IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 26, 2020, accepted April 7, 2020, date of publication April 10, 2020, date of current version April 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2987099

Robotic System Specification Methodology Based on Hierarchical Petri Nets

MAKSYM FIGAT, (Graduate Student Member, IEEE), AND CEZARY ZIELIŃSKI, (Senior Member, IEEE)
Warsaw University of Technology, Institute of Control and Computation Engineering, 00-665 Warsaw, Poland

Corresponding author: Maksym Figat (maksym.figat@pw.edu.pl)

This work was supported in part by the National Science Centre, Poland, under Grant 2017/25/N/ST7/00900, and in part by the Dean of the Faculty of Electronics and Information Technology, Warsaw University of Technology, Poland.

ABSTRACT The paper presents a methodology of creating a Hierarchical Petri Net modelling the activities of a multi-agent robotic system. The methodology follows the separation of concerns approach to the design of robot control software, thus five layers resulted, representing: the system composed of agents, agents' subsystems, behaviours of subsystems, behaviour pattern, and finally inter-subsystem communication and transition function calculation. Blocking and non-blocking communication modes are taken into account. The robotic system structure and its activities are specified using the developed Robotic System HPN Tool. It facilitates modeling HPNs, verification of the activities of a robotic system through the HPN simulation and automatic code generation of an equivalent ROS based system. The specification methodology is presented on a simple example of designing a controller for the LWR4+ robot.

INDEX TERMS Robotic system specification methodology, robotic system design methodology, communication model, hierarchical petri net.

I. INTRODUCTION

Robotic control systems are inherently complex, hence the process of developing them requires both appropriate development methods and tools [1]. Usually robot programming frameworks supplemented by design experience are exploited. Frameworks provide use patterns, communication middleware and libraries of modules, all treated as building blocks out of which systems are constructed. This approach is completely focused on the implementation of a robotic system, and the quality of the architecture of the created system depends on the experience of the programmer. Among the best-known frameworks used in robotics are: OpenRTMaist [34], MARIE [34], ASEBA [34], Player [35], CLARAty [36], MIRO [37], ORCA [38], ROS [39], OROCOS [40], [41], MRROC++ [42], G^{en}oM [43], DCA [44], TCA [45], TDL [46], Generis [47], CoolBOT [48]. Nevertheless, the architecture of many of the already developed robotic systems is not obvious, because they have been most often created without a clear architectural design [5].

The development method reflects the underlying robotic system architecture, thus its definition is of relevance. The robotic architecture [2] provides the principled way of organizing a control system. It provides the structure of the robotic system and imposes constraints on the way the control problem can be solved. Another definition [3] states that the architecture is an abstract design of a class of interconnected components in which perception, reasoning and action occur. The briefest definition states that an architecture describes a set of architectural components and their interaction [4]. The common part of those definitions is the distinction of the structure and the activities of the system.

An agent is a system rationally affecting its surroundings, relying on the information collected from the environment [6]. Diverse structures of robotic systems composed of agents have been defined, e.g., [2]–[4], [7]. Usually the structure is defined in terms of a class of agents composed of a set of structural components in which perception, reasoning and action occur [3]. Communicating agents together with devices forming the control system were presented in [8], [9].

Many attempts have been made to treat the subject formally [10]–[13]. One of the formal approaches is based on Petri nets, which have been also utilised in robotics, e.g.: [14]–[18], [20]–[22]. In the above-mentioned works Petri nets are used for modeling, planning, fault tolerance analysis and supervisory tasks. However, the Petri net potential is also visible in the context of designing robotic systems. Some of those works treat Petri nets mainly as a formal tool

The associate editor coordinating the review of this manuscript and approving it for publication was Christopher Kitts.

used to model robotic systems and verify their correctness. However, some designers use Petri nets in the whole cycle of the development of a robotic system, i.e. system modeling, verification, code generation and validation of the developed model. Unfortunately, most of the above works do not take into account all aspects of the robotic system activities, e.g. they do not disclose all aspects of interactions between the communicating subsystems. Even if they reveal the communication models, they limit the discussion to only two basic models: fully asynchronous – the producer and the consumer act in the non-blocking mode or fully synchronous – both act in the blocking mode. However, there are other possibilities. In the majority of papers, the methodology presenting how to build a Petri net describing the activity of the robotic system is missing. As a result, it is difficult or even impossible to assign the obtained Petri net or a part of it to a particular element of the robotic system. Such Petri nets do not foster any particular robot system architecture. There are many different types of Petri networks, e.g.: colored, time, timed, stochastic, hierarchical. General networks, which are not limited to a specific domain, robotics in our case, significantly hamper the development of a robotic system, especially its modeling and analysis. Thus we introduce in this article a multi-agent robotic system specification methodology (RSSM), which is based on a specific hierarchical Petri net (HPN) [23], [24] tailored to fit the model of a multi-agent robotic system. The RSSM methodology is based on the architecture of an embodied agent [8], [13], [25]. However, in the mentioned papers the communication and computation model within the HPN [23], [24] were not treated comprehensively. In this article, we introduce the RSSM for defining the HPN, which describes all aspects of the structure and activities of a robotic system. The resulting HPN consists of layers connected in a strictly defined way. By tailoring the HPN to the requirements of an embodied agent meta-model, we get a simplified network, which is easier to verify than the general networks used in the above-mentioned articles. Moreover, in contrast to the above articles, our approach is supplemented by a methodology indicating how to design a robotic system and how to specify it utilising a HPN. This methodology uses HPN to define system activities at its all levels. Verification of the thus constructed HPN can be done using RSHPN Tool or using general purpose tools available for that purpose, and if the net is positively verified, it is transformed automatically a robotic controller C++ code implemented as a ROS system [39]. Subsequently, the obtained robot controller is also validated by simulation.

The produced HPN models the activities of subsequent layers of the designed robotic system, i.e. multi-agent robotic layer, agent layer, subsystem layer, behaviour layer, and the action layer. The action layer models: 1) the transition function decomposed into two sublayers based on: canonical decomposition and data availability, and 2) the interaction between communicating subsystems decomposed into two sublayers based on: communication arrangement and communication mode. Subsystems may communicate with each other using other than just synchronous and asynchronous models.

The structure of the paper is as follows. Introduction to HPNs used here is presented in Sec. II. The next section reveals the RSSM methodology defining the structure and activities of a robotic system. The activities are modeled by a five layered HPN. Sec. IV provides a detailed description of each layer of the HPN modeling a multi-agent robotic system. Sec. V provides an example of a single-robot system specified utilising the presented methodology. The conclusions are drawn in Sec. VI.

II. HIERARCHICAL PETRI NET

A HPN is a bipartite graph consisting of transitions t and either places p or pages $\mathcal{P}$, alternatively connected by directed arcs [23], [24]. Each place is associated with a single operation $\mathcal{O}$, while a transition with a single condition $\mathcal{C}$. Each page $\mathcal{P}$, which is a HPN itself, has a distinguished single input place $p_{\text{in}}^{\text{fusion}}$ and a single output place $p_{\text{out}}^{\text{fusion}}$. A HPN containing tokens in its places is a marked HPN $\mathcal{H}$ [23], [24]. Two places may be fused with each other. Fused places p^{fusion} represent the same place, however appearing in different nets [27]. Fusing places enables the connection of nets. The fusion of two places: $p_{\mathcal{H}_1,\alpha}^{\text{fusion}}$ and $p_{\mathcal{H}_2,\beta}^{\text{fusion}}$, belonging respectively to nets: $\mathcal{H}_1$ and $\mathcal{H}_2$ is represented by a single place $p_{(\mathcal{H}_1,\mathcal{H}_2),(\alpha,\beta)}^{\text{fusion}}$. A simple example of fused places is presented in Fig. 5.

Graphical representation of the components of HPN $\mathcal{H}$ is as follows: rectangles represent transitions, single circles are places, while double circles are pages, arrows are directed arcs, black filled circles represent tokens, and predicates forming conditions $\mathcal{C}$ are placed within square brackets. If a condition $\mathcal{C}$ is always fulfilled (i.e. is True) then it may be omitted. Similarly, if an operation $\mathcal{O}$ associated with a place does not produce any result it may also be omitted. This paper considers only safe HPNs, i.e. each place contains at the most one token.

III. ROBOTIC SYSTEM SPECIFICATION METHODOLOGY (RSSM)

RSSM introduces the general robotic system architecture, which is used to define any multi-agent robotic system. It determines both the robotic system structure and its activities. The structure presents the entities out of which the robotic system consists and interconnections between them. The activities define how those entities behave and interact in order to perform the entrusted task. Furthermore, RSSM contains the robotic system development procedure, which shows the consecutive steps indicating how to obtain the required model of the robotic system. Last but not least, RSSM provides the Robotic System HPN Tool (RSHPN Tool) facilitating the system development process: construction of HPN, verification of HPN through simulating the execution of the HPN (e.g. detection of deadlocks) and C++ code generation of the robotic system controller implemented

as a ROS system [39]. The RSHPN Tool is only mentioned in this paper, but it is available online at [28].

