Journal of Intelligent & Robotic Systems (2019) 93:787–805
https://doi.org/10.1007/s10846-018-0869-6

Communication Within Multi-FSM Based Robotic Systems

Cezary Zieliński[1] · Maksym Figat[1] · René Hexel[2]

Received: 2 October 2017 / Accepted: 8 May 2018 / Published online: 2 June 2018

Abstract

The paper presents a robotic system design methodology based on the concept of an embodied agent decomposed into communicating subsystems, whose activities are specified in terms of FSMs invoking behaviours parameterised by transition functions and terminal conditions. In the implementation phase, this specification is transformed into a system composed of a whiteboard providing communication means and logically labelled FSMs (LLFSMs) defining the system behaviour. These concepts are used to generate the code of the robot controller. The inclusion of inter-subsystem communication model completes the resulting system design with respect to our previous work that did not account for this model. Thus communication plays a central role in this presentation. The design methodology is exemplified with a rudimentary table tennis ball-collecting robot. The presented methodology and the implementation tools are suitable and beneficial for application to the design of other robotic systems.

Keywords Robotic system specification · Robotic system design methodology · Communication model

1 Introduction

Good practises of software engineering require that the creation of systems involves a specification phase that defines the model of what has to be created, and an implementation phase that discloses how this is done [84]. This paper uses the specification method based on the concept of an embodied agent [54, 92]. The novelty is twofold. Our first contribution is the method of implementation of robotics systems using the concepts of logic-labelled finite-state machines (LLFSMs) and whiteboards. Secondly, we show the simplicity of the transformation of the embodied agent based specification into an (executable) implementation using LLFSMs and the whiteboard for communication. Especially the decoupling achieved by the method of communication between the components of a robotic system is of interest here. We start with a motivation for the presented research and the structure of the paper.

✉ Cezary Zieliński
c.zielinski@ia.pw.edu.pl

Maksym Figat
m.figat@ia.pw.edu.pl

René Hexel
r.hexel@griffith.edu.au

1 Institute of Control and Computation Engineering, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland

2 School of ICT, Griffith University, 170 Kessels Road, Nathan, QLD 4111, Australia

1.1 Motivation

The design of a robotic system starts with the specification of its architecture. As Andrew Tanenbaum points out in his seminal work [86], the term architecture pertains to: the instruction set, memory organisation, I/O, and bus structure. This obviously refers to the architecture of computer systems. However current robotic systems are computer based, thus their architecture has the same foundation. The instruction set defines what the processing capabilities of the system, i.e. possible activities, are. The remaining elements refer to the structure, i.e. the components and their interconnections [23, 55]. In the approach that we follow in our work, the structure refers to the division of the system into subsystems and their interconnections, whilst the activities are specified by a set of such concepts as: Finite State Machines (FSMs), behaviours, transition functions and their arguments, and inter subsystem communication means, all of which will be detailed later in the paper. However, in many existing robotic systems, it is difficult to determine their architecture [55]. Robot control systems are inherently complex, nonetheless their authors rarely formally specify them. Lack of a formal system specification usually results in an obscure architecture that is difficult to modify or extend it in future projects or to integrate with other systems. The process of developing robotic systems requires both appropriate development methods (procedures, architectures, etc.) and tools (modelling languages, middleware and other frameworks). Many attempts have been made to introduce formal languages based on mathematics to determine both the controller structure and its activities [18, 63, 90, 92, 96]. However, the majority of the robot systems were developed based on software engineering [16, 17, 68], downplaying the domain-specific information required by robotics. The proposed method of modelling and specifying robotic systems, based on the concept of the embodied agent [54, 92, 94], relies on such fundamental concepts of robotics as: effectors, i.e. devices influencing the physical environment, and receptors, i.e. sensors gathering the information about the state of that environment, as well as mathematical concepts underlying decision making and sub-system behaviour. However, up till now this design methodology lacked a detailed representation of a vital component: *inter-subsystem communication*. Hence, the present paper delves into that issue. The main purpose of this paper is therefore to present the system design methodology split into two phases, as required by good practises in software engineering. The first phase produces the system model specification based on the concept of an embodied agent, whilst the second one is the implementation phase, which first transforms the model into the composition of a whiteboard and LLFSMs as an interim implementation model that is subsequently transformed into executable robotic system controller code.

1.2 Structure of the Paper

In Section 2 the classification of communication methods from the robotics perspective is introduced. Section 3 briefly presents the concept of an embodied agent, showing both its structure and its activities. The embodied agent is the primary concept of the robotic system design method used in this paper. The universality of an embodied agent is expounded in Section 4. Section 5 describes the interim implementation model, i.e. the elements necessary for the implementation of an embodied agent, focusing on the communication aspect.

As ROS currently is the most popular robot programming framework, the comparison of whiteboard and ROS communication mechanisms is presented in Section 6. Section 7 showcases the proposed design and implementation methods on a robot collecting table-tennis balls. This example is kept concise to enable the reader to focus on the presented methods rather than being forced to deal with the complexity of the robotic system. The last section presents the conclusions derived from the presented work.

2 Communication

In telecommunications, the concept of the communication channel is central to the operation of a communication system [44, 83], distinguishing between free propagation (broadcast, one-to-many) and guided propagation (point-to-point, one-to-one) communication types. Telecommunication deals with such characteristics of the communication channel as its linearity or non-linearity, time variability or invariance, bandwidth, or power limitation. However those physical properties are of secondary importance and can be abstracted away for communication of software implemented subsystems of robots. In the most rudimentary view, the communication model consists of the communicating systems/subsystems and the communication channel. The communication channel (the transmission medium) can be either based on technological means (e.g. wired or wireless) or use natural means (e.g. voice communication using air) [44, 83]. In the latter case the environment itself can be used as a communication channel [4], then stigmergy results [12]. Both technological and natural communication channels can contain buffers, which can store the transmitted information for some time, or exist without a buffer. Transmission delays should not be treated as a capacity of the channel to store data. For communication to take place not only the transmission medium is necessary, but also a common protocol has to exist, as the transmitter and the receiver of the information must have a common understanding of the transmitted information.

In robotics, diverse communication methods have been considered. Three different types of communication between robots were discussed in [25]: two-way communication, explicit one-way communication, and completely implicit communication. This work also points out that there are several difficulties that should be considered in communication between collective robots, i.e. efficiency, fault tolerance and cost, and that in contrast to centralised communication approaches, e.g. [88], communication between robots should be distributed. Comparison of three types of communication, i.e. through the environment, using transmission of state between the agents, and utilising the transmission of the detected goal, was introduced in [8]. Similar criteria were discussed in [65], distinguishing direct (purely communicative act) and indirect communication (based on observed behaviour of other agents). Direct communication was further decomposed into one-to-one (peer-to-peer) and one-to-many (broadcasting), based on the the number of communicating entities. Another division of communication is direct vs indirect communication [34]. Direct communication occurs when a dedicated on-board hardware device is utilised, whilst indirect communication occurs through the environment. The paper also notices that indirect communication is particularly useful from the point of view of Multirobot Systems (MRS), in contrast to direct communication, which in MRS may result in much more expensive and unreliable solution. However, that paper also points out that direct communication in Multi-Agent Systems (MAS) may be used to guarantee locality of interactions and to avoid synchronisation procedures amongst agents. A similar criterion based on how the information may be obtained by robots was introduced in [75], where three types of communication were distinguished: implicit communication through the world, passive action recognition (robots use sensors to directly observe the actions of other robots) and explicit communication (robots communicate directly and intentionally through an artificial communication channel). Either way, care must be taken to avoid the hidden channel problem that results when differing temporal precedence between these means of communication creates inconsistencies between the intent of the system designer and the actual timing and order in which information is delivered [21].

Criteria pertaining to the communication range, communication topology and communication bandwidth were discussed in [24, 26]. However, those criteria concern mainly physical aspects of communication, already discussed in [44, 85]. Yet other criteria pertain to interaction distance (distance between the agents during the communication, i.e. direct physical contact, visual range, hearing range and long range), interaction simultaneity (period between the signal emission and reception, i.e. immediate, long time), signalling explicitness (explicitness of the emitter's signalling behaviour) and sophistication of interpretation (the complexity of the interpretation process that gives meaning to the signal) [48].

The above analysis shows that whilst similar terminology is used for the purpose of communication classification, the meaning attributed to the same names is slightly different, making it difficult to propose a comprehensive communication model. Some of the subdivision criteria are based on physical aspects of communication (topology, place, types of communication channels, bandwidth, etc.), neglecting other also very important criteria (e.g. whether the communicating entities block each other whilst sending/receiving data). Thus, to make the presentation clear, we summarise the criteria most relevant to the subject of this paper as follows. From the point of view of robot control system design, of importance is the possible number of subsystems writing to or reading from the communication channel. This leads to four communication types [39], depending on the number of message producers and the number of consumers: one-to-one (peer-to-peer), one-to-many (e.g. broadcast), many-to-one and many-to-many (e.g. multicast). An example of one-to-one communication is presented in [66] where a system composed of many robots executes a box-pushing task. Cases of one-to-many communication are presented in [7, 45, 65, 74, 76], where the consumers either utilise broadcast messages at their will or ignore them. The many-to-many communication can be organised as e.g. a blackboard system [41] or by stigmergy [12].

Another criterion worth considering is based on the location of the communication channel: inter-system or intra-system. Here the terms *intra-system* and *inter-system* need to be disambiguated. We assume that in the case of intra-system communication, the transmission occurs between nodes existing on a single computer (composed of a single or multiple processors), and in the case of inter-system communication transmission is between nodes residing on many computers. Thus in this case the word *system* does not refer to the robotic system as a whole, but only to the communicating nodes that are within the robotic system, located either on a single computer or on several computers forming a network. The importance of this distinction is due to each of these types of communication requiring different software and hardware means of implementation.

Still another criterium pertains to whether the producer or consumer block until message delivery has completed. Blocking producer behaviour requires the consumer to respond in order to unblock the producer, whilst in the non-blocking mode this is not required, i.e. after initiating the communication, the producer can continue with other activities. Similarly the consumer can either be blocked whilst waiting for the message or not. In the latter case, the consumer can poll the communication channel when required to find out whether there is a message for it to attend to. Still another criterion is storage, i.e. whether the communication channel contains a buffer or not. In the latter case the communication channel does not have the capability to store messages. In the former case the size and type of the buffer is of relevance. The commonly used buffer types in robotics are either queues or cyclic buffers.

Moreover, it should be noted that communication is sometimes associated with cooperation. However the former refers to the ability to transmit information between systems and the latter with the ability to jointly execute a task. Cooperation can be achieved both by using implicit and explicit communication. The former refers to a situation where the consumer of the message is not clearly defined, and thus the interpretation of the message by a consumer may be difficult (as in the case of stigmergy) and in the latter case the consumer is clearly defined, and thus the interpretation of the message using the commonly known protocol is straightforward (as is the case with the majority of technologically based channels or such natural channels as air used for voice communication). In robotics, there is a vast body of literature dealing with cooperation using implicit communication, especially stigmergy. For example [4] presents a schema based control of a multi-robot system used for common object retrieval, [81] describes robots communicating using sensors for detecting infrared radiation reflected by the environment, [87] presents robots communicating using traction forces, [25] describes robots communicating indirectly using on-board visual sensing, whilst [20] reports on the system composed of 20 e-puck robots pushing three different objects based on inputs from infrared proximity sensors, [89] describes a multi-robot system where individual robots use an adaptive behaviour selection strategy based on sensoric information about the location of the transferred object and the other robots, [95] presents multi-robot box pushing based on the detection of the motions of the translocated box. On the one hand the examples of implicit communication are abundant in the literature and on the other hand this form of communication does not influence the system specification from the point of view of communication, as it is realised by the perception subsystem. Thus this paper will limit itself to explicit forms of communication.

3 Embodied Agent

A single- or multi-robot system (a robotic system, in short) is represented by communicating agents [14, 46]. An agent which has a physical body is called an embodied agent [14, 54, 94, 96] – its general structure is presented in Fig. 1. An embodied agent a_j (j is its designator) is decomposed into 5 types of subsystems: control subsystem c_j, virtual receptors $r_{j,k}$, virtual effectors $e_{j,n}$, real receptors $R_{j,l}$ and real effectors $E_{j,m}$, where k, n, l and m are the designators of particular subsystems of the agent a_j [94].

