

The description above distinguishes between *system* and *module*. *System* denotes the whole system under control, whereas *module* a part of the system. A system may actually have internal connections between the modules specified by means of the stereotypes `<< in >>` and `<< out >>`, and a module may provide to the controller the connections that are specified by means of the stereotypes `<< input >>` and `<< output >>`. Two system modules may also be interconnected by an event which is an action which occurrence may be detected by another module in the system. An event is different from an input/output, since the first is just a signal informing that a certain action took place. The `<< eventInput >>` and `<< eventOutput >>` stereotypes respectively represent the event inputs and outputs that a module may have.

The information provided by a class diagram can be formally written as a tuple:

$$CID = (C, A, M, S, \alpha, \beta) \quad (4.1)$$

where:

- $C = \{cl_1, cl_2, \dots, cl_n\}$ is a finite set of classes in a class diagram CID;
- $A = \{attr_1, attr_2, \dots, attr_n\}$ is a finite set of attributes that belong to the classes;
- $M = \{setInput, resetInput, setOutput, resetOutput, setCeiling\}$ is a finite set of methods of the classes;
- S is a finite set of stereotypes / $S = \{\ll in \gg, \ll out \gg, \ll input \gg, \ll output \gg, \ll eventInput \gg, \ll eventOutput \gg, \ll integer \gg, \ll boolean \gg\}$;
- $\alpha : st_i \rightarrow attr_j$ is a function which maps the stereotype st_i from S to the $attr_j$ from A ;
- $\beta : attr_i \rightarrow cl_j$ is a function which maps attribute to the class.

According to the previous class diagram definition, we create two classes to model the running example of Section 4.3:

- a class named *Task* to model, as its name suggests, the different tasks of the system,
- a second class, named *Resource*, to model the different reconfigurable shared resources. We instantiate for each task or resource an object from the corresponding class.

The class *Task*, as showed in Figure 4.4, has an integer-stereotyped attribute, named *priority*, translating the task's priority. It also has a boolean-stereotyped one, *added*, indicating whether the task is added to the system (*added=TRUE*) or not (*added=FALSE*), depending on the applied reconfiguration scenario. The Figure 4.5 shows that the class *Resource* features an integer-stereotyped attribute, named ***ceiling***, translating the ceiling that each resource has according to PCP definition in Section 2.2.7. The class also features a method named *setCeiling* that recompute a resource's ceiling after applying a reconfiguration scenario. This method's code will be detailed later. Just as tasks, resources have a boolean-stereotyped attribute, *added*, because a reconfiguration scenario may add or remove a task or a resource [Salem 2014].

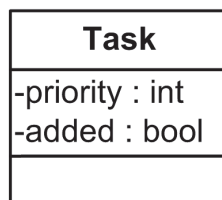


Figure 4.4: Task class.

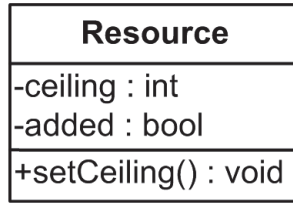


Figure 4.5: Resource class.

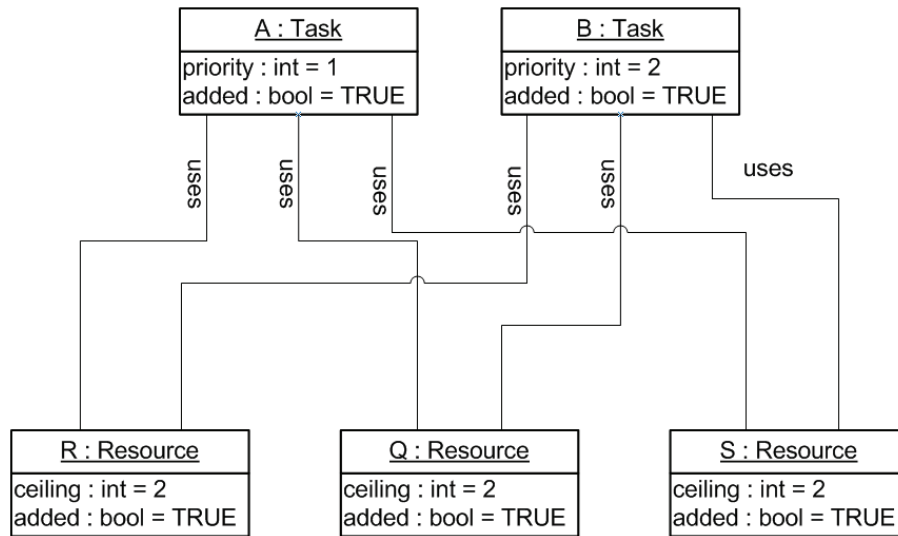


Figure 4.6: Running example’s object diagram.

Running Example 1

The static description of a system is often made through the class diagram. This simplifies the modeling by synthesizing the common characteristics and covering a large number of objects. However, it is sometimes useful or even necessary to add an object diagram. The latter allows, depending on the situation, to illustrate the class diagram (showing an example that explains the model), clarify certain aspects of the system (by highlighting imperceptible details in the class diagram), express an exception (by modeling specific cases of non-generalizable knowledge) or take an image (snapshot) of a system at a given time. The class diagram models the rules, whereas the object diagram models facts. Often the class diagram is a model to instantiate the binders in order to obtain the object diagram [Rumbaugh 1991]. Thus, we propose here to realize the object diagram of the running example described in Section

4.3. The said diagram illustrated in Figure 4.6 features two objects of the Task class (modeling the tasks A and B) and three of the Resource class (modeling the resources R , Q and S) while highlighting the links between them.

4.4.2 Behavior Modeling

UML features the State diagram as powerful tool to represent the behavior of an object which is the implementation of a particular class. We define for the system or its components a set of states which they may take. Each state is distinguished by its name. The change of the states is represented via transitions. The latter specify the laws that cause the change of the state and the consequences of the change. The rules which fire transitions may be expressed by event and guard which is a boolean expression that has to be evaluated to TRUE to fire the transition. A given transition may be fired through three manners: an event (if a certain action took place somewhere in the system), a guard (if the certain properties are assigned with the particular values) or combination of both. The different states are interconnected by transitions which determine the rules that cause transition to fire and the consequences of a transition's firing. Events, guards and the combination of both specify these rules. A time event, after (n) where n is a positive integer, is also used to specify that n time units should elapse before the transition may fire. Events may also be specified by `<< eventInput >>` or `<< eventOutput >>` stereotyped attributes. A transition firing may be accompanied by the activation of an action which can modify some properties of the system. This activation may call attribute-modifying methods defined in the classes, such as `setInput`, `resetInput`, `setOutput`, `resetOutput` and `setCeiling`.

We extend the contribution proposed in [Lobov 2005] and formally represent a state diagram by the tuple:

$$StD = (St, Tr, Ev, G, Ac, \gamma, \delta, \epsilon, \zeta) \quad (4.2)$$

where:

- $St = \{st_1, st_2, \dots, st_n\}$ is a finite set of states in a state diagram StD ;
- $Tr = \{tr_1, tr_2, \dots, tr_m\}$ is a finite state of transitions in a state diagram StD ;
- Ev is a finite set of events in transitions of StD ;
- G is a finite set of the guards in StD ;
- Ac is a finite set of actions;

- $\gamma : ev_i \rightarrow tr_j$ is a function that maps the event ev_i of Ev to the transition tr_j of Tr ;
- $\delta : gr_k \rightarrow tr_j$ is a function which maps the guard gr_k of Gr to the transition tr_j of Tr ;
- $\epsilon : act_l \rightarrow tr_j$ is a function which maps the action act_l of Ac to the transition tr_j of Tr ;
- $\zeta : tr_j \rightarrow \{st_b, st_e\}$ is a function which maps transition tr_j of Tr to the 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.

According to the reconfiguration feature expected from the system, we define a reconfigurable state diagram as a structure:

$$R - StD = (B, R) \quad (4.3)$$

where:

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

A reconfiguration function r_i makes the necessary changes to the system after a reconfiguration scenario in accordance with the definition given in Section 4.2. Hence, we define r as the structure:

$$r = (\eta, \theta, \iota) \quad (4.4)$$

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_j is added to the system and $\theta(res_j) = 0$ otherwise;
- $\iota : (res_j, t_i) \rightarrow \{0, 1\}$, $\iota(res_j, t_i) = 1$ if res_j is used by t_i in this triggered reconfiguration scenario, $\iota(res_j, t_i) = 0$ otherwise.

According to the previous definitions, we define in this section the Resource class's method, *setCeiling*, as follows:

```

if  $\theta(res) == 1$ 
  for i:=1 to |Tasks|
    if  $\eta(t_i) == 1$  AND  $\iota(res, t_i) == 1$ 
      AND  $t_i.priority > res.ceiling$ 
         $res.ceiling := t_i.priority$ 

```

We propose, then, R-StD diagrams to model a task and a resource on the basis of PCP definition and the reconfiguration feature expected from the system. Thus, we propose the R-StD illustrated in Figure 4.7 to model a reconfigurable shared resource:

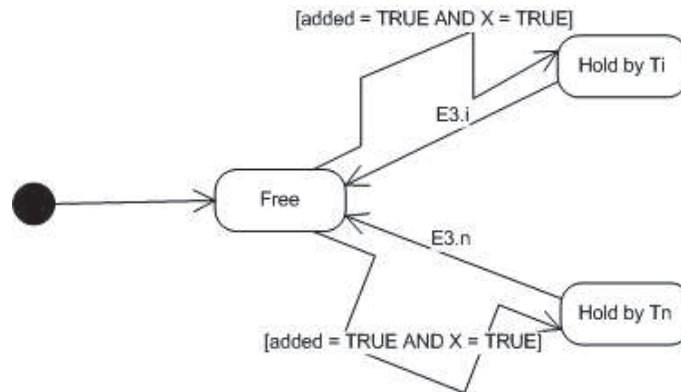


Figure 4.7: Shared resource's R-StD.

A resource may actually be free or hold by a task T_i . Thus, we propose the states "Free", "Hold by T_i " and "Hold by T_n " where R may be exclusively hold by a task from a set of n different tasks (n is an integer $\in (1, +\infty)$). The guards associated to the transitions leaving the state *Free* guarantee the respect of PCP rules before locking a resource, i.e. a task T may hold a given resource if, first, the latter is free and, secondly, the resources hold by other tasks have a ceiling lower than T 's dynamic priority, a condition verified by the guard named X . $E3.i$ is an event coming from T_i and asking to unlock R.

We propose, then, a second R-StD, illustrated in Figure 4.8 to model a reconfigurable task:

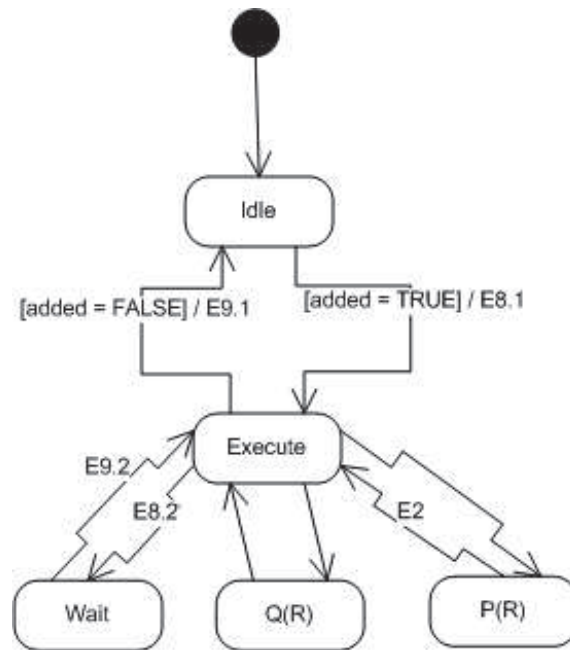


Figure 4.8: Task's R-StD.

The task's R-StD is composed of the following states:

- *Idle*: as its names suggests, the task is idle,
- *Execute*: the task is running,
- *Wait*: the task was interrupted by another one, so it is waiting,
- $P(R)$: the task T is asking to lock a resource R ,
- $Q(R)$: the task T is unlocking the resource R .

The R-StD of a task T should include as many $P(R)$ and $Q(R)$ as the resources it may lock, but, in this running example, we decide that T will use just one resource (R). The different events indicated on the figure above stand for:

- $E2$: an event confirming the lock of R by T ,
- $E8.1$ and $E8.2$: when T switches from *Idle* to *Execute*, the event $E8.1$ forces the running tasks with lower priorities to switch from *Execute* to *Wait*. T 's $E8.1$ is actually the $E8.2$ of tasks with lower priorities. Whence, the $E8.2$ on Figure 4.8 is an event announcing that a task with a higher priority than T 's switched from *Idle* to *Execute*,

- *E9.1*: when T switches from *Execute* to *Idle*, the event *E9.1* will force the waiting tasks with lower priorities to switch from *Wait* to *Execute*. T 's *E9.1* is actually the *E9.2* of tasks with lower priorities. Whence, the *E9.2* is translating that a task with a higher priority than T 's has switched from *Execute* to *Idle*.

Running Example 2

Now that we formalized R-StD and proposed patterns to model control tasks and shared resources, we can model our running example. We propose, as examples and respectively in Figure 4.9 and Figure 4.10, the modeling of the control task A , which uses the resources Q , R and S , and the resource R which is shared by the tasks A and B .

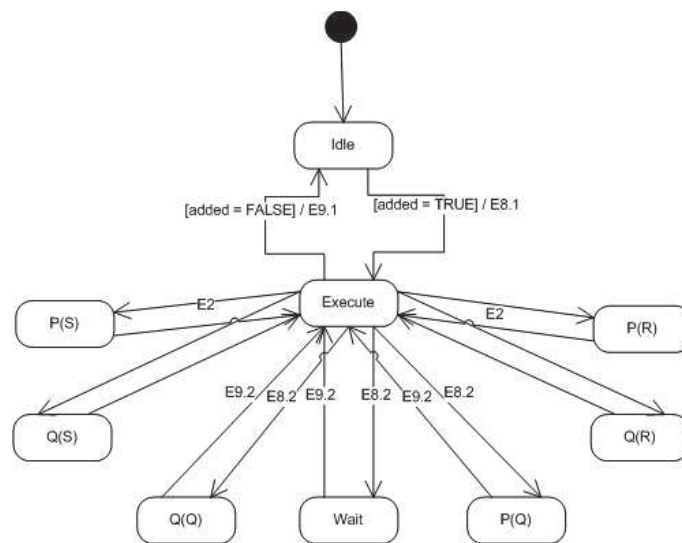


Figure 4.9: Task A's R-StD.