A. RSSM – STRUCTURE

RSSM defines the structure of the designed system in terms of embodied agents [8], [25]. An embodied agent a_j (j – represents the name of the agent) consists of the following subsystems: the control subsystem c_j, real receptors $R_{j,l}$ (l – represents the name of a real receptor), real effectors $E_{j,h}$ (h – represents the name of a real effector), virtual effectors $e_{j,n}$ (n – represents the name of a virtual effector) and virtual receptors $r_{j,k}$ (k – represents the name of a virtual receptor). Receptors $R_{j,l}$ read the data from the environment and deliver it in the form aggregated by $r_{j,k}$ to c_j. Based on the received data, c_j sends control commands, which are adequately transformed by $e_{j,n}$, to $E_{j,h}$, in order to affect the environment. The internal structure of an embodied agent a_j is presented in Fig. 1.

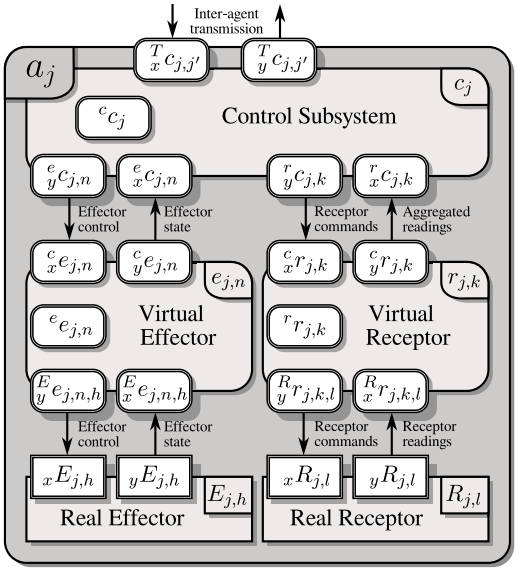


FIGURE 1. General embodied agent structure.

Each subsystem of an agent contains: internal memory, input buffers and output buffers. Input and output buffers are used to communicate between subsystems. Communication between agents occurs only between their control subsystems. Internal memory and buffers are named systematically [13], [25]. The name consists of a center letter indicating the type of the subsystem s, where $s \in \{c, e, r, E, R\}$; the left subscript, which denotes: the buffer type, x – input buffer, y – output buffer, no subscript – internal memory; the left superscript s' defines the type of subsystem with which the buffer communicates: $s' \in \{c, e, r, E, R, T\}$, where T stands for inter-agent transmission buffer; the right superscript denotes the discrete time stamp i; the right subscripts for inter-agent communication determine the names of two communicating agents, while for intra-agent communication denote the names of the agent and subsystem in which the buffer resides, as well as, if necessary, the buffer component. Although each subsystem may have its own internal clock, running at its specific frequency, the discrete time stamp is always denoted as i – thus those are different i-s. If a certain subsystem is unique within an agent a_j its name is omitted, e.g. c_j, as only a single control subsystem exists within an agent. Examples: 1) ${}^{T}_{x}c^{i}_{j,j'}$ is the input buffer receiving data from the control subsystem $c_{j'}$ of the agent $a_{j'}$ at a discrete time i; this input buffer is a part of the control subsystem c_j of the agent a_j, 2) ${}^{e}e^{i}_{j,n}$ is the internal memory of the virtual effector named n of and agent a_j at time i. No distinction is made for the name of a buffer or memory and its contents, for contextual obviousness.

B. RSSM – ACTIVITIES

The overall activity of a multi-agent robotic system results from the activities of its agents a_j, and those depend on the activities of their subsystems $s_{j,v}$. Each subsystem executes its behaviours ${}^{s}\mathcal{B}_{j,v,\omega}$ (ω – behaviour designator). Behaviour iteratively executes an elementary action ${}^{s}\mathcal{A}_{j,v,\omega}$ [13] and checks the terminal condition ${}^{s}f^{\tau}_{j,v,\xi}$ and error condition ${}^{s}f^{\varepsilon}_{j,v,\beta}$, where ξ and β are the designators of the terminal and error conditions respectively. The behaviour terminates if one of the above-mentioned conditions is fulfilled. In that case the subsystem $s_{j,v}$ switches to another behaviour, selected on the basis of the fulfilled initial condition ${}^{s}f^{\sigma}_{j,v,\alpha}$($\alpha$ – predicate designator) [13].

An elementary action ${}^{s}\mathcal{A}_{j,v,\omega}$ does the following: 1) calculates the transition function ${}^{s}f_{j,v,\omega}$, which takes as arguments the current data in the input buffers ${}_{x}s_{j,v}$ and internal memory ${}^{s}s_{j,v}$, and calculates the new values inserted into output buffers ${}_{y}s_{j,v}$ and internal memory ${}^{s}s_{j,v}$; 2) transmits data from ${}_{y}s_{j,v}$ to the associated subsystems, 3) increments the discrete time counter i, 4) receives data from the associated subsystems inserting it into ${}_{x}s_{j,v}$. Each elementary action employs its own transition function. A transition function ${}^{s}f_{j,v,\omega}$ is composed of partial transition functions ${}^{s,s'}f_{j,v,\omega,\psi}$, where ψ designates the partial function:

$$\left({}^{s}s^{i+1}_{j,v},\ {}_{y}s^{i+1}_{j,v}\right) := {}^{s,s'}f_{j,v,\omega,\psi}\left({}^{s}s^{i}_{j,v},\ {}_{x}s^{i}_{j,v}\right), \tag{1}$$

The two components of the left superscript denote the type of subsystem to which the transition function belongs (s) and type of the subsystem that this function calculates its results for (s'). Partial functions result from the canonical decomposition of ${}^{s}f_{j,v,\omega}$ based on the subsystem that output buffer is connected to or the internal memory. The canonical decomposition requires that each of the partial functions produces output for a disjoint subsystem or its own memory, thus no data collision occurs.

Communicating subsystems during data transfer may block their other activities or not [1], thus they operate either in blocking or non-blocking mode. Blocking mode used by the producer results in it waiting until the consumer confirms data reception, the non-blocking mode enables it to resume its further activities immediately after dispatching the data.

Blocking mode used by the consumer causes it to wait until it receives fresh data, while the non-blocking mode causes it to read the new data, if it is available, and, if not, immediately resume its other activities. As a result, four possible communication models emerge: 1) fully asynchronous – both the producer and the consumer act in the non-blocking mode, 2) fully synchronous – both act in the blocking mode, 3) producer acts in blocking mode while the consumer acts in non-blocking mode, 4) producer acts in non-blocking while consumer in blocking mode.

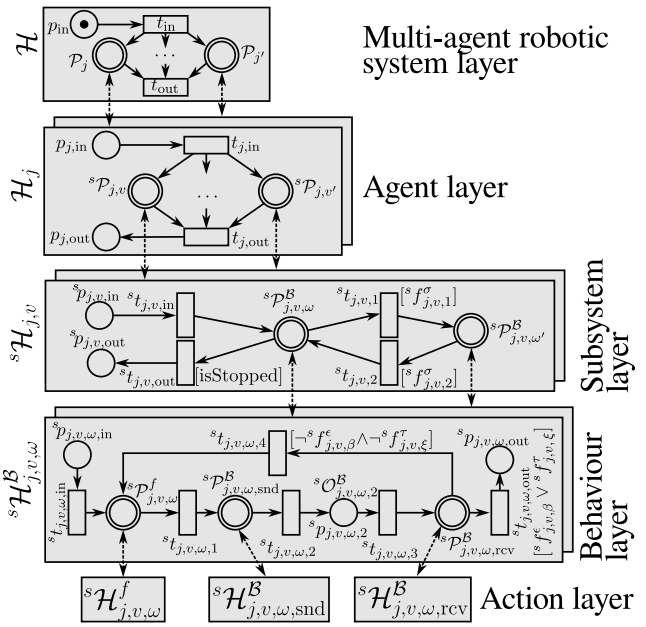


FIGURE 2. Robotic system activities defined by HPN $\mathcal{H}$.

C. HPN MODELLING A MULTI-AGENT ROBOTIC SYSTEM

The HPN $\mathcal{H}$ (Fig. 2) modeling a multi-agent robotic system activities is composed of five layers:

1) multi-agent robotic system layer (section IV-A) – defines a single net $\mathcal{H}$ representing the activities of each agent a_j defined by individual pages $\mathcal{P}_j$,
2) agent layer (section IV-B) – defines nets $\mathcal{H}_j$ represented by pages $\mathcal{P}_j$. Each net describes the activities of an agent a_j, which consists of several subsystems. The activities of each subsystem $s_{j,v}$ are represented by an individual page ${}^s\mathcal{P}_{j,v}$. It should be emphasized that within the agent layer only a single control subsystem c_j, zero or more virtual effectors $e_{j,n}$, as well as zero or more virtual receptors $r_{j,k}$ can exist. $E_{j,h}$ and $R_{j,l}$ are omitted, because they are treated as external devices supplied by their vendors, thus they are not subject of this design procedure,
3) subsystem layer (section IV-C) – defines nets ${}^s\mathcal{H}_{j,v}$ represented by pages ${}^s\mathcal{P}_{j,v}$. Each net describes the activities of subsystem $s_{j,v}$ switching between behaviours ${}^s\mathcal{B}_{j,v,\omega}$ represented by pages ${}^s\mathcal{P}^{\mathcal{B}}_{j,v,\omega}$,
4) behaviour layer (section IV-D) – defines nets ${}^s\mathcal{H}^{\mathcal{B}}_{j,v,\omega}$ represented by pages ${}^s\mathcal{P}^{\mathcal{B}}_{j,v,\omega}$. Each net describes the activities of behaviour ${}^s\mathcal{B}_{j,v,\omega}$ executing an elementary action ${}^s\mathcal{A}_{j,v,\omega}$. It contains pages: ${}^s\mathcal{P}^{f}_{j,v,\omega}$ (describing how $s_{j,v}$ executes the transition function ${}^s f_{j,v,\omega}$), ${}^s\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd}}$ (defining the communication mode of $s_{j,v}$ when sending data from ${}_y s_{j,v}$) and ${}^s\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv}}$ (defining the communication mode of $s_{j,v}$ when receiving data into ${}_x s_{j,v}$) – each behaviour is thus represented by a parameterised pattern, where the parameters are the definitions of: ${}^s\mathcal{P}^{f}_{j,v,\omega}$, ${}^s\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd}}$ and ${}^s\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv}}$,
5) action layer (section IV-E) – defines three independent nets ${}^s\mathcal{H}^{f}_{j,v,\omega}$, ${}^s\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd}}$, ${}^s\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv}}$ represented by pages ${}^s\mathcal{P}^{f}_{j,v,\omega}$, ${}^s\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd}}$, ${}^s\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv}}$, respectively. Each net is hierarchical and consists of two sublayers described later.