Virtual receptors $r_{j,k}$ aggregate data acquired from the environment through real receptors $R_{j,l}$. Virtual effectors $e_{j,n}$ transform control commands received from the control subsystem c_j into a form acceptable by the real effectors $E_{j,m}$. An embodied agent contains a single control subsystem and zero or more subsystems of each of the other types.

The activity of a system as a whole depends on the individual activities of its subsystems and their interactions. The activity of each subsystem $s \in \{c, e, r\}$ is represented by a hierarchical FSM ${}^{s}\mathcal{F}_{j,v}$ [35], where v indicates the subsystem name. An FSM is represented as a directed graph composed of nodes and directed arcs. Each node represents a state, whilst each arc represents a state transition. Arcs are labelled by initial conditions, being predicates, which,

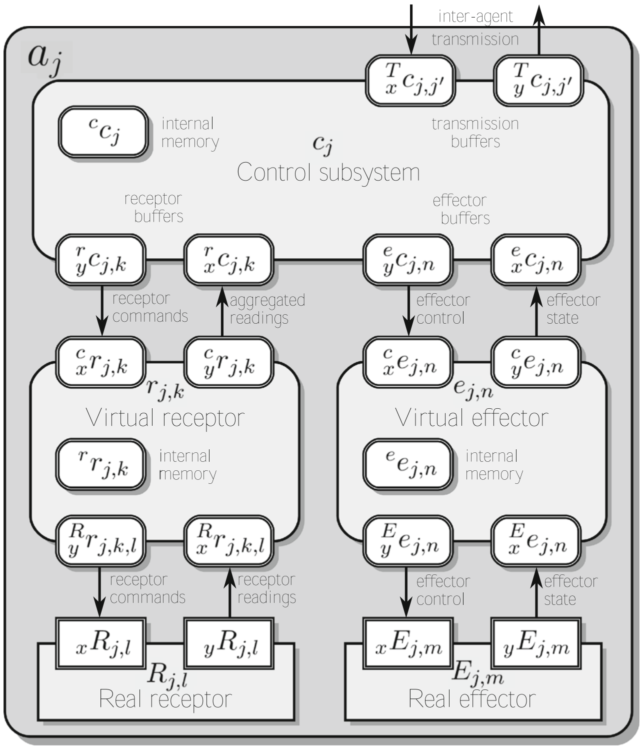


Fig. 1 Internal structure of an agent a_j

if true, cause a state transition from the current to the state pointed at by the arc labelled by this predicate. With each state of an FSM ${}^{s}\mathcal{F}_{j,v}$ there is an associated subFSM ${}^{s}\mathcal{F}^{\mathcal{B}}_{j,v,\omega}$, specifying the behaviour ${}^{s}\mathcal{B}_{j,v,\omega}$, where ω names the behaviour. The subFSM template modelling a behaviour ${}^{s}\mathcal{B}_{j,v,\omega}$ is presented in Fig. 2. Each iteration of a behaviour consists of calculation of a transition function and two-way transfer of data. The transition function ${}^{s}f_{j,v,\omega}$ takes as arguments the current data from the subsystem s internal memory ${}^{s}S^{i}_{j,v}$ and its input buffers ${}_{x}S^{i}_{j,v}$, and calculates the values which are inserted into the output buffers ${}_{y}S^{i+1}_{j,v}$ and the internal memory ${}^{s}s^{i+1}_{j,v}$. Superscripts i and $i+1$ indicate the elapse of a single tick of the discrete time,

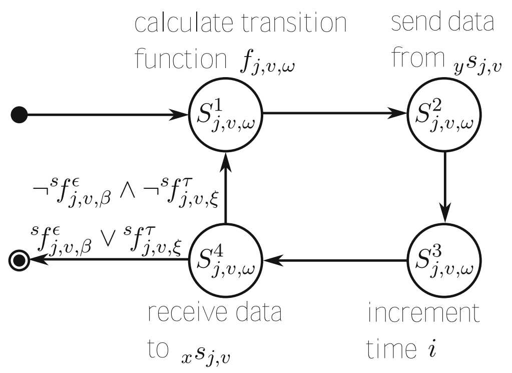

calculate transition function $f_{j,v,\omega}$
send data from ${}_{y}s_{j,v}$
$S^{1}_{j,v,\omega}$
$S^{2}_{j,v,\omega}$
$\neg{}^{s}f^{\epsilon}_{j,v,\beta} \wedge \neg{}^{s}f^{\tau}_{j,v,\xi}$
${}^{s}f^{\epsilon}_{j,v,\beta} \vee {}^{s}f^{\tau}_{j,v,\xi}$
$S^{4}_{j,v,\omega}$
$S^{3}_{j,v,\omega}$
receive data to ${}_{x}s_{j,v}$
increment time i

Fig. 2 FSM ${}^{s}\mathcal{F}^{\mathcal{B}}_{j,v,\omega}$ specifying the behaviour ${}^{s}\mathcal{B}_{j,v,\omega}$

which determines the duration of the behaviour iteration. The behaviour subsequently sends the results of calculations from the output buffers ${}_{y}S^{i+1}_{j,v}$ to the inputs of the associated subsystems and acquires from them new data loading it into input buffers ${}_{x}S^{i}_{j,v}$. Based on the obtained data an associated error condition ${}^{s}f^{\varepsilon}_{j,v,\beta}$ and terminal condition ${}^{s}f^{\tau}_{j,v,\xi}$ are tested (β and ξ are their designators). The duration of a single iteration of the subsystem behaviour defines its sampling period. The iterations of the behaviour terminate when the associated error or terminal condition is fulfilled. At this moment the main subsystem FSM ${}^{s}\mathcal{F}_{j,v}$ resumes its activity, changing its state.

4 Universality of the Embodied Agent

The structure of the embodied agent and its ability to connect to other such agents, thus producing networks, enables the reproduction of other well known and studied architectures. Hence, the embodied agent can be treated as a general means for designing robotics systems. The most popular architectures employed in robotics are: Sense-Plan-Act (SPA), multi-tier (usually either two- (2T) or three-tier (3T) are used), subsumption, and the ones based on some forms of schemata. The universality of the concept of an embodied agent will be briefly justified here by showing its relationship to those architectures. Robotic system software reflects its architecture. Although some work has been done on formal specification of robot control software, e.g. [6, 62, 63, 91, 93] and its formal verification, e.g. [49, 50], unfortunately this has not gained widespread adoption [56]. Usually, architectures are presented by informal text descriptions supplemented by block diagrams, with varying level of details. Early approaches to formalisation of the specification of a robotic system treated such systems as composed of computational modules and followed operational semantics in describing their structure and activities [62, 63]. One of the few systems that employs formal specification and generation of code for the implementation of the lowest layer of robotic system controller is G$^{\text{en}}$oM (Generator of Modules) [1, 36]. It is integrated with the BIP (Behaviour Interaction Priorities) [10] framework and toolset, which is used to formally specify and verify the correctness of the produced system, treated as a real-time, component-based complex structure. However, as the majority of architectures presented in the literature is not specified formally this discussion will not rely on a formal description.

Frequently, hierarchical layered architectures are designed. One of the oldest hierarchical architectures is the sense-plan-act (SPA) structure, used for instance by the Shakey robot in the 1960s [71]. Shakey sensed its environment, invoked action planning, and then executed the thus derived plan. Later, different criterions have been used for the decomposition into layers. Usually, the time of the execution of a single behaviour iteration of a component within the layer is the discriminating factor. The higher the layer within the system, the longer the time of its single iteration execution. In some cases, task abstraction is the criterion, i.e. a single activity at an upper layer is decomposed into several lower layer activities, e.g. NASREM (NASA Standard Reference Model) [60]. Usually, as a consequence, the utilised ontology becomes more abstract going up the layers. The lowest layer is behavioural and is tightly coupled with actuators and sensors. The middle layer is the sequencing layer (sometimes termed as executive layer [56]), responsible for selecting behaviours that will eventually realise the task. The uppermost tier usually involves deliberation. In robotics, deliberation is associated with planning. From what has been said, it is evident that planning and sequencing layers act together to accomplish the plan and in some systems are tightly bound together (e.g. CRAM [9], CLARAty [68]), hence 2T instead of 3T architecture results. The proposed embodied agent based architecture can accommodate all of these possibilities. By design, an embodied agent forms a two tier architecture: virtual entities controlling the real devices form the first tier and the control subsystem forms the second one. Above that, a hierarchy of computational agents (embodied agents lacking effectors and receptors) can form supplementary tiers. A computationally demanding planning task can be decomposed into several parts, implemented as several interacting computational agents.

As planning is a time-consuming task, the subsumption architecture was proposed [13–15]. Its overall behaviour is the result of the interaction of the activities of modules, which react to input from other modules and sensoric stimulus. If lower layer reactions can not cope with the task, upper layer reactions suppress or inhibit them, taking over control, thus subsuming their actions. As the SPA architecture tends to be prohibitively slow and the subsumption architecture, although very reactive, lacks foresight, hybrid architectures emerged [5]. Those were usually designed as three tier systems (e.g. [1, 38]), having: planning (deliberative layer), sequencing (usually implemented as a finite state machine governing the actions of the system), and device control layers (responsible for motor control and sensor data aggregation – a reactive layer resulting if both were combined). Behavioural systems [5, 6], based on reactivity, have gained widespread attention. In order to satisfy the requirement of the fastest possible response, reactivity should be implemented at the lowest level. The shortest possible loop starts with proprioceptors and directly influences the real effector. As the activities of the reactive layer rely mainly on sensor input and to a very limited degree on internal state (i.e. memory) it is very easy to reproduce this layer within an embodied agent, where the quickest responses are organised within the virtual effector and are based on proprioceptive input. An example of this approach is the utilisation of force/torque sensing directly within the virtual effector (e.g. [97]). If a subsumption architecture [13, 15] is to be reproduced, its modules can be represented by blocks within the data flow diagram definition of a transition function by using inhibiting and suppressing links (as presented in [54]). Otherwise, the modules can deliver the results of their computations into a composition unit, which will compute the final outcome, subsequently dispatched to a real effector for execution. If a slower reaction is permissible, exteroceptive input from the virtual receptors can be used by the control subsystem, which will use the same scheme of operation as the virtual effector. Thus, the same pattern will be reproduced at different levels of the control hierarchy. This can be continued at even higher levels of the hierarchy by using extra computational agents [98].

Schema Theory was formulated for the purpose of representing systems created to study both artificial intelligence and brain theory [2, 3, 5, 62, 63]. Schema Theory provides guidance to building models of animal and artificial agent behaviour, stating that behaviour is the result of competition and cooperation between schemata, which are conglomerates of knowledge and methods of processing [2]. Instances of schemas are created either by other schemas or as a reaction to sensory stimulus. Perceptual schemas process sensory input, whilst motor schemas produce action patterns. A network of schemas is called an assemblage. The overall behaviour of an agent is produced as a result of interactions between schemas composing the network. Such schemas can be embedded in an embodied agent, as virtual receptors reflect perceptual schemas, whilst virtual effectors reflect motor schemas. At a higher level, embodied agents without effectors match perceptual schemas, whilst those without receptors match motor schemas, this multi-agent networks can also reproduce assemblages. The application of Schema Theory in robotics followed different paths. One of them is described in [3, 5]. In this case, motor schemas are basic units of behaviour. Motor schemas act in conjunction with embedded perceptual schemas. Concurrently acting behaviours, i.e. motor schemas, compute individual results that are composed together into the final control outcome. This architecture is easily reproduced in an embodied agent. Perceptual schemas are reproduced by virtual sensors. The control subsystem implements a behaviour that is parameterised by a transition function which in turn is decomposed into concurrently computed functions that define the computations of individual motor schemas. The transition function composes the results produced by partial transition function into an outcome that is dispatched to the virtual effector. A more formalised approach to Schema Theory is called Robot Schemas (RS) [63]. RS form a formal model of distributed computing in the robotics domain. The model is built of concurrent computing agents, each being a Schema Instance (SI). SIs communicate through ports, with which data types are associated. At SI instantiation time, its ports are connected to the ports of other SIs. Communication occurs when an SI writes to one of its output ports. Basic schemas are defined by the port connections, local variables and the procedural behaviour section, which specifies how the schema reacts to input. Basic schemas, once initiated, iteratively execute the behaviour section, synchronously reading or writing to ports, performing computations on local variables and input ports and instantiating and deinstantiating other SIs. Complex schemas, i.e. assemblages, are a recursive composition of basic schemas into ever more complex ones. Assemblages do not have the behaviour definition part, as that is defined by the schemas they are composed of. Only the port connection part exists. SIs form a dynamical network which grows or shrinks as the activities of the system take place. The task is represented as the outermost assemblage. Thus, the system is composed of sensory schemas, motor schemas and schemas that connect them, i.e. task schemas. In general, this reflects the structure of an embodied agent or can be reproduced as a network of embodied agents with some of their internal components missing. It should be noted that the design of an RS-based system focuses on computation, however this involves also instantiation and deinstantiation as well as communication, all considered at the same computational level. Thus, the basic level is flat and all aspects of system activities are entangled, hence intelligibility is sacrificed. Hierarchy emerges from assembling lower level schemas into higher level ones, but as the design principles of the lowest level are not expressed explicitly the creation of such systems is difficult. An embodied agent introduces a hierarchy of concepts starting from the lowest level: transition function and terminal condition (purely computational), behaviour (adds communication), FSM (selects behaviours), subsystems (distinguish perception, action and task control), agents (form networks of any size). Each of those concepts can be dealt with separately, thus it is easier for the designer to focus on a single entity at a time, simplifying the process of system creation. Systems composed of embodied agents, besides being capable of reproducing the well known architectures, can be used to create other architectural patterns [94, 98].