In our case study, the different resources have the same modeling since they have the same ceiling and are used by the same tasks. We choose to model the resource R as shown in Figure 4.10. The guards named X are used to guarantee that, when a task T tries to lock the resource, all the other resources, whose ceilings are not lower than the task's priority, are free or hold by T . Thus, we avoid any eventual deadlock and see the relevance of the PCP.

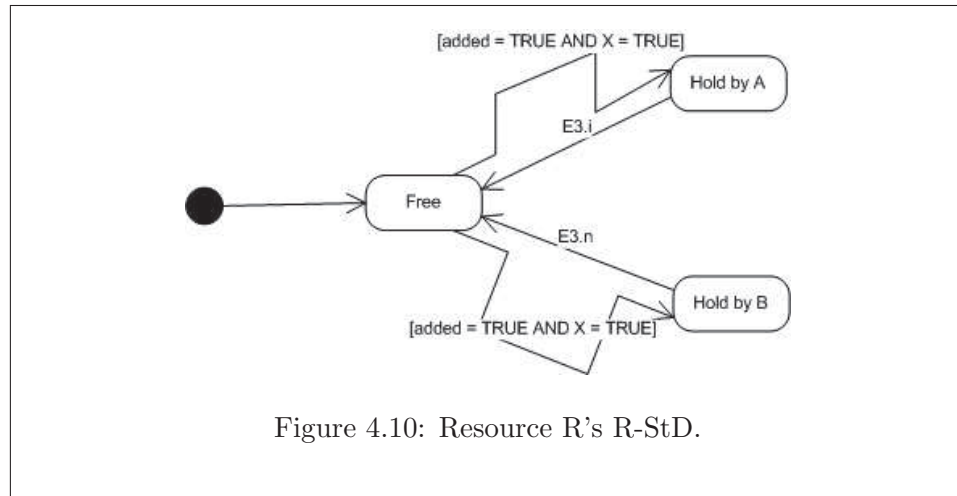


Figure 4.10: Resource R's R-StD.

4.5 Transformation from R-UML to R-TNCES

We propose, in this section, a new solution to translate R-StD models into the ones R-TNCES ones. A formal verification is, then, performed to prove the relevance of our contribution. Finally, an algorithm is proposed to automatize this transformation.

4.5.1 R-Std Translation into R-TNCES

Table 4.1: Correspondence table for R-StD translation into R-TNCES.

Rules	R-StD	R-TNCES
Rule 1	St (4.2)	P (2.2.3)
Rule 2	Tr (4.2)	T (2.2.3)
Rule 3	$\{st_b, st_e\} := \zeta(tr)$ (4.2)	$\{p_{out}, p_{to}\} \subseteq P ; \{fa_1, fa_2\} \subseteq F$ (2.2.3)
Rule 4	$gr := \delta^{-1}(tr)$ (4.2)	$ci \in C^{in}$ (2.3) ; $co \in C^{out}$ (2.3) ; $ca \in CN$ (2.2)
Rule 5	$ac := \epsilon^{-1}(tr)$ (4.2)	$ei \in E^{in}$ (2.3) ; $eo \in E^{out}$ (2.3) ; $ea \in EN$ (2.2)
Rule 6	$ev := \zeta^{-1}(tr)$ (4.2) AND $<< eventInput >> := \alpha^{-1}(ev)$ (4.1)	$ei \in E^{in}$ (2.3) ; $eo \in E^{out}$ (2.3) ; $ea \in EN$ (2.2)
Rule 7	$ev := \zeta^{-1}(tr)$ (4.2) AND ev is an after(n) event	$n \in DR ; \infty \in DL$ (2.2) (2.3) (2.4)

The paper proposes Table 4.2 which is given above to show the correspondence between R-StD and R-TNCES. The numbers given in parentheses show the reference to the formulas that give details on the syntax used in the table.

The seven translation rules are explained hereafter:

- **Rule 1:** A state St in an R-StD corresponds to a place P in an R-TNCES;
- **Rule 2:** A transition Tr in an R-StD corresponds to a transition too (T) in an R-TNCES;
- **Rule 3:** Each transition tr in an R-StD is mapped to a pair of states, st_b and st_e , where the first is the state from which tr is taken and the second is the next state if tr fires. The corresponding transition (t) and two places (p_{out} and p_{to}) will be created using, respectively, Rule 2 and Rule 1. Rule 3 creates actually in the R-TNCES a flow arc, fa_1 , linking p_{out} to t , and another one, fa_2 , linking t to p_{to} ;
- **Rule 4:** In an R-StD, some guards can be mapped to some transitions. A guard gr corresponds to 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 , to the place which is pointed by ca ;
- **Rule 5:** In an R-StD, some actions can be mapped to some transitions. An action ac corresponds to an event arc, ea , in an R-TNCES. An event output signal, eo , is added to the place from which ea is leaving and an event input signal, ei , to the place which is pointed by ea ;
- **Rule 6:** In an R-StD, each `<<eventInput>>`-stereotyped event, ev , is translated into an event arc, ea , in the corresponding R-TNCES. An event output signal, eo , is added to the place from which ea is leaving and an event input signal, ei , to the place which is pointed by ea ;
- **Rule 7:** An R-StD may feature after(n)-typed events, where $n \in \mathbf{N}^*$. If so, n is added to DR , the set of minimum times that the token should spend at particular place before the transition can fire, and ∞ to DL , the set of limitation time that defines maximum time that the place may hold a token, since the place from which the after(n)-typed event is leaving may indefinitely hold the token.

4.5.2 Verification

We propose in this section to check the relevance of the proposed solution and the contribution of PCP in solving several issues threatening a DRCS's

safety and deadlock-freedom. Thus, we start by modeling the running example of Section 4.3 in UML and then transforming the latter in R-TNCES according to [Zhang 2013]. Thus, we don't call out the PCP. We obtain the model illustrated in Figure 4.11. To verify it, we use model-checking which is a technique for automatically verifying the correctness properties of finite-state systems. Model checking for R-TNCES is based on its reachability graphs. ZiZo [Salem 2015b] is a new and effective software environment for the analysis of R-TNCES, which computes the set of reachable states exactly. It exports, then, files exploitable by the model-checker SESA [Starke 2002]. Typical properties which can be verified are boundedness of places, liveness of transitions, and reachability of states. In addition, temporal/functional properties based on Computation Tree Logic (CTL) specified by users can be checked manually. We apply, then, the CTL formula $AG\ EX\ TRUE$ which checks the deadlock-freedom of the system. The said formula turned out to be false as shown in Figure 4.12, meaning that the system features a deadlock issue.

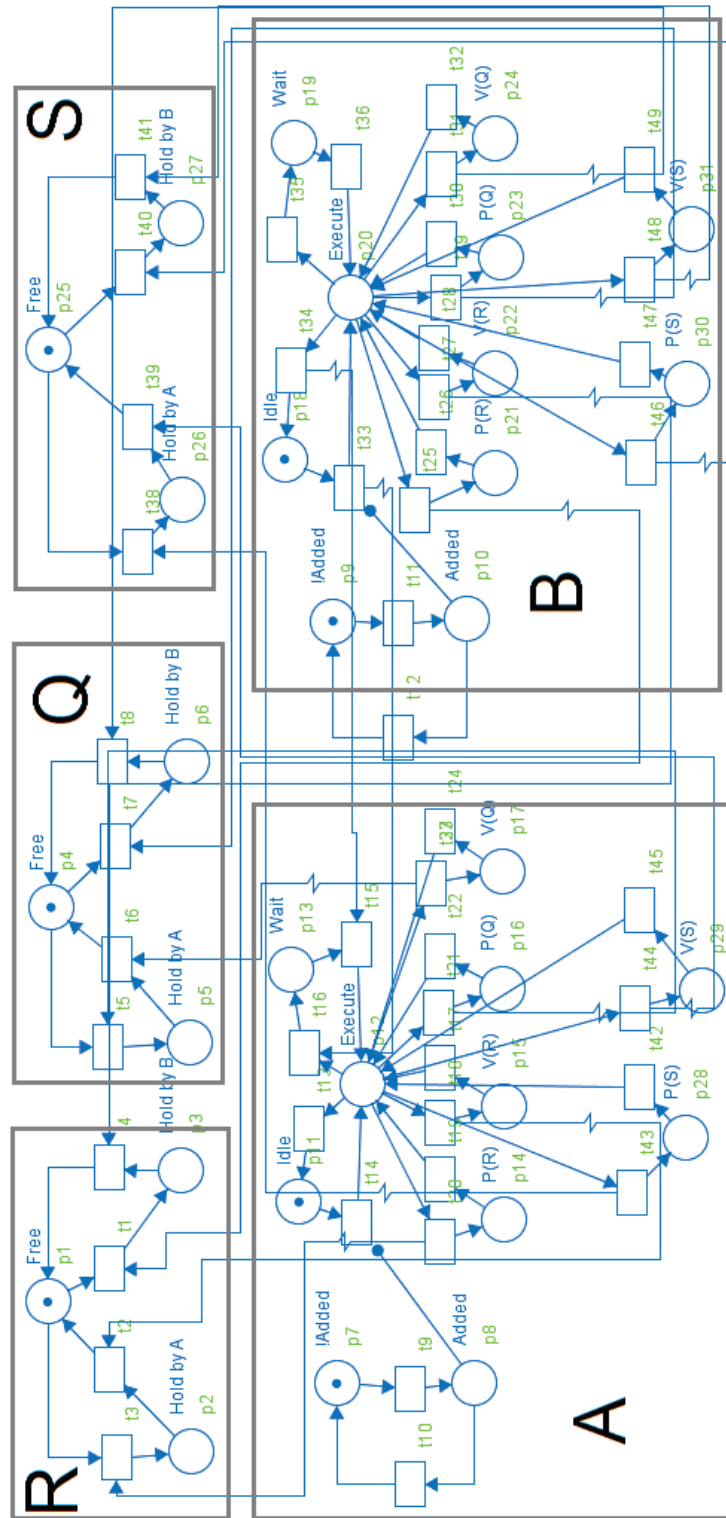


Figure 4.11: Illustrative example's R-TNCEs.



```
Parsing formula:  
AG EX TRUE  
The formula is FALSE.
```

Figure 4.12: Screenshot from SESA.

Whence, we call out the solution we proposed in the previous sections. We start by modeling the two tasks and the three resources in R-StD and transform, then, the models to R-TNCES based on the transformation rules specified in Section 4.5.1. We obtain, thus, the R-TNCES model of the tasks A and B illustrated in Figure 4.13.

Once the R-TNCES model of the DRCS is enriched with PCP, the next step is to verify whether the models meet users requirements. So, any reconfiguration scenario dealing with adding/removal of resources does not lead to a blocking situation. The following e-CTL formula is applied:

$$AG EX true \tag{4.5}$$

This formula is proven to be true by SESA as shown in the screenshot in Figure 4.14, so there is no deadlock in our R-TNCES.

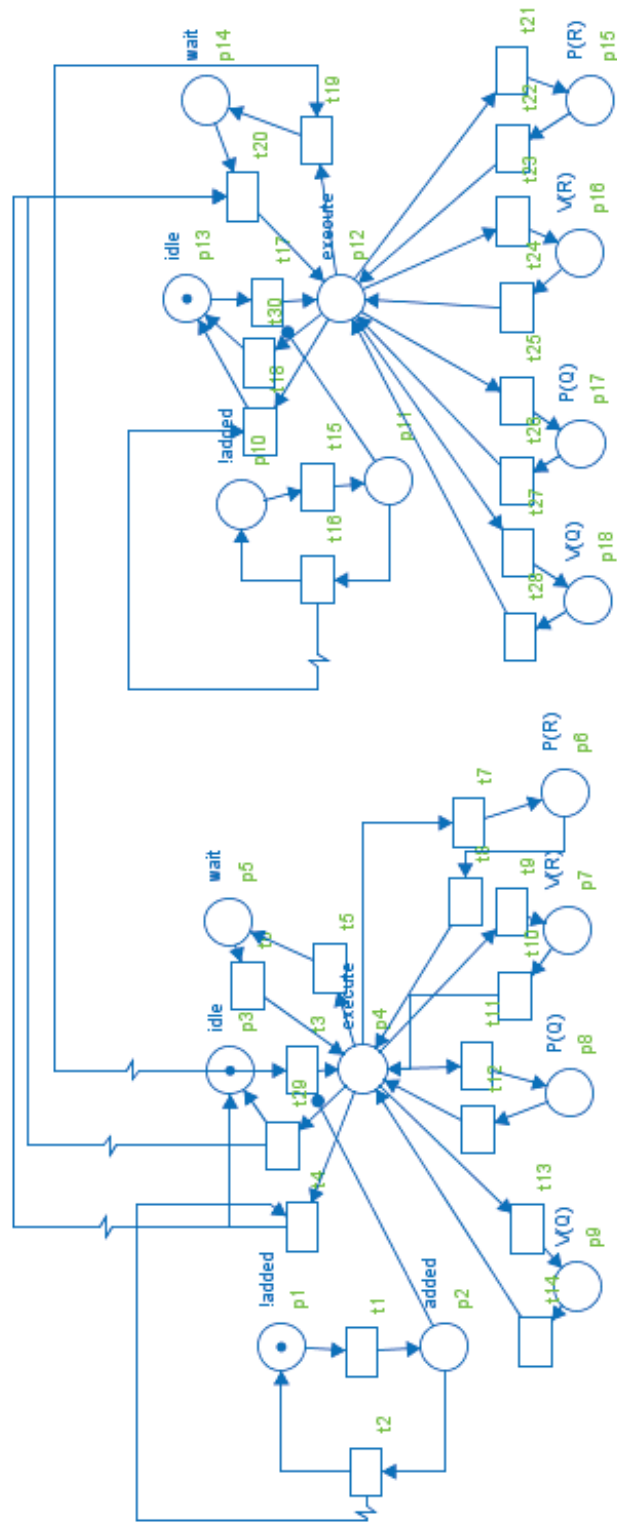


Figure 4.13: Tasks A and B's modeling using PCP.

```

Parsing formula:
AG EX TRUE
The formula is TRUE.