Fig. 2, presenting the HPN, uses the following convention. Looking at any two consecutive layers, in the upper one pages appear and in the lower one panels (stacked rectangles) are drawn. Each panel is the HPN representing its respective page. Thus each such page is represented by a panel, hence in the lower layer there are as many panels as there are pages in the upper one. Each dashed arrow connects one upper layer page to its respective panel in the lower layer.

IV. DESCRIPTION OF ROBOTIC SYSTEM LAYERS

Regardless of the designed system the only layer of the HPN $\mathcal{H}$ which has a fixed structure is the behaviour layer, as it is a pattern. The structure of each of the first two layers differs only by the number of pages that they contain. The structure of the subsystem layer depends on the task that the subsystem has to execute, thus it varies a lot. The structure of the action layer depends on the task that is to be executed, i.e. the organisation of the computations of Eq. (1), and to a lesser extent on the selected communication modes.

A. MULTI-AGENT ROBOTIC SYSTEM LAYER – $\mathcal{H}$

The HPN $\mathcal{H}$ consists of as many pages $\mathcal{P}_j$ as there are agents a_j in the designed robotic system. On system initiation t_{in} fires instantaneously. Once all pages complete their activities t_{out} fires and the system terminates.

B. AGENT LAYER – $\mathcal{H}_j$

The agent layer HPN $\mathcal{H}_j$ contains one page ${}^s\mathcal{P}_{j,v}$ for each subsystem $s_{j,v}$ within the agent a_j, i.e. one c_j, zero or more of $e_{j,n}$ and zero or more of $r_{j,k}$. $p_{j,in}$ and $p_{j,out}$ are the initial and the terminal places of $\mathcal{H}_j$. When the token appears in $p_{j,in}$ the transition ${}^s t_{j,in}$ fires, starting the activity of all subsystems of the a_j. Upon completion of activities of all subsystems $t_{j,out}$ fires and the token exits through the place $p_{j,out}$.

C. SUBSYSTEM LAYER – ${}^s\mathcal{H}_{j,v}$

The structure of HPN ${}^s\mathcal{H}_{j,v}$ depends on the task and thus defines the order of execution of behaviours ${}^s\mathcal{B}_{j,v,\omega}$.

The activity of each behaviour ${}^{s}\mathcal{B}_{j,v,\omega}$ is represented by page ${}^{s}\mathcal{P}^{\mathcal{B}}_{j,v,\omega}$. For each pair of consecutive behaviors (e.g. ${}^{s}\mathcal{B}_{j,v,\omega}$ and ${}^{s}\mathcal{B}_{j,v,\omega'}$, where the former is the current behaviour and the latter is one of the possible next behaviours), a single transition ${}^{s}t_{j,v,\alpha}$ exists with an associated initial condition ${}^{s}f^{\sigma}_{j,v,\alpha}$. If it is true the switch between ${}^{s}\mathcal{B}_{j,v,\omega}$ and ${}^{s}\mathcal{B}_{j,v,\omega'}$ occurs. At least one condition associated with transitions ${}^{s}t_{j,v,\alpha}$ connected by directed arcs emerging from ${}^{s}\mathcal{P}^{\mathcal{B}}_{j,v,\omega}$ must be true upon termination of ${}^{s}\mathcal{B}_{j,v,\omega}$. $p_{j,v,in}$ and $p_{j,v,out}$ are the initial and the terminal places of ${}^{s}\mathcal{H}_{j,v}$.

D. BEHAVIOUR LAYER – ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega}$

The structure of HPN ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega}$, representing behaviour ${}^{s}\mathcal{B}_{j,v,\omega}$, is fixed, as it conforms to a universal pattern. It contains a sequence of pages forming an elementary action ${}^{s}\mathcal{A}_{j,v,\omega}$ as well as the terminal ${}^{s}f^{\tau}_{j,v,\xi}$ and error ${}^{s}f^{\varepsilon}_{j,v,\beta}$ conditions. $p_{j,v,\omega,in}$ and $p_{j,v,\omega,out}$ are the initial and the terminal places of ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega}$. The elementary action ${}^{s}\mathcal{A}_{j,v,\omega}$ consists of 3 pages and one operation executed unconditionally as a sequence: ${}^{s}\mathcal{P}^{f}_{j,v,\omega}$ calculates the transition function, ${}^{s}\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd}}$ sends out the results inserted into the buffer ${}_{y}s_{j,v}$ to the associated subsystems, operation ${}^{s}\mathcal{O}^{\mathcal{B}}_{j,v,\omega,2}$ associated with the place ${}^{s}p_{j,v,\omega,2}$ increments the discrete time stamp i, and page ${}^{s}\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv}}$ inserts data received from the associated subsystems into the buffer ${}_{x}s_{j,v}$. Subsequently both the error ${}^{s}f^{\varepsilon}_{j,v,\beta}$ and terminal ${}^{s}f^{\tau}_{j,v,\xi}$ conditions are checked. If none is fulfilled, then transition ${}^{s}t_{j,v,\omega,4}$ fires leading to the next iteration of ${}^{s}\mathcal{B}_{j,v,\omega}$. Otherwise transition ${}^{s}t_{j,v,\omega,out}$ fires terminating ${}^{s}\mathcal{B}_{j,v,\omega}$. The activity subsequently returns to ${}^{s}\mathcal{H}_{j,v}$.

E. ACTION LAYER – ${}^{s}\mathcal{H}^{f}_{j,v,\omega'}$ ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd}'}$ ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv}}$

The action layer is composed of three independent HPNs: ${}^{s}\mathcal{H}^{f}_{j,v,\omega}$, ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd}}$ and ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv}}$. As each one of them contains pages, thus this layer consists of two sublayers.

1) TRANSITION FUNCTION ${}^{s}f_{j,v,\omega}$

It is defined by hierarchically composed nets: ${}^{s}\mathcal{H}^{f}_{j,v,\omega}$ and ${}^{s}\mathcal{H}^{f}_{j,v,\omega,\psi}$ (Fig. 3). Each one of them decomposes this function, thus facilitating both its definition and future implementation. ${}^{s}\mathcal{H}^{f}_{j,v,\omega}$ results from the canonical decomposition (Eq. (1)), thus it has as many pages as there are output buffers ${}_{y}s_{j,v}$ (μ here) in the subsystem plus one (for internal memory, thus $\mu+1$ pages). Each partial function ${}^{s,s'}f_{j,v,\omega,\psi}$, $\psi = 1,\ldots,\mu+1$, (Eq. 1), is represented by ${}^{s}\mathcal{H}^{f}_{j,v,\omega,\psi}$ which further decomposes this function by taking into account the availability of new data in the input buffers ${}_{x}s_{j,v}$. Obsolete data should not be used in the computations, hence a different form of the function has to be used for different combinations of available data [29]. Subsystem $s_{j,v}$ has ν input buffers, thus 2^{ν} possibilities result.

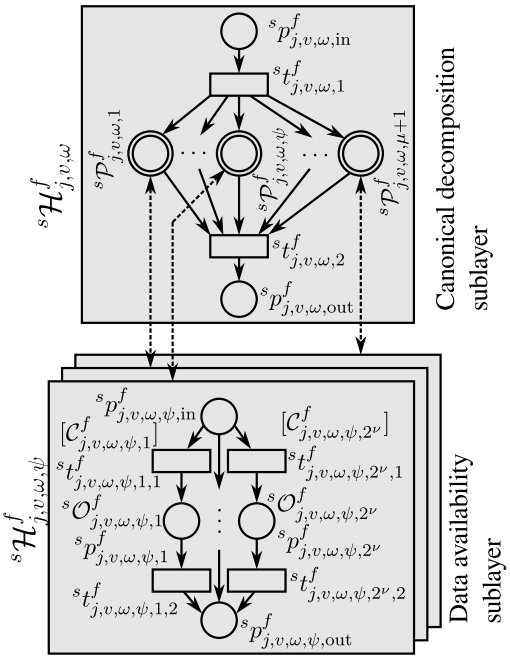


FIGURE 3. Two sublayers defining the transition function ${}^{s}f_{j,v,\omega}$ within the action layer.