5 Implementation of an Embodied Agent

The architecture of an embodied agent presented in Section 3 is implemented using several earlier-mentioned concepts. Here we shall concentrate on two important ones associated with the communication and behaviour selection [31]: the whiteboard, which is used as the main communication means between subsystems, and Logic-Labelled Finite State Machines (LLFSMs), which govern the actions of each of the subsystems present in the agent, including the exchange of information between subsystems. The whiteboard is our implementation of the general concept of a blackboard [42], thus whenever we refer here to the former we refer to our implementation and for the general concept, the latter is referred to.

The concept of a blackboard was created for the purpose of studying artificial intelligence (AI) [69]. The blackboard model consists of three major components [22, 43, 70, 82]: knowledge sources (KSs) – corresponding to the domain experts, blackboard data structure – the global resource where information to be shared is written, and control mechanism – responsible for processing the knowledge contained in the blackboard and ensuring that only one domain expert has access to the blackboard at a given time. The blackboard can be any data structure accessible by all the KSs. Each KS operates in the following sequence: reads the information from the blackboard, acts based on the information found and writes its conclusions and hypothesis into the blackboard. The work on blackboards in AI concentrated on several aspects: the organisation of knowledge data structures, reasoning utilising those data structures as well as providing access to them. The concept of the whiteboard focuses on data access. The use of data-centric communication similar to blackboards has transcended AI and found its way into the wider software engineering community. The Data Distribution Service (DDS) is a standard formulated by the Object Management Group (OMG), implementing Data-Centric Publish/Subscribe (DCPS) communication model [40, 58]. The main purpose of DDS is to assure dependable communication in dynamic, networked environments between distributed producers and consumer fulfilling real-time constraints. Whiteboard implements the pull approach, whilst the DDS, which is based on Publish/Subscribe model, is based on the push approach.

Blackboards have also been used in robotics. The following examples show only a fraction of the diversity of their applications. A reconfigurable mobile robot used a blackboard enabling a horizontal system decomposition allowing the cooperation between all the subsystems considered as knowledge sources executing a specific task [72]. The CrunchBot [37] used a Bayesian blackboard to solve the mapping problem for a mobile robot equipped with whisker sensors. Autonomous mobile robots dealing with hazardous material spill emergency situation used a blackboard system to integrate information from various sensor-based and knowledge-based subsystems [73]. Those examples and the plethora of other works on the utilisation of blackboards in robotics show that the effort has concentrated on either different aspects of knowledge processing or on providing a balanced access to knowledge to all involved actors or on both. The concept of a whiteboard is derived from blackboards, however it focuses on information sharing, thus the communication aspect. The contents of the whiteboard is of secondary importance as is the purpose of information processing. The focus is on providing efficient communication means. A similar communication concept was introduced in [1], namely a poster. It uses shared memory for the purpose of communication. A poster is a structured shared memory only writable by its owner and readable by any element of the architecture. Posters are completely independent from each other. They are not regions of a common blackboard and cannot be written to by many producers, thus they do not have all the properties of a blackboard. A whiteboard has all the communication abilities of a blackboard, so it is more general than a poster.

5.1 Whiteboard

The `gusimplewhiteboard` (Griffith University Simple Whiteboard) is a library for organising inter-process communication [31]. It implements a mutation of the general blackboard concept [42]. It enables the transfer of C++11 objects through shared memory. As it is linked in, no broker is necessary. The message transfer is idempotent, i.e., if the transfer is repeated the result does not change. No queues need to be involved, however, if necessary, they can be organised, as the whiteboard does not determine the data structures it will be storing – this is the choice of the system designer using the whiteboard.

Shared memory communication exhibits certain advantages over the traditional approach taken by, e.g. ROS messages and services, [47]. Producers and consumers use local instances of classes to communicate. When a producer sends a message, it copies an object containing the message from its local store to the static instance of the class in the `gusimplewhiteboard`. When a consumer wants to retrieve the message, the current instance of the object in the `gusimplewhiteboard` is transferred to the local memory of the consumer. When operating in a distributed system, a UDP-based sharing algorithm that also makes use of the Time-Triggered Architecture (TTA) [52] is used to broadcast data to other receivers [61]. The implementation of the distributed version of the whiteboard uses efficient, fault-tolerant communication for distributed systems as presented in [11, 51, 57] [52, 53]. Compared to the DCPS [58], the whiteboard does not offer inherent mechanisms for distributed reconfiguration, but offers a statically defined data classes and O(1), fail-silent communication within bounded real-time constraints [31]. A more detailed discussion of the networked version of the whiteboard [61] is presented in Section 6.

5.2 Communication Model

The blackboard can be considered as a broker, which enables the producer to depose in it, in a non-blocking way, whatever data it has to send out, whilst the consumer may act in one of the following modes. Either it acts in the push mode (similar to the DCPS), where the blackboard invokes a specific function of the consumer-subscriber, thus notifying it of the arrival of new data, or in pull mode, where the consumer accesses the data when it requires. The former usually involves queuing of messages, whilst the latter implies the delivery of only the most current data. The push model implies close coupling of modules, where the subscriber must react to all messages issued by the producer [30]. In the pull model, the components are decoupled, the consumer may react in its own time, and only to some of the produced messages. In both communication models, the publisher introduces a message into the blackboard. In the case of the push model the blackboard notifies the consumer that the message it has subscribed to is available and the subscriber invokes a call back function in response. The messages are queued, so the consumer should somehow react to all of them. An event driven architecture results. It relies on an assumption that there is enough of computational power to handle the messages as they come. However, in the case of the pull model, such an assumption is not necessary. The consumer is not notified about the arrival of a new message. When the consumer needs the new message it retrieves form the blackboard the most current one, neglecting the obsolete ones (in reality usually they will even not be available). The publisher and subscriber are very loosely coupled, without assumptions as to the available computational power, and thus the ability to respond timely to the incoming stream of messages results. Obviously, the pull model works on the principle that we get a better response, if the system at any moment uses the most current data. Both the publisher and the subscriber act with their own frequencies, without having to synchronise their activities. This way, no global timing schedule is required, fully decoupling the development of both components. The pull model is immune to messages being lost by definition, as it does not require response to every message, whilst the push model requires delivery guarantees, because every message attracts attention. The majority of middleware utilising the concept of a blackboard still relies on the push model. For instance, ROS topics act in push mode. However the blackboard provides an opportunity to base the communication on the pull model. Whilst the proposed whiteboard can be used in both push and pull modes, the latter should be the method of choice. In the majority of middleware, the structure of the contents of the messages transferred is limited, whilst in `gusimplewhiteboard` any C++11 objects can be used [31].

5.3 LLFSM

LLFSMs are FSMs, in which transitions from one state to another state are based on the fulfilment of logic expressions rather than events. As events are usually queued and logic expressions are computed just on the basis of current data, as is more appropriate for robot control, this form of an FSM is preferred. In the implementation presented here, the LLFSMs, in principle, operate concurrently, but in reality are scheduled sequentially, thus critical sections and synchronisation points become unnecessary. This inherently removes the possibility of race conditions. The communication between LLFSMs uses whiteboards. The utilised LLFSMs are synchronous automatons. With every state, three portions of code are associated: the code that is executed on entering the state, the exit code invoked when the logic expression with one of the outgoing transitions (directed arcs) is satisfied, and the code that is executed when none of those logic labels is true, hence the execution of this code may be repeated many times.

The LLFSMs can be composed hierarchically. Each LLFSM can be suspended, thus enabling the execution of a subLLFSM. It can also be restarted from the state it was suspended in or restarted from its initial state[1]. The subLLFSMs act like subprograms and need not act simultaneously – this prevents state replication [19]. The higher level LLFSM becomes dormant during the actions of the subLLFSMs, increasing the degree of determinism of the system performance.

From what has been said here, it is evident that the implementation of both the subsystem FSM ${}^{s}\mathcal{F}^{\mathcal{B}}_{j,v}$ and the behaviour subFSM ${}^{s}\mathcal{F}^{\mathcal{B}}_{j,v,\omega}$ (specifying the behaviour ${}^{s}\mathcal{F}_{j,v,\omega}$), as LLFSMs is straightforward, as the directed arcs of those FSMs are labelled by predicates, i.e. logic expressions, hence their equivalence is ascertained.

5.4 Scheduler

An embodied agent is composed of one control subsystem and usually several virtual effectors and receptors. Each of

[1]Incidentally, this property also enables the implementation of the subsumption architecture [13] by using those automatons [19].

those subsystems is governed by one FSM – here implemented as an LLFSM. The proposed solution schedules sequentially the LLFSMs representing producers and consumers, thus minimising the problems of determining the delays in servicing the transfer of data. The majority of middleware, e.g. ROS, Microsoft Robotics Studio, relies on concurrency, data queuing and the non-deterministic TCP/IP suite. Hence real-time performance is compromised. Deterministic scheduling has the following advantages over the non-deterministic multi-threaded approach. Full concurrency, where switching between threads is operating system dependent, is much more difficult to envision and correctly apply by the robot control system designer, as it necessitates: thread synchronisation, vigilance of nondeterministic communication delays, managing critical sections, problems with fairness as well as avoiding deadlocks, live-locks, and starvation. Moreover, CPU context switch overhead has to be taken into account. The sequential LLSFSM switching procedure enables fairly simple formal correctness verification of models, as it avoids the usual combinatorial explosion of concurrent threads. The sequential switching is time-triggered in contrast to the frequently used event-triggered approach, where due to peak-load situations it is often impossible to assure reliability. Specifications using LLFSMs constitute executable models that can be formally verified. A more detailed treatment of this subject can be found in [29, 32, 33].

Each iteration of an LLFSM operation starts with reading the relevant variables from the whiteboard and it ends with depositing the results of computations in it. Each iteration is treated as atomic, thus no inconsistencies can arise. The LLFSMs are executed in a round-robin fashion one iteration of each LLFSM at a time.

To show how the scheduler works in conjunction with the LLFSMs, a simple example is presented here. It is assumed that within the robotic system, only two subsystems v and h exist, where v and h are the specific names of the subsystems. The exemplary activities of each subsystem may be represented by hierarchical FSMs: ${}^{s}\mathcal{F}_{j,v}$ and ${}^{s}\mathcal{F}_{j,h}$, presented in Fig. 3.

With each FSM state a behaviour is associated. It is represented as a subFSM based on behaviour template from Fig. 2. For simplicity it was assumed that the terminal condition of each behaviour is always fulfilled, whilst error condition is not. This implies that all behaviours are executed only once in each state of the FSM. The functioning of the scheduler is presented assuming that initially FSM ${}^{s}\mathcal{F}_{j,v}$ is in the state $S^1_{j,v}$ and the FSM ${}^{s}\mathcal{F}_{j,h}$ is in the state $S^1_{j,h}$. The scheduler activates one state of each FSM or subFSM at a time, switching between all FSMs and subFSMs. Obviously this causes the execution of the code associated with that state. In this case the scheduler starts with state $S^1_{j,v}$ of ${}^{s}\mathcal{F}_{j,v}$ and then switches to $S^1_{j,h}$ of ${}^{s}\mathcal{F}_{j,h}$. Being in $S^1_{j,v}$ ${}^{s}\mathcal{F}_{j,v}$ transits to $S^1_{j,v,1}$ of the subFSM executing the behaviour ${}^{s}\mathcal{B}_{j,v,1}$, so this will be the next state activated by the scheduler. The fourth state activated will be $S^1_{j,h,1}$ of the subFSM ${}^{s}\mathcal{F}_{j,h,1}$. The full switching sequence for both FSMs ${}^{s}\mathcal{F}_{j,v}$ and ${}^{s}\mathcal{F}_{j,h}$ and their subFSMs executing behaviours is presented in Fig. 4.