```

Figure 4.14: Screenshot from SESA.

We also check the safety property by checking if a given resource may be simultaneously locked by two different tasks. The following CTL formula is checked:

$$EF p22 \text{ AND } p23 \quad (4.6)$$

where $p22$ is the place translating that the resource R is locked by the task A ; $p23$ means that B locks R . This formula is proven to be false as illustrated in Figure 4.15.

```

Parsing formula:
(EF p22 AND p23)
Current model checking options are:
  write a proof
  but only witnesses and counterexamples
  but only for the top level formula
  to the session file
.....Reset options? Y/N M
States: 53
The formula is FALSE.

```

Figure 4.15: Screenshot from SESA.

The formula 5.3.2 is applied six times of the R-TNCES modeling, changing at each time $p22$ and $p23$ by the places which correspond to the ones translating that the resource R (and then Q and S) is locked by the task A (and then B). We check thus whether a given resource can be locked by the two tasks at the same time. The six formulas turned out to be false. We are sure, then, that our system doesn't feature a deadlock issue caused by a concurrent access to shared resources after a reconfiguration scenario.

4.5.3 Implementation

Having basic elements of R-UML used for system modeling defined by the equations 4.1 and 4.3, the R-UML project defining a reconfigurable system with adaptive shared resources model can be represented via 4-tuple:

$$R - UML_{system} = (CIDs, R - StDs, O, \Omega) \quad (4.7)$$

where:

- $CIDs = \{CID_1, CID_2, \dots, CID_n\}$ is a finite set of class diagrams, where each CID_i element is defined by (4.1);

- $R - StDs = \{R - StD_1, StD_2, \dots, StD_l\}$ is a finite set of reconfigurable state diagrams, where each $R - StD_j$ element is defined by (4.3);
- O is a finite set of objects, where each one is an instance of CID_i and have its corresponding $R - StD_j$;
- $\Omega : R - StD_a \rightarrow cl_b$ is function that maps the reconfigurable state diagram $R - StD_a$ to the class cl_b of C (4.1).

The algorithm 1 defines the rules to translate R-UML to R-TNCES. The numbers given in parentheses show the reference to the formula that gives details on the used syntax.

4.6 Transformation from R-UML to ROS

4.6.1 State of the Art

Many works in our community have recently established concepts to improve robustness, interoperability, maintainability and reusability in robotics to face the latter's growing complexity. They proposed means of component-based architectures and model-driven software development. This gave rise to the creation of robotic frameworks and architectures such as ROS [Quigley 2009], SmartSoft [Schlegel 2004] and Orocos [Bruyninckx 2001]. Current activities focus on composition towards reuse as black boxes and configuration at run-time for both, parameters and the component's life-cycle.

Among the works promoting the component-based development, we cite the BRICS component model (BCM) [Bruyninckx 2013] which introduces the concept of composition with components grouped or nested together. They include a lifecycle coordinator to form a new reusable component. Restricted finite state machines (rFSM) [Klotzbücher 2012] were also developed in the context of BRICS/BCM. rFSM are a minimal variant of state charts and are integrate into the robotic framework OROCOS/RTT. This new concept was developed with a focus on component coordination for robotics. BCM and rFSM certainly make valuable progress towards reuse and composition of components. However, they cannot handle reconfiguration, an important feature in new robotic systems [Murata 2002]. They control the component's life-cycle rather than the component's skills at task level.

4.6.2 Transformation Rules from R-UML to ROS

We propose in this section a new approach to generate ROS code from R-UML diagrams. We define, thus, a server to manage the different reconfiguration scenarios. We also explain how ROS nodes communicate. We introduce then a solution to translate R-UML into ROS code.

Algorithm 1 R-UML translation into R-TNCES

```

Input:  $R - UML_{system}$ 
Output:  $R - TNCES_{system}$ 
for  $obj_j \in O$  (4.7);  $j \in [0, |O|]$  do
  Initialize  $R - TNCES_k$ , each element is  $\emptyset$  (2.2.3)
  Define class of the object  $Cl := \Omega(obj_j)$  (4.7)
  for  $attr_l \in Cl(l \in [0, |A_{Cl}|])$  do
    if  $\alpha^{-1}(attr_l) = \langle\langle input \rangle\rangle$  (4.1) then
      Add condition input  $ci$  to  $C^{in}$  of  $R - TNCES_k$  (2.3)
      Add condition arc  $ca$  to  $CN$  of  $R - TNCES_k$  (2.2)
    end if
    if  $\alpha^{-1}(attr_l) = \langle\langle output \rangle\rangle$  (4.1) then
      Add condition output  $co$  to  $C^{out}$  of  $R - TNCES_k$  (2.3)
      Add condition arc  $ca$  to  $CN$  of  $R - TNCES_k$  (2.2)
    end if
    if  $\alpha^{-1}(attr_l) = \langle\langle in \rangle\rangle$  (4.1) then
      Add condition input  $ci$  to  $C^{in}$  of  $R - TNCES_k$  (2.3)
    end if
    if  $\alpha^{-1}(attr_l) = \langle\langle out \rangle\rangle$  (4.1) then
      Add condition input  $co$  to  $C^{out}$  of  $R - TNCES_k$  (2.3)
    end if
    if  $\alpha^{-1}(attr_l) = \langle\langle eventInput \rangle\rangle$  (4.1) then
      Add event input  $ei$  to  $E^{in}$  of  $R - TNCES_k$  (2.3)
    end if
    if  $\alpha^{-1}(attr_l) = \langle\langle eventOutput \rangle\rangle$  (4.1) then
      Add event input  $eo$  to  $E^{out}$  of  $R - TNCES_k$  (2.3)
    end if
    if  $\alpha^{-1}(attr_l) = \langle\langle integer \rangle\rangle$  and  $Attr_l == x$  (4.1) then
      Add  $x$  to  $DR$  of  $DC$  (2.4)
    end if
    if  $\alpha^{-1}(attr_l) = \langle\langle boolean \rangle\rangle$  and  $Attr_l == y$  (4.1) then
      Add  $y$  to  $G$  of  $StD$  (4.2)
    end if
  end for
  Define a state diagram for each object  $obj_j : R - StD := \Omega^{-1}(Cl)$  (4.7)
  for  $tr \in Tr$  of  $StD$  (4.2) do
    Add transition  $t$  to  $T$  of  $R - TNCES_k$  (2.2.3)
    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  $R - TNCES_k$  (2.2.3)
      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  $tr:gr := \delta^{-1}(tr)$  (4.2)
    Define action for transition  $tr:ac := \epsilon^{-1}(tr)$  (4.2)
    Define event for transition  $tr:ev := \gamma^{-1}(tr)$  (4.2)
    for  $\langle operand \rangle (P_{guard}) \in gr$  do
      Add condition input  $ci$  to  $C^{in}$  of  $R - TNCES_k$  (2.2.3)
      Add condition arc  $ca$  to  $CN$  of  $R - TNCES_k$  (2.2)
    end for
    for  $\langle action \rangle (P_{action}) \in ac$  do
      Add event input  $ei$  to  $E^{in}$  of  $R - TNCES_k$  (2.2.3)
      Add event arc  $ea$  to  $EN$  of  $R - TNCES_k$  (2.2)
    end for
    if  $ev$  is of eventInputi stereotype then
      Add event input  $ei$  to  $E^{in}$  of  $R - TNCES_k$  (2.2.3)
      Add event arc  $ea$  to  $EN$  of  $R - TNCES_k$  (2.2)
    end if
    if  $ev$  is after(n) event then
      Add  $n$  to  $DR$  of  $DC$  for  $fa_1$  (2.4)
      Add  $\infty$  to  $DL$  of  $DC$  for  $fa_1$  (2.4)
    end if
  end for
end for

```

Configuration Server

Due to the potential large number of parameters in a reconfigurable robotic system, it is highly unpractical or even infeasible to configure and tune the performance of the said system if these parameters are incorporated in the source code as hard-coded constants. Modifying hard-coded constants requires recompilation and redeployment to the robot, potentially a very time-consuming process, and does not allow for runtime changes to parameters. To address this issue, the configuration server was developed. Implemented as a ROS node, the configuration server is a centralized manager and storage location for system parameters. The existing *dynamic_reconfigure* ROS package provides a means to change node parameters at any time without having to restart the node [Allgeuer 2013].

Running Example

To begin, we create a package called *running_example* which depends on the packages *rospy*, *roscpp* and *dynamic_reconfigure*:

```
catkin_create_pkg -roscpp indigo running_example rospy roscpp dynamic_reconfigure
```

Then, we create a *cfg* directory, this is where all configuration files live:

```
mkdir cfg
```

Lastly, we need to create two configuration files, *task.cfg* and *resource.cfg*:

```
task.cfg
```

```
#!/usr/bin/env python PACKAGE = "running_example"

from dynamic_reconfigure.parameter_generator_catkin import *

gen = ParameterGenerator()

gen.add("priority", int_t, 0, "the priority of the task", 1, 0, 100)

gen.add("added", bool_t, 0, "defines whether the task is added to the system", False)
```



```
exit(gen.generate(PACKAGE, "running_example", "run-
ning_example"))

task.cfg

#!/usr/bin/env python PACKAGE = "running_example"

from dynamic_reconfigure.parameter_generator_catkin import *

gen = ParameterGenerator()

gen.add("ceiling", int_t, 0, "the priority of the task", 1, 0, 100)

gen.add("added", bool_t, 0, "defines whether the task is added to
the system", False)

exit(gen.generate(PACKAGE, "running_example", "run-
ning_example"))
```

Communication with ROS Nodes

Communication between ROS nodes and the external world is performed in the form of YAML strings. This approach has a number of advantages. First, this approach avoids the requirement of actually running ROS on the lightweight device featuring the HMI (human-machine interface). This is a critical issue as ROS is not designed for the small memory, bandwidth and computational limits of most of devices featuring HMI. Second, interfacing with the robot's control environment can be separated to a large extent from the ROS-based computation that is providing vehicle control. Finally, the approach only exposes the specific messages that are key for device operation/management [Speers 2013].

Dynamic_reconfigure Package

dynamic_reconfigure ROS package, which extends the ROS parameter server, permits easy adjustment of filtering parameters on-the-fly (during execution). This tool is used to change the parameters of ROS nodes dynamically. The parameters are stored in a hierarchical structure in the server, with local copies of the relevant parameters being kept in each of the nodes for performance reasons. The appropriate nodes are notified via service calls whenever a parameter is changed on the server. Appropriate functionality exists to be able to load and save the entire parameter hierarchy. The

Table 4.2: Correspondence table for R-UML translation into ROS configuration file.

Rules	R-StD	ROS
Rule 1	Class	Node
Rule 2	Attribute	Parameter
Rule 3	Stereotype	Enumerated type

default parameter values on system startup are taken from a well-defined configuration file, stored on each robot. This feature is very interesting in this paper, since configuration parameters are dynamically changed.

R-UML Translation into ROS Configuration File

The paper proposes Table 4.2 which is given above to show the correspondence between R-UML class diagrams (Formula 4.1) and ROS configuration file.

The three translation rules are explained hereafter:

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 are pretty simple, they just initialize ros and import the parameter generator.

```
from dynamic_reconfigure.parameter_generator_catkin import *
```

We create then a generator.

```
gen = ParameterGenerator()
```

Now that we have a generator we can start to define parameters. The add function adds a parameter to the list of parameters. It takes few different arguments:

- **name** - a string which specifies the name under which this parameter should be stored.
- **type** - defines the type of value stored, and can be any of `int_t`, `double_t`, `str_t`, or `bool_t`.
- **level** - A bitmask which will later be passed to the dynamic reconfigure callback. When the callback is called all of the level values for parameters that have been changed are ORed together and the resulting value is passed to the callback.
- **description** - string which describes the parameter.

- **default** - specifies the default value.
- **min** - specifies the min value (optional and does not apply to strings and bools).
- **max** - specifies the max value (optional and does not apply to strings and bools).

```
gen.add("int_param", int_t ,
0, "An Integer parameter", 50, 0, 100)
gen.add("double_param", double_t, 0, "A double parameter",
.5, 0, 1)
gen.add("str_param", str_t ,
0, "A string parameter", "Hello World")
gen.add("bool_param", bool_t ,
0, "A Boolean parameter", True)
```

The last line simply tells the generator to generate the necessary files and exit the program. The second parameter is the name of a node this could run in, the third parameter is a name prefix the generated files will get

```
exit(gen.generate(PACKAGE, "dynamic-example", "example"))
```

4.7 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 [Salem 2015b]. Thus, the latter will be able to, first, edit R-UML models and, secondly, translate them into R-TNCES ones as indicated in Figure 4.16. The user can, lately, simulate the generated models and apply CTL formulas on them to check different properties.

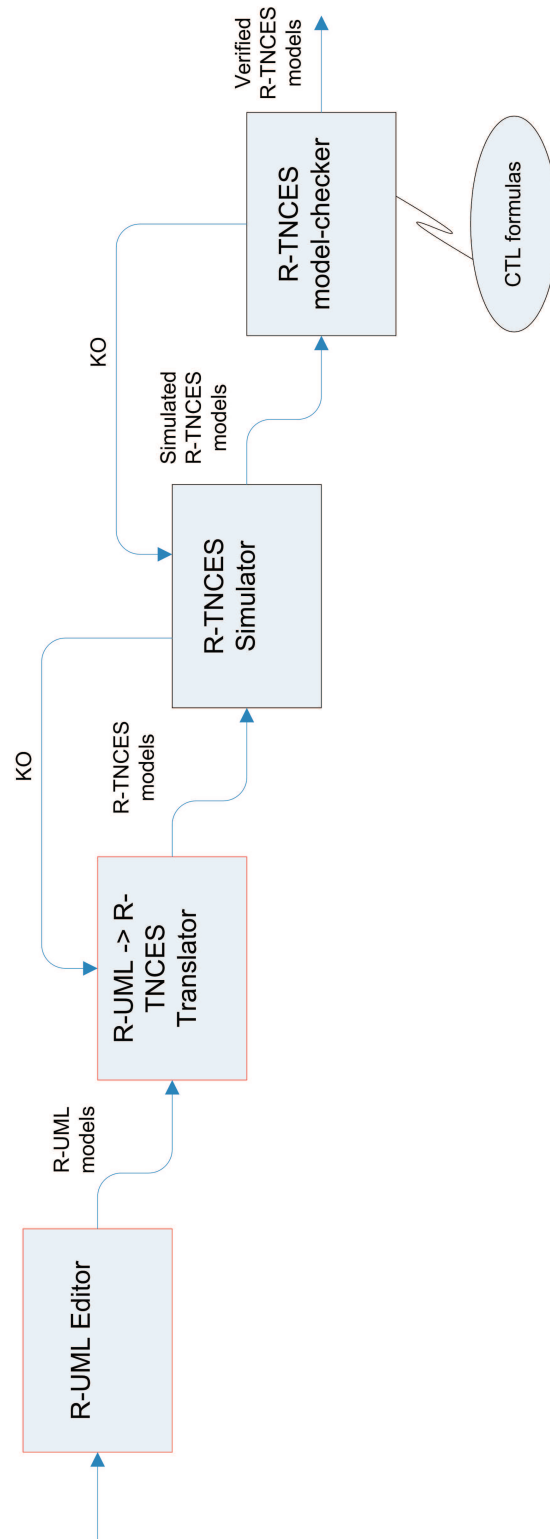


Figure 4.16: ZiZo's different modules.