2) COMMUNICATION

HPNs ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd}}$ and ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv}}$ define how the data is sent and received by $s_{j,v}$ while executing behaviour ${}^{s}\mathcal{B}_{j,v,\omega}$. Each of those HPNs is decomposed into two sub-layers: arrangement sublayer – determining the order in which data is sent/received, and mode sublayer – determining the mode of $s_{j,v}$ communication with the associated subsystems. The arrangement sublayer is composed of ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd}}$ containing pages ${}^{s}\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd},\rho}$ and ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv}}$ containing pages ${}^{s}\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv},\kappa}$, where $\rho = 1,\ldots,\mu$ and $\kappa = 1,\ldots,\nu$. The activation order of pages within each HPN may be arbitrary: sequential, parallel or mixed. Fig. 4 uses parallel arrangement both for sending and receiving data. Pages ${}^{s}\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd},\rho}$ and ${}^{s}\mathcal{P}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv},\kappa}$, defined by ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd},\rho}$ and ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv},\kappa}$ respectively, determine the communication mode utilised by $s_{j,v}$ while sending/receiving data to/from each of the associated subsystems. As it was mentioned in sec. III-B, there are four possible communication modes. Both communicating subsystems acting in blocking mode or both acting in non-blocking mode [23], [24]; PN for the sender using the non-blocking mode and receiver using the blocking mode is presented in Fig. 5a – PN is split into two nets (Fig. 5b and Fig.5c) by using fusion places.

When a token appears in the input place ${}^{s}p_{j,v,\omega,\mathrm{snd},\rho,in}$ of ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd},\rho}$ and there is a token in place ${}^{s}p^{fusion}_{j,v,\omega,\mathrm{snd},\rho,2}$, signalling that the data was consumed by $s_{j,h}$, the transition ${}^{s}t_{j,v,\omega,\mathrm{snd},\rho,1}$ fires and thus activates operation ${}^{s}\mathcal{O}_{j,v,\omega,\mathrm{snd},\rho}$, producing new data for $s_{j,h}$. When operation ${}^{s}\mathcal{O}_{j,v,\omega,\mathrm{snd},\rho}$ is complete, and thus transition ${}^{s}t_{j,v,\omega,\mathrm{snd},\rho,3}$

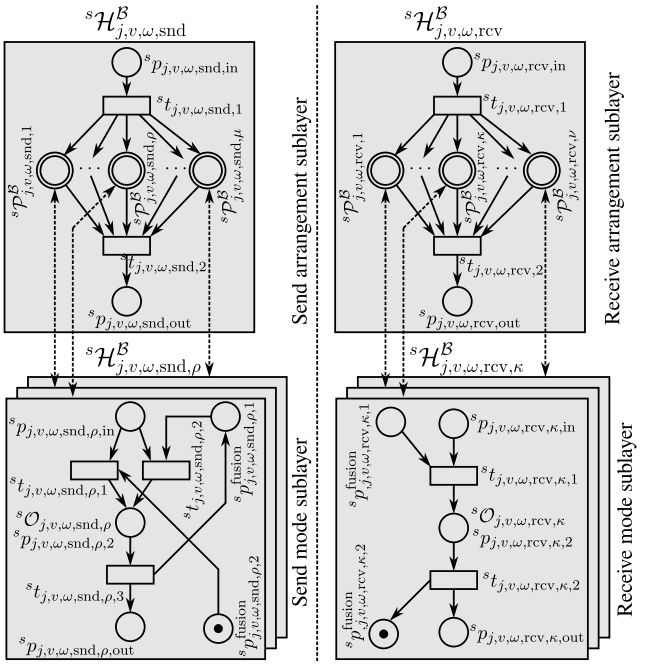


FIGURE 4. HPNs ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd}}$ and ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{rcv}}$ defining $s_{j,v}$ communication while executing ${}^{s}\mathcal{B}_{j,v,\omega}$; $s_{j,v}$ sends data using non-blocking mode and receives data using blocking mode.

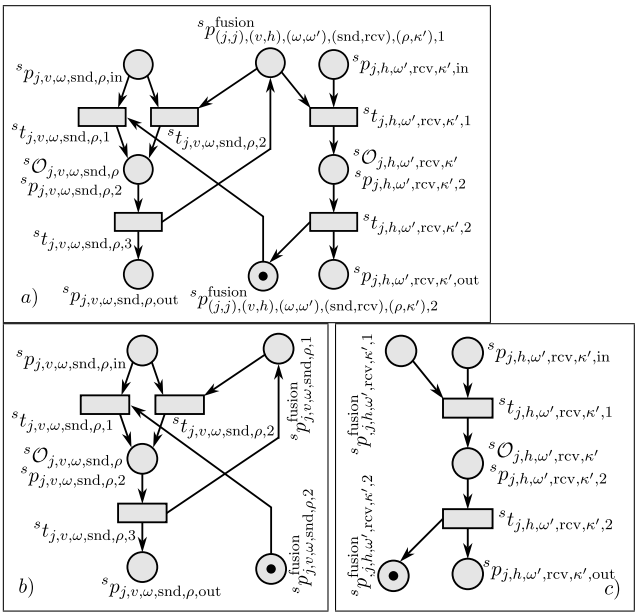


FIGURE 5. (a) Communication mode used to send data from subsystem $s_{j,v}$ to subsystem $s_{j,h}$. Subsystem $s_{j,v}$ sends data in non-blocking mode, while $s_{j,h}$ receives data in blocking mode. (b) ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,v,\omega,\mathrm{snd},\rho}$ defining the communication mode used by subsystem $s_{j,v}$ executing behaviour ${}^{s}\mathcal{B}_{j,v,\omega}$. (c) ${}^{s}\mathcal{H}^{\mathcal{B}}_{j,h,\omega',\mathrm{rcv},\kappa'}$ defining the communication mode used by subsystem $s_{j,h}$ executing behaviour ${}^{s}\mathcal{B}_{j,h,\omega'}$.

fires, a token appears in place ${}^{s}p^{fusion}_{j,v,\omega,\mathrm{snd},\rho,1}$. Because two places: ${}^{s}p^{fusion}_{j,v,\omega,\mathrm{snd},\rho,1}$ and ${}^{s}p^{fusion}_{j,h,\omega',\mathrm{rcv},\kappa',1}$ are fused, a single token also appears in ${}^{s}p^{fusion}_{j,h,\omega',\mathrm{rcv},\kappa',1}$. If $s_{j,h}$ is ready to read data, i.e. a single token resides in ${}^{s}p^{fusion}_{j,h,\omega',\mathrm{rcv},\kappa',in}$, the transition ${}^{s}t_{j,h,\omega',\mathrm{rcv},\kappa',1}$ fires activating ${}^{s}\mathcal{O}_{j,h,\omega',\mathrm{rcv},\kappa'}$ and thus receiving new data. If $s_{j,h}$ is not ready to read data ${}^{s}t_{j,v,\omega,\mathrm{snd},\rho,2}$ fires when new data is produced by $s_{j,v}$, thus consuming the token from fused places: ${}^{s}p^{fusion}_{j,v,\omega,\mathrm{snd},\rho,1}$ and ${}^{s}p^{fusion}_{j,h,\omega',\mathrm{rcv},\kappa',1}$ and hence activating ${}^{s}\mathcal{O}_{j,v,\omega,\mathrm{snd},\rho}$. This assures that $s_{j,v}$ does not wait for confirmation that the data was received by $s_{j,h}$ and that $s_{j,h}$ blocks until new data is available.

V. EXPERIMENT

To show how the RSSM is to be used a rudimentary experiment is presented here. A controller for a LWR4+ manipulator having 7 d.o.f. utilising impedance control [30] was designed. The task is to move the end-effector along a circular trajectory (Fig. 18). The experiment is purposefully kept simple in order not to hinder the understanding of the RSSM methodology presented in this article.

A. SPECIFICATION – SINGLE-ROBOT SYSTEM STRUCTURE

The exemplary robot system consists of a single embodied agent a_1 containing: c_1, $e_{1,\mathrm{lwr}}$ and $E_{1,\mathrm{lwr}}$, as presented in Fig. 6. The control subsystem c_1 is responsible for calculating subsequent positions of the end-effector (w.r.t. Cartesian reference frame) and inserting them in the output buffer ${}^{e}_{y}c_{1,\mathrm{lwr}}$, from which the data is sent to $e_{1,\mathrm{lwr}}$. Virtual effector

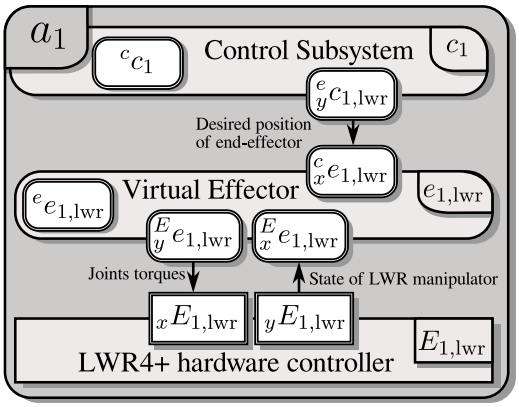


FIGURE 6. Structure of an agent a_1 modeling the LWR4+ robot system controller executing a rudimentary task.