Whiteboard communication in conjunction with this way of switching the LLFSMs by the scheduler avoids deadlocks, because only one FSM accesses the whiteboard at a time. Moreover, because the variables contained in the whiteboard are located in cyclic buffers, always there is available data that can be accessed, alas not always the most current.

5.5 Communication Organisation

As in many cases, the example that follows requires only intra-system communication between subsystems. Whilst the utilised whiteboard is capable of providing inter-system communication [61], this is not our focus here. It should be noted that the whiteboard intra-system communication gives real-time guarantees (when the underlying operating system is real-time), whereas the inter-system one would depend on the network protocol.

The whiteboard is a table with slots. Each slot creates a separate communication channel containing a cyclic buffer. Hence, buffered communication based on a cyclic buffer of limited size is realised. The whiteboard as a whole supports concurrent many-to-many communication, but a message slot (buffer) with multiple producers requires locking and is therefore not recommended. We therefore recommend one-to-one or one-to-many communication, which is

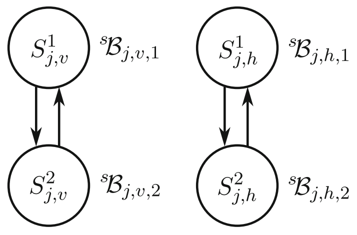

$S^1_{j,v}$
${}^{s}\mathcal{B}_{j,v,1}$
$S^2_{j,v}$
${}^{s}\mathcal{B}_{j,v,2}$
$S^1_{j,h}$
${}^{s}\mathcal{B}_{j,h,1}$
$S^2_{j,h}$
${}^{s}\mathcal{B}_{j,h,2}$

Fig. 3 The activities of two subsystems v and h of a_j

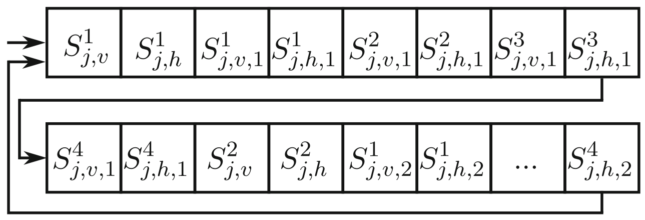


Fig. 4 Sequence of state activations of FSMs ${}^{s}\mathcal{F}_{j,v}$ and ${}^{s}\mathcal{F}_{j,h}$ and their subFSMs executing particular behaviours

lock-free with O(1) complexity for access operations. In the following example one-to-one communication will be utilised. The LLFSMs together with the whiteboard enable the implementation of both blocking and non-blocking communication modes both in the case of the producer and the consumer. However, in the presented example both the producers and the consumers operate in the non-blocking mode. In the experiment, a single sequential scheduler is used, thus subsystem LLFSMs are executed sequentially.

6 Comparison of Whiteboard and ROS Communication Mechanisms

Initially, ROS was designed for large service robot control software, which was meant for implementation on several computers connected via Ethernet [78], thus ROS was in fact designed for communication between computers. However later it was mainly used for communication within a single computer. Hence below we introduce ROS communication mechanisms provided for a single computer and compare them with the whiteboard approach.

ROS uses: topics, services, nodelets, and a parameter server. Those communication concepts may be utilised only for certain types of communication (based on the localisation of the communicating nodes), i.e. inter-process and intra-process. The inter-process communication occurs between ROS nodes (implemented as processes) utilising TCP/IP server socket [80] and peer-to-peer topology, which requires some sort of lookup mechanism to allow processes to find each other at runtime. This mechanism is implemented as the ROS master [78]. ROS nodes may communicate with each other utilising topics, services and parameter service.

A ROS topic is a named bus implementing communication between nodes. Communication between two nodes may only occur if both nodes exist and are already registered with the ROS master (utilising Remote Procedure Call (RPC)). Nodes communicate utilising TCP. Since a ROS topic uses a publish/subscribe mode of operation, thus it implements the *push* paradigm, which forces the subscriber to act in an event-driven fashion. Events are deposited into the middleware which relays them to call-backs from the subscribers. It is assumed that there will be sufficient computational resources to enact all the threads generated and to execute the subscriber callbacks. An additional assumption is that events would occur with enough sparsity that the call-backs would be completed by the time they are executed again or alternatively require handling of concurrency issues. This event-driven approach also results in coupling and has timing consequences. The push approach compels the subscriber to keep up with message reading in order not to lose any data, even if a specific update rate is not necessarily convenient for the subscriber. In congested systems using TCP, if the acknowledgments do not reach the publisher on time, the packets will be retransmitted, thus the subscriber will eventually be flooded with repeated packets [77]. This means that the subscriber activity is tightly coupled with the publisher update rate. Additionally, the push approach assumes that the subscriber reacts to transmitted data immediately (otherwise data is queued), thus the data is presumed to be fresh and as recent as possible, what might not be the case. This will result in reaction delays. The negative influence of such delays on the performance of a drone are presented in [47]. Timeliness, sequencing and reliability of the data are taken for granted. Such scenario is very optimistic, which in fact, does not occur in reality often. Therefore, it requires that the producer must eventually be slowed down or complex handshaking protocols must be deployed, otherwise the data will be lost [30]. The push paradigm, used by topics, does not assure idempotent transfer of messages, thus an additional effort must be done to ensure that the data was received exactly once [30].

The ROS service is a simple RPC, thus the client is blocked until it obtains the response from the server. This type of communication has three additional features. It uses only plain old data (POD) structures, collections of basic types, thus object oriented concepts using inheritance and polymorphism can hardly be modelled [27], whilst the approach based on whiteboards implements object oriented approach [30]. Additionally, ROS service forces client-server relationship between the two communicating nodes, possibly resulting in delays due to waiting for the server reply. Moreover, the ROS service again uses the TCP/IP communication protocol.

Parameter server is the last of the inter-process ROS communication concepts for intra-system inter-process communication. It is a shared, multi-variate dictionary stored in the ROS master. It is accessible by nodes via network APIs. However, it was not designed to be high-performance, as it is to be used for static data such as configuration parameters [79].

Nodelets are threads that use intra-process communication of ROS. Data is exchanged between nodlets executed within the same process by sharing memory. However, the developers have to take care to avoid contention due to concurrent access to the data, e.g. by using mutexes or by allocating new memory whenever new data is written. This again results in a large performance penalty as shown in [27].

There exist robotic systems that have special communication requirements. In some of them, both the producer and the consumer should be able to operate in a non-blocking fashion (i.e. the communicating nodes should be decoupled), as blocking implies possible lack of reaction from the system, when such a reaction is necessary. The communication should be capable of operating in the time-triggered mode, because in this mode of operation the system can react both to the message and its lack (usually those reactions are different), i.e. lack of a message is also a message. It is required that the communication be low latency and low jitter in the case of intra-system transmission, as quick reactions to events occurring in the environment may influence the survival of the robot. Finally, communication should be implemented in the same way both for threads and processes, because the imposition of single process system structure (even with many threads) is not acceptable, i.e. multi-process implementation is treated as an advantage.

None of the four ROS communication methods fulfils all of these requirements. The parameter server introduces high latency. ROS services use RPC, thus they block both the producer and the consumer. In the case of ROS topics, the consumer is activated by an event associated with the producer depositing a message in the communication channel, thus communication is event triggered, not time triggered. However non-blocking operation of the consumer using ROS and C++ can be realised through a separate thread dealing with acquisition of the message and using nodlets – rather a complex solution introducing significant communication delays. Unfortunately, nodelets can be used only for communicating threads, but not processes. As none of the ROS communication mechanisms satisfies the above-mentioned requirements, the whiteboard was chosen (in fact, a ROS package for the whiteboard is available at [67]). This approach provides intra-system inter-process communication, which does not block neither the producer nor the consumer; as it operates on common memory it is very low-latency [31, 47]; by using the pull mode of operation it operates in time triggered mode; and moreover it can be used both by threads and processes in the same fashion.

If whiteboards could be used only for intra-system communication the proposed approach would not be scalable. However there exists a network implementation of the whiteboards [61], but here the latency introduced both by them and ROS are similar. In general, network based implementation of the concept of the whiteboard requires that each computer in the system instantiates its own whiteboards. One of those whiteboards contains variables for local communication whilst the others reflect the variables in the remote whiteboards. The latter variables are updated periodically broadcasted to other computers in the network utilising the UDP Bridge module. The UDP Bridge module is used for all network connectivity and connects directly to the shared data structures of the whiteboard. The UDP Bridge can either be run as an independent process (for debugging purposes) or can be started as a thread by an instance of the Remote Whiteboard class. The UDP Bridge has three main components: Listener – listening for incoming packets, Broadcaster – broadcasting packets when scheduled, and Manager – interacting with the whiteboards. Due to the fact that whiteboars contain cyclic buffers that are transmitted as a whole, even if (due to the use of UDP) a transmitted message is lost, in the next transmission period both the most recent data and previous data will be delivered.

7 Experiment

The robot system design methodology described in, e.g. [92], utilising hierarchical FSMs [35], has been applied to the design of the control system of a mobile robot collecting table-tennis balls. The robot (inspired by [64]) uses a suction mechanism to collect table-tennis balls and a camera to detect them, as well as a sonar to avoid obstacles. Based on the estimated ball position, the robot moves in the direction of the closest ball and vacuums it into its container. This simple, yet useful, robot is used here for the purpose of exemplifying the proposed general robotic system specification method and the implementation of the resulting system. Both the system architecture and the behavioural approach used resemble the well known solutions presented in Section 4. As the purpose of the example is the presentation of the proposed specification and implementation method, a well known architectural solution and control paradigm have been used, to enable the reader to focus only on the methodology.

7.1 Specification

The specification describes both the structure of the designed system and its activities. For the purpose of briefness, only relevant excerpts from the full system specification are presented here. Only a single behaviour of the control subsystem is specified formally here to provide an example – the others follow the same pattern.

Structure It suffices to represent the structure of the ball collecting robot (bc) by a single embodied agent a_{bc} (Fig. 5). This agent contains: the control subsystem c_{bc} and four

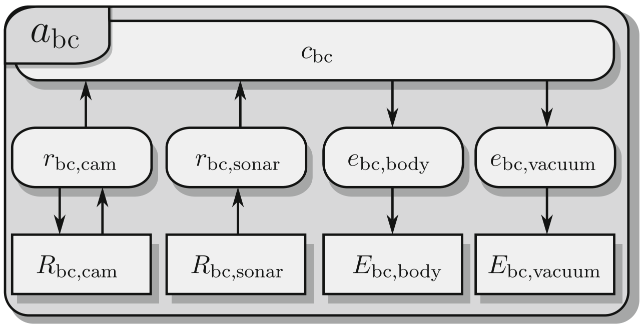


Fig. 5 Embodied agent representing the table-tennis ball collecting robot

virtual subsystems: $r_{\text{bc,cam}}$, $r_{\text{bc,sonar}}$, $e_{\text{bc,body}}$, $e_{\text{bc,vacuum}}$ and four corresponding real subsystems: $R_{\text{bc,cam}}$, $R_{\text{bc,sonar}}$, $E_{\text{bc,body}}$ and $E_{\text{bc,vacuum}}$. The real receptor $R_{\text{bc,cam}}$ detects balls, whilst the real receptor $R_{\text{bc,sonar}}$ detects obstacles. The real effector $E_{\text{bc,body}}$ controls four motors to which wheels are attached (a skid-steering mobile platform is used), whereas $E_{\text{bc,vacuum}}$ controls the vacuum cleaner providing suction. $R_{\text{bc,cam}}$ acquires images of the environment and sends them to the virtual receptor $r_{\text{bc,cam}}$, so that it can detect table-tennis balls and subsequently inform the control subsystem c_{bc} whether it detected at least one ball. If a ball is detected $r_{\text{bc,cam}}$ sends to c_{bc} an estimated position of the ball within the image space. $R_{\text{bc,sonar}}$ provides to the virtual receptor $r_{\text{bc,sonar}}$ the measured time (i.e. interval between the emission of the ultrasonic signal and receiving its echo), based on which it calculates the distance to the obstacle. Next it sends the information to c_{bc} whether an obstacle was detected. The c_{bc} commands both virtual effectors $e_{\text{bc,body}}$ and $e_{\text{bc,vacuum}}$ using the information obtained from $r_{\text{bc,cam}}$ and $r_{\text{bc,sonar}}$. Finally $e_{\text{bc,body}}$ controls $E_{\text{bc,body}}$, whilst $e_{\text{bc,vacuum}}$ controls $E_{\text{bc,vacuum}}$.