We start, then, by editing an R-StD describing the behavior of BROS as illustrated in Figure 4.17. The second step consists, according to Figure 4.16, in translating the R-UML model into an R-TNCES one. The latter is shown in Figure 4.19. Upon definition of the model, ZiZo can simulate it. Simulation can be tracked by selection of a token game. Once simulation is finished, a report is displayed at the debug window. The obtained report displayed in Figure 3.19 proves that, after exploring 3057 places by ZiZo, our system is deadlock-free.

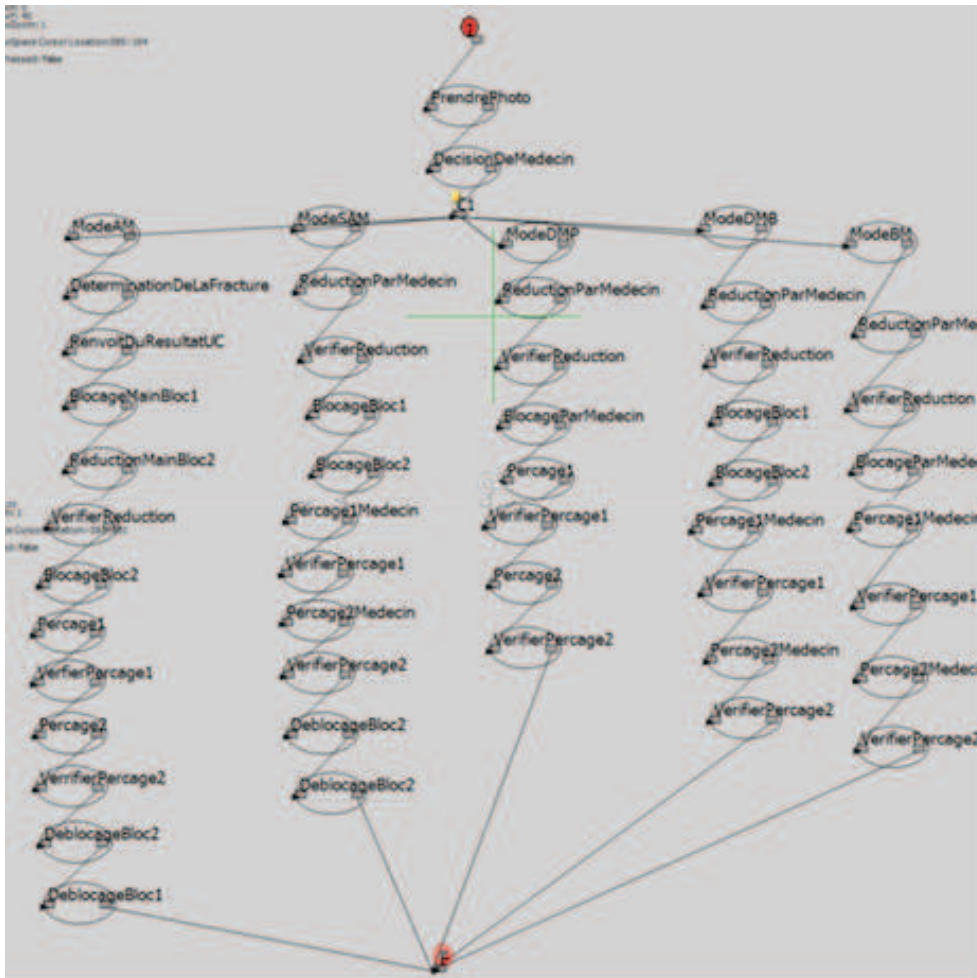


Figure 4.17: R-StD of BROS.