$e_{1,\mathrm{lwr}}$ has two input buffers: 1) ${}^{E}_{x}e_{1,\mathrm{lwr}}$ acquiring data from $E_{1,\mathrm{lwr}}$, namely: current joint positions, current end-effector pose and its velocity, 2) ${}^{c}_{x}e_{1,\mathrm{lwr}}$ holding data obtained from c_1, i.e. the desired end-effector pose. The received data and the model of the manipulator enable the calculation of the necessary joint torques: torque compensating the gravity and torque resulting from impedance control. The calculated torques are inserted into the output buffer ${}^{E}_{y}e_{1,\mathrm{lwr}}$ and subsequently sent to $E_{1,\mathrm{lwr}}$, which represents the LWR4+ hardware controller. $E_{1,\mathrm{lwr}}$ reads the desired joint torques sent by $e_{1,\mathrm{lwr}}$ and uses them as the desired values for hardware controllers of the motors.

B. SPECIFICATION – SINGLE-ROBOT SYSTEM ACTIVITIES

The activities of the exemplary robotic system are represented by a five-layer HPN presented in section III-C.

1) MULTI-AGENT ROBOTIC SYSTEM LAYER – $\mathcal{H}$

The activities of the discussed robot system are modeled by $\mathcal{H}$ (Fig. 7). The robot system consists of a single agent a_1, hence $\mathcal{H}$ contains a single page $\mathcal{P}_1$.

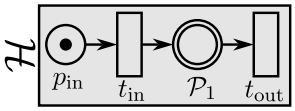


FIGURE 7. Robot system layer $\mathcal{H}$ for the LWR4+ robot system controller.

2) AGENT LAYER – $\mathcal{H}_1$

The agent a_1 contains the control subsystem c_1 and the virtual effector $e_{1,\text{lwr}}$, thus $\mathcal{H}_1$ consists of two pages ${}^{c}\mathcal{P}_{1,c}$ and ${}^{e}\mathcal{P}_{1,\text{lwr}}$ (Fig. 8). Note that the $E_{1,\text{lwr}}$ is not represented within $\mathcal{H}_1$, because it is treated as a real device, which has a fixed structure that cannot be modified, i.e. $e_{1,\text{lwr}}$ contacts it through its ${}^{E}_{y}e_{1,\text{lwr}}$ buffer, which must match ${}_{x}E_{1,\text{lwr}}$.

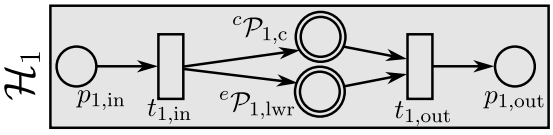


FIGURE 8. Agent layer $\mathcal{H}_1$ modeling the activities of the agent a_1 representing the LWR4+ robot system controller.

3) SUBSYSTEM LAYER – ${}^{c}\mathcal{H}_{1,c}$, ${}^{e}\mathcal{H}_{1,\text{lwr}}$

a: CONTROL SUBSYSTEM – ${}^{c}\mathcal{H}_{1,c}$

The control subsystem c_1 switches between executing two behaviours ${}^{c}\mathcal{B}_{1,c,\text{cw}}$ and ${}^{c}\mathcal{B}_{1,c,\text{ccw}}$, thus two pages ${}^{c}\mathcal{P}^{\mathcal{B}}_{1,c,\text{cw}}$ and ${}^{c}\mathcal{P}^{\mathcal{B}}_{1,c,\text{ccw}}$ appear within ${}^{c}\mathcal{H}_{1,c}$ (Fig. 9).

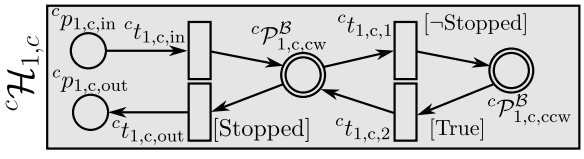


FIGURE 9. Subsystem layer ${}^{c}\mathcal{H}_{1,c}$ modeling the activities of the control subsystem c_1.

b: VIRTUAL EFFECTOR – ${}^{e}\mathcal{H}_{1,\text{lwr}}$

The virtual effector $e_{1,\text{lwr}}$ executes only a single behaviour ${}^{e}\mathcal{B}_{1,\text{lwr},move}$ calculating the desired torques based on the data received from c_1 and $E_{1,\text{lwr}}$, thus ${}^{e}\mathcal{H}_{1,\text{lwr}}$ contains only a single page ${}^{e}\mathcal{P}^{\mathcal{B}}_{1,\text{lwr},move}$ (Fig. 10).

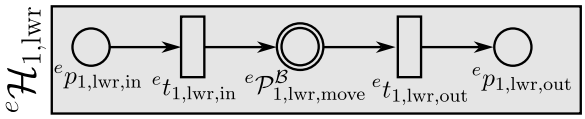


FIGURE 10. Subsystem layer ${}^{e}\mathcal{H}_{1,lwr}$ modeling the activities of the virtual effector $e_{1,\text{lwr}}$.

4) BEHAVIOUR LAYER – ${}^{c}\mathcal{H}^{\mathcal{B}}_{1,c,\text{cw}}$, ${}^{c}\mathcal{H}^{\mathcal{B}}_{1,c,\text{ccw}}$, ${}^{e}\mathcal{H}^{\mathcal{B}}_{1,\text{lwr,move}}$

Behaviours of both subsystems follow the same pattern (Fig. 2), however they are parameterized by different transition functions, send and receive communication modes as well as terminal and error conditions.

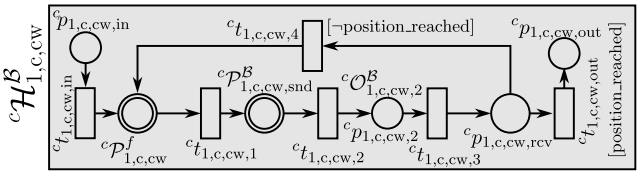


FIGURE 11. Behaviour layer ${}^{c}\mathcal{H}_{1,c,\text{cw}}$ for c_1.

a: BEHAVIOUR HPNs OF THE CONTROL SUBSYSTEM – ${}^{c}\mathcal{H}^{\mathcal{B}}_{1,c,\text{cw}}$ AND ${}^{c}\mathcal{H}^{\mathcal{B}}_{1,c,\text{ccw}}$

Petri nets modeling behaviours ${}^{c}\mathcal{B}_{1,c,\text{cw}}$ and ${}^{c}\mathcal{B}_{1,c,\text{cw}}$ have the same structure (Fig. 11) – they differ by the names of behaviours only. For both behaviours c_1 calculates the transition function and subsequently sends the computed data from the output buffer to $e_{1,\text{lwr}}$. c_1 does not receive data from other subsystems thus the operations associated with places ${}^{c}p_{1,c,\text{cw},2}$ and ${}^{c}p_{1,c,\text{ccw},2}$ are empty. Terminal conditions of both behaviours are the same, i.e. they become fulfilled when the end effector reaches once again the start position on the circle (left side of Fig. 19a). Once the terminal condition is fulfilled c_1 terminates the current behaviour and switches to the other behaviour, which simply changes the direction of the end effector motion along the same circular path.

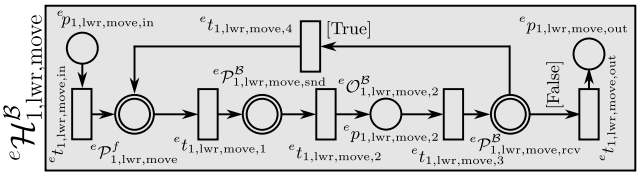


FIGURE 12. Behaviour layer ${}^{e}\mathcal{H}_{1,\text{lwr,move}}$ for $e_{1,\text{lwr}}$.

b: BEHAVIOUR HPN OF THE VIRTUAL EFFECTOR – ${}^{e}\mathcal{H}^{\mathcal{B}}_{1,\text{lwr,move}}$

The virtual effector $e_{1,\text{lwr}}$ endlessly executes a single behaviour ${}^{c}\mathcal{B}_{1,\text{lwr,move}}$ (i.e. the terminal condition is always not fulfilled) modeled by ${}^{e}\mathcal{H}^{\mathcal{B}}_{1,\text{lwr,move}}$ presented in Fig. 12. In each iteration of this behaviour $e_{1,\text{lwr}}$ calculates its transition function, sends the computed data from the output buffer

to $E_{1,\mathrm{lwr}}$ and subsequently receives new data from c_1 and $E_{1,\mathrm{lwr}}$, thus ${}^{e}\mathcal{H}^{\mathcal{B}}_{1,\mathrm{lwr,move}}$ contains page ${}^{e}\mathcal{P}^{\mathcal{B}}_{1,\mathrm{lwr,move,rcv}}$.