Each subsystem has its own input, output and internal buffers. Below only those control subsystem c_{bc} buffer contents are introduced, which are relevant for this partial presentation of the specification.

– Internal memory ${}^{c}c_{\text{bc}}$:

  r_min – minimum ball size triggering the vacuum cleaner (constant)

– Output buffer ${}^{e}_{y}c_{\text{bc,body}}$:

  cmd – command for $e_{\text{bc,body}}$,
  cmd ∈ {MOVE, LEFT, RIGHT, STOP},
  d_x – X coordinate of the location of the centre of the ball with respect to the image centre (expressed in pixels)
  vel – velocity of the robot (range: 0–100)

– Output buffer ${}^{e}_{y}$ $c$bc, vacuum:

  cmd – command for $e_{\text{bc,vacuum}}$,
  cmd ∈ {TURN_ON, TURN_OFF},

– Input buffer ${}^{r}_{x}$ $c$bc, sonar:

  obst_det – signals whether an obstacle was detected or not;

– Input buffer ${}^{r}_{x}$ $c$bc, cam:

  ball_det – ball was detected or not;

  x, y – X and Y coordinates of the detected ball centre (in pixels)
  w, h – image width and height (in pixels)
  r – detected ball radius (in pixels)

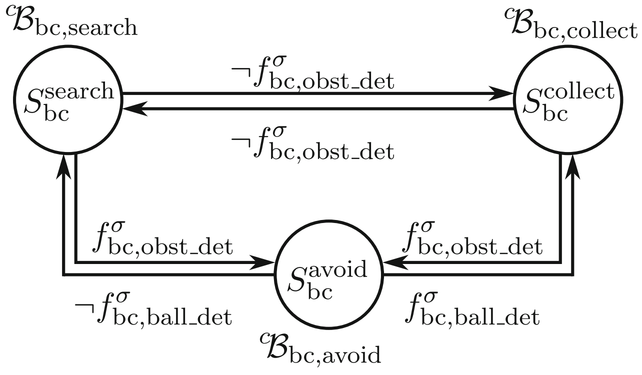
Activities The behaviour of a_{bc} is defined by the activities of its subsystems and the interactions between them. The activities of c_{bc} are defined using a hierarchical FSM ${}^{c}\mathcal{F}_{\text{bc}}$ presented in Fig. 6. Each state of the FSM ${}^{c}\mathcal{F}_{\text{bc}}$ is associated with the corresponding behaviour: ${}^{c}\mathcal{B}_{\text{bc,search}}$ (robot searches for the balls within the environment), ${}^{c}\mathcal{B}_{\text{bc,collect}}$ (robot collects balls) and ${}^{c}\mathcal{B}_{\text{bc,avoid}}$ (robot avoids obstacles). Each behaviour is modelled using the general behaviour template presented in Fig. 2, requiring the definition of a transition function, terminal condition, and the determination of the communication model. Below, the definition of the behaviour ${}^{c}\mathcal{B}_{\text{bc,collect}}$ is presented, thus transition function ${}^{c}f_{\text{bc,collect}}$ and terminal condition ${}^{c}f^{\tau}_{\text{bc,collect}}$ are presented. Other behaviours are defined in a similar way.

Canonical decomposition of the transition function ${}^{c}f_{\text{bc,collect}}$, based on the components of the output buffer, is as follows:

$$
{}^{c,e}f_{\text{bc,collect}} \triangleq
\begin{cases}
{}^{e}_{y}c^{i+1}_{\text{bc,body}}[\text{cmd}] &= \text{MOVE} \\
{}^{e}_{y}c^{i+1}_{\text{bc,body}}[\text{d\_x}] &= {}^{r}_{x}c^{i}_{\text{bc,cam}}[\text{x}] - {}^{r}_{x}c^{i}_{\text{bc,cam}}[\text{w}]/2 \\
{}^{e}_{y}c^{i+1}_{\text{bc,body}}[\text{vel}] &= \text{calculate\_vel}({}^{r}_{x}c^{i}_{\text{bc,cam}}[\text{r}]) \\
{}^{e}_{y}c^{i+1}_{\text{bc,vacuum}}[\text{cmd}] &= \text{TURN\_ON}, \quad \text{for}\, {}^{r}_{x}c^{i}_{\text{bc,cam}}[\text{r}] \geqslant {}^{c}c^{i}_{\text{bc}}[\text{r\_min}] \\
{}^{e}_{y}c^{i+1}_{\text{bc,vacuum}}[\text{cmd}] &= \text{TURN\_OFF}, \quad \text{for}\, {}^{r}_{x}c^{i}_{\text{bc,cam}}[\text{r}] < {}^{c}c^{i}_{\text{bc}}[\text{r\_min}]
\end{cases}
\tag{1}
$$

The internal memory contains only constants, thus it does not require a transition function. The virtual receptors do not have input buffers fed by the control subsystem, thus those transition functions are not needed.

The partial function (1) calculates the parameters of the control commands for $e_{\text{bc,body}}$ and $e_{\text{bc,vacuum}}$. In every iteration of the behaviour, ${}^{c}\mathcal{B}_{\text{bc,collect}}$ a new desired velocity of the robot is calculated using the function calculate_vel, which takes into account the size of the detected ball image. Using the X coordinate of the the centre of the ball with respect to the centre of the image, an offset is computed that is used to navigates the robot towards the ball.



${}^{c}\mathcal{B}_{\text{bc,search}}$
${}^{c}\mathcal{B}_{\text{bc,collect}}$
$S^{\text{search}}_{\text{bc}}$
$S^{\text{collect}}_{\text{bc}}$
$\neg f^{\sigma}_{\text{bc,obst\_det}}$
$\neg f^{\sigma}_{\text{bc,obst\_det}}$
$f^{\sigma}_{\text{bc,obst\_det}}$
$f^{\sigma}_{\text{bc,obst\_det}}$
$S^{\text{avoid}}_{\text{bc}}$
$\neg f^{\sigma}_{\text{bc,ball\_det}}$
$f^{\sigma}_{\text{bc,ball\_det}}$
${}^{c}\mathcal{B}_{\text{bc,avoid}}$

Fig. 6 Activities of the control subsystem represented by an FSM ${}^{c}\mathcal{F}_{\text{bc}}$

The behaviour stops iterating when the terminal condition ${}^{c}f^{\tau}_{\mathrm{bc,collect}}$ is fulfilled:

$${}^{c}f^{\tau}_{\mathrm{bc,collect}} \triangleq \neg^{r}_{x}c_{\mathrm{bc,cam}}[\mathrm{ball\_det}] \vee^{r}_{x} c_{\mathrm{bc,sonar}}[\mathrm{obst\_det}]$$

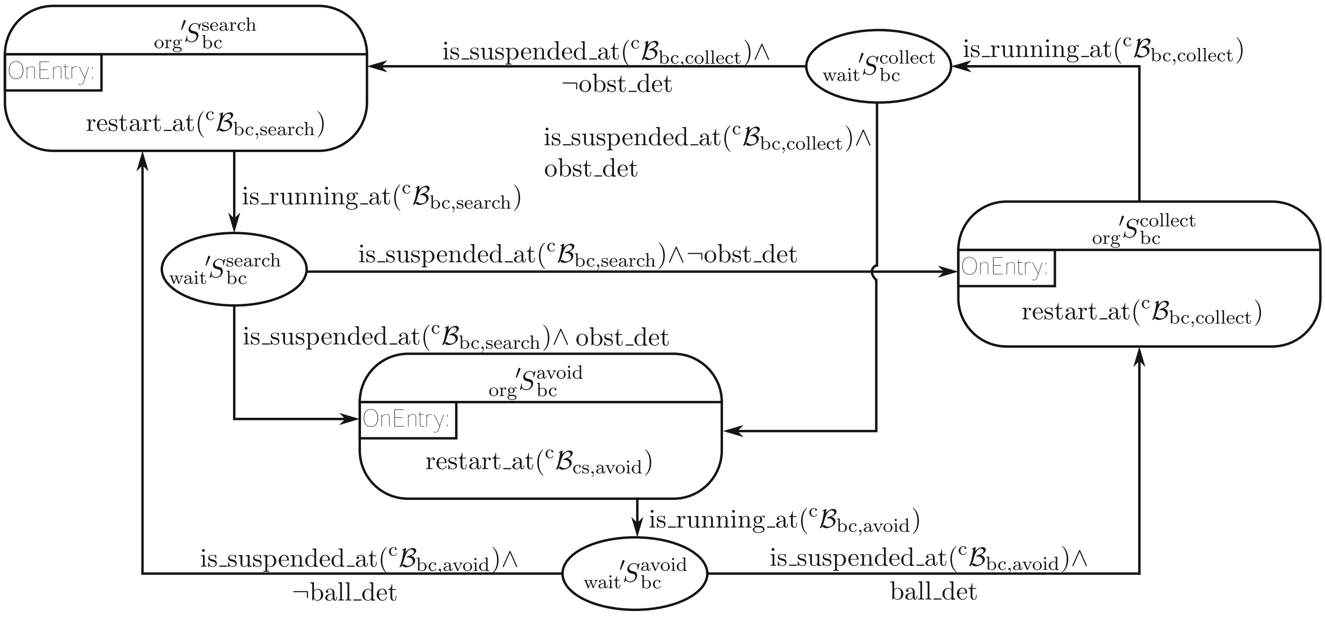
Other two behaviours are specified similarly. Behaviour ${}^{c}\mathcal{B}_{\mathrm{bc,search}}$ is responsible for moving the robot in search for the balls. It terminates when either a ball or an obstacle is detected. Whenever an obstacle is detected behaviour ${}^{c}\mathcal{B}_{\mathrm{bc,avoid}}$ is invoked. It uses the bug algorithm [59] for obstacle avoidance.

All other subsystems of a_{bc}, i.e. virtual receptors and virtual effectors, are governed by single state FSMs. The associated behaviours execute in their endless loops the subFSM template presented in Fig. 2 (thus the terminal conditions of those behaviours are always false). The virtual receptor $r_{\mathrm{bc,cam}}$ executes the behaviour ${}^{r}\mathcal{B}_{\mathrm{bc,cam,detect}}$. The transition function of this behaviour uses the RGB image loaded into the buffer ${}^{R}_{x}r_{\mathrm{bc,cam}}$ connected to $R_{\mathrm{bc,cam}}$, transforms this image into the HSV space, cheques whether pixels are in the appropriate colour range, erodes the image, dilates it, finds contours, cheques the shape of the detected contours, assuming that all circular contours represent balls, so the one with the largest circumscribed area is used to calculate the estimated ball position within the image space. Thus the transition function is defined as a composition of the above functions, as a result producing a pure function (no side effects are involved). This is an intrinsic property of all transition functions. Virtual receptor $r_{\mathrm{bc,sonar}}$ executes ${}^{r}\mathcal{B}_{\mathrm{bc,sonar,detect}}$. Its transition function calculates the distance from the obstacle, based on the measured time of flight received from $R_{\mathrm{bc,sonar}}$. The virtual effectors: $e_{\mathrm{bc,body}}$ and $e_{\mathrm{bc,vacuum}}$ execute: ${}^{e}\mathcal{B}_{\mathrm{bc,body,move}}$ and ${}^{e}\mathcal{B}_{\mathrm{bc,vacuum,set}}$, respectively. ${}^{e}\mathcal{B}_{\mathrm{bc,body,move}}$ using its transition function transforms the commands received from c_{bc} into PWM values and sends them to $E_{\mathrm{bc,body}}$. ${}^{e}\mathcal{B}_{\mathrm{bc,vacuum,set}}$ transforms the commands obtained from c_{bc} to an appropriate PWM value and dispatches it to $E_{\mathrm{bc,vacuum}}$. Only two states of the vacuum cleaner are considered: turned on or off, thus PWM is either of maximum value or zero.

The intra-system communication between subsystems of a_{bc} is realised as one-to-one communication. The communication between c_{bc} and its virtual subsystems uses cyclic buffers. Both producers and consumers use non-blocking mode of operation, thus subsystems must react appropriately to lack of new data.