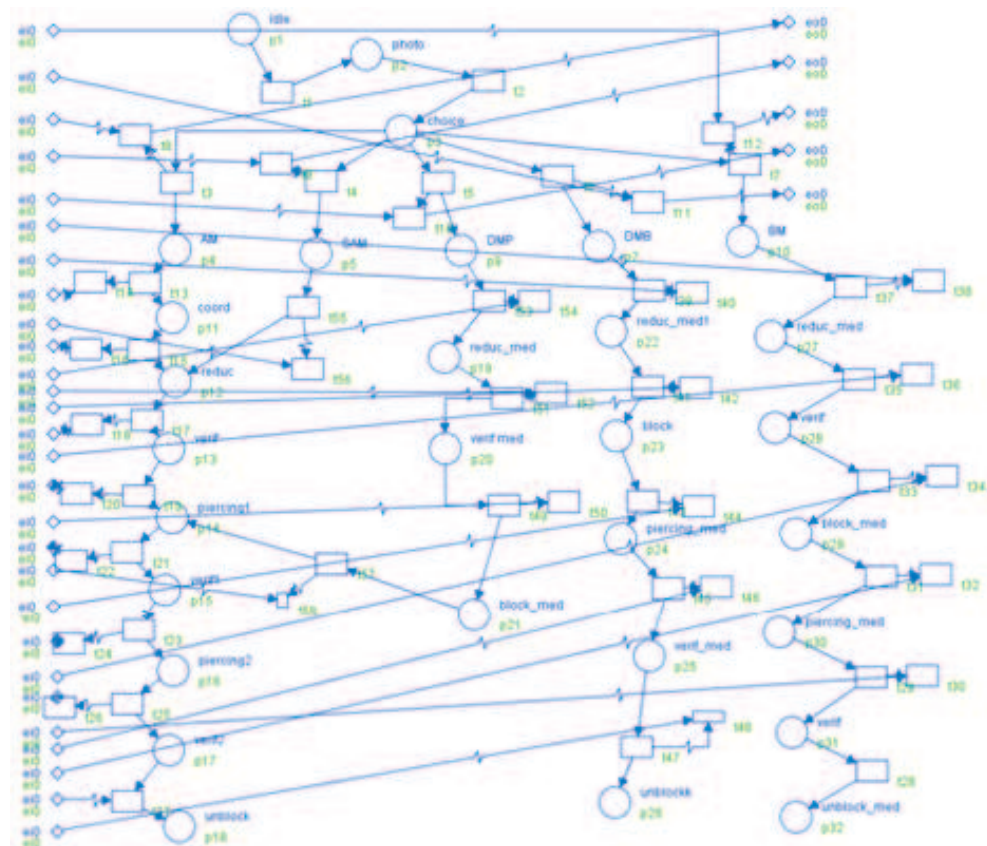


Figure 4.18: R-TNCES of BROS.

```

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

Figure 4.19: BROS's simulation report.

After proving, by simulation, the non-existence of problems related to concurrent access on BROS's reconfigurable shared resources, we perform again the exhaustive CTL-based verification previously done in Section 3.4.2 to check the potential existence of several problems that may be faced at BROS's runtime. The results are as positive as in Section 3.4.2.

4.8 Summary

Our work consisted, through this chapter, in proposing a new UML profile, the R-UML, to model and verify flexible control systems sharing adaptive

resources. Whence, we chose to enhance class and statecharts diagrams to support PCP. We proposed, then, a new and original solution to translate the generated R-UML models into R-TNCES-based patterns which were proposed in [Salem 2014]. This aims at proving the correctness of the R-UML models by performing model-checking on the generated R-TNCES models. The relevance of our contribution was proved thanks to model-checking using ZiZo, a new R-TNCES editor, simulator and model-checker [Salem 2015b]. This approach is original since R-TNCES is a new formalism dedicated to flexible control systems modeling and ZiZo is a new tool supporting the said formalism.

We applied, then, this contribution on BROS which is a flexible system since it can run under different operating modes: it is reconfigurable. The concurrent access to adaptive shared resources is present in the said system, which can be rather hazardous in such medical systems. Whence, applying our contribution to BROS turned out to be very relevant to certify that the robotic platform is safe and does not run any risk after any reconfiguration scenario. Thus, we can move to the next step: implementation of BROS.

Implementation of BROS

Contents

5.1	Introduction	86
5.2	Platform and Environment	88
5.3	Middleware	89
5.3.1	Architecture	89
5.3.2	Image Processing	91
5.4	Control Unit	94
5.4.1	Station Definition	95
5.4.2	B-BROS1 Module	96
5.4.3	B-BROS2 Module	96
5.4.4	P-BROS Module	97
5.4.5	Synchronization Module	98
5.4.6	CU-MW Communication Module	99
5.4.7	Surgeon-Robot Interface	99
5.4.8	Simulation and Test of BROS	100
5.5	Summary	101

5.1 Introduction

Once the system is specified and certified, we move to the implementation step. It takes as input the verified models we got from the certification step and hardware/software libraries (algorithms, programming languages, OS,..etc) which were chosen by designers to be used for the implementation of the system. This step is composed of two sub-steps as illustrated in Figure 5.1. It produces the code that will be used for the deployment step. If any issue is detected during the simulation step, we have to go back to the specification step. Note that this simulation step is different from the one we find in the certification step: the first uses simulators provided by the manufacturers of the hardware/software we are intending to use, whereas the second uses ZiZo to detect any potential deadlock in the R-TNCES-based models.

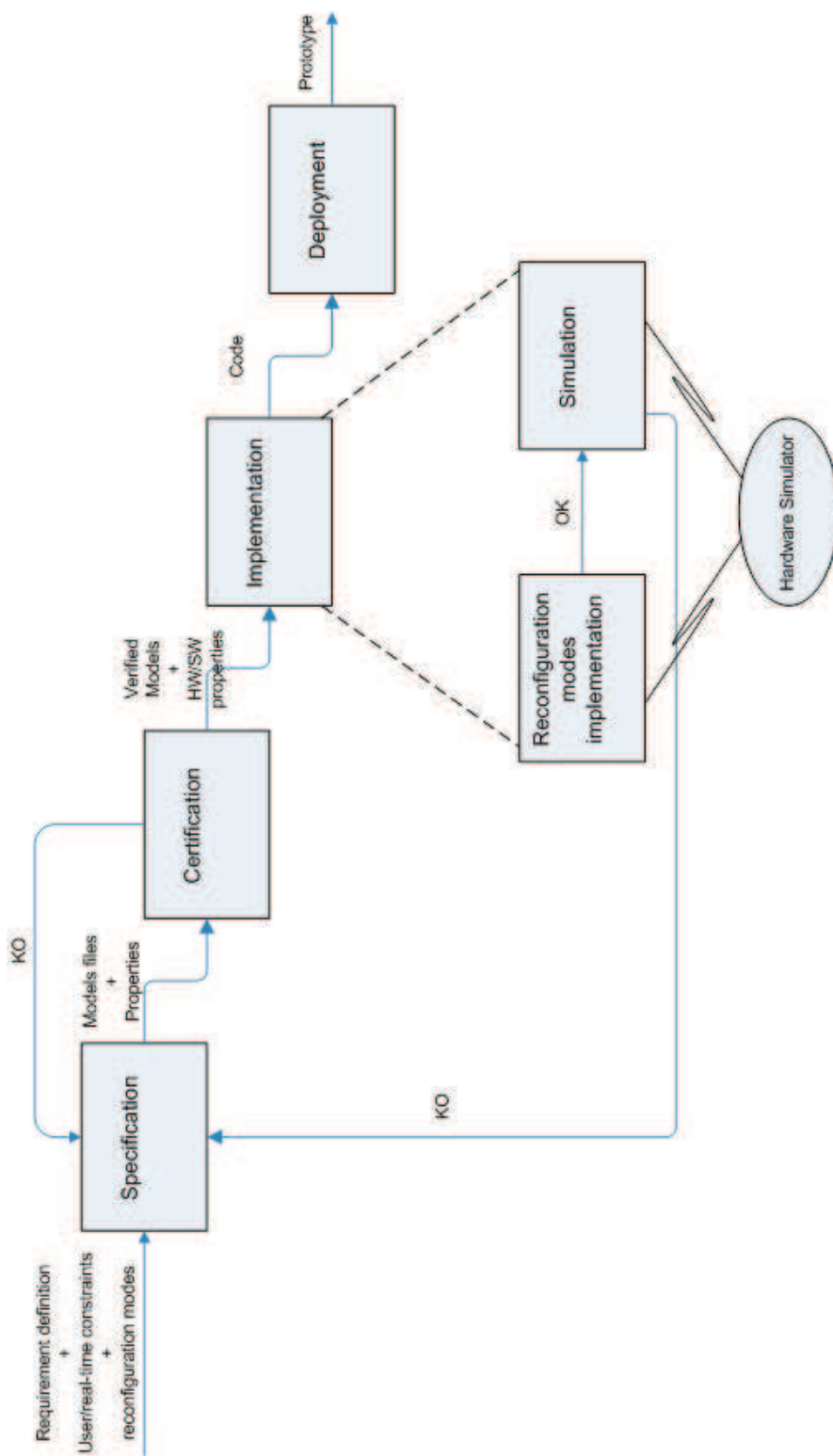


Figure 5.1: Implementation step.

5.2 Platform and Environment

As the smallest robot from ABB, the IRB 120 offers all the functionality and expertise of the ABB range in a much smaller package. Like all ABB robots, the IRB 120 is a particularly agile 6-axis robot which, thanks to its compact turning radius, can be mounted closer to other equipment. Besides, it is ideal for a wide range of industries including the electronic, food and beverage, machinery, solar, pharmaceutical, medical and research sectors. With its lightweight but strong aluminum structure and small powerful engines, the IRB 120 weighs only 25 kg, which explains its rapid and precise acceleration. In fact, this featherweight has all the traditional features of ABB robots, including leading performance in terms of trajectory tracking and motion control. Thus, the IRB 120 won many manufacturers' spurs [Mikaelsson 2009].

IRB 120 can be programmed offline with RobotStudio ABB's software that allows to simulate an industrial manufacturing cell to find the optimal position of the robot and avoid costly downtime and production delays. RobotStudio from ABB Robotics is a powerful off-line robot programming and simulation tool. What makes it unique is the fact that, when the code is fully developed off-line, it downloads to the actual controller with no translation stage, reducing time-to-market. RobotStudio is able to create the robot movements using graphical programming, edit and debug the robot system, and simulate and optimize existing robot programs. It is widely used in universities to educate engineering students in the capabilities and applications of robots, as well as in the automation industry by mechanical designers and robot programmers. RobotStudio is also used in remote maintenance and troubleshooting. It actually connects to the live system to take an instant virtual copy, and then goes off-line to enable the situation to be studied in depth. RobotStudio also features a RAPID Editor which enables the user to write a robot program. The user can watch a single robot execute the RAPID program in the graphical environment [Connolly 2009].

Running example 1

To test our new robotized platform, we decided to simulate the surgery that would be performed on a real case. Thus, we chose a new patient, a nine-year-old girl, suffering from a fracture similar to the one presented in the case study of Section 2.3.2 (a type III fracture). We simulated the whole surgery on June 9th 2014 using the software RobotStudio and the developed middleware and control unit. We will present the obtained results as we introduce these two components in the next sections.

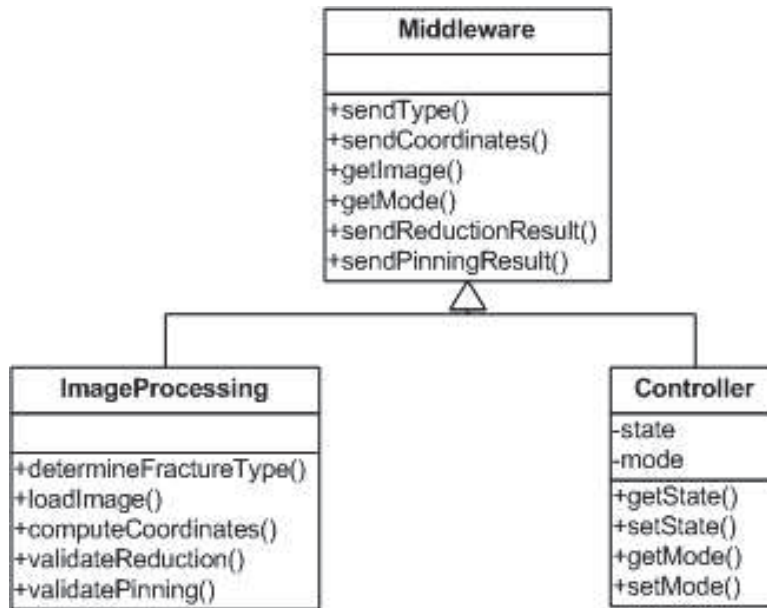


Figure 5.2: Middleware's class diagram.

5.3 Middleware

We introduce in this section the architecture of the middleware and its image processing module.

5.3.1 Architecture

The Middleware features two important modules: the first performs operations relating to image processing and the second insures the synchronization and communication with the whole robotized platform. Middleware's class diagram is illustrated in Figure 5.3.1. Since the middleware acts as a mediator between the browser and the control unit, several data are exchanged between MW and CU during the surgery. First, the control unit notifies the start of the intervention and the activated operating mode to the middleware. Then, it asks it to compute necessary parameters like fracture's type and spatial coordinates. It also informs MW about the end of reduction and pinning. The middleware and the control unit are connected through an ad hoc network. We illustrate the different exchanges between MW and CU by a sequence diagram as shown in Figure 5.3.1.

The controller is a module that saves the current status reached by the intervention. Indeed, the control unit informs the middleware of each fired transition and the current triggered operating mode. The control unit updates these information as the intervention advances in time. Thus, the middleware is kept aware of the progress of the surgery. This module syn-

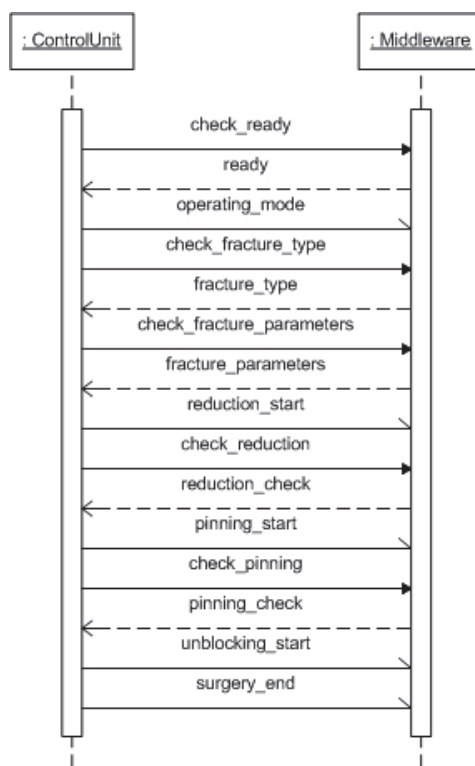


Figure 5.3: Sequence diagram of communication with CU.

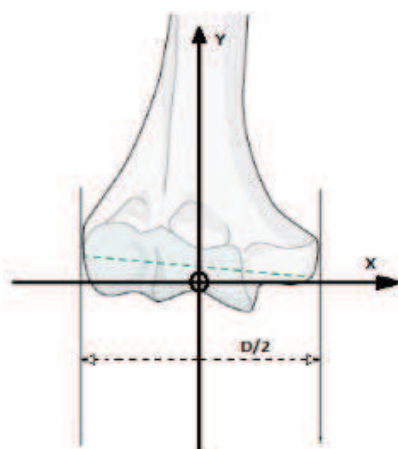


Figure 5.4: Coordinate system axes.

chronizes, then, the middleware with the whole operation. The image processing module is deeply detailed in the next section.

5.3.2 Image Processing

Image processing is the most important module of the middleware and provides a number of features that we detail below.

Locating

Locating is an important feature that involves setting a spatial reference which is considered during the whole intervention. The middleware and the control unit must use the same coordinate system since several points coordinates computed by MW are, firstly, sent to CU so the latter performs a preoperative simulation and, secondly, to B-BROS and P-BROS to realize the fracture reduction and pinning. We choose to fix the coordinate system origin at the patient's elbow as illustrated in Figure 5.3.2. The X, Y and Z axes respectively represent the elbow's rotation axis, the humeral palette length's median and the normal to (XY) plan.

Determination of the Fracture Type

MW starts by receiving from BW a first image of the fracture to determine its type. It compares the acquired image with the ones stored in its database. To do this, the middleware uses two image processing techniques, ensuring, thus, proper detection of the fracture type. The first one is image matching and consists in comparing images in order to obtain a measure of their similarity. It extracts invariant local features for all images, and then uses voting to rank the database images in similarity with the query

image [Grauman 2005b]. The second used image processing technique is contour comparison. It consists in detecting an image contour by quantifying the presence of a boundary at a given image location through local measurements [?]. The contour comparison is applied on the patient's elbow image acquired from BW and images stored at the database, one at a time.

Running example 2

Figure 5.3.2 shows the result of image matching applied on the running example's fractured elbow (on the left) and an image from the MW database (on the right). Figure 5.3.2, for its part, shows a contour comparison with another image from the database. The type III is confirmed.

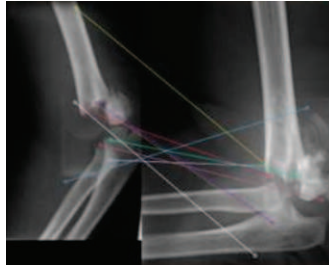


Figure 5.5: Image matching applied on two fractured elbow images.



Figure 5.6: Contour comparison performed by MW.

Coordinates Transformation

The middleware acquires images from the browser. The latter uses a system camera composed of two lenses to geometrically triangulate the spatial coordinates of each light source on the instrument, reference frame, and C-Arm Target. However, the images it sends to MW are two-dimensional, and MW needs to operate in a three-dimensional environment to properly ensure the different steps of the surgery, such as the fracture reduction and pinning. Thus, we must, first, realize a camera calibration which consists in finding the relationship between the spatial coordinates of a point in space (i.e. the operating theatre) and the associated point in the image taken by the camera [Tsai 1987]. To achieve the desired transformation, two types of parameters must be determined:

- the camera extrinsic parameters which define the position and orientation of camera relative to the space in which we work. Technically,

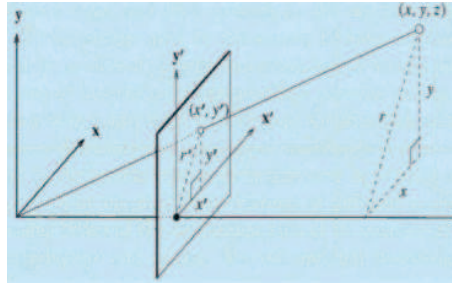


Figure 5.7: Different coordinate systems.

determining these parameters consists in finding the translation vector between the relative positions of the origins of two references: the camera reference and the operating theatre's. A rotation vector aligning the axes of the two references must also be computed.

- the camera intrinsic parameters which are required to bind the image pixels coordinates with the corresponding ones in the camera coordinate system. These parameters present the camera optical, digital and geometric features like the focal length, the geometric distortion and image magnification factors.

Figure 5.3.2 illustrates the different used coordinate systems where: (i) (x, y) plan is the image pixels reference, (ii) (x', y', z') is the camera coordinate system, (iii) (x, y, z) is the operating theatre reference.

To translate the coordinates of a point in the image from the latter's reference to the operating theatre's and vice versa, we use the following formula:

$$S \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = CM(RV \begin{bmatrix} X \\ Y \\ Z_{const} \end{bmatrix} + TV) \quad (5.1)$$

where :

- $S \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$ are the coordinates of a point in the image,
- CM is the camera matrix,
- RV represents the rotation vector,
- TV is the translation vector,
- (X, Y, Z_{const}) are the coordinates corresponding to the point S in the operating theater reference.

Fracture Reduction Validation

The validation of fracture reduction consists in checking whether the bone fragments regained their original places or not. Thus, this module detects, based on the acquired image, the bone discontinuity and, then, computes the distance between the displaced bone fragments. We hereafter explain this technique with the most common fracture types of Lagrange classification: II and III.

Validating the reduction of a type II fracture involves calculating the distances AC and BD as illustrated in Figure 5.3.2. A reduction is considered successful when:

$$|AC| = |BD| = 0 \quad (5.2)$$

BROS has only three attempts to achieve a successful reduction before switching to the semi-automatic mode (SAM) to let the surgeon manually perform it. The type III fractures usually present a rotary disorder. Their reduction consists, therefore, in the rotation of the forearm with an α angle which is $\arcsin(Z_b - Z_a)$ as illustrated in Figure 5.3.2.

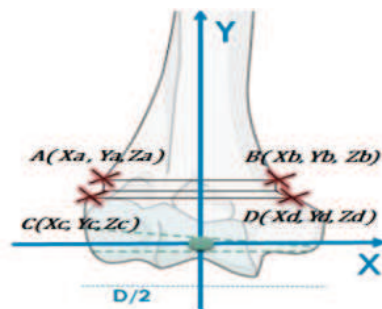


Figure 5.8: Reduction of a type II fracture.

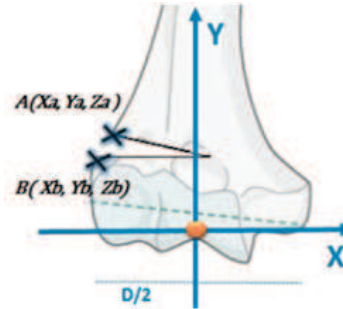


Figure 5.9: Reduction of a type III fracture.

Pinning Validation

Pinning validation amounts to checking the respect of the formula which is introduced in Section 1.2.1 by computing the humeral palette's width and the distance separating the two pins.