5) ACTION LAYER

The action layer is composed of: 1) send arrangement sublayer for both subsystems, 2) receive arrangement sublayer for $e_{1,\mathrm{lwr}}$ and 3) canonical decomposition sublayer for both subsystems.

a: SEND ARRANGEMENT SUBLAYER – ${}^{c}\mathcal{H}^{\mathcal{B}}_{1,c,\mathrm{cw,snd}}$, ${}^{c}\mathcal{H}^{\mathcal{B}}_{1,c,\mathrm{ccw,snd}}$ *and* ${}^{e}\mathcal{H}^{\mathcal{B}}_{1,\mathrm{lwr,move,snd}}$

It was assumed that all subsystems send out data using the non-blocking communication mode [24]. The control subsystem c_1 sends data only to a single subsystem, thus the send arrangement sublayer ${}^{c}\mathcal{H}^{\mathcal{B}}_{1,c,\mathrm{cw,snd}}$ for ${}^{c}\mathcal{B}_{1,c,\mathrm{cw}}$, presented in Fig. 13, contains only a single page ${}^{c}\mathcal{P}^{\mathcal{B}}_{1,c,\mathrm{cw,snd},1}$, which represents the non-blocking mode utilised to communicate with $e_{1,\mathrm{lwr}}$. The send arrangement sublayer ${}^{c}\mathcal{H}^{\mathcal{B}}_{1,c,\mathrm{ccw,snd}}$ of ${}^{c}\mathcal{B}_{1,c,\mathrm{ccw}}$ is the same, if cww is substituted for cw. ${}^{e}\mathcal{H}^{\mathcal{B}}_{1,\mathrm{lwr,move,snd}}$, presented in Fig. 14, contains a single page ${}^{e}\mathcal{P}^{\mathcal{B}}_{1,\mathrm{lwr,move,snd},1}$, because $e_{1,\mathrm{lwr}}$ contains a single output buffer. This net defines the non-blocking communication mode used to send data to $E_{1,\mathrm{lwr}}$.

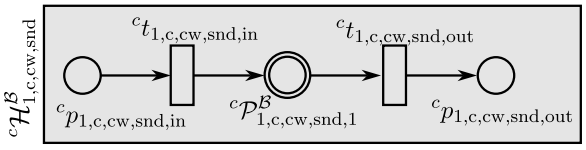


FIGURE 13. Send arrangement sublayer for ${}^{c}\mathcal{B}_{1,c,\mathrm{cw}}$.

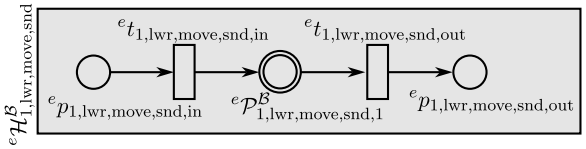


FIGURE 14. Send arrangement sublayer for ${}^{e}\mathcal{B}_{1,\mathrm{lwr,move}}$.

b: RECEIVE ARRANGEMENT SUBLAYER – ${}^{e}\mathcal{H}^{\mathcal{B}}_{1,\mathrm{lwr,move,rcv}}$

The $e_{1,\mathrm{lwr}}$ uses non-blocking mode to receive data from both subsystems: c_1 and $E_{1,\mathrm{lwr}}$. Thus its receive arrangement sublayer ${}^{e}\mathcal{H}^{\mathcal{B}}_{1,\mathrm{lwr,move,rcv}}$ contains two pages ${}^{e}\mathcal{P}^{\mathcal{B}}_{1,\mathrm{lwr,move,rcv},1}$ and ${}^{e}\mathcal{P}^{\mathcal{B}}_{1,\mathrm{lwr,move,rcv},2}$, executed sequentially, as presented in Fig. 15. The former page represents the non-blocking mode used to receive data from c_1, while the latter page also represents the non-blocking mode, however to receive data from $E_{1,\mathrm{lwr}}$. In this case, the parallel receipt of data is counterproductive, because the time required by the parallel data reception mechanism exceeds the time of sequential data reception. However it should be noted that this is an implementation issue considered at the specification stage.

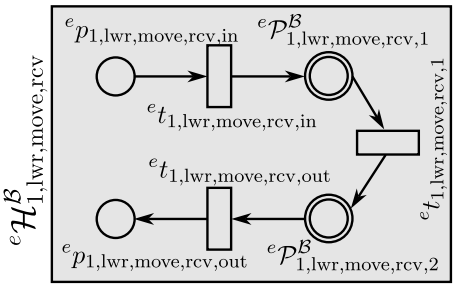


FIGURE 15. Receive arrangement sublayer for ${}^{e}\mathcal{B}_{1,\mathrm{lwr,move}}$.

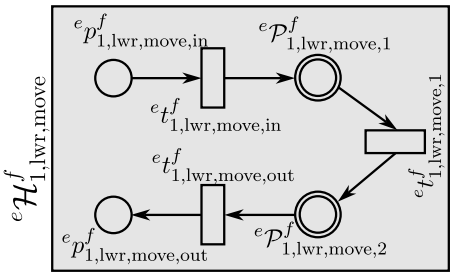


FIGURE 16. Canonical decomposition sublayer for ${}^{e}\mathcal{B}_{1,\mathrm{lwr,move}}$.

c: CANONICAL DECOMPOSITION SUBLAYER – ${}^{c}\mathcal{H}^{f}_{1,c,\mathrm{cw}}$, ${}^{c}\mathcal{H}^{f}_{1,c,\mathrm{ccw}}$ *AND* ${}^{s}\mathcal{H}^{f}_{1,\mathrm{lwr,move}}$

The virtual effector $e_{1,\mathrm{lwr}}$ behaviour ${}^{e}\mathcal{B}_{1,\mathrm{lwr},move}$ in its each iteration computes the transition function ${}^{e}f_{1,\mathrm{lwr,move}}$ (Eq. 2), which takes as arguments the contents of two input buffers: ${}^{c}_{x}e_{1,\mathrm{lwr}}$ and ${}^{E}_{x}e_{1,\mathrm{lwr}}$, and internal memory ${}^{e}e_{1,\mathrm{lwr}}$.

$$\left({}^{e}e^{i+1}_{1,\mathrm{lwr}},\ {}^{E}_{y}e^{i+1}_{1,\mathrm{lwr}}\right) := {}^{e}f_{1,\mathrm{lwr,move}}\left({}^{c}_{x}e^{i}_{1,\mathrm{lwr}},\ {}^{E}_{x}e^{i}_{1,\mathrm{lwr}},\ {}^{e}e^{i}_{1,\mathrm{lwr}}\right) \tag{2}$$

As $e_{1,\mathrm{lwr}}$ contains only a single output buffer and internal memory, ${}^{e}f_{1,\mathrm{lwr,move}}$ is in effect decomposed into two partial transition functions, i.e. ${}^{e,E}f_{1,\mathrm{lwr,move},1}$ (Eq. 3) and ${}^{e,e}f_{1,\mathrm{lwr,move},2}$(Eq. 4).

$${}^{E}_{y}e^{i+1}_{1,\mathrm{lwr}} := {}^{e,E}f_{1,\mathrm{lwr,move},1}\left({}^{c}_{x}e^{i}_{1,\mathrm{lwr}},\ {}^{E}_{x}e^{i}_{1,\mathrm{lwr}},\ {}^{e}e^{i}_{1,\mathrm{lwr}}\right) \tag{3}$$

$${}^{e}e^{i+1}_{1,\mathrm{lwr}} := {}^{e,e}f_{1,\mathrm{lwr,move},2}\left({}^{c}_{x}e^{i}_{1,\mathrm{lwr}},\ {}^{E}_{x}e^{i}_{1,\mathrm{lwr}},\ {}^{e}e^{i}_{1,\mathrm{lwr}}\right) \tag{4}$$

Transition function ${}^{e}f_{1,\mathrm{lwr,move}}$ is defined by the canonical decomposition sublayer ${}^{e}\mathcal{H}^{f}_{1,\mathrm{lwr,move}}$, (presented in Fig. 16), while ${}^{e,E}f_{1,\mathrm{lwr,move},1}$ and ${}^{e,e}f_{1,\mathrm{lwr,move},2}$ are represented by ${}^{e}\mathcal{P}^{f}_{1,\mathrm{lwr,move},1}$ and ${}^{e}\mathcal{P}^{f}_{1,\mathrm{lwr,move},2}$ defined by the data availability sublayer ${}^{e}\mathcal{H}^{f}_{1,\mathrm{lwr,move},1}$ and ${}^{e}\mathcal{H}^{f}_{1,\mathrm{lwr,move},2}$, respectively. As $e_{1,\mathrm{lwr}}$ receives the data from two subsystems, the transition function ${}^{e,E}f_{1,\mathrm{lwr,move},1}$ is decomposed into four partial transition functions ${}^{e,E}f_{1,\mathrm{lwr,move},1,\varpi}$ based on data availability criterium, as presented in Eq. 5, where $\varpi = 1,\ldots,4$. Predicate *new* becomes *TRUE* whenever a new value of its argument is obtained and *FALSE* when this value is utilised

by the virtual effector.

$$
{}^{e,E}f_{1,\mathrm{lwr},\mathrm{move},1} \triangleq \begin{cases} {}^{e,E}f_{1,\mathrm{lwr},\mathrm{move},1,1} & \text{for } new({}^{c}_{x}e^{i}_{1,\mathrm{lwr}}) \wedge new({}^{E}_{x}e^{i}_{1,\mathrm{lwr}}) \\ {}^{e,E}f_{1,\mathrm{lwr},\mathrm{move},1,2} & \text{for } \neg new({}^{c}_{x}e^{i}_{1,\mathrm{lwr}}) \wedge new({}^{E}_{x}e^{i}_{1,\mathrm{lwr}}) \\ {}^{e,E}f_{1,\mathrm{lwr},\mathrm{move},1,3} & \text{for } new({}^{c}_{x}e^{i}_{1,\mathrm{lwr}}) \wedge \neg new({}^{E}_{x}e^{i}_{1,\mathrm{lwr}}) \\ {}^{e,E}f_{1,\mathrm{lwr},\mathrm{move},1,4} & \text{for } \neg new({}^{c}_{x}e^{i}_{1,\mathrm{lwr}}) \wedge \neg new({}^{E}_{x}e^{i}_{1,\mathrm{lwr}}), \end{cases} \tag{5}
$$