7.2 Implementation

The above defined subsystems are implemented using the MiCASE modelling tool [28]. It enables the high-level specification of the activities of the software-implemented subsystem (i.e. control subsystem and virtual subsystems) using hierarchical LLFSMs that constitute an executable model. To this end, a slight modification of the original FSM ${}^{s}\mathcal{F}_{\mathrm{bc}}$ is required [35]. Each ${}^{s}\mathcal{F}_{\mathrm{bc}}$ is implemented as ${}^{s}\mathcal{F}'_{\mathrm{bc}}$. The transformation of ${}^{c}\mathcal{F}_{\mathrm{bc}}$, presented in Fig. 6, into its implementation version ${}^{c}\mathcal{F}'_{\mathrm{bc}}$, presented in Fig. 7, is straightforward. Each state of ${}^{c}\mathcal{F}_{\mathrm{bc}}$, denoted as S^{ω}_{bc} (ω is the designator of the behaviour associated with the considered state of ${}^{c}\mathcal{F}_{\mathrm{bc}}$), is implemented as a pair of consecutive states of ${}^{c}\mathcal{F}'_{\mathrm{bc}}$: the original state ${}_{\mathrm{org}'}S^{\omega}_{\mathrm{bc}}$ (depicted as a rectangle



${}_{\mathrm{org}}{}'S^{\mathrm{search}}_{\mathrm{bc}}$
OnEntry:
restart_at(${}^{c}\mathcal{B}_{\mathrm{bc,search}}$)
is_suspended_at(${}^{c}\mathcal{B}_{\mathrm{bc,collect}}$)∧ ¬obst_det
${}_{\mathrm{wait}}{}'S^{\mathrm{collect}}_{\mathrm{bc}}$
is_running_at(${}^{c}\mathcal{B}_{\mathrm{bc,collect}}$)
is_suspended_at(${}^{c}\mathcal{B}_{\mathrm{bc,collect}}$)∧ obst_det
is_running_at(${}^{c}\mathcal{B}_{\mathrm{bc,search}}$)
${}_{\mathrm{org}}{}'S^{\mathrm{collect}}_{\mathrm{bc}}$
OnEntry:
restart_at(${}^{c}\mathcal{B}_{\mathrm{bc,collect}}$)
${}_{\mathrm{wait}}{}'S^{\mathrm{search}}_{\mathrm{bc}}$
is_suspended_at(${}^{c}\mathcal{B}_{\mathrm{bc,search}}$)∧¬obst_det
is_suspended_at(${}^{c}\mathcal{B}_{\mathrm{bc,search}}$)∧ obst_det
${}_{\mathrm{org}}{}'S^{\mathrm{avoid}}_{\mathrm{bc}}$
OnEntry:
restart_at(${}^{c}\mathcal{B}_{\mathrm{cs,avoid}}$)
is_running_at(${}^{c}\mathcal{B}_{\mathrm{bc,avoid}}$)
is_suspended_at(${}^{c}\mathcal{B}_{\mathrm{bc,avoid}}$)∧ ¬ball_det
${}_{\mathrm{wait}}{}'S^{\mathrm{avoid}}_{\mathrm{bc}}$
is_suspended_at(${}^{c}\mathcal{B}_{\mathrm{bc,avoid}}$)∧ ball_det

Fig. 7 Implementation of ${}^{c}\mathcal{F}_{\mathrm{bc}}$ FSM as ${}^{c}\mathcal{F}'_{\mathrm{bc}}$ FSM modelling the activities of the control subsystem

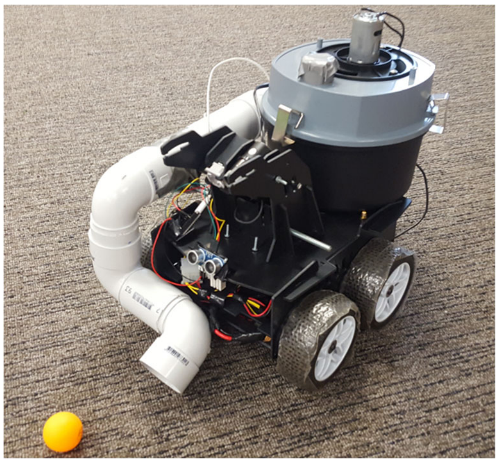


Fig. 8 Table-tennis ball collecting robot

with oval corners) and the supplementary state ${}_{\mathrm{wait}'}S^{\omega}_{\mathrm{bc}}$ (depicted as a small oval). With each state ${}_{\mathrm{org}'}S^{\omega}_{\mathrm{bc}}$ an action is associated, where a new submachine ${}^{\prime}\mathcal{F}^{\mathcal{B}}_{\mathrm{bc},\omega}$ is executed by the restart_at(SUBMACHINE) command. It models the behaviour ${}^{c}\mathcal{B}_{\mathrm{bc},\omega}$. The FSM ${}^{c}\mathcal{F}'_{\mathrm{bc}}$ waits in the current state ${}_{\mathrm{org}'}S^{\omega}_{\mathrm{bc}}$ until the submachine ${}^{\prime}\mathcal{F}^{\mathcal{B}}_{\mathrm{bc},\omega}$ starts executing (i.e. the Boolean expression is_running_at(SUBMACHINE) returns true). When the submachine ${}^{\prime}\mathcal{F}^{\mathcal{B}}_{\mathrm{bc},\omega}$ starts executing, the FSM ${}^{c}\mathcal{F}'_{\mathrm{bc}}$ changes its state to the state ${}_{\mathrm{wait}'}S^{\omega}_{\mathrm{bc}}$. The FSM ${}^{c}\mathcal{F}'_{\mathrm{bc}}$ being in the state ${}_{\mathrm{wait}'}S^{\omega}_{\mathrm{bc}}$ waits for the termination of activities of the subFSM ${}^{\prime}\mathcal{F}^{\mathcal{B}}_{\mathrm{bc},\omega}$ by testing the Boolean expression is_suspended_at(SUBMACHINE) (it returns true, if the submachine has finished its activities).

The ball collecting robot system requires five FSMs, because it contains five software subsystems. Additionally, for each behaviour within each subsystem an additional subFSMs must be produced. Thus in total there are 12 FSMs/subFSMs. Each of them is governed by the same, single sequential scheduler, switching their activities. The communication between subsystems is implemented using the whiteboard [30].

7.3 Robot Hardware

The table-tennis ball collecting robot (Fig. 8) has four motors actuating the four wheels, an obstacle detecting sonar HC-SR04 with a 2–400 cm range, a ball detecting camera (Microsoft LifeCam Studio Webcam), a 12 V car vacuum cleaner producing suction, a ball container, a Raspberry Pi 3B on-board computer, Arduino Micro microcontroller board and a Xaxxon MALG PCB motor controller. The Pi 3B executes the software of the control subsystem and all virtual subsystems, sending commands to the real effectors: $E_{\mathrm{bc,body}}$ (to the motors through the Xaxxon MALG motor controller), vacuum cleaner $E_{\mathrm{bc,vacuum}}$ (Arduino Mirco DC motor velocity controller MCU-60127) and receives data from the real receptors: camera $R_{\mathrm{bc,cam}}$ (Microsoft LifeCam Studio Webcam camera), sonar $R_{\mathrm{bc,sonar}}$ (Arduino Micro and HC-SR04 sonar). The Raspberry Pi 3B is connected to both the Arduino Micro and Xaxxon MALG PCB using USB serial port and to the camera by a SCCB interface (serial camera control bus).

7.4 Performance of the Communication within the Robot

If a real-time operating system is used, the execution times of particular behaviour steps are as presented in Table 1. The produced system requires both intra-system communication (i.e. between c_{bc} and virtual subsystems) and access to the hardware (i.e. between virtual subsystems and the real subsystems). Communication with the hardware is organised as either: send operations of behaviours ${}^{e}\mathcal{B}_{\mathrm{bc,body,move}}$ and ${}^{e}\mathcal{B}_{\mathrm{bc,vacuum,set}}$, or receive operations of behaviours ${}^{r}\mathcal{B}_{\mathrm{bc,cam,detect}}$ and ${}^{r}\mathcal{B}_{\mathrm{bc,sonar,detect}}$. Hardware access uses standard driver interfaces whilst intra-system communication utilises the whiteboard. The former communication uses USB serial port which introduces significant delays, hence those send/receive operations may take even several milliseconds. In the latter case, the

Table 1 Maximum measured time [μs] required by behaviour steps: calculation of transition function ${}^{s}f_{j,v}$, sending and receiving data, where vac is vacuum and wb is the whiteboard

Behaviour	${}^{s}f_{j,v}$	Receive		Send	
${}^{c}\mathcal{B}_{\mathrm{bc,search}}$	3	wb$\rightarrow c_{\mathrm{bc}}$	16	$c_{\mathrm{bc}} \rightarrow$ wb	18
${}^{c}\mathcal{B}_{\mathrm{bc,collect}}$	3	wb $\rightarrow c_{\mathrm{bc}}$	10	$c_{\mathrm{bc}} \rightarrow$ wb	29
${}^{c}\mathcal{B}_{\mathrm{bc,avoid}}$	9	wb $\rightarrow c_{\mathrm{bc}}$	21	$c_{\mathrm{bc}} \rightarrow$ wb	14
${}^{r}\mathcal{B}_{\mathrm{bc,cam,detect}}$	20345	$R_{\mathrm{bc,cam}} \rightarrow r_{\mathrm{bc,cam}}$	4720	$r_{\mathrm{bc,cam}} \rightarrow$ wb	21
${}^{r}\mathcal{B}_{\mathrm{bc,sonar,detect}}$	2	$R_{\mathrm{bc,sonar}} \rightarrow r_{\mathrm{bc,sonar}}$	1501	$r_{\mathrm{bc,sonar}} \rightarrow$ wb	9
${}^{e}\mathcal{B}_{\mathrm{bc,body,move}}$	2	wb $\rightarrow e_{\mathrm{bc,body}}$	10	$e_{\mathrm{bc,body}} \rightarrow E_{\mathrm{bc,body}}$	5093
${}^{e}\mathcal{B}_{\mathrm{bc,vacuum,set}}$	3	wb $\rightarrow e_{\mathrm{bc,wb}}$	9	${}^{e}_{\mathrm{bc,vac}} \rightarrow E_{\mathrm{bc,wb}}$	1384

messages are transferred using shared memory, thus the time required for execution a single send/receive operation is at the most 39μs (10μs+29μs).

The hardware out of which the simple robot was built to present both the design method and the software used for the implementation of the controller was inexpensive. The camera enables image acquisition at the rate of 30 Hz. Thus the virtual receptor $r_{\mathrm{bc,cam}}$ collects data every 34 ms. The balls are detected by the camera at a maximum distance of 2.5 m. The sonar enables the detection of obstacles in the range 2-400 cm, thus due to the velocity of sound in air, a data sample is obtained every 60 ms. As the maximum velocity of the robot is $0.25\frac{\mathrm{m}}{\mathrm{s}}$, the robot is able to stop and change the direction of motion well ahead of possible collisions. The decisions as to the selection of the nearest ball to collect are also made well in advance, so the robot does not have to veer overly rapidly. The balls scattered between obstacles were collected efficiently.

8 Conclusions

This paper extends the robotic system design methodology (presented in e.g. [35, 92, 96]) by inter-subsystem communication model [31]. It assumes that the activities of each subsystem and each of their behaviours is represented by a LLFSMs. The communication between the subsystems takes place in specific states of the resulting hierarchic FSM, being the composition of the higher level FSM defining the task of the subsystem and the lower level ones governing the execution of this subsystem's particular behaviours. The subsystems in those states access the whiteboard, providing or acquiring the necessary data. As the scheduler switches the hierarchic FSMs sequentially no contentions arise. To refer the proposed communication model to the state of the art, a classification of communication methods from the robotics perspective has been introduced.

The choice of LLFSMs and the whiteboard as an implementation means for the proposed specification method was dictated by the fact that the specification method utilises the concept of FSM and transition function embedded in a behaviour, which organises the communication, i.e. the concepts into which it was easy to translate the specification. Importantly, the whiteboard ensures reliable access to data. The whiteboard gives precise communication semantics in terms of bounded timing as well as atomicity. Moreover always the latest readings are delivered and no buffer overflow occurs due to the use of a cyclic buffer. If only one scheduler is employed the LLFSMs ensure reliable sequential execution of the behaviours of the subsystems. Multi-scheduler implementation on a multi-processor system is also possible, but then contention resolution has to be used. However for simple one-computer implementation of the system this is not necessary. The presented implementation approach ensures that at a given time only a single subsystem is active so there is no contention regarding data access.