5.4 Control Unit

The control unit, the entity responsible of the smooth running and the safety of surgery, is composed of several modules which we detail hereafter. We use RobotStudio to implement it and RobotWare [Rampersaud 2000b] as the robot controller. Both are ABB's products.

5.4.1 Station Definition

This module implements the station which is, in our case, the operating room with all its components. The latter can be grouped into two categories: the mechanisms and the static components. The mechanisms are objects that perform 3D motion during simulations, whereas static components, as their name suggests, remain fixed during all surgery.

Running example 3

Figure 5.10 shows the implementation of our operating theater with its different robotic arms, the patient's limb modeling and the surgical bed.

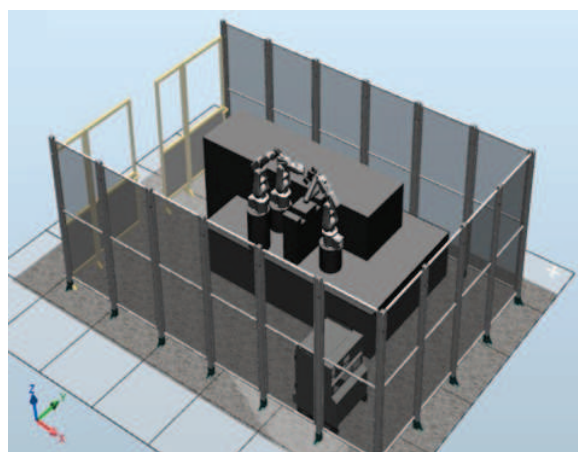


Figure 5.10: Operating room definition.

Mechanisms

Our operating theatre's mechanisms are B-BROS1, B-BROS2 and PBROS. They are all ABB's IRB 120. "Blocker 1" is the used tool to block the patient's limb at humerus and lately unblock it according to coordinates computed by the blocking module. To reduce the fracture and block the limb at forearm, "Blocker 2" is used according to coordinates received from the reduction module. Blocker 1 and Blocker 2 have the same 3D modeling illustrated in Figure 5.4.1. "Pinning", as its name suggests, is the used tool to perform pinning at the patient's elbow according coordinates computed by the pinning module. Its 3D modeling is showed in Figure 5.4.1.

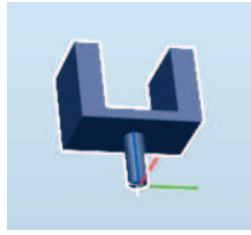


Figure 5.11: Blocker 1 and Blocker 2's 3D modeling.

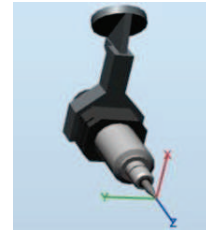


Figure 5.12: Pinning's 3D modeling.

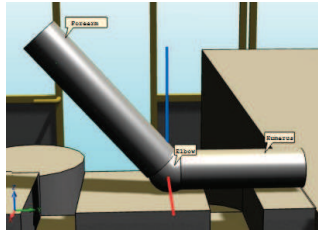


Figure 5.13: Limb's 3D modeling.

To simulate the progress of the surgery on the patient's limb, we model the latter as illustrated in Figure 5.4.1. It is modeled by a mechanism that rotates about the X axis (in red).

Static Components

Static components are the different 3D objects which are useful to the simulation like the robotic arms' racks and the surgical bed.

5.4.2 B-BROS1 Module

B-BROS1 module describes the behavior of the robotic arm B-BROS1 and how it blocks the patient's limb at the humerus and unblocks it once the surgery is completed. Thus, this module features two procedures:

- (i) `B_BROS1_humerusBlock ()` : it blocks the arm at a distance of $y + 100mm$ where y is the coordinate on Y axis of the intersection point of the humeral palette and its median. Figure 5.4.2 illustrates how the blocking is performed,
- (ii) `B_BROS1_humerusUnblock ()` : it releases the patient's limb once the fracture treatment is completed.

5.4.3 B-BROS2 Module

This module features several procedures which allow robotized fracture reduction when the automatic mode is triggered and direct robotized arm blocking when AM, SAM or DMB is triggered. B-BROS2 module releases

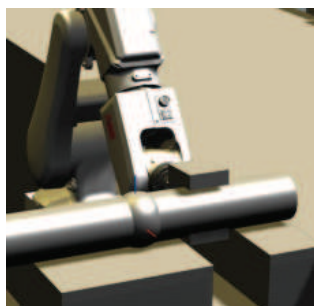


Figure 5.14: Blocking the patient's limb.

the patient's limb once the surgery is completed. We, hereafter, detail the procedures:

1. `B_BROS2_reduce_II (A, B, C, D)` : it performs the reduction of a type II fracture and takes into account the parameters that we defined in Section 5.2. Figure 5.4.3 illustrates a robotized fracture reduction,
2. `B_BROS2_unblock_II ()` : this procedure unblock the patient's limb suffering from a type II fracture once the surgery is completed,
3. `B_BROS2_reduce_III (A, B)` : it computes the rotation angle of the rotary disorder in the case of a type III fracture and, then, reduces the latter,
4. `B_BROS2_block ()` : the procedure blocks the limb at the forearm once a manual reduction is performed during SAM or DMB. Figure 5.4.3 shows how this is performed.



Figure 5.15: Robotized fracture re-
duction.



Figure 5.16: Blocking the fractured
limb at the forearm.

5.4.4 P-BROS Module

This section describes the behavior of P-BROS, the robotic arm performing fracture reduction according to its type and the triggered operating mode.

We point out that the used pinning technique is Judet's which we mentioned in Section 1.2.1. The orientation of the tool "Pinning" (Section 6.1), relatively to the coordinate system defined in Section 5.3.2, depends on the type of the fracture. Thus, the figures 5.17 and 5.18 respectively shows the orientation of "Pinning" in the case of a type II and a type III fractures.

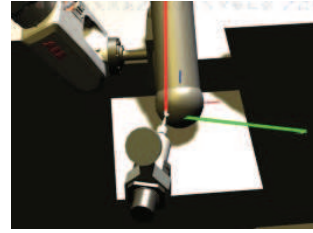
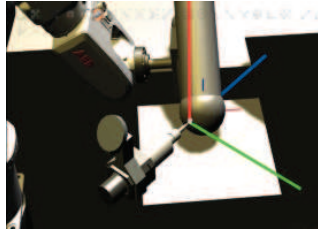


Figure 5.17: Orientation of "Pinning" in the case of a type II fracture. Figure 5.18: Orientation of "Pinning" in the case of a type III fracture.

The P-BROS module features several procedures that we hereafter detail:

1. P_BROS_DoublePin (A, B, C, D, HP) : it performs a parallel pinning using two pins inserted from the external condyle to the lateral humeral column in the case of a type II fracture which requires a double pinning. The procedure uses as parameters the four points of the distal dissolution and the width of the humeral palette (HP),
2. P_BROS_SinglePin_III (A, B, HP) : this procedure performs a percutaneous pinning for a type III fracture. The pin is actually inserted from the external condyle throughout the medial column in a rectilinear direction by keeping a fixed (XY) plane,
3. P_BROS_SinglePin_IV (A, B, HP) : it realizes a percutaneous pinning for a type IV fracture. Indeed, for this type of fracture, the pin is inserted in the lateral condyle and makes an angle of 45 relative to the orientation of the pin in the case of a type III fracture. The pin is inserted until reaching the lateral column.

5.4.5 Synchronization Module

We present, in this section, the synchronization module of the control unit. It is the entity that insures the coordination between the tasks of B-BROS1, B-BROS2 and P-BROS modules. To insure this function, we use interruptions through binary logic signals. Indeed, each signal corresponds to a very specific task. The signal is high when the task is running and low when it is idle or finished executing. We note that the used signals represent the steps

of a fracture treatment based on the operating mode and regardless to the nature of a given action (robotized or manual).

We define for the control unit the following logic signals which we detail in Table 5.4.5.

Table 5.1: Synchronization logic signals.

Logic Signal	Description
HandBlocking	This signal controls the first step of a fracture treatment which is blocking the patient's limb at the humerus. It is the highest priority task. The signal is high when B-BROS1 starts blocking the humerus and it switches to low once blocking is finished.
HandReduction	The signal controls the fracture reduction and the forearm blocking. It switches to high when HandBlocking is low and either B-BROS2 starts the robotized reduction and/or blocking or the surgeon starts the manual reduction and/or blocking. It is the second priority task.
HandPinning	HandPinning controls pinning, whether it is manual or robotized. It changes to high when the signal HandReduction changes to low informing, thus, that reduction and blocking are finished. When it switches to high, HandPinning starts pinning and switches to low once it is finished.

5.4.6 CU-MW Communication Module

A good communication between the control unit and the middleware is critical to the smooth functioning of BROS. For example, the control unit cannot start the different processing until it receives key parameters like the fracture type and the coordinates of the points of the distal fragment discontinuity. The module respects the diagram presented in Section 5.3.1.

5.4.7 Surgeon-Robot Interface

It is the graphical interface through which the surgeon communicates with the platform and oversees the progress of the operation. The surgeon can, using this interface, choose the operating mode to start with. Through this GUI, the surgeon consults any medical parameter like the fracture type, the displacement nature or the angle of the rotational trouble in the case of type III fractures. This interface meets the man-machine requirements like:

- Guidance: All resources used to guide the surgeon during the use of the interface like grouping/distinction, immediate feedback and legibility,
- Workload: Minimum and explicit actions ("start reduction", "start pinning"), informational density more or less acceptable for a surgeon,

- Error management: This is to protect sensitive actions against errors with error messages,
- Ergonomics: The interface must be flexible and adaptable to a surgeon and especially in an operating room.

Running example 4

The whole surgery was successfully performed by BROS under the automatic operating mode and simulated using RobotStudio and RobotWare. Only 4 fluoroscopic images were needed, what makes 21 images less than in the case study introduced in Section 3. BROS insured all the intervention steps and the surgeon had only to remotely check the smooth running of the surgery and be ready to intervene in the case where the robotized platform would not be able to perform one of the surgery's steps or he would judge that a human intervention is necessary.

5.4.8 Simulation and Test of BROS

To test the relevance and the accuracy of BROS, we decided to compare the pinning coordinates computed by the middleware (MC) and those where BROS really inserts the Kirschner wires into the fractured elbow (RC). Our medical partners provided us, thus, with forty-two real radiographies of supracondylar humeral fracture. The middleware sorted them according to Smida's classification as illustrated in Table 2.3.

Table 5.2: Sorting 42 SCH fractures' radiographies according to Lagrange's classification.

Type	Number
I	4
IIA	6
IIB	12
IIC	7
III	13

We assume that the error rate is the ratio of RC to MC . Since type-I fractures don't need pinning to be treated, the error rate is only calculated for type-IIA, IIB, IIC and III fractures [Smida 2007]. We obtain the graph illustrated in Figure 5.19.

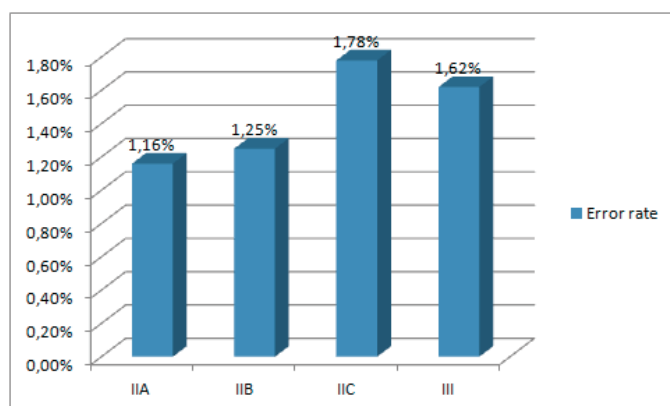


Figure 5.19: Error rates per SCH fracture type.

Based on the obtained results, the error rates never exceeded 2%. According to our medical partners, these error rates are very acceptable in themselves, which allows us to confirm that BROS is a safe and trustful platform for the robotized treatment of supracondylar humeral fractures.

5.5 Summary

Our work consisted, through this chapter, in implementing BROS on dedicated software. Through the simulation of a real case of BROS-assisted surgery, we proved the usefulness of this robotic platform to avoid the complications that may be generated because of the blind pinning and prevent the danger posed by the recurrent exposition to radiations. We can, now, certify that BROS is an innovating project which will be of a great help to pediatric orthopedic surgeons. The next step is to proceed to the real implementation of BROS using the ABB robotic arms.

Conclusion and Perspectives

Contents

6.1	Conclusion	104
6.2	Perspectives	104

6.1 Conclusion

This dissertation presents a new methodology called BROMETH to validate medical robotized platforms and insure their safety from specification to deployment. This methodology is original since it uses new technologies like the R-TNCES and a new tool, ZiZo. BROMETH is mainly established to validate medical robotized platforms, 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 during the last step of BROMETH. The latter will be the subject of our future work.

In this work, we defined new Petri Nets-based to model both tasks and resources of a reconfigurable system. A new Petri Nets-based editor and random-simulator named ZiZo is also developed to model and simulate the generated models. These new concepts and tools are applied on BROS as part of going through the specification and certification steps of BROMETH.

We also proposed a new UML profile, baptized R-UML (Reconfigurable UML), to model flexible control systems sharing adaptive shared resources. The profile is enriched with a PCP-based solution for the management of resource sharing which is defined in Chapter 3. The chapter 4 also presents an automatic translation of R-UML into R-TNCES to support model checking. BROS serves as an application of this new concept on a real study case. This chapter also introduces new solutions to automatize the generation of codes usable by Robot Operating System from R-UML models.

We, then, discussed the implementation step of BROS. It exposes and evaluates the results of the treatment of supracondylar humeral fracture by BROS. The development of the middleware and control unit of BROS are detailed in this chapter.

We, finally, discussed the shown results and drawn conclusions of the presented work presented. Future improvements that could enrich the work developed during this dissertation are proposed.

6.2 Perspectives

The BROMETH methodology turned out to be relevant to design, certify and implement a surgical robotic system dedicated to the treatment of humeral supracondylar fracture. Our purpose is now to continue developing this system to treat other sensitive surgeries, like the spinal cord and femoral neck ones. BROMETH is absolutely useful to design reconfigurable industrial systems featuring adaptive shared resources issues.

Bibliography