Each partial transition function ${}^{e,E}f_{1,\mathrm{lwr},\mathrm{move},1,\varpi}$ is represented by a respective operation ${}^{e}\mathcal{O}_{1,\mathrm{lwr},\mathrm{move},1,\varpi}$. In the presented system, if either of the two buffers does not hold new data (i.e. when $\varpi = 2, 3, 4$), the operation ${}^{e}\mathcal{O}_{1,\mathrm{lwr},\mathrm{move},1,\varpi}$ is assumed to be void. If new data is available ${}^{e}\mathcal{O}_{1,\mathrm{lwr},\mathrm{move},1,1}$ executes ${}^{e,E}f_{1,\mathrm{lwr},\mathrm{move},1,1}$, defined as ${}^{E}_{y}e_{1,\mathrm{lwr}} = \mathfrak{T}_{grav} + \mathfrak{T}_{imp}$, where $\mathfrak{T}_{grav}$ is the vector of joint torques necessary to compensate the gravitation [31] and $\mathfrak{T}_{imp}$ is the vector of joint torques resulting from impedance control, i.e. $\mathfrak{T}_{imp} = J^T \mathfrak{F}_{ext}$, where J^T is a 7×6 transpose of the Jacobian matrix and $\mathfrak{F}_{ext}$ is the vector of external forces and torques acting on the end effector: $\mathfrak{F}_{ext} = K\Delta P - D\dot{P}$, where K is the stiffness matrix, ΔP is the difference between desired P_{des} and current P_{cur} end-effector poses, D is the damping matrix and $\dot{P}$ is a vector of linear and angular velocities of the end-effector. Transition functions of the two behaviours of c_1 generate consecutive desired end-effector poses P_{des} along the circular path. Their formal definition is omitted here, because of obviousness.

C. TOOLS AND SIMULATION

The RSHPN Tool facilitating specification of multi-agent robotic systems utilising HPN has been developed. Fig. 17 presents an exemplary screen-shot of a HPN representing the activities of agent a_1. It is used to create the HPN graphically. The thus created HPN can be saved and edited. The RSHPN Tool enables the specification of operations $\mathcal{O}$ appearing within layers of $\mathcal{H}$ (Fig. 2) in the form of C++ code. It contains two modules enabling automatic generation of the robot controller code based on the presented specification methodology.

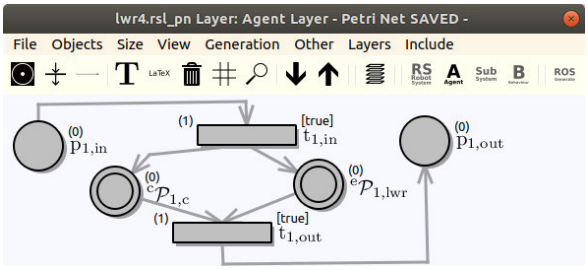


FIGURE 17. Exemplary HPN presenting the activities of agent layer viewed using RSHPN Tool.

The first module generates C++ code that interprets the defined HPN [24]. The module is used to generate the code which simulates the execution of the HPN, thus facilitates its verification (e.g. detection of deadlocks). The generated code consists of the scheduler, which searches for active transitions (enabled transition with fulfilled condition). One of such transitions is fired, i.e. a token is removed from each input place (place directly connected to the fired transition by a directed arc pointing at the transition) and inserted into each output place (place directly connected to the transition by a directed arc pointing at the place). The operations associated with the output places are executed in separate threads. When the directed arc connects the firing transition with an output page, the input place of that page receives a new token and the associated operation of that input place starts its execution in a new thread. The scheduler repeats the above-mentioned steps either endlessly or until a behaviour commands it to terminate its activities.

The second module of RSHPN Tool generates C++ code for ROS packages. For each agent a single ROS package is automatically generated. The package consists of automatically generated files: 1) C++ code defining a ROS node for each subsystem within an agent, 2) the required include files, 3) scripts utilised to launch the agent's subsystems and 4) the necessary ROS configuration files, i.e. CMakeLists.txt and package.xml. The generated C++ code for the ROS system is currently constrained to only ROS topics assuring asynchronous communication. Directly after generating the above files the package is ready to be compiled and executed. The functionality of the RSHPN Tool is presented in the video material [28].

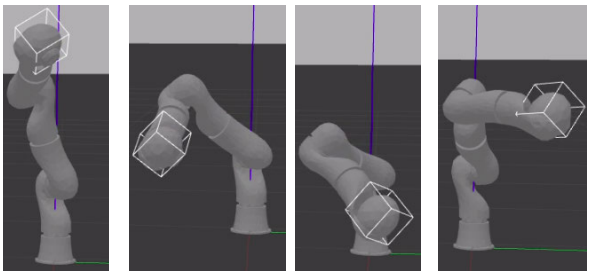
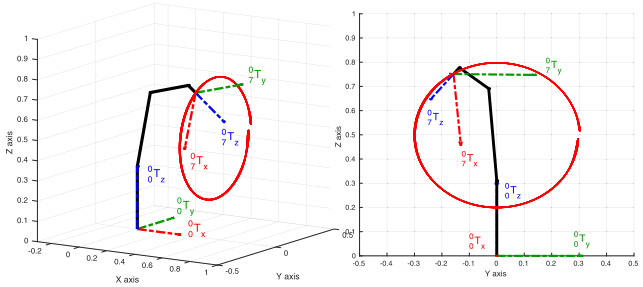


FIGURE 18. Simulated LWR4+ manipulator with 7 degrees of freedom controlled using impedance control. Its end-effector moves along a circular trajectory (only its Cartesian position is controlled).

For additional types of communication two libraries have been developed: 1) communication library – enabling communication for single robot systems based on shared memory (all four communication modes are available) and 2) LWR4+ dynamics library – providing functions necessary to calculate joint torques compensating gravitation and resulting from impedance control. Based on the specification, in the HPN format, the robot controller code for ROS system was automatically generated and run with a LWR4+ manipulator using the Gazebo simulator (Fig. 18). The consecutive positions of the end-effector were saved to a file and the plots were reproduced by a Matlab program (Fig. 19). The ROS topics have been utilised for communication between c_1 and



(a) 3D view (b) Y-Z plane

FIGURE 19. The consecutive robot end-effector positions form a circular trajectory.

$e_{1,\text{lwr}}$ (lower frequency), while shared memory library for communication with $E_{1,\text{lwr}}$ (higher frequency).

VI. CONCLUSION

The paper presents the general RSSM methodology of designing multi-agent robot systems. Following the principle of separation of concerns five layer HPN is specified. In our previous papers (e.g., [23], [24]) a robot system was also decomposed into several layers, however the action layer was not presented in detail. Thus the decomposition of transition functions as well as send and receive communication modes were absent. The canonical decomposition of a transition function based on output buffers and further decomposition based on the availability of new data in input buffers generates $\mu \cdot 2^{\nu}$ partial transition functions. Those partial functions can further be decomposed if necessary following the presented method of creating HPNs. Moreover, the decomposition of a communication model into two sublayers enables both parallel/sequential composition of individual interactions between subsystems and the definition of operation of both communicating parties (blocking/non-blocking mode).

The resulting HPN representing the system can be verified using existing PN verification tools [32]. The resulting system is modular, what greatly simplifies future modifications of any of the defined layers. The specification of the LWR4+ robot control system was both verified and validated. The verification consisted in checking whether the HPN is safe and whether there are no deadlocks. For that purpose the place invariants method was used [33]. Another verification method was based on HPN simulation utilising automatically generated C++ HPN interpreter [24]. Based on the resulting HPN the robot system controller (for the ROS system) was automatically generated and subsequently validated in simulation.

Currently work is in progress on control of the real LWR4+ robot and on the automatic generation of the HPN structure based on textual specification. In our further work, we plan to extend the HPN model by including communication timeouts. It will enable modeling temporal dependencies in the system and verification in terms of meeting the real-time requirements.