An embodied agent is a set of subsystems communicating with the control subsystem utilising the whiteboard as a communication means. The constraint imposed on the internal structure of an embodied agent, which requires that all virtual entities contact only the control subsystem, limits the internal number of communication links. Each subsystem of an embodied agent is implemented as a separate process. An additional agent added to the system linearly increases the number of utilised processes. The number of new connections required within the system is equal to the number of the subsystems within the embodied agent and usually low number of interconnections of the added agent with the already existing agents, thus is scalable.

Embodied agents can be implemented not only by using LLFSMs and the whiteboard, but also by utilising OROCOS components, however the transformation procedure is not that straightforward. OROCOS is a component based robotics framework, where each component encapsulates an executable algorithm (primary code). Within the component there exists an Execution Engine, which processes asynchronous operations, calls plug-in functions and executes the primary code, when the component is in a running state. The FSM is related to the states of the component (pre-operational, stopped, running, exception etc.) not the states of the executed tasks. Thus, the FSM within the component is completely different from the FSM in the embodied agent. The former changes the state of the component, and only the running state is of relevance to us, as only in that state the task is executed. In the latter case, the FSM is related to the states of the task execution. With every one of those states a transition function parameterised behaviour is associated. Thus, if a subsystem would have to be implemented as an OROCOS component both the FSM and the behaviours would have to be recreated within the primary code. Hence no benefit is obtained. Our multi-layer structure is being flattened within this code. The other method of using OROCOS is to associate a component with a behaviour. Then the primary code would represent a transition function. However then the subsystem FSM would have to be implemented by an extra component marshalling the other components (i.e. it would play the role of a scheduler). The activity of the agent would have to be produced by still another component executing the FSM of the control subsystem. However in such a system, not only the communication between subsystems would have to be organised, but also between behaviours (behaviours of subsystems of embodied agents have access to all the input, output and internal data of that subsystem). Thus, data transfers between behaviours would have to be arranged through ports, which was not the intention of the model employed by embodied agents.

Both the utility of the design methodology and the employed communication method have been presented on a rudimentary robot searching for table-tennis balls. The system was implemented by using the MiCASE tool [28]. It enables both modelling of the activities of the designed system and automatic code generation, resulting in an executable, high-level behaviour model. The resulting robot control system functions as specified.

Acknowledgments This work was supported by the Erasmus Mundus Action 2 PANTHER (Pacific Atlantic Network for Technical Higher Education and Research) grant and by the National Science Centre, Poland (grant number 2017/25/N/ST7/00900).

References

1. Alami, R., Chatila, R., Fleury, S., Ghallab, M., Ingrand, F.: An architecture for autonomy. Int. J. Robot. Res. **17**(4), 315–337 (1998)
2. Arbib, M.: Handbook of Physiology – The Nervous System II. Motor Control, chap. Perceptual structures and distributed motor control, pp. 1449–1480. Wiley Online Library (1981)
3. Arkin, R.C.: Motor schema-based mobile robot navigation. Int. J. Robot. Res. **8**, 92–112 (1989)
4. Arkin, R.C.: Cooperation without communication: Multiagent schema-based robot navigation. J. Robot. Syst. **9**(3), 351–364 (1992)
5. Arkin, R.C.: Behavior-Based Robotics. MIT press, Cambdrige (1998)
6. Armbrust, C., Kiekbusch, L., Ropertz, T., Berns, K.: Soft Robot Control with a Behaviour-Based Architecture. In: Soft Robotics, pp. 81–91. Springer (2015)
7. Asama, H., Ozaki, K., Matsumoto, A., Endo, I.: Development of task assignment system using communication for multiple autonomous robots. J. Robot. Mechatron. **4**(2), 122–127 (1992)
8. Balch, T., Arkin, R.C.: Communication in reactive multiagent robotic systems. Auton. Robot. **1**(1), 27–52 (1994)
9. Beetz, M., Mösenlechner, L., Tenorth, M.: CRAM – a Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, October 18–22, 2010, Taipei, Taiwan, pp. 1012–1017. IEEE (2010)
10. Bensalem, S., de Silva, L., Ingrand, F.R.Y.: A verifiable and correct-by-construction controller for robot functional levels. J. Softw. Eng. Robot. **2**(1), 1–19 (2011)
11. Birman, K., Schiper, A., Stephenson, P.: Lightweight causal and atomic group multicast. ACM Trans. Comput. Syst. **9**(3), 272–314 (1991)
12. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm intelligence: From natural to artificial systems. Oxford university press, New York (1999)
13. Brooks, R.A.: A robust layered control system for a mobile robot. IEEE J. Robot. Autom. **2**(1), 14–23 (1986)
14. Brooks, R.A.: Intelligence without reason. Artif. Intell. Crit. Concepts **3**, 107–163 (1991)
15. Brooks, R.A.: Intelligence without representation. Artif. Intell. **47**(1-3), 139–159 (1991)
16. Brugali, D.: Software Engineering for Experimental Robotics. In: Brugali, D. (ed.): Stable Analysis Patterns for Robot Mobility. Software Engineering for Experimental Robotics, pp. 9–30. Springer Berlin Heidelberg, Berlin (2007). https://doi.org/10.1007/978-3-540-68951-5_2
17. Brugali, D., Agah, A., MacDonald, B., Nesnas, I.A.D., Smart, W.D.: Software Engineering for Experimental Robotics. In: Brugali, D. (ed.): Trends in robot software domain engineering. Software Engineering for Experimental Robotics, pp. 3–8. Springer Berlin Heidelberg, Berlin (2007). https://doi.org/10.1007/978-3-540-68951-5_1
18. Cabrera-Gámez, J., Domínguez-Brito, A.C., Hernández-Sosa, D.: Sensor based intelligent robots. chap. CoolBOT: A Component-Oriented Programming Framework for Robotics, pp. 282–304. Springer Berlin / Heidelberg (2002)
19. Chen, D., Hexel, R., Raja, F.: Engineering Real-Time Communication through Time-Triggered Subsumption: Towards Flexibility with Incus and Llfsms. In: Maciaszek, L., Filipe, J. (eds.) 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering, pp. 272 – 281, Rome (2016). https://doi.org/10.5220/0005915602720281
20. Chen, J., Gauci, M., Li, W., Kolling, A., Groß, R.: Occlusion-based cooperative transport with a swarm of miniature mobile robots. IEEE Trans. Robot. **31**(2), 307–321 (2015). https://doi.org/10.1109/TRO.2015.2400731
21. Cheriton, D.R., Skeen, D.: Understanding the limitations of causally and totally ordered communication. SIGOPS Oper. Syst. Rev. **27**(5), 44–57 (1993). https://doi.org/10.1145/173668.168623
22. Corkill, D.: Blackboard systems. AI Expert. **6**(9), 40–47 (1991)
23. Coste-Maniere, E., Simmons, R.: Architecture, the Backbone of Robotic Systems. In: 2000. Proceedings. ICRA '00. IEEE International Conference On Robotics and Automation, vol. 1, pp. 67–72 (2000). https://doi.org/10.1109/ROBOT.2000.844041
24. Dudek, G., Jenkin, M., Milios, E., Wilkes, D.: A taxonomy for swarm robots. In: IROS '93. Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems '93, vol. 1, pp. 441–447 (1993). https://doi.org/10.1109/IROS.1993.583135
25. Dudek, G., Jenkin, M., Milios, E., Wilkes, D.: Experiments in Sensing and Communication for Robot Convoy Navigation. In: Proceedings. 1995 IEEE/RSJ International Conference On Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots, vol. 2, pp. 268–273 (1995). https://doi.org/10.1109/IROS.1995.526171
26. Dudek, G., Jenkin, M.R.M., Milios, E., Wilkes, D.: A taxonomy for multi-agent robotics. Auton. Robot. **3**(4), 375–397 (1996). https://doi.org/10.1007/BF00240651
27. Einhorn, E., Langner, T., Stricker, R., Martin, C., Gross, H.M.: MIRA - Middleware for robotic applications. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, pp. 2591–2598 (2012). https://doi.org/10.1109/IROS.2012.6385959. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6385959&isnumber=6385431
28. Estivill-Castro, V., Hexel, R.: Arrangements of finite-state machines - semantics, simulation, and model checking. In: Proceedings of the 1st International Conference on

Model-Driven Engineering and Software Development - Volume 1: MODELSWARD, pp. 182–189 (2013). https://doi.org/10.5220/0004317101820189
29. Estivill-Castro, V., Hexel, R.: Correctness by Construction with Logic-Labeled Finite-State Machines – Comparison with Event-B. In: Zhu, L., Steel, J. (eds.) Proceedings of 23Rd Australasian Software Engineering Conference (ASWEC), pp. 38–47. IEEE Computer Society Conference Publishing Services (CPS), Milsons Point (2014)
30. Estivill-Castro, V., Hexel, R.: Simple, Not Simplistic — the Middleware of Behaviour Models. In: Filipe, J., Maciaszek, L.A. (eds.) ENASE 10 International Conference on Evaluation of Novel Approaches to Software Engineering, pp. 189–196. INSTCC, SciTePress, Av. Dom Manuel i, 2910-592, Setúbal (2015)
31. Estivill-Castro, V., Hexel, R., Lusty, C.: High Performance Relaying of C++11 Objects across Processes and Logic-Labeled Finite-State Machines. In: Brugali, D., Broenink, J.F., Kroeger, T., MacDonald, B.A. (eds.) Simulation, Modeling, and Programming for Autonomous Robots - 4th International Conference, SIMPAR 2014, Lecture Notes in Computer Science, vol. 8810, pp. 182–194. Springer (2014)
32. Estivill-Castro, V., Hexel, R., Rosenblueth, D.A.Wang, P. (ed.): Efficient Model Checking and FMEA Analysis with Deterministic Scheduling of Transition-Labeled Finite-State Machines. IEEE Computer Society Conference Publishing Services (CPS), Wuhan (2012)
33. Estivill-Castro, V., Hexel, R., Rosenblueth, D.A.: Efficient Modelling of Embedded Software Systems and Their Formal Verification. In: Leung, K.R., Muenchaisri, P. (eds.) The 19Th Asia-Pacific Software Engineering Conference (APSEC 2012), pp. 428–433. IEEE Computer Society, Conference Publishing Services, Hong Kong (2012)
34. Farinelli, A., Iocchi, L., Nardi, D.: Multirobot systems: a classification focused on coordination. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **34**(5), 2015–2028 (2004). https://doi.org/10.1109/TSMCB.2004.832155
35. Figat, M., Zieliński, C., Hexel, R.: Fsm Based Specification of Robot Control System Activities. In: 2017 11Th International Workshop on Robot Motion and Control (Romoco), pp. 193–198. https://doi.org/10.1109/RoMoCo.2017.8003912 (2017)
36. Fleury, S., Herrb, M., Chatila, R.: G^{en}om: A tool for the specification and the implementation of operating modules in a distributed robot architecture. Proc. 1997 IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS'97) **2**, 842–849 (1997). https://doi.org/10.1109/IROS.1997.655108
37. Fox, C., Evans, M., Pearson, M., Prescott, T.: Towards hierarchical blackboard mapping on a whiskered robot. Robot. Auton. Syst. **60**(11), 1356–1366 (2012). https://doi.org/10.1016/j.robot.2012.03.005. Towards Autonomous Robotic Systems 2011
38. Gat, E.: On Three-Layer Architectures. In: Kortenkamp, D., Bonnasso, R.P., Murphy, R. (eds.) Artificial Intelligence and Mobile Robots, pp. 195–210. AAAI Press, Cambridge (1998)
39. Giffin, S.d.: A taxonomy of Internet applications for project management communication. Proj. Manag. J. **33**(4), 39–47 (2002)
40. Group, O.M.: Data distribution service (dds). https://www.omg.org/spec/DDS/1.4
41. Hayes-Roth, B.: A blackboard architecture for control. Artif. Intell. **26**(3), 251–321 (1985)
42. Hayes-Roth, B.: A blackboard architecture for control. In: Bond, A.H., Gasser, L. (eds.) Distributed Artificial Intelligence, pp. 505–540. Morgan Kaufmann Publishers Inc., San Francisco (1988). http://dl.acm.org/citation.cfm?id=60204.60241
43. Hayes-Roth, B., Hayes-Roth, F.: A cognitive model of planning. Cogn. Sci. **3**(4), 275–310 (1979)
44. Haykin, S. Communication Systems, 5th edn. Wiley Publishing, Hoboken (2009)
45. Huntsberger, T.L., Trebi-Ollennu, A., Aghazarian, H., Schenker, P.S., Pirjanian, P., Nayar, H.D.: Distributed control of multi-robot systems engaged in tightly coupled tasks. Auton. Robot. **17**(1), 79–92 (2004). https://doi.org/10.1023/B:AURO.0000032939.08597.62
46. Jennings, N.R., Sycara, K., Wooldridge, M.: A roadmap of agent research and development. Auton. Agents Multi-Agent Syst. **1**(1), 7–38 (1998). https://doi.org/10.1023/A:1010090405266
47. Joukoff, D., Estivill-Castro, V., Hexel, R., Lusty, C.: Fast MAV Control by Control/Status OO-Messages on Shared-Memory Middleware. In: Robot Intelligence Technology and Applications 4 - Results from the 4Th International Conference on Robot Intelligence Technology and Applications, RiTA 2015, Advances in Intelligent Systems and Computing, Vol. 345, pp. 195–211. Springer (2015)
48. Jung, D., Zelinsky, A.: Grounded symbolic communication between heterogeneous cooperating robots. Auton. Robot. **8**(3), 269–292 (2000). https://doi.org/10.1023/A:1008929609573
49. Kiekbusch, L., Armbrust, C., Berns, K.: Formal verification of behaviour networks including sensor failures. Robot. Auton. Syst. **74**, 331–339 (2015)
50. Kiekbusch, L., Armbrust, C., Berns, K.: Formal Verification of Behaviour Networks including Hardware Failures. In: Intelligent Autonomous Systems 13, pp. 1571–1582. Springer (2016)
51. Kopetz, H.: Should responsive systems be event-triggered or time-triggered? IEICE Trans. Inf. Syst. **76**(11), 1325 (1993)
52. Kopetz, H. Real-Time Systems - Design Principles for Distributed Embedded Applications, 2nd edn. Real-Time Systems Series. Springer, Berlin (2011)
53. Kopetz, H., Bauer, G.: The time-triggered architecture. Proc. IEEE **91**(1), 112–126 (2003)
54. Kornuta, T., Zieliński, C.: Robot control system design exemplified by multi-camera visual servoing. J. Intell. Robotic Syst. **77**(3–4), 499–524 (2013). https://doi.org/10.1007/s10846-013-9883-x
55. Kortenkamp, D., Simmons, R.: Robotic Systems Architectures and Programming. In: Khatib, O., Siciliano, B. (eds.) Springer Handbook of Robotics, pp. 187–206. Springer (2008)
56. Kortenkamp, D., Simmons, R., Brugali, D.: Robotic Systems Architectures and Programming. In: Siciliano, B., Khatib, O. (eds.) Springer Handbook of Robotics. 2nd edn., pp. 283–306. Springer (2016)
57. Lamport, L.: Using time instead of timeout for fault-tolerant distributed systems. ACM Trans. Progr. Lang. Syst. **6**, 254–280 (1984)
58. Lee, C., Hwang, J., Lee, J., Ahn, C., Suh, B., Shin, D.H., Nah, Y., Kim, D.H.: Self-Describing and Data Propagation Model for Data Distribution Service. In: Brinkschulte, U., Givargis, T., Russo, S. (eds.) Software Technologies for Embedded and Ubiquitous Systems, pp. 102–113. Springer, Berlin (2008)
59. Lumelsky, V.J., Stepanov, A.A.: Path-planning strategies for a point mobile automaton moving amidst obstacles of arbitrary shape. Algorithmica **2**, 403–430 (1987)
60. Lumia, R., Fiala, J., Wavering, A.: The nasrem robot control system standard. Robot. Comput.-Integr. Manuf. **6**(4), 303 – 308 (1989). https://doi.org/10.1016/0736-5845(89)90120-8. Special Issue Robots in Manufacturing
61. Lusty, C.: Cooperative Time-Triggered Wireless Communication in Mobile Robotics. Honours thesis, Griffith University (2012)
62. Lyons, D.M.: Prerational intelligence, Studies in cognitive systems, vol. 2: Adaptive behavior and intelligent systems without symbols and logic, chap. A Schema-Theory Approach to Specifying and Analysing the Behavior of Robotic Systems, pp. 51–70 Kluwer Academic (2001)