- [Allgeuer 2013] Philipp Allgeuer, Max Schwarz, Julio Pastrana, Sebastian Schueller, Marcell Missura and Sven Behnke. *A ROS-based software framework for the NimbRo-OP humanoid open platform*. In Proceedings of 8th Workshop on Humanoid Soccer Robots, IEEE-RAS Int. Conference on Humanoid Robots, Atlanta, USA, 2013. (Cited on page 76.)
- [Alur 1991] Rajeev Alur and Thomas A Henzinger. *Logics and models of real time: A survey*. In Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems), pages 74–106. Springer, 1991. (Cited on page 21.)
- [Arbelaez 2011] Pablo Arbelaez, Michael Maire, Charless Fowlkes and Jitendra Malik. *Contour detection and hierarchical image segmentation*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 33, no. 5, pages 898–916, 2011. (Cited on page 22.)
- [Bahill 2003] Terry Bahill and Jesse Daniels. *Using objected-oriented and UML tools for hardware design: A case study*. Systems Engineering, vol. 6, no. 1, pages 28–48, 2003. (Cited on page 54.)
- [Baier 2008] Christel Baier, Joost-Pieter Katoen *et al.* Principles of model checking, volume 26202649. MIT press Cambridge, 2008. (Cited on page 19.)
- [Barton 2001] Kelly L Barton, Cornelia K Kaminsky, Daniel W Green, Christopher J Shean, Steven M Kautz and David L Skaggs. *Reliability of a modified Gartland classification of supracondylar humerus fractures*. Journal of Pediatric Orthopaedics, vol. 21, no. 1, pages 27–30, 2001. (Cited on page 25.)
- [Baumgartl 2013] Johannes Baumgartl, Thomas Buchmann, Dominik Henrich and Bernhard Westfechtel. *Towards Easy Robot Programming-Using DSLs, Code Generators and Software Product Lines*. In IC-SOFT, pages 548–554, 2013. (Cited on page 25.)
- [Bernardi 2007] Simona Bernardi and José Merseguer. *Performance evaluation of UML design with Stochastic Well-formed Nets*. Journal of Systems and Software, vol. 80, no. 11, pages 1843–1865, 2007. (Cited on page 23.)
- [Bertaud 2008] Valérie Bertaud, Jérémy Lasbleiz, Fleur Mougin, Anita Burgun and Régis Duvauferrier. *A unified representation of findings in*

- clinical radiology using the UMLS and DICOM*. International journal of medical informatics, vol. 77, no. 9, pages 621–629, 2008. (Cited on page 2.)
- [Bondavalli 1999] Andrea Bondavalli, Istvan Majzik and Ivan Mura. *Automated dependability analysis of UML designs*. In Object-Oriented Real-Time Distributed Computing, 1999.(ISORC'99) Proceedings. 2nd IEEE International Symposium on, pages 139–144. IEEE, 1999. (Cited on page 55.)
- [Bradski 2008] Gary Bradski and Adrian Kaehler. Learning opencv: Computer vision with the opencv library. ” O'Reilly Media, Inc.”, 2008. (Cited on page 22.)
- [Bruyninckx 2001] Herman Bruyninckx. *Open robot control software: the OROCOS project*. In Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on, volume 3, pages 2523–2528. IEEE, 2001. (Cited on page 74.)
- [Bruyninckx 2013] Herman Bruyninckx, Markus Klotzbücher, Nico Hochgeschwender, Gerhard Kraetzschmar, Luca Gherardi and Davide 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, pages 1758–1764. ACM, 2013. (Cited on page 74.)
- [Cardoso 2001] Janette Cardoso and Christophe Sibertin-Blanc. *Ordering actions in sequence diagrams of UML*. In Information Technology Interfaces, 2001. ITI 2001. Proceedings of the 23rd International Conference on, pages 3–14. IEEE, 2001. (Cited on page 54.)
- [Cleary 2001] Kevin Cleary and Charles Nguyen. *State of the art in surgical robotics: clinical applications and technology challenges*. Computer Aided Surgery, vol. 6, no. 6, pages 312–328, 2001. (Cited on page 2.)
- [Clein 1954] Norman W Clein. *How safe is X-ray and fluoroscopy for the patient and the doctor?* The Journal of pediatrics, vol. 45, no. 3, pages 310–315, 1954. (Cited on page 4.)
- [Connolly 2009] Christine Connolly. *Technology and applications of ABB RobotStudio*. Industrial Robot: An International Journal, vol. 36, no. 6, pages 540–545, 2009. (Cited on page 88.)
- [Cortellessa 2000] Vittorio Cortellessa and Raffaella Mirandola. *Deriving a queueing network based performance model from UML diagrams*. In Proceedings of the 2nd international workshop on Software and performance, pages 58–70. ACM, 2000. (Cited on page 54.)

- [Costa 2012] Carlos Costa and José Luís Oliveira. *Telecardiology through ubiquitous Internet services*. International journal of medical informatics, vol. 81, no. 9, pages 612–621, 2012. (Cited on page 2.)
- [D’Ambrogio 2005] Andrea 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, pages 75–86. ACM, 2005. (Cited on pages 23 and 24.)
- [Douira-Khomsni 2012] Wiem Douira-Khomsni, Mahmoud Smida, Hela Louati, Zied Jlalía, Maher Ben Ghachem and Ibtissem Bellagha. *Multi slice computed tomography approach in the assessment of supracondylar humeral fractures in children*. Acta Orthopædica Belgica, vol. 78, no. 4, page 458, 2012. (Cited on page 25.)
- [Dubinin 2006] VN Dubinin, HM Hanisch and S Karras. *Building of reachability graph extractions using a graph rewriting system*. In (2006): . V . -. , pages 160–171, 2006. (Cited on page 19.)
- [Dumitru 2015] Violeta Cristina Dumitru and Mirela Cherciu. *Application of the FMEA concept to medical robotic system*. In Advanced Engineering Forum, volume 13, pages 324–331. Trans Tech Publ, 2015. (Cited on page 5.)
- [Fenton 1999] Norman E Fenton and Martin Neil. *A critique of software defect prediction models*. Software Engineering, IEEE Transactions on, vol. 25, no. 5, pages 675–689, 1999. (Cited on page 54.)
- [Flynn 1974] Joseph C Flynn, Joseph G Matthews and Roger L Benoit. *Blind pinning of displaced supracondylar fractures of the humerus in children*. J Bone Joint Surg Am, vol. 56, no. 2, pages 263–272, 1974. (Cited on page 4.)
- [Garg 2013] Adesh Garg, Timmy Siau, Dmitry Berenson, J Adam M Cunha, I-Chow Hsu, Jean Pouliot, Dan Stoianovici and Ken Goldberg. *Robot-guided open-loop insertion of skew-line needle arrangements for high dose rate brachytherapy*. Automation Science and Engineering, IEEE Transactions on, vol. 10, no. 4, pages 948–956, 2013. (Cited on page 2.)
- [Genter 2007] Goran Genter, Stjepan Bogdan, Zdenko Kovacic and Ivor Grubisic. *Software tool for modeling, simulation and real-time implementation of Petri net-based supervisors*. In 2007 IEEE International Conference on Control Applications, pages 664–669. IEEE, 2007. (Cited on page 19.)

- [Gérard 2010] Sébastien Gérard, Huascar Espinoza, François Terrier and Bran Selic. *6 Modeling Languages for Real-Time and Embedded Systems*. In *Model-Based Engineering of Embedded Real-Time Systems*, pages 129–154. Springer, 2010. (Cited on page 23.)
- [Gherbi 2006] Abdelouahed Gherbi and Ferhat Khendek. *UML Profiles for Real-Time Systems and their Applications*. *Journal of Object Technology*, vol. 5, no. 4, pages 149–169, 2006. (Cited on page 23.)
- [Gomes 2011] Paula Gomes. *Surgical robotics: Reviewing the past, analysing the present, imagining the future*. *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 2, pages 261–266, 2011. (Cited on page 2.)
- [Goodenough 1988] John B Goodenough and Lui Sha. The priority ceiling protocol: A method for minimizing the blocking of high priority ada tasks, volume 8. ACM, 1988. (Cited on pages 22 and 33.)
- [Gosens 2003] Taco Gosens and Karst J Bongers. *Neurovascular complications and functional outcome in displaced supracondylar fractures of the humerus in children*. *Injury*, vol. 34, no. 4, pages 267–273, 2003. (Cited on page 4.)
- [Grauman 2005a] Kristen Grauman and Trevor Darrell. *Efficient image matching with distributions of local invariant features*. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 627–634. IEEE, 2005. (Cited on page 22.)
- [Grauman 2005b] Kristen Grauman and Trevor Darrell. *Efficient image matching with distributions of local invariant features*. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 627–634. IEEE, 2005. (Cited on page 92.)
- [Group 2005] OM Groupet al. *Uml profile for schedulability, performance and time specification*. Version 1.1, formal/05-01, vol. 2, 2005. (Cited on page 23.)
- [Gu 2005] Gordon P Gu and Dorina C Petriu. *From UML to LQN by XML algebra-based model transformations*. In *Proceedings of the 5th international workshop on Software and performance*, pages 99–110. ACM, 2005. (Cited on page 23.)
- [Hamid 2010] Brahim Hamid and Fatma Krichen. *Model-based engineering for dynamic reconfiguration in DRTEs*. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pages 269–276. ACM, 2010. (Cited on page 23.)

- [Hanisch 1997] H-M Hanisch, J Thieme, A Luder and O Wienhold. *Modeling of PLC behavior by means of timed net condition/event systems*. In Emerging Technologies and Factory Automation Proceedings, 1997. ETFA'97., 1997 6th International Conference on, pages 391–396. IEEE, 1997. (Cited on pages 17 and 32.)
- [Hanisch 1999] HM Hanisch and A Lüder. *Modular modelling of closed-loop systems*. In Colloquium on Petri Net Technologies for Modelling Communication Based Systems, Berlin, Germany, pages 103–126, 1999. (Cited on page 16.)
- [Haque 2006] Maahir Ul Haque, Harry L Shufflebarger, Michael OBrien and Angel Macagno. *Radiation exposure during pedicle screw placement in adolescent idiopathic scoliosis: is fluoroscopy safe?* Spine, vol. 31, no. 21, pages 2516–2520, 2006. (Cited on page 4.)
- [Haux 2006] Reinhold Haux. *Health information systems—past, present, future*. International journal of medical informatics, vol. 75, no. 3, pages 268–281, 2006. (Cited on page 2.)
- [Hoeckelman 2015] Mathias Hoeckelman, Imre Rudas, Paolo Fiorini, Frank Kirchner and Tamas Haidegger. *Current capabilities and development potential in surgical robotics*. International Journal of Advanced Robotic Systems, vol. 12, no. 61, pages 1–39, 2015. (Cited on page 2.)
- [Intel 2012] Willow Garage Intel. *Open Source Computer Vision Library*. URL <http://opencv.willowgarage.com>. Retrieved May, 2012. (Cited on page 22.)
- [ISO 2005] ISO. *ISO/IEC 19502:2005 Information Technology Meta Object Facility (MOF)*. 2005. (Cited on page 24.)
- [Judet 1953] JEAN Judet. *Traitement des fractures sus-condyliennes transversales de l'humérus chez l'enfant*. Rev Chir Orthop, vol. 39, pages 199–212, 1953. (Cited on page 9.)
- [Khalgui 2011] Mohamed Khalgui, Olfa Mosbahi, Zhiwu Li and Hans-Michael Hanisch. *Reconfiguration of distributed embedded-control systems*. IEEE/ASME Transactions on Mechatronics, vol. 16, no. 4, pages 684–694, 2011. (Cited on page 32.)
- [King 1999] Peter King and Rob Pooley. *Using UML to derive stochastic Petri net models*. In Proceedings of the 15th UK Performance Engineering Workshop, pages 45–56, 1999. (Cited on page 55.)
- [Klotzbücher 2012] Markus Klotzbücher and Herman Bruyninckx. *Coordinating robotic tasks and systems with rFSM statecharts*. JOSER:

- Journal of Software Engineering for Robotics, vol. 3, no. 1, pages 28–56, 2012. (Cited on page 74.)
- [Koh 1991] I Koh and F DiCesare. *Checking liveness in Petri nets using synchronic concepts*. In Proceedings of the Korean Automatic Control Conference, 1991. (Cited on page 33.)
- [Kooijmans 2007] Tijn Kooijmans, Takayuki Kanda, Christoph Bartneck, Hiroshi Ishiguro and Norihiro Hagita. *Accelerating Robot Development Through Integral Analysis of Human–Robot Interaction*. Robotics, IEEE Transactions on, vol. 23, no. 5, pages 1001–1012, 2007. (Cited on page 2.)
- [Koubâa 2016] Anis Koubâa. Robot operating system (ros): The complete reference, volume 1. Springer, 2016. (Cited on page 24.)
- [Kouskoulas 2013] Yanni Kouskoulas, David Renshaw, André Platzer and Peter Kazanzides. *Certifying the safe design of a virtual fixture control algorithm for a surgical robot*. In Proceedings of the 16th international conference on Hybrid systems: computation and control, pages 263–272. ACM, 2013. (Cited on page 5.)
- [Kumar 2015] Pranav Srinivas Kumar, William Emfinger, Amogh Kulkarni, Gabor Karsai, Dexter Watkins, Benjamin Gasser, Cameron Ridgewell and Amrutur Anilkumar. *ROSMOD: a toolsuite for modeling, generating, deploying, and managing distributed real-time component-based software using ROS*. In 2015 International Symposium on Rapid System Prototyping (RSP), pages 39–45. IEEE, 2015. (Cited on page 24.)
- [Kuo 2004] Christina E Kuo and Roger F Widmann. *Reduction and percutaneous pin fixation of displaced supracondylar elbow fractures in children*. Techniques in Shoulder & Elbow Surgery, vol. 5, no. 2, pages 90–102, 2004. (Cited on page 25.)
- [Kutz 2003] Myer Kutz. Standard handbook of biomedical engineering and design. McGraw-Hill, 2003. (Cited on page 2.)
- [Kwoh 1988] Yik San Kwoh, Joahin Hou, Edmond A Jonckheere and Samad Hayati. *A robot with improved absolute positioning accuracy for CT guided stereotactic brain surgery*. Biomedical Engineering, IEEE Transactions on, vol. 35, no. 2, pages 153–160, 1988. (Cited on page 2.)
- [Lagrange 1962] J Lagrange and P Rigault. *Fractures supra-condyliennes*. Rev Chir Orthop, vol. 48, pages 337–414, 1962. (Cited on page 25.)

- [Lam 2007] Vitus SW Lam. *A formalism for reasoning about UML activity diagrams*. Nordic Journal of Computing, vol. 14, no. 1, pages 43–64, 2007. (Cited on page 55.)
- [Landin 1983] Lennart A Landin. *Fracture Patterns in Children: Analysis of 8,682 Fractures with Special Reference to Incidence, Etiology and Secular Changes in a Swedish Urban Population 1950–1979*. Acta Orthopaedica Scandinavica, vol. 54, no. sup202, pages 3–109, 1983. (Cited on page 4.)
- [Landin 1986] Lennart A Landin and Lars G Danielsson. *Elbow fractures in children: an epidemiological analysis of 589 cases*. Acta orthopaedica Scandinavica, vol. 57, no. 4, pages 309–312, 1986. (Cited on page 4.)