REFERENCES

[1] C. Zieliński, M. Figat, and R. Hexel, "Communication within multi-FSM based robotic systems," *J. Intell. Robotic Syst.*, vol. 93, nos. 3–4, pp. 787–805, Mar. 2019.
[2] M. Mataric, "Behavior-based control: Main properties and implications," in *Proc. IEEE Int. Conf. Robot. Automat., Workshop Archit. Intell. Control Syst.*, May 1992, pp. 46–54.
[3] B. Hayes-Roth, "An architecture for adaptive intelligent systems," *Artif. Intell.*, vol. 72, pp. 329–365, May 1995.
[4] T. L. Dean and M. P. Wellman, *Planning and Control*. San Francisco, CA, USA: Morgan Kaufmann, 1991.
[5] D. Kortenkamp and R. Simmons, "Robotic systems architectures and programming," in *Springer Handbook Robotics*, O. Khatib and B. Siciliano, Eds. Cham, Switzerland: Springer, 2008, pp. 187–206.
[6] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, 1995.
[7] R. C. Arkin, *Behavior-Based Robotics*. Cambridge, MA, USA: MIT Press, 1998.
[8] R. A. Brooks, "Intelligence without reason," *Artif. Intell., Crit. Concepts*, vol. 3, pp. 107–163, Sep. 1991.
[9] N. R. Jennings, K. Sycara, and M. Wooldridge, "A roadmap of agent research and development," *Auto. Agents Multi-Agent Syst.*, vol. 1, pp. 7–38, Jan. 1998.
[10] D. M. Lyons and M. A. Arbib, "A formal model of computation for sensory-based robotics," *IEEE Trans. Robot. Autom.*, vol. 5, no. 3, pp. 280–293, Jun. 1989.
[11] C. Zielićski, "Description of semantics of robot programming languages," *Mechatronics*, vol. 2, no. 2, pp. 171–198, Apr. 1992.
[12] J. Cabrera-Gámez, *Sensor Based Intelligent Robots* (CoolBOT: A Component-Oriented Program). Berlin, Germany: Springer, 2002, pp. 282–304.
[13] C. Zieliński and M. Figat, "Robot system design procedure based on a formal specification," in *Recent Advances in Automation, Robotics and Measuring Technique*, vol. 440. Cham, Switzerland: Springer, 2016, pp. 511–522.
[14] P. Freedman, "Time, Petri nets, and robotics," *IEEE Trans. Robot. Autom.*, vol. 7, no. 4, pp. 417–433, Dec. 1991.
[15] T. Cao and A. C. Sanderson, "Task decomposition and analysis of robotic assembly task plans using Petri nets," *IEEE Trans. Ind. Electron.*, vol. 41, no. 6, pp. 620–630, Dec. 1994.
[16] A. Caloini, G. Magnani, and M. Pezze, "A technique for designing robotic control systems based on Petri nets," *IEEE Trans. Control Syst. Technol.*, vol. 6, no. 1, pp. 72–87, Jan. 1998.
[17] P. Oliveira, A. Pascoal, V. Silva, and C. Silvestre, "Mission control of the MARIUS autonomous underwater vehicle: System design, implementation and sea trials," *Int. J. Syst. Sci.*, vol. 29, no. 10, pp. 1065–1080, Oct. 1998.
[18] L. Montano, F. J. García, and J. L. Villarroel, "Using the time Petri net formalism for specification, validation, and code generation in robot-control applications," *Int. J. Robot. Res.*, vol. 19, no. 1, pp. 59–76, Jan. 2000.
[19] G. Kim and W. Chung, "Navigation behavior selection using generalized stochastic Petri nets for a service robot," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 37, no. 4, pp. 494–503, Jul. 2007.
[20] J. López, D. Pérez, and E. Zalama, "A framework for building mobile single and multi-robot applications," *Robot. Auto. Syst.*, vol. 59, nos. 3–4, pp. 151–162, Mar. 2011.
[21] V. A. Ziparo, L. Iocchi, P. U. Lima, D. Nardi, and P. F. Palamara, "Petri net plans: A framework for collaboration and coordination in multi-robot systems," *Auto. Agents Multi-Agent Syst.*, vol. 23, no. 3, pp. 344–383, Nov. 2011.
[22] C. Lesire and F. Pommereau, "ASPiC: An acting system based on skill Petri net composition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 6952–6958.
[23] M. Figat and C. Zieliński, "Hierarchical Petri net representation of robot systems," in *Automation*. Cham, Switzerland: Springer, 2019, pp. 492–501.
[24] M. Figat and C. Zielinski, "Methodology of designing multi-agent robot control systems utilising hierarchical Petri nets," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 3363–3369.
[25] T. Kornuta and C. Zielićski, "Robot control system design exemplified by multi-camera visual servoing," *J. Intell. Robotic Syst.*, vol. 77, nos. 3–4, pp. 499–523, Mar. 2015.
[26] M. Quigley, "ROS: An open-source Robot Operating System," in *Proc. ICRA Workshop Open Source Softw.*, 2009, pp. 1–15.

[27] P. Huber, K. Jensen, and R. M. Shapiro, "Hierarchies in coloured Petri nets," *Lect. Notes Comput. Science; Adv. Petri Nets*, vol. 483, pp. 313–341, Dec. 1991.
[28] M. Figat. *Robotic System HPN Tool (RSHPN Tool)*. Accessed: Mar. 20, 2020. [Online]. Available: https://github.com/mfigat/public_rshpn_tool
[29] P. Trojanek, T. Kornuta, and C. Zielinski, "Design of asynchronously stimulated robot behaviours," in *Proc. 9th Int. Workshop Robot Motion Control*, Jul. 2013, pp. 129–134.
[30] N. Hogan, "Impedance control: An approach to manipulation: Part I—Theory," *J. Dyn. Syst., Meas., Control*, vol. 107, no. 1, pp. 1–7, Mar. 1985.
[31] J. J. Craig, *Introduction to Robotics, Mechanics & Control*. Reading, MA, USA: Addison-Wesley, 1986.
[32] W. Thong and M. Ameedeen, "A survey of Petri net tools," in *Proc. Adv. Comput. Commun. Eng. Technol.*, 2015, pp. 537–551.
[33] K. Jensen, "Coloured Petri nets and the invariant-method," *Theor. Comput. Sci.*, vol. 14, no. 3, pp. 317–336, 1981.
[34] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Middleware for robotics: A survey," in *Proc. IEEE Conf. Robot., Autom. Mechatronics*, Sep. 2008, pp. 736–742.
[35] T. Collett, B. MacDonald, and B. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proc. Australas. Conf. Robot. Autom. (ACRA)*, Dec. 2005, pp. 1–5. [Online]. Available: http://www.ai.sri.com/ gerkey/papers/acra2005.pdf
[36] I. Nesnas, "Claraty: Challenges and steps toward reusable robotic," *Int. J. Adv. Robotic Syst.*, vol. 3, no. 1, pp. 23–30, 2006.
[37] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro–middleware for mobile robot applications," *IEEE Trans. Robot. Autom.*, vol. 18, no. 4, pp. 493–497, Aug. 2002.
[38] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Orebäck, "Orca: A component model and repository," *Softw. Eng. Experim. Robot.*, vol. 30, pp. 231–251, Dec. 2007.
[39] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. and Ng, "ROS: An open-source Robot Operating System," in *Proc. ICRA*, vol. 3, 2009, p. 2.
[40] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," in *Proc. IEEE Int. Conf. Robot. Autom.*, Sep. 2003, pp. 2766–2771.
[41] A. Peekema, D. Renjewski, and J. Hurst, "Open-source real-time robot operation and control system for highly dynamic, modular machines," in *Proc. 9th Int. Conf. Multibody Syst., Nonlinear Dyn., Control*, Aug. 2013, pp. 1–5.
[42] C. Zieliński, "The MRROC++ system," in *Proc. 1st Workshop Robot Motion Control*, Kiekrz, Polska, Jul. 1999, pp. 147–152.
[43] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An architecture for autonomy," *Int. J. Robot. Res.*, vol. 17, no. 4, pp. 315–337, 1998.
[44] L. Petersson, D. Austin, and H. Christenseni, "DCA: A distributed control architecture for robotics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. Expanding Societal Role Robot. Next Millennium*, May 2001, pp. 2361–2368.
[45] R. Simmons, R. Goodwin, C. Fedor, and J. Basista, "Task control architecture: Programmer's guide to version 8.0," Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., May 1997. [Online]. Available: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/TCA/release/tca-8.5/doc/tcaManual.ps
[46] R. Simmons and D. Apfelbaum, "A task description language for robot control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Systems. Innov. Theory, Pract. Appl.*, Oct. 1998, pp. 1931–1937.
[47] E. Ruiz Morales, "GENERIS: The EC–JRC generalised software control system for industrial robots," *Ind. Robot, Int. J.*, vol. 26, no. 1, pp. 26–32, Feb. 1999.
[48] A. C. Dominguez-Brito, D. Hernandez-Sosa, J. Isern-González, and J. and Cabrera-Gámez, "CoolBOT: A component model and software infrastructure for robotics," in *Software Engineering for Experimental Robotics*, (Tracts in Advanced Robotics Springer), vol. 30, D. Brugali, Ed. Berlin, Germany: Springer, 2007, pp. 143–168.

MAKSYM FIGAT (Graduate Student Member, IEEE) received the M.Sc.Eng. degree in computer science (modeling CAD/CAM systems) from the Faculty of Mathematics and Information Science, Warsaw University of Technology (WUT), Warsaw, Poland, in 2013, where he is currently pursuing the Ph.D. degree with the Faculty of Electronics and Information Technology (FEIT), Institute of Control and Computation Engineering (ICCE). He is also a member of the Robotics Group in ICCE working on the design of robot controllers and programming methods. His main scientific interests focus on automatic methods of robot control system generation based on a formal specification. His research in robotics is based on model driven engineering, domain specific language, embodied agent approach, and the formal specification for robotic control systems.

CEZARY ZIELIŃSKI (Senior Member, IEEE) received the M.Sc.Eng., Ph.D., and Habilitation degrees in control and robotics from the Warsaw University of Technology (WUT), in 1982, 1988, and 1996, respectively. He is currently a Full Professor of WUT. He specializes in robotics, heads the Robotics Group, Institute of Control and Computation Engineering, working on the design of robot controllers and programming methods. His research interests focus on robotics in general and especially include robot programming methods, multirobot system controllers, robot kinematics, robot force control, visual servo control, utilization of sensors in robot control, and design of digital circuits. He is also the Vice-Dean of the General Affairs of the Faculty of Electronics and Information Technology, WUT.