63. Lyons, D.M., Arbib, M.A.: A formal model of computation for sensory-based robotics. IEEE Trans. Robot. Autom. **5**(3), 280–293 (1989). https://doi.org/10.1109/70.34764
64. Majcher, P.: Autonomous Mobile Robot for Collecting Table Tenis Balls (Polish). Master's thesis, Warsaw University of Technology (2012)
65. Matarić, M.J.: Issues and approaches in the design of collective autonomous agents. Robot. Auton. Syst. **16**(2), 321 – 331 (1995). https://doi.org/10.1016/0921-8890(95)00053-4
66. Matarić, M., Nilsson, M., Simsarian, K.: Cooperative Multi-Robot Box-Pushing. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 556–561 (1995)
67. MiPal: Downloads (checked: 23 March 2018). http://mipal.net.au/downloads.php
68. Nesnas, I.A.D.: Software Engineering for Experimental Robotics. In: Brugali, D. (ed.): The CLARAty project: Coping with hardware and software heterogeneity, pp. 31–70. Springer Berlin Heidelberg, Berlin (2007). https://doi.org/10.1007/978-3-540-68951-5_3
69. Newell, A.: Some Problems of Basic Organization in Problem-Solving Programs. In: Yovitz, M., Jacobi, G., Goldstein, G. (eds.) Self-Organizing Systems, pp. 393–423. Spartan, Washington (1962)
70. Nii, H.P.: Blackboard systems, part two: Blackboard application systems, blackboard systems from a knowledge engineering perspective. AI Mag. **7**, 82–106 (1986)
71. Nilsson, N.: Shakey the robot, technical note 323. Technical report, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA, pp. 94025 (1984)
72. Occello, M., Chaouiya, C., Thomas, M.C.: A Parallel Blackboard Model for Mobile Robotics control, pp. 439–447. Springer Netherlands, Dordrecht (1992). https://doi.org/10.1007/978-94-011-2526-0_51
73. Pang, G.K., Shen, H.C.: Intelligent control of an autonomous mobile robot in a hazardous material spill accident — a blackboard structure approach. Robot. Auton. Syst. **6**(4), 351–365 (1990)
74. Parker, L.: ALLIANCE: An architecture for fault tolerant multirobot cooperation. IEEE Trans. Robot. Autom. **14**(2), 220–240 (1998)
75. Parker, L., Rus, D., Sukhatme, S.G.: Multiple Mobile Robot Systems. In: Khatib, O., Siciliano, B. (eds.) Springer Handbook of Robotics, pp. 1335–1384. Springer (2016)
76. Parker, L.E.: Alliance: an architecture for fault tolerant, cooperative control of heterogeneous mobile robots. In: IROS '94. Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems '94. Advanced Robotic Systems and the Real World, vol. 2, pp. 776–783 (1994)
77. Postel, J.: Transmission Control Protocol, RFC 793. Technical report, University of Southern California, Information Sciences Institute. https://tools.ietf.org/html/rfc793 (1981)
78. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an open-source Robot Operating System. In: Proceedings of the Open-Source Software workshop at the International Conference on Robotics and Automation (ICRA) (2009)
79. ROS: Parameter Server. http://wiki.ros.org/ParameterServer
80. ROS: Technical Overview. http://wiki.ros.org/ROS/TechnicalOverview
81. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science **345**(6198), 795–799 (2014). https://doi.org/10.1126/science.1254295
82. Schwartz, D.G.: Cooperating heterogeneous systems. Springer Science & Business Media, NO (1995)
83. Shannon, C.E., Weaver, W.: A Mathematical Theory of Communication. University of Illinois Press, Baltimore (1963)
84. Sommerville, I. Software Engineering, 9th edn. Addison-Wesley Publishing Company, USA (2010)
85. Song, D., Goldberg, K., Chong, N.Y.: Networked Robots. In: Khatib, O., Siciliano, B. (eds.) Springer Handbook of Robotics, pp. 1109–1131. Springer (2016)
86. Tanenbaum, A.S., Woodhull, A.S. Operating Systems Design and Implementation, 3rd edn. Prentice-Hall, Inc., Upper Saddle River (2005)
87. Trianni, V., Nolfi, S., Dorigo, M.: Cooperative hole avoidance in a swarm-bot. Robot. Auton. Syst. **54**(2), 97–103 (2006). Intelligent Autonomous Systems 8th Conference on Intelligent Autonomous Systems (IAS-8)
88. Ueyama, T., Fukuda, T., Arai, F.: Configuration of communication structure for distributed intelligent robotic system. Proc. - IEEE Int. Conf. Robot. Autom. **1**, 807–812 (1992)
89. Yamada, S., Saito, J.: Adaptive action selection without explicit communication for multirobot box-pushing. IEEE Trans. Syst. Man Cybern. Part C **31**(3), 398–404 (2001)
90. Zieliński, C.: Description of semantics of robot programming languages. Mechatronics **2**(2), 171–198 (1992)
91. Zieliński, C.: A Quasi-Formal Approach to Structuring Multi-Robot System Controllers. In: Second International Workshop on Robot Motion and Control, Romoco'01, pp. 121–128 (2001)
92. Zieliński, C., Figat, M.: Robot System Design Procedure Based on a Formal Specification. In: Recent Advances in Automation, Robotics and Measuring Techniques, Advances in Intelligent Systems and Computing (AISC), vol. 440, pp. 511–522. Springer (2016). https://doi.org/10.1007/978-3-319-29357-8_45
93. Zieliński, C., Kornuta, T.: Diagnostic requirements in multi-robot systems. In: Intelligent Systems in Technical and Medical Diagnostics, vol. 230, pp. 345–356. Springer (2014). https://doi.org/10.1007/978-3-642-39881-0_29
94. Zieliński, C., Kornuta, T., Winiarski, T.: A Systematic Method of Designing Control Systems for Service and Field Robots. In: 19-Th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR, pp. 1–14. IEEE (2014). https://doi.org/10.1109/MMAR.2014.6957317
95. Zieliński, C., Trojanek, P.: Stigmergic cooperation of autonomous robots. J. Mech. Mach. Theory **44**, 656–670 (2009)
96. Zieliński, C., Winiarski, T.: General specification of multi-robot control system structures. Bullet. Pol. Acad. Sci. – Techn. Sci. **58**(1), 15–28 (2010). https://doi.org/10.2478/v10175-010-0002-x
97. Zieliński, C., Winiarski, T.: Motion generation in the MRROC++ robot programming framework. Int. J. Robot. Res. **29**(4), 386–413 (2010). https://doi.org/10.1177/0278364909348761
98. Zieliński, C., Winiarski, T., Kornuta, T.: Agent-based structures of robot systems. In: K. J., et al. (eds.) Trends in Advanced Intelligent Control, Optimization and Automation, Advances in Intelligent Systems and Computing, vol. 577, pp. 493–502 (2017). https://doi.org/10.1007/978-3-319-60699-6_48

Cezary Zieliński M.Sc./Eng. (1982), Ph.D. (1988) and habilitation (1996) all in control and robotics, from Warsaw University of Technology (WUT), full professor of WUT. Specialises in robotics, heads the Robotics Group in 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, multi-robot system controllers, robot kinematics, robot force control, visual servo control, utilisation of sensors in robot control, design of digital circuits. Currently he is the vice-dean for General Affairs of the Faculty of Electronics and Information Technology of WUT.

Maksym Figat M.Sc./Eng. (2013) in computer science (modelling CAD/CAM systems), from Warsaw University of Technology (WUT), Faculty of Mathematics and Information Science. Since 2013, he is Ph.D. student at the Warsaw University of Technology (WUT), Faculty of Electronics and Information Technology (FEIT), Institute of Control and Computation Engineering (ICCE), Warsaw, Poland. He is 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.

René Hexel M.Sc./Eng. (1995), PhD (1999), has been a leading researcher in software engineering of safety-critical real-time systems. His achievements include methodologies for modelling, model-checking, evaluation, and validation of safety-critical real-time systems, including failure mode effects analysis (FMEA) through fault injection. His research interests include Software Engineering of Complex Distributed Systems and autonomous robots as well as robust and reliable safety critical systems. René Hexel is Deputy Head of School of ICT at Griffith University and Co-Director of the Machine Intelligence and Pattern Analysis Lab (MiPal) at Institute for Integrated and Intelligent Systems (IIIS) in Brisbane, Australia.