- [Lilius 1999] Johan Lilius and Iván Porres Paltor. *The production cell: An exercise in the formal verification of a UML model*. 1999. (Cited on page 54.)
- [Llobet 2007] Rafael Llobet, Juan C Pérez-Cortés, Alejandro H Toselli and Alfons Juan. *Computer-aided detection of prostate cancer*. international journal of medical informatics, vol. 76, no. 7, pages 547–556, 2007. (Cited on page 2.)
- [Lobov 2005] Andrei Lobov, JL Martinez Lastra and Reijo Tuokko. *Application of UML in plant modeling for model-based verification: UML translation to TNCES*. In Industrial Informatics, 2005. INDIN’05. 2005 3rd IEEE International Conference on, pages 495–501. IEEE, 2005. (Cited on pages 58 and 61.)
- [Mallet 2009] Frédéric Mallet and Charles 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, pages 305–312. IEEE, 2009. (Cited on page 23.)
- [Marcus 2013] Hani Marcus, Dipanjan Nandi, Ara Darzi and Guang-Zhong Yang. *Surgical robotics through a keyhole: from today’s translational barriers to tomorrow’s disappearing robots*. Biomedical Engineering, IEEE Transactions on, vol. 60, no. 3, pages 674–681, 2013. (Cited on page 2.)
- [Martelli 2000] M Martelli, M Marcacci, L Nofrini, F La Palombara, A Malvisi, F Iacono, P Vendruscolo and M Pierantoni. *Computer- and robot-assisted total knee replacement: analysis of a new surgical procedure*. Annals of biomedical engineering, vol. 28, no. 9, pages 1146–1153, 2000. (Cited on page 2.)

- [Martelli 2003] Sandra Martelli, Laura Nofrini, Paolo Vendruscolo and Andrea Visani. *Criteria of interface evaluation for computer assisted surgery systems*. International journal of medical informatics, vol. 72, no. 1, pages 35–45, 2003. (Cited on page 2.)
- [Martin 2001] Grant Martin, Luciano Lavagno and Jean Louis-Guerin. *Embedded UML: a merger of real-time UML and co-design*. In Proceedings of the ninth international symposium on Hardware/software codesign, pages 23–28. ACM, 2001. (Cited on page 23.)
- [Marzolla 2004] Moreno Marzolla and Simonetta Balsamo. *UML-PSI: the UML performance simulator*. In Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the, pages 340–341. IEEE, 2004. (Cited on pages 23 and 24.)
- [Membarth 2012] Richard Membarth, Frank Hannig, Jürgen Teich, Mario Körner and Wieland Eckert. *Generating device-specific GPU code for local operators in medical imaging*. In Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, pages 569–581. IEEE, 2012. (Cited on page 22.)
- [Merseguer 2002] José Merseguer, Javier Campos, Simona Bernardi and Susanna Donatelli. *A compositional semantics for UML state machines aimed at performance evaluation*. In Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on, pages 295–302. IEEE, 2002. (Cited on page 23.)
- [Mikaelsson 2009] Pierre Mikaelsson and Mark 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, pages 39–41, 2009. (Cited on page 88.)
- [Mikk 1998] Erich Mikk, Yassine Lakhnech, Michael Siegel and Gerard J Holzmann. *Implementing statecharts in PROMELA/SPIN*. In Industrial Strength Formal Specification Techniques, 1998. Proceedings. 2nd IEEE Workshop on, pages 90–101. IEEE, 1998. (Cited on page 55.)
- [Murata 2002] Satoshi Murata, Eiichi Yoshida, Akiya Kamimura, Haruhisa Kurokawa, Kohji Tomita and Shigeru Kokaji. *M-TRAN: Self-reconfigurable modular robotic system*. IEEE/ASME transactions on mechatronics, vol. 7, no. 4, pages 431–441, 2002. (Cited on page 74.)
- [Nolden 2013] Marco Nolden, Sascha Zelzer, Alexander Seitel, Diana Wald, Michael Müller, Alfred M Franz, Daniel Maleike, Markus Fangerau, Matthias Baumhauer, Lena Maier-Heinet *al.* *The medical imaging interaction toolkit: challenges and advances*. International journal of

- computer assisted radiology and surgery, vol. 8, no. 4, pages 607–620, 2013. (Cited on page 22.)
- [Petriu 2002] Dorina C Petriu and Hui Shen. *Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications*. In *Computer Performance Evaluation: Modelling Techniques and Tools*, pages 159–177. Springer, 2002. (Cited on page 24.)
- [Petriu 2007] Dorin B Petriu and Murray Woodside. *An intermediate meta-model with scenarios and resources for generating performance models from UML designs*. *Software & Systems Modeling*, vol. 6, no. 2, pages 163–184, 2007. (Cited on page 23.)
- [Pirone 1988] AM Pirone, HK Graham and Joseph Ivan Krajbich. *Management of displaced extension-type supracondylar fractures of the humerus in children*. *J Bone Joint Surg Am*, vol. 70, no. 5, pages 641–650, 1988. (Cited on page 25.)
- [Quigley 2009] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler and Andrew Y Ng. *ROS: an open-source Robot Operating System*. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009. (Cited on pages 24, 25 and 74.)
- [Rahimi 1991] Mansour Rahimi and Xia Xiadong. *A framework for software safety verification of industrial robot operations*. *Computers & industrial engineering*, vol. 20, no. 2, pages 279–287, 1991. (Cited on page 5.)
- [Rampersaud 2000a] Y Raja Rampersaud, Kevin T Foley, Alfred C Shen, Scott Williams and Milo Solomito. *Radiation exposure to the spine surgeon during fluoroscopically assisted pedicle screw insertion*. *Spine*, vol. 25, no. 20, pages 2637–2645, 2000. (Cited on page 4.)
- [Rampersaud 2000b] Y Raja Rampersaud, Kevin T Foley, Alfred C Shen, Scott Williams and Milo Solomito. *Radiation exposure to the spine surgeon during fluoroscopically assisted pedicle screw insertion*. *Spine*, vol. 25, no. 20, pages 2637–2645, 2000. (Cited on page 94.)
- [Ratzer 2003] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen and Kurt Jensen. *CPN tools for editing, simulating, and analysing coloured Petri nets*. In *International Conference on Application and Theory of Petri Nets*, pages 450–462. Springer, 2003. (Cited on page 19.)

- [Rausch 1995] Mathias Rausch and H-M Hanisch. *Net condition/event systems with multiple condition outputs*. In Emerging Technologies and Factory Automation, 1995. ETFA'95, Proceedings., 1995 INRIA/IEEE Symposium on, volume 1, pages 592–600. IEEE, 1995. (Cited on page 16.)
- [Riviere 2006] Cameron N Riviere, Jacques Gangloff and Michel De Mathelin. *Robotic compensation of biological motion to enhance surgical accuracy*. PROCEEDINGS-IEEE, vol. 94, no. 9, page 1705, 2006. (Cited on page 2.)
- [Roch 2000a] Stephan Roch. *Extended computation tree logic*. In Workshop Concurrency, Speci & Programming, number 140 in Informatik-Bericht. Citeseer, 2000. (Cited on pages 20 and 21.)
- [Roch 2000b] Stephan Roch. *Extended computation tree logic: Implementation and application*. 2000. (Cited on pages 20 and 21.)
- [Rouff 2012] Christopher Rouff, Richard Buskens, Laura Pullum, Xiaohui Cui and Mike Hinchey. *The AdaptiV approach to verification of adaptive systems*. In Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering, pages 118–122. ACM, 2012. (Cited on page 19.)
- [Rumbaugh 1991] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William E. Lorensen *et al.* Object-oriented modeling and design, volume 199. Prentice-hall Englewood Cliffs, 1991. (Cited on page 60.)
- [Salem 2014] Mohamed Oussama Ben Salem, Olfa Mosbahi and Mohamed Khalgui. *PCP-based Solution for Resource Sharing in Reconfigurable Timed Net Condition/Event Systems*. In ADECS 2014, Proceedings of the 1st International Workshop on Petri Nets for Adaptive Discrete-Event Control Systems, co-located with 35th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2014), Tunis, Tunisia, June 24, 2014., pages 52–67, 2014. (Cited on pages 56, 59 and 83.)
- [Salem 2015a] Mohamed Oussama Ben Salem, Olfa Mosbahi, Mohamed Khalgui and Georg Frey. *BROS - A New Robotic Platform for the Treatment of Supracondylar Humerus Fracture*. In HEALTH-INF 2015 - Proceedings of the International Conference on Health Informatics, Lisbon, Portugal, 12-15 February, 2015, pages 151–163, 2015. (Cited on page 55.)

- [Salem 2015b] Mohamed Oussama Ben Salem, Olfa Mosbahi, Mohamed Khalgui and Georg Frey. *ZiZo: Modeling, Simulation and Verification of Reconfigurable Real-Time Control Tasks Sharing Adaptive Resources. Application to the Medical Project BROS*. In HEALTH-INF 2015 - Proceedings of the International Conference on Health Informatics, Lisbon, Portugal, 12-15 February, 2015, pages 20–31, 2015. (Cited on pages 55, 69, 79 and 83.)
- [Schlegel 2004] Christian Schlegel. *Navigation and execution for mobile robots in dynamic environments: an integrated approach*. PhD thesis, University of Ulm, 2004. (Cited on page 74.)
- [Selic 1998] Bran Selic. *Using UML for modeling complex real-time systems*. In Languages, Compilers, and Tools for Embedded Systems, pages 250–260. Springer, 1998. (Cited on page 23.)
- [Sha 1990] Lui Sha, Rangunathan Rajkumar and John P Lehoczky. *Priority inheritance protocols: An approach to real-time synchronization*. IEEE Transactions on computers, vol. 39, no. 9, pages 1175–1185, 1990. (Cited on page 33.)
- [Shousha 2012] Marwa Shousha, Lionel Briand and Yvan 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, pages 354–374, 2012. (Cited on page 23.)
- [sit a] *The Open Source Robotics Foundation*. <http://www.osrfoundation.org/>. Accessed: 2016-09-14. (Cited on page 24.)
- [sit b] *ZiZo: a tool to model, simulate and verify reconfigurable real time control systems*. <http://www.aut.uni-saarland.de/forschung/forschung-zizo-tool-bensalem/>. Accessed: 2016-06-16. (Cited on page 45.)
- [Smida 2007] M Smida, H Smaoui, T Ben Jlila, W Saeid, H Safi, C Ammar, C Jalel and M Ben Ghachem. *Un index de stabilité pour lembrochage percutané latéral parallèle des fractures supracondyliennes du coude chez lenfant*. Revue de Chirurgie Orthopédique et Réparatrice de l’Appareil Moteur, vol. 93, no. 4, page 404, 2007. (Cited on pages 9 and 100.)
- [Smith 2005] Connie U Smith, Catalina M Lladó, Vittorio Cortellessa, Antiniscia Di Marco and Lloyd 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, pages 87–98. ACM, 2005. (Cited on page 23.)

- [Speers 2013] Andrew Speers, P Mojiri Forooshani, Michael Dicke and Michael Jenkin. *Lightweight tablet devices for command and control of ROS-enabled robots*. In Advanced Robotics (ICAR), 2013 16th International Conference on, pages 1–6. IEEE, 2013. (Cited on page 77.)
- [Staines 2007] Anthony Spiteri Staines. *A Comparison of Software Analysis and Design Methods for Real Time Systems*. In Proceedings of World Academy of Science, Engineering and Technology, volume 21. Citeseer, 2007. (Cited on page 23.)
- [Starke 2002] Peter H Starke and Stephan Roch. Analysing signal-net systems. Professoren des Inst. für Informatik, 2002. (Cited on pages 19, 43, 45, 51 and 69.)
- [Suender 2011] Christoph Suender, Valeriy Vyatkin and Alois Zoitl. *Formal validation of downtimeless system evolution in embedded automation controllers*. ACM Transactions on Embedded Control Systems, 2011. (Cited on page 19.)
- [Sun 2007] Yan Sun, Bruce McMillin, Xiaoqing Liu and David Cape. *Verifying noninterference in a cyber-physical system the advanced electric power grid*. In Seventh International Conference on Quality Software (QSIC 2007), pages 363–369. IEEE, 2007. (Cited on page 5.)
- [Tsai 1987] Roger Tsai. *A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses*. IEEE Journal on Robotics and Automation, vol. 3, no. 4, pages 323–344, 1987. (Cited on page 92.)
- [Vendruscolo 2001] Paolo Vendruscolo and Sandra Martelli. *Interfaces for computer and robot assisted surgical systems*. Information and software technology, vol. 43, no. 2, pages 87–96, 2001. (Cited on page 2.)
- [Wang 2006] Yulun Wang, Steven E Butner and Ara Darzi. *The developing market for medical robotics*. PROCEEDINGS-IEEE, vol. 94, no. 9, page 1763, 2006. (Cited on page 2.)
- [Wang 2015] Xi Wang, Imen Khemaissia, Mohamed Khalgui, ZhiWu Li, Olfa Mosbahi and MengChu Zhou. *Dynamic low-power reconfiguration of real-time systems with periodic and probabilistic tasks*. Automation Science and Engineering, IEEE Transactions on, vol. 12, no. 1, pages 258–271, 2015. (Cited on page 18.)
- [Warmer 1998] Jos B Warmer and Anneke G Kleppe. *The Object Constraint Language: Precise Modeling With UML (Addison-Wesley Object Technology Series)*. 1998. (Cited on page 54.)

- [Woodside 2005] Murray Woodside, Dorina C Petriu, Dorin B Petriu, Hui Shen, Toqeer Israr and Jose Merseguer. *Performance by unified model analysis (PUMA)*. In Proceedings of the 5th international workshop on Software and performance, pages 1–12. ACM, 2005. (Cited on page 23.)
- [Xu 2015] Yi Xu, Ying Mao, Xianqiao Tong, Huan Tan, Weston B Griffin, Balajee Kannan and Lynn A DeRose. *Robotic Handling of Surgical Instruments in a Cluttered Tray*. Automation Science and Engineering, IEEE Transactions on, vol. 12, no. 2, pages 775–780, 2015. (Cited on page 2.)
- [Zhang 2013] Jiafeng Zhang, Mohamed Khalgui, Zhiwu Li, Olfa Mosbahi and Abdulrahman 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, pages 757–772, 2013. (Cited on pages 18, 32, 33, 36 and 69.)
- [Zhang 2015] Jiafeng Zhang, Mohamed Khalgui, Zhiwu Li, Georg Frey, Olfa Mosbahi and Hela Ben Salah. *Reconfigurable coordination of distributed discrete event control systems*. Control Systems Technology, IEEE Transactions on, vol. 23, no. 1, pages 323–330, 2015. (Cited on page 18.)
- [Zhou 1991] MengChu Zhou and Frank DiCesare. *Parallel and sequential mutual exclusions for Petri net modeling of manufacturing systems with shared resources*. IEEE Transactions on Robotics and Automation, vol. 7, no. 4, pages 515–527, 1991. (Cited on page 33.)

Tag des Kolloquiums:

Dekan:

Berichterstatte:

Vorsitz:

Akad. Mitarbeiter: