EARL – Embodied Agent-Based Robot Control Systems Modelling Language - version 1.0 - reference manual

EARL developers group - https://www.robotyka.ia.pw.edu.pl/projects/earl/
tomasz.winiarski@pw.edu.pl

April 4, 2020

Important citation notice

If you are to use EARL, in your papers, please at first cite the Electronics journal article [14], where the initial version of EARL is presented.

1 Introduction

EARL is developed by the robotic team at Warsaw University of Technology, Institute of Control and Computation Engineering. EARL proposes a standardized approach to the control system specification of cyber-physical systems. The Embodied Agent [18] is its foundation. EARL maps the concepts associated with Embodied Agents into SysML blocks with theirs properties, i.e., parts, references, values and operations. It extends the set of best practices, by answering the following questions.

- How to organize a specification into SysML packages?
- For what purposes should the graphical tools be used and where the mathematical notation should be applied directly?
- How to map the specification into component systems?
- How to describe systems with a time-varying structure?

Figure 1 presents the dependencies of EARL packages. The model utilised by EARL is defined in the Model package (Section 2). The system instances that «realize» EARL model constraints are defined in the System Instance package (Section 3). This package «uses» independently defined computational structures from the Calculation Components package and data types from the DataTypes package.

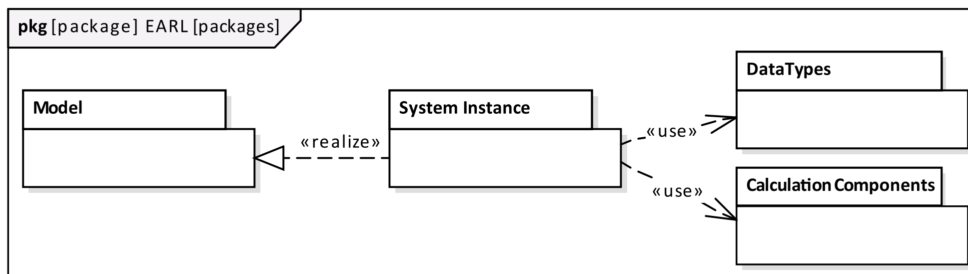


Figure 1: EARL package dependencies.

2 Model Formulation

The model of a system specified in EARL is composed of concepts describing its structure and behaviour. The structure of the model is specified with SysML Block Definition Diagrams (bdd) and

Internal Block Diagrams (ibd) [9]. For clarity of presentation, the various aspects of the structure are presented by separate diagrams. The model is composed of a set of diagrams. Each of the diagrams presents only a part of the structure, however the whole set has to be consistent. Some of the model constraints are defined by mathematical equations.

2.1 System and Its Parts

System is the most general EARL concept. It is structurally defined as in Figure 2. A System must contain at least one Robot r. A Robot is composed of at least one Agent a. A system may contain agents that are not elements of robots, e.g., an Agent coordinating the work of a group of Robots [15]. Agents are connected with aa inter-agent communication Links. Each aa Link can be referred by a Robot. In general, the Links parts names are created by combining the source block part name at the beginning of the Link part name and destination block part name at the end of the Link part name.

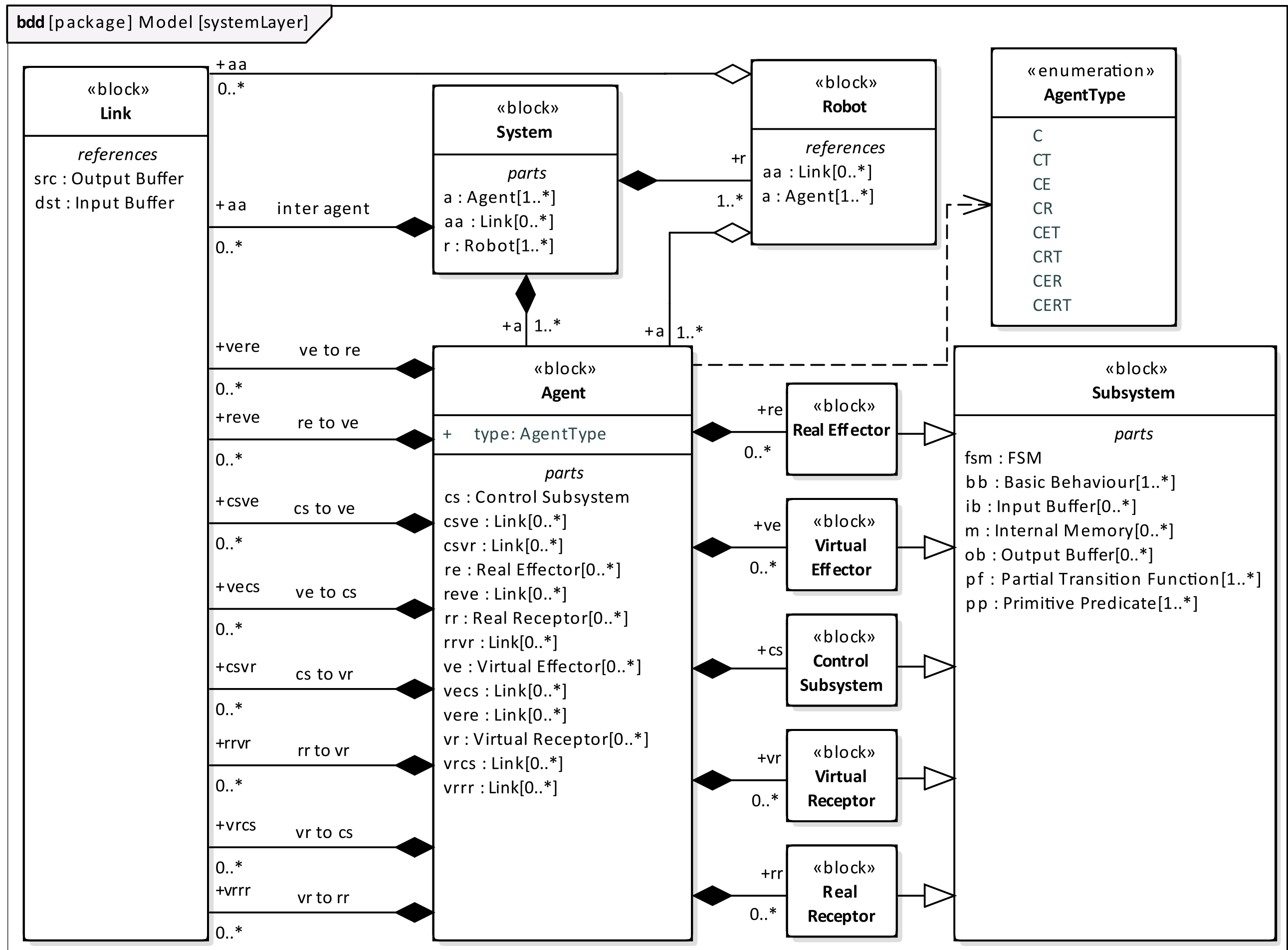


Figure 2: System and its parts.

In cyber-physical systems an Agent usually has a physical body, thus it is an Embodied Agent. It represents either a whole or a part of a robot [16]. The structure of an Agent is defined in Figure 2. The specific features of robotics, where an Agent can take on various roles, from real-time control, through sensor data processing, to execution of computationally demanding tasks [19], require its decomposition into various types of Subsystems and specialized Links between them. The variety of link names was introduced to distinguish the types of Subsystems or Agents that communicate with each other and the direction of data transmission. The blocks cardinality presented in Figure 2 is general, but particular system structure may introduce more strict constraints according to the extra rules presented further.

There are five different specialisations of Subsystems (right side of Figure 2). The main one (in-

dispensable for an Agent) is a Control Subsystem cs, which coordinates the Agent's Subsystems and communicates with other Agents. Real Effectors re are Subsystems which affect the environment, whereas Real Receptors rr (exteroceptors) gather information from the environment. Virtual Subsystems (Virtual Receptors vr and Virtual Effectors ve) supervise the work of Real Subsystems. Therefore, the Real Subsystems of a particular type, cannot exist without virtual ones and vice versa, see Equation (1).

$$|vr| \geqslant 1 \iff |rr| \geqslant 1, \quad |ve| \geqslant 1 \iff |re| \geqslant 1. \tag{1}$$

Inequalities Equation (1) represents the necessary conditions ensuring the preservation of system integrity. Additional constraints have to be imposed on the number of Subsystems due to the specificity of inter-subsystem communication Links (Section 2.3).

2.2 Subsystem and its Parts

The structure of a Subsystem is defined in Figures 3 and 4a. It contains Input Buffers ib and Output Buffers ob, Internal Memory m and other entities that are used to model both structural and behavioural aspects of a Subsystem, i.e., FSM fsm (Finite State Machine), Primitive Predicates pp, Basic Behaviours bb and Partial Transition Functions pf.

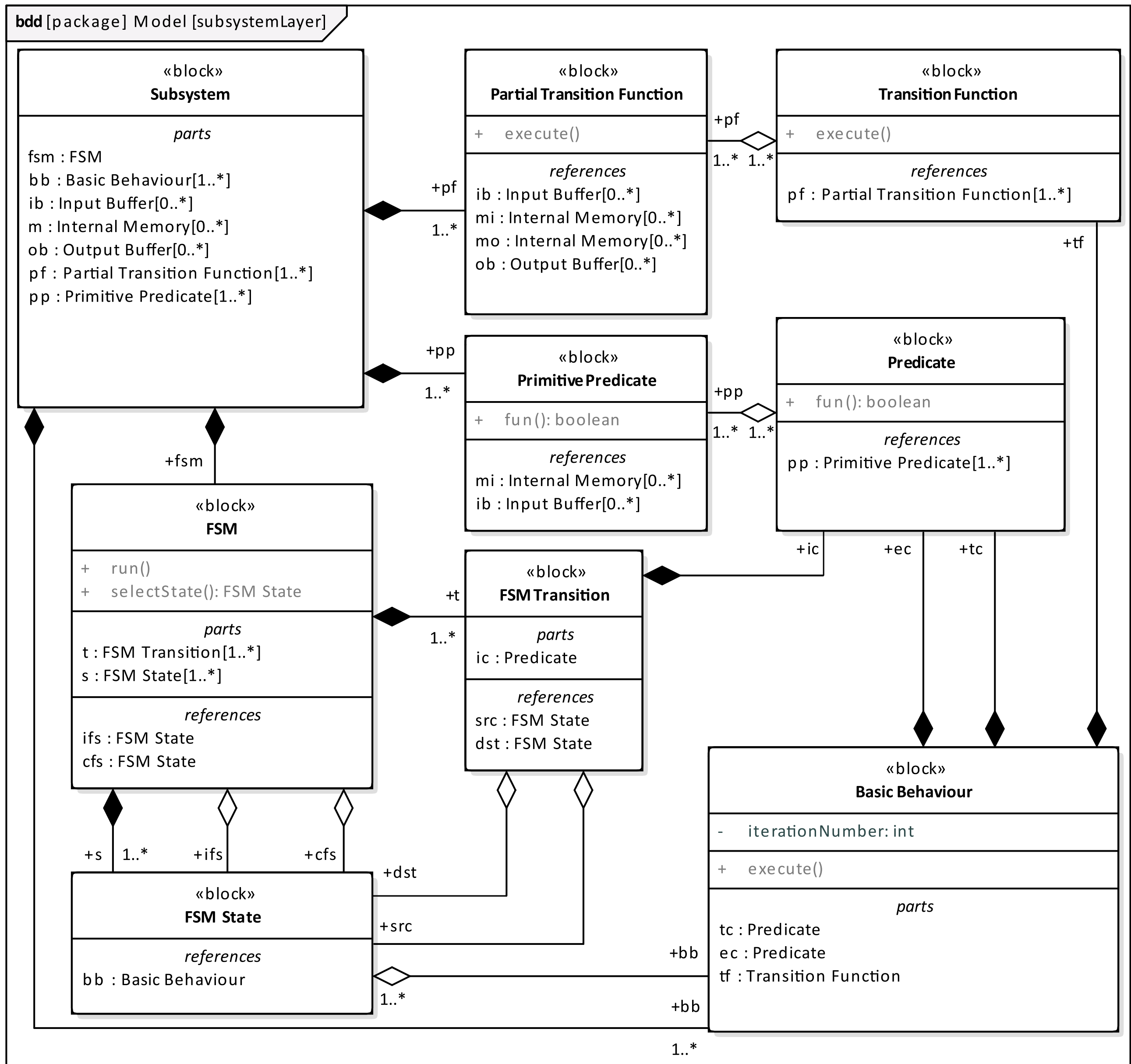
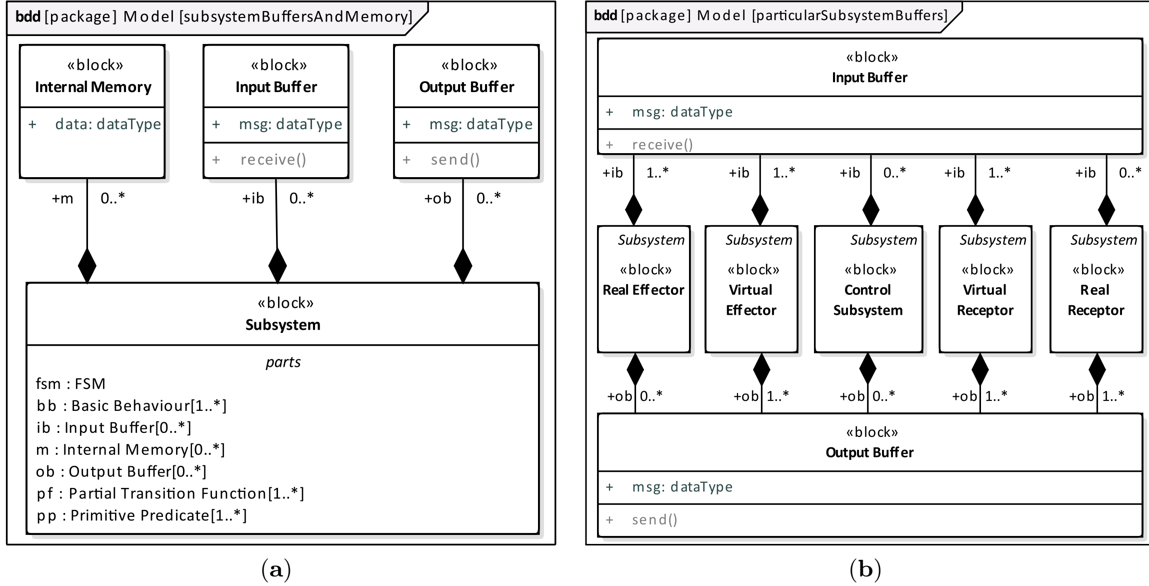


Figure 3: Subsystem and its parts (Input, Output Buffers and Internal Memory are excluded).

Figure 4b depicts relations between a particular Subsystem and its communication buffers. The communication constraints depicted in Section 2.3 cause that each Virtual Receptor or Virtual Effector must have at least one Input Buffer and one Output Buffer. A Real Effector needs at least one Input Buffer to receive commands, and a Real Receptor needs at least one Output Buffer to send sensory data.



(a) (b)

Figure 4: Subsystems and Buffers. (**a**) Subsystem Buffers and Internal Memory; (**b**) Relation of particular Subsystems to Communication buffers.

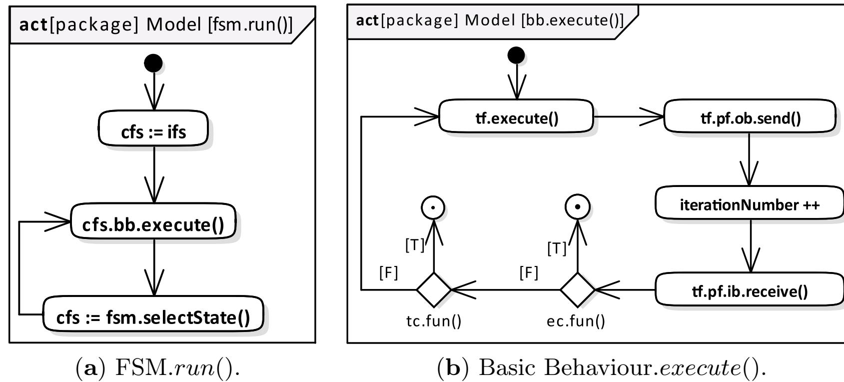
Input Buffer, Output Buffer and Internal Memory are defined analogically as in [12]. Each Buffer contains a data structure *msg*, which stores data of type dataType. The dataType can be defined either as a primitive type or a composite and nested structure. Input Buffer possesses an operation *receive*(), which enables communication with Output Buffers, and stores the received data in the Input Buffer. Analogically, Output Buffer has a *send*() operation, which dispatches the data stored in the Output Buffer to the connected Input Buffers. Internal Memory stores *data*, which is a value of type dataType. Input and Output Buffers are graphically represented by squares connected by an arrow showing the direction of data transfer. Internal Memory is represented by a square with a bidirectional arrow. Various forms of communication between Subsystems have been described in the paper [17].

Similarly to [4, 8], the EARL Subsystem structural model contains a Finite State Machine (FSM) that determines its activities (Figure 3). To define the FSM, the set s of FSM States and the set t of FSM Transitions are distinguished. With each of the states a behaviour bb is associated. Figure 5a defines how the $run()$ operation works. The FSM starts in the initial FSM State ifs. Then, while the Subsystem is running, the $bb.execute()$ operation executes a behaviour associated with the current state, which is represented by cfs. The $fsm.selectState()$ operation evaluates the predicates associated with the FSM Transitions emerging from cfs to select the next FSM State. FSM Transition (Figure 3) is defined by the source and destination FSM States as well as the associated Initial Condition, i.e., predicate ic.

In the following part of the article a SysML dot "." notation [9] is used to depict the nesting of the part instances as well as other block properties. The dot "." can be treated as an extraction operator. It is assumed that if a specific instance of a part is not indicated, the set of all instances of the part is taken into account. In particular, if there is only one instance, there is no need to name it explicitly, only the part name is needed. The same rule applies to references. As the particular parts compose objects of the same type, they can be interpreted as sets in mathematical formulas.

The structure of a Basic Behaviour is defined in Figure 3. The Basic Behaviour includes a Transition Function tf; a Terminal Condition tc , which is a Predicate determining when the execution of the Basic Behaviour should terminate; and an error condition ec, which is a predicate indicating that

an error has been detected in the execution of the Basic Behaviour. Basic Behaviour also posses an $execute()$ operation (Figure 5b). That operation, first executes a Transition Function $tf.execute()$, then all calculated Output Buffers values are sent out by $tf.pf.ob.send()$. Next, $iterationNumber$ is incremented, and $tf.pf.ib.receive()$ gets new values into Input Buffers. Finally, Error Condition $ec.fun$ and Terminal Condition $tc.fun$ are tested. If both values are false starts a new iteration of operations composing the Basic Behaviour; otherwise, the $fsm.run()$ operation designates the next FSM State (Figure 5a).



(a) FSM.$run()$.

(b) Basic Behaviour.$execute()$.

Figure 5: FSM and Basic Behaviour operations.

The structure of a Transition Function is defined in Figure 3. A Transition Function is decomposed into Partial Transition Functions. This sometimes reduces the redundancy of the specification, making it more comprehensible. Moreover, if the implementation of the specified system is based on components, a Partial Transition Function can be identified with a separate component or a set of components [6, 13]. In this case, a Partial Transition Function can be reused in more than one Transition Function similarly as a component can be reused in more than one of the separate groups of components, where one group implements one specific behaviour of a system. Partial Transition Functions composing a Transition Function can be executed in diverse orders, see, e.g., in [16]. To define the execution of Partial Transition Functions within a Transition Function, the operation $execute()$ was introduced. The operation may vary between particular instances of Subsystems.

The structure of a Partial Transition Function is defined in Figure 3. It refers to Input Buffers, Output Buffers as well as Subsystem Internal Memory (Figure 6). A Partial Transition Function can read from the Internal Memory (using the mi reference) or write to it (using the mo reference). It can be defined as a composition of components from the Calculation Components Package (Figure 1). The composition is defined by a $tf.execute()$ operation. The Partial Transition Function algorithm is executed by an $pf.execute()$ operation. The concept of the Embodied Agent as presented in this paper introduces no restrictions on how to implement both of these operations.

Terminal Conditions used by a Basic Behaviour and Initial Conditions utilised within an FSM Transition can be decomposed into Primitive Predicates. A Primitive Predicate takes its arguments from Subsystem Buffers, see Figures 3 and 6. Both Predicate and Primitive Predicate execute an operation called fun producing a Boolean output.

2.3 Embodied Agent Communication

The general system architecture is defined by the Agents and their Subsystems, being the building blocks forming the system structure, and the communication links between those entities. In a way, the architecture is defined by the constraints that are imposed on permissible connections. If no constraints are imposed on the communication links, then the system designer has an excessive freedom of choice, which in the case of his limited experience might lead to an obscure structure. Therefore, architectures limiting this choice are preferred, thus leading to freedom from choice [3]. This provides guidance to the designers, which results in a clear system structure.

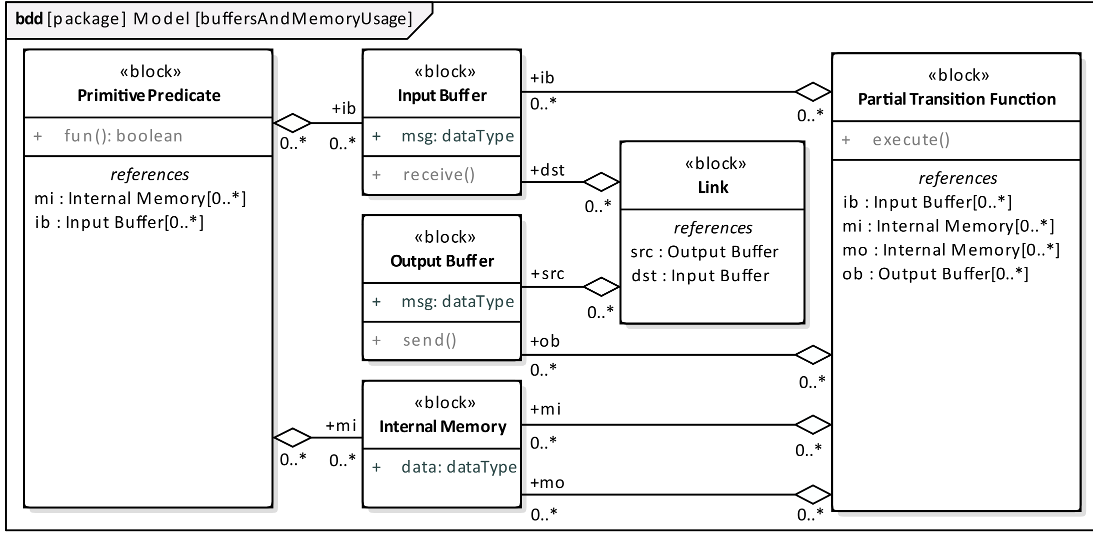


Figure 6: The utilisation of Buffers and Internal Memory by: Partial Transition Function, Primitive Predicate and Links.

In the case of EARL, inter-agent and inter-subsystem communication [16] is defined by unidirectional communication Links (see Figures 2 and 6). The communication takes place between Input Buffers and Output Buffers of Subsystems. Figure 7 presents acceptable communication links between pairs of Subsystems. Note that the inter-agent communication is realized between the Control Subsystems of the communicating agents. Additionally, Figure 7 shows that for each Real Effector present in the system at least one transmission chain should exist. The commands produced by the Control Subsystem, transformed by the Virtual Effector, must reach the Real Effector. Analogically, for each Real Receptor, there is one compulsory communication chain that transmits and processes sensory data. The Real Receptor provides data to the Virtual Receptor which in an agregated form passes it to the Control Subsystem. The other communication Links appearing in Figure 7 are not obligatory. To define bidirectional communication, a pair of unidirectional communication Links is used. Detailed discussion of communication in Embodied Agent systems is presented in [17].

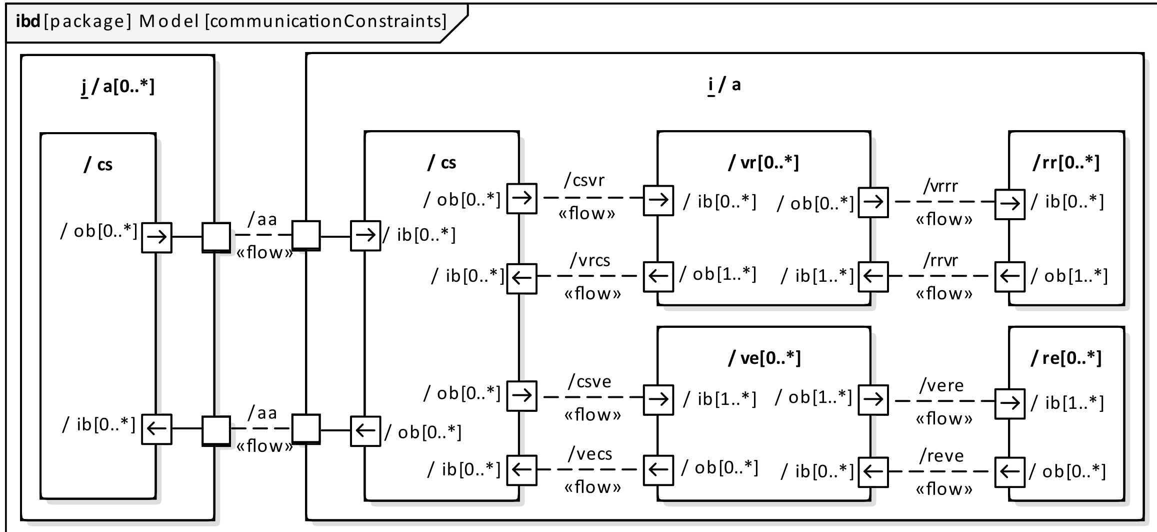


Figure 7: Communication constraints, where $i \neq j$.

2.4 Types of Agents

Four general activities of an Agent can be distinguished [15]:

C – overall control of the agent,

E – exerting influence over the environment by using effectors,

R – gathering the information from the environment by using receptors, and

T – inter-agent communication (transmission).

The first activity is indispensable, but the other three are optional, thus eight types of Agents result (Table 1), depending on their capabilities. However, only seven are of utility, as an agent without any of the optional capabilities is useless.

Table 1: Type of Agent, number of its Subsystems ($|ve|$, $|re|$, $|vr|$, $|rr|$) and number of inter agent communication Links ($|aa|$) expressed with respect to the number of Buffers of the considered Agent.

	$\|cs\|$	$\|ve\|$	$\|re\|$	$\|vr\|$	$\|rr\|$	$\|aa\|$	**Description**
C	1	0	0	0	0	0	zombie (useless)
CT	1	0	0	0	0	1..*	purely computational agent
CE	1	1..*	1..*	0	0	0	blind agent
CR	1	0	0	1..*	1..*	0	monitoring agent
CET	1	1..*	1..*	0	0	1..*	teleoperated agent
CRT	1	0	0	1..*	1..*	1..*	remote sensor
CER	1	1..*	1..*	1..*	1..*	0	autonomous agent
CERT	1	1..*	1..*	1..*	1..*	1..*	full capabilities

2.5 Specification of a Particular Robot Control System

The particular structure of a system is specified by application of instances of specializations of blocks [7] constituting the general model presented above. The names of instances should be long enough to be descriptive and intuitive to interpret, thus reducing the need for additional glossaries. In our approach, each instance can set the number of parts and references (e.g., associated Buffers), however within the limits imposed by the general model. Similarly, each instance can redefine the particular operations of parent blocks present in the general model (e.g., each instance of Partial Transition Function redefines $pf.execute$ operation).

In general, a system instance is defined as a graph. Its nodes represent Agents a and the directed arcs represent the communication Links aa between them. It is a good practice to name Links by using the names of communicating Agents: first the source Agent name, then a comma, and finally the destination Agent name. Input Buffers and Output Buffers of the Control Subsystems are depicted as sources and destinations of dataTypes being transmitted through the Links. The Buffer names reflect the content of dataType being transmitted. The Subsystems are defined analogically.

Specification refers to a system with a static structure and invariable behaviour, or a system with a variable structure at a certain time instant that both can be efficiently solved by using advanced optimization techniques proposed in [1, 2]. To specify a particular system, instances of the relevant concepts appearing in the general system model should be concretised. The SysML diagrams [10] are a part of the EARL-based system (Table 2). Some of the EARL concepts are specified mathematically:

- model and system instance constraints that can not be practically formulated in diagrams,
- fun operations of Predicates and Primitive Predicates, and
- some calculations performed inside actions of Activity Diagrams of Partial Transition Functions, e.g., control laws.

In addition, mathematical notation is used to express formal conditions ascertaining the correctness of the composition of Partial Transition Functions.

Table 2: SysML diagrams describing system parts in EARL.

System Part and Function	SysML Diagrams
System and its parts, initial analysis	req, uc
System and Agent internal structure, Links, Input Buffer, Output Buffer	ibd
FSM, FSM State	stm
Operations of blocks	act

3 Example of a System Specified Using EARL

This section is devoted to the illustration of how to use the EARL language to specify a robot control system. The example presents a single robot multi-agent system containing **CT** and **CET** agents. For the obvious reason of briefness, this description is not a complete specification, but contains only examples of important aspects of the general model and its use:

- Structure of the whole System with Buffers, Internal Memories, inter Agent communication Links, and dataTypes used by them.
- Structure of the particular Agent with Buffers, Internal Memories, inter Subsystem communication Links, and dataTypes used by them.
- Specification of a particular Subsystem, its structure and behaviour, i.e., Buffers, Internal Memories, dataTypes, FSM, Basic Behaviours and their Terminal Conditions and Error Conditions; Primitive Predicates, FSM Transitions and their Initial Conditions; method of both composition and execution of Partial Transition Functions and control law utilised in the activity diagram of Partial Transition Function.

A manipulation robot with N degrees of freedom and a gripper is considered, capable to perform e.g. pick and place task. The specification process starts with the definition of the System structure. Tips on the specification of requirements and use cases using SysML can be found in [5, 11].

3.1 Structure of the System Composed of Agents

There are three Agents in the System (Figure 8). The Agent $task/a$ supervises the task execution, i.e., picking and placing objects; the Agent $manip/a$ controls the N-DOF manipulator; and the Agent $grip/a$ controls the gripper. The gripper controller is separate from the manipulator controller, because different grippers can be attached to the manipulator, thus separate Agents facilitate system modification.

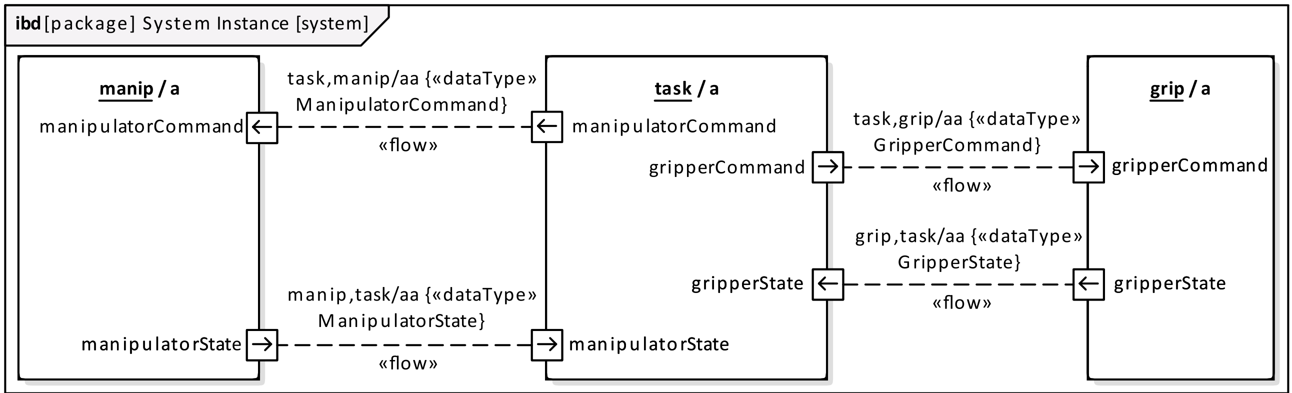


Figure 8: Structure of the considered exemplary System.

Figure 9 presents the dataTypes transmitted between the Agents. The Task Agent *task/a* sends ManipulatorCommands to the Manipulator Agent *manip/a*. The commands contain parameters, e.g., operational or joint position setpoints and a command to perform emergency stop. In return *task/a* gets a ManipulatorState dataType containing: the current operational or joint position, status of the manipulator movement and information whether an emergency stop occurred. The Task Agent *task/a* sends GripperCommand messages to the Gripper Agent *grip/a* and receives GripperStatus in return. Similarly to messages exchanged between *manip/a* and *task/a* Agents, the GripperCommand and GripperStatus messages contain parameters describing the desired and current gripper finger positions.

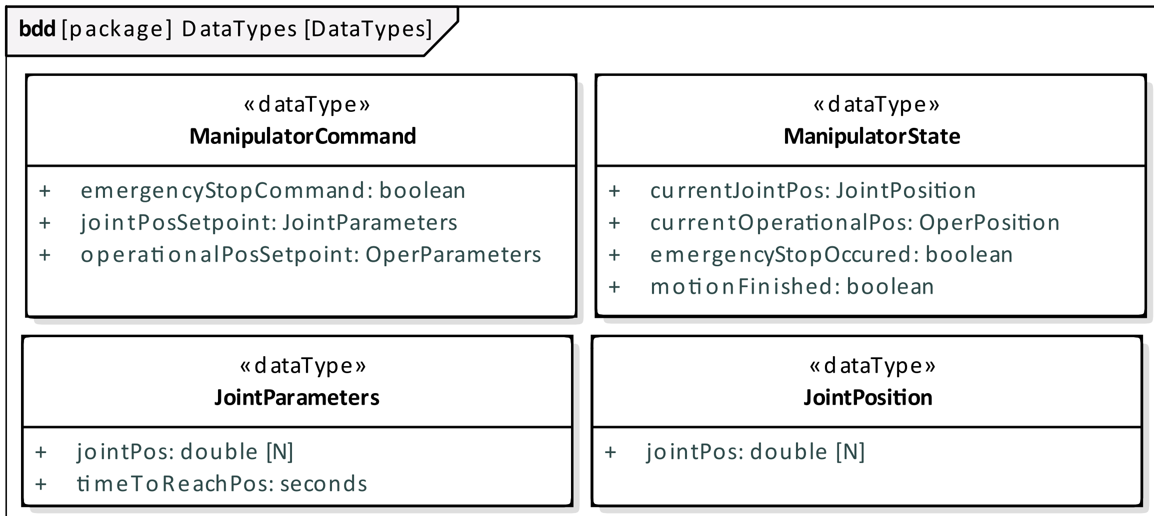


Figure 9: dataTypes transmitted within the System.

3.2 Manipulator Agent *manip/a*

The structure of the Manipulator Agent *manip/a* is presented in Figure 10. Each Real Effector *re* represents one of the N drives of manipulator joints. Each drive is controlled by a Virtual Effector that, e.g., implements a motor position regulator. All N Virtual Effectors *ve* are controlled by a single Control Subsystem *cs*, which causes the manipulator to move either in joint space, where it interpolates between joint positions, or in operational space, where it interpolates between Cartesian poses of a frame affixed to a chosen link of the kinematic chain.

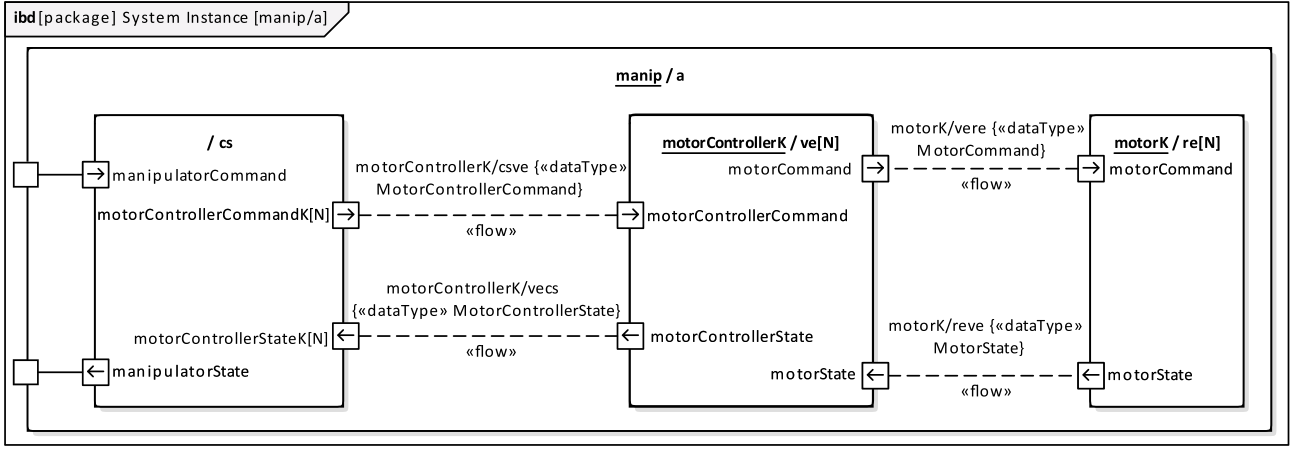


Figure 10: Structure of the Agent *manip/a*; letter K placed at the end of the instance name should be substituted by a number, i.e. $K \in \{1, ..., N\}$.

The dataTypes transmitted inside the Manipulator Agent *manip/a* are presented in Figure 11. The Control Subsystem *cs* sends MotorControllerCommand to each Virtual Effector *motorControllerK/ve*. The dataType contains the desired winding current value or a command to switch the hardware driver to the emergency stop state. Each Virtual Effector *motorControllerK/ve* sends to the Control Subsystem *cs* information about the current motor position and whether the hardware driver is in an emergency stop state. Each Virtual Effector *motorControllerK/ve* sends the desired motor winding current to its respective Real Effector *motorK/re*, and in return receives the encoder readings. Table 3 describes types of data stored in the *manip/a.cs*.

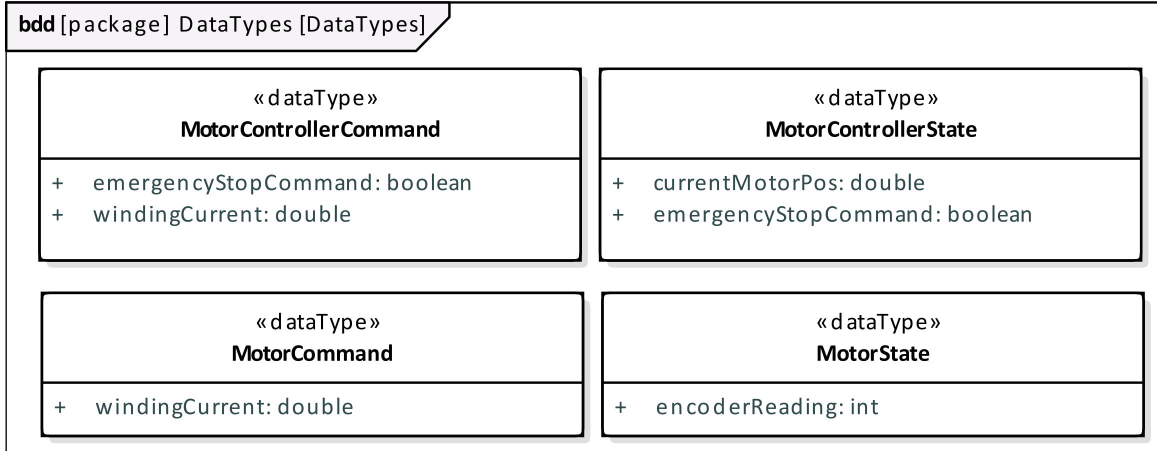


Figure 11: Definition of *manip/a* dataTypes.

Table 3: $manip/a.cs.\boldsymbol{m}$ dataTypes.

$\boldsymbol{m}$	**dataType**
motionFinished/m	boolean
currentOperationalPos/m	OperPosition
currentJointPos/m	JointPosition
emergencyStopCommandK/m	boolean
windingCurrentK/m	double

Table 4 describes Primitive Predicates *pp* used in the Control Subsystem *manip/a.cs*. They take

as arguments the contents of the buffers and memory. The $newData(InputBuffer.msg)$ function producing Boolean values, returns TRUE if there is new data in the Input Buffer, and FALSE if the data is obsolete.

Table 4: Definitions of $manip/a.cs.\boldsymbol{pp}.\boldsymbol{fun}$.

$\boldsymbol{pp}$	$\boldsymbol{fun}$
$emergencyStop/pp$	$manipulatorCommand/ib.msg.emergencyStopCommand \vee$ $motorControllerState1/ib.msg.emergencyStopCommand \vee \ldots \vee$ $motorControllerStateN/ib.msg.emergencyStopCommand$
$motionFinished/pp$	$motionFinished/mi.msg$
$newJointPos/pp$	$newData(manipulatorCommand/ib.msg.jointPosSetpoint)$
$newOperationalPos/pp$	$newData(manipulatorCommand/ib.msg.operationalPosSetpoint)$
$false/pp$	FALSE

Table 5 describes the Predicates utilised by $manip/a.cs$. Figure 12 shows possible transitions between the FSM States of the Control Subsystem $manip/a.cs$ as well as the association of Basic Behaviours to particular FSM States.

Table 5: Initial conditions labelling $manip/a.cs.fsm$ transitions and terminal conditions of $manip/a.cs.bb$. It is assumed that $task/a$ can not set simultaneously a new joint position and an operational space pose.

Labels of transitions between FSM States $cs.fsm.\boldsymbol{t}.ic.fun \triangleq$ **PREDICATE**	
$\boldsymbol{t}$	**PREDICATE**
$idle, jointMove/t$	$newJointPos/pp.fun \wedge \neg emergencyStop/pp.fun$
$jointMove, jointMove/t$	$newJointPos/pp.fun \wedge \neg emergencyStop/pp.fun$
$idle, operationalMove/t$	$newOperationalPos/pp.fun \wedge$ $\neg emergencyStop/pp.fun$
$operationalMove, operationalMove/t$	$newOperationalPos/pp.fun \wedge$ $\neg emergencyStop/pp.fun$
$jointMove, idle/t$	$\neg emergencyStop/pp.fun$
$operationalMove, idle/t$	$\neg emergencyStop/pp.fun$
$i, emergencyStop/t$; where $i \neq emergencyStop$	$emergencyStop/pp.fun$
Definitions of Terminal Conditions $cs.\boldsymbol{bb}.tc.fun \triangleq$ **PREDICATE**	
$\boldsymbol{bb}$	**PREDICATE**
$idle/bb$	$newJointPos/pp.fun \vee newOperationalPos/pp.fun \vee$ $emergencyStop/pp.fun$
$jointMove/bb$	$motionFinished/pp.fun \vee emergencyStop/pp.fun$
$operationalMove/bb$	$motionFinished/pp.fun \vee emergencyStop/pp.fun$
$emergencyStop/bb$	$false/pp.fun$

The Control Subsystem $manip/a.cs$ uses the following Partial Transition Functions.

- $calculatePosition/pf$—calculates manipulator joint positions and end-effector operational space pose.
- $jointMove/pf$ / $operationalMove/pf$—generates the joint/operational space trajectory and calculates the winding current needed to realize the motion (Figure 13).
- $passiveRegulation/pf$—calculates the winding current needed to keep the manipulator in a stationary position.

- *emergencyStop/pf*—copies the information about the occurrence of an emergency stop to Output Buffers that are linked to the associated Subsystem Input Buffers.
- *outputManipState/pf*—composes *ManipulatorState/ob* (Figure 14a).
- *outputMotorCon/pf*—composes *DriveControllerCommandK/ob* (Figure 14b).

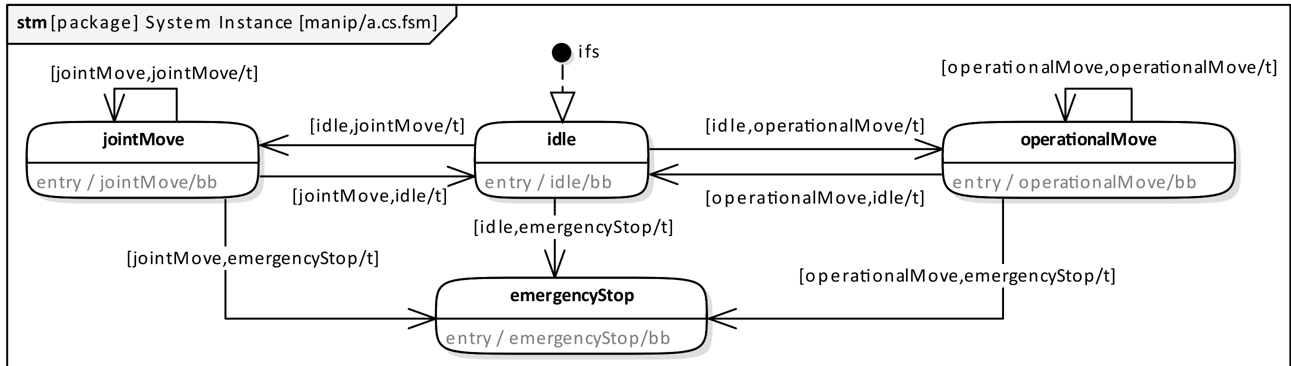


Figure 12: *manip/a.cs.fsm* definition. Conditions of transitions between FSM States are specified in Table 5.

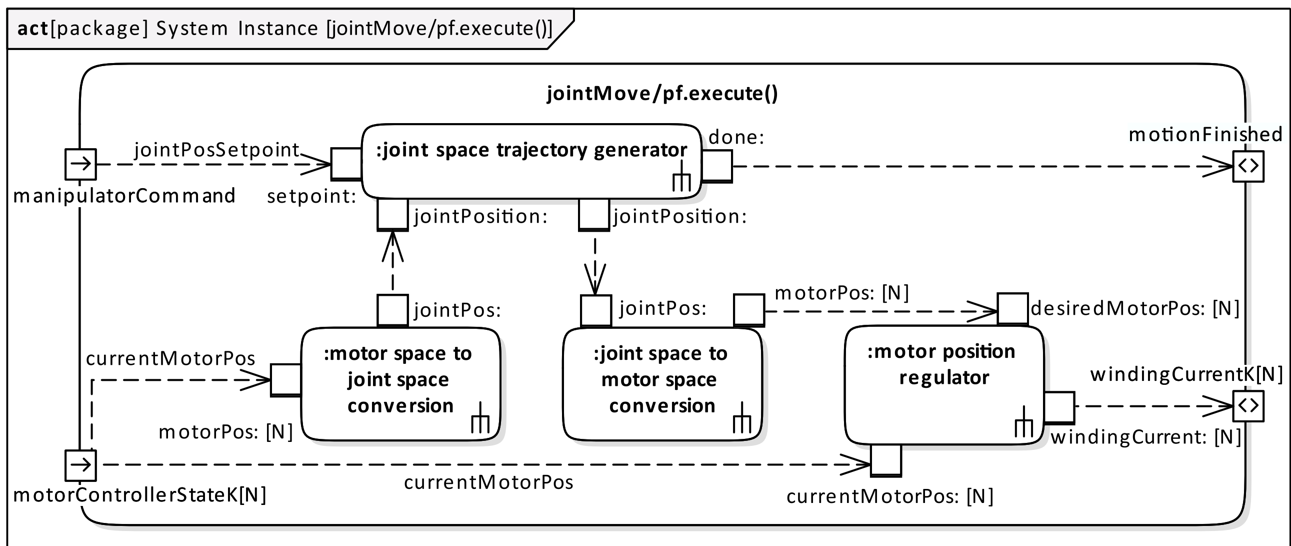
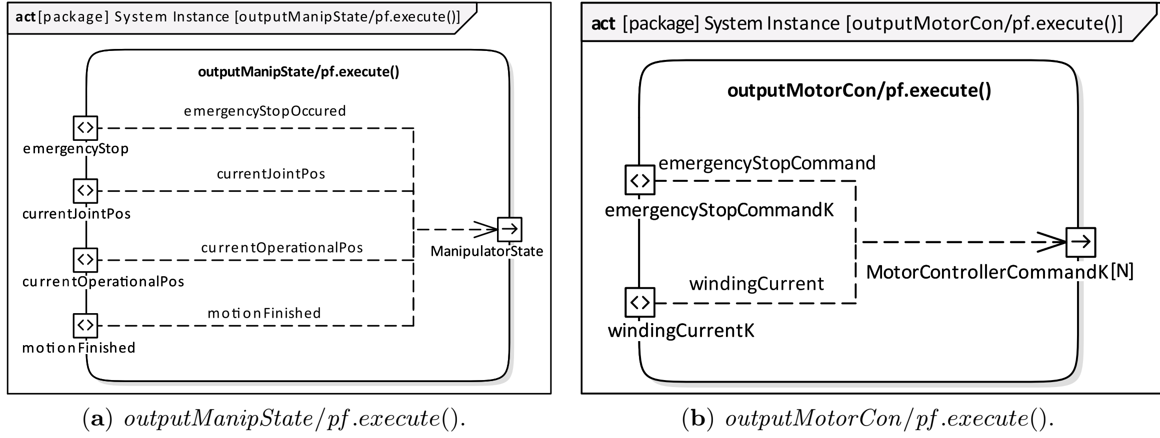


Figure 13: *manip/a.cs.jointMove/pf.execute*() – operation definition.



(a) $outputManipState/pf.execute()$. (b) $outputMotorCon/pf.execute()$.

Figure 14: $manip/a.cs.pf.execute()$—operations definition.

Figure 13 shows the execute operation of a $jointMove/pf$ Partial Transition Function. This Partial Transition Function realises, e.g., the PI type *motor position regulator* for each joint Equation (2), Equation (3):

$$windingCurrent = K_p e(t) + K_i \int_0^t e(t')dt', \tag{2}$$

$$e = desiredMotorPos - currentMotorPos, \tag{3}$$

where K_p and K_i are, respectively, proportional and integral gain factors, e is the position error, t is time.

Table 6 shows which Partial Transition Functions constitute the definitions of Transition Function compositions used by Basic Behaviours of $manip/a.cs.bb$.

The Partial Transition Functions $manip/a.cs.pf$ are subdivided into two disjoint sets: pfc and pfo. The pfc set Equation (4) contains Partial Transition Functions that take as arguments Input Buffers: $manipulatorCommand/ib$ and $motorControllerCommandK/ib[N]$

$$\begin{aligned} pfc = \{ & calculatePosition/pf, jointMove/pf, operationalMove/pf, \\ & passiveRegulation/pf, emergencyStop/pf\}. \end{aligned} \tag{4}$$

The values produced by them are inserted into the Internal Memory mbo Equation (5)

$$\begin{aligned} mbo = \{ & motionFinished/mo, currentOperationalPos/mo, currentJointPos/mo, \\ & emergencyStopCommandK/mo, windingCurrentK/mo\}. \end{aligned} \tag{5}$$

Functions from the pfo set Equation (6) take arguments from the Internal Memory mbo Equation (5) and produce the Output Buffer values: $ManipulatorState/ob$ and $MotorControllerCommandK/ob[N]$, hence they produce output of the whole Subsystem

$$pfo = \{outputManipState/pf, outputMotorCon/pf\}. \tag{6}$$

It was assumed that any two Partial Transition Functions used by a particular Transition Function do not produce data to the same Output Buffers and Internal Memories, therefore the following conditions are formulated for the pfc set Equation (7) and the pfo set Equation (8), respectively,

$$(\forall x/bb)(\forall x/bb.i/pf, x/bb.j/pf \in pfc,\ i \neq j)(x/bb.i/pf.k/mo \nsim x/bb.j/pf.k/mo,\ k/mo \in mbo), \tag{7}$$

$$(\forall x/bb)(\forall x/bb.i/pf, x/bb.j/pf \in pfo,\ i \neq j)(x/bb.i/pf.k/ob \nsim x/bb.j/pf.k/ob), \tag{8}$$

where $\nsim$ stands for „is not the same entity”.

Table 6: Compositions of Transition Functions $manip/a.cs.\boldsymbol{bb}.tf.\boldsymbol{pf}$. The right part of the table presents what parts of output buffers and internal memory are produced by the specific Partial Transition Functions.

		/mo					/ob	
bb	***pf***	*motionFinished*	*currentJointPos*	*currentOperationalPos*	*emergencyStop*	*windingCurrentK*	*ManipulatorState*	*MotorControllerCommandK*
idle/bb	*outputManipState/pf*						•	
	outputMotorCon/pf							•
	calculatePosition/pf		•	•				
	passiveRegulation/pf					•		
jointMove/bb	*outputManipState/pf*						•	
	outputMotorCon/pf							•
	calculatePosition/pf		•	•				
	jointMove/pf	•				•		
operationalMove/bb	*outputManipState/pf*						•	
	outputMotorCon/pf							•
	calculatePosition/pf		•	•				
	operationalMove/pf	•				•		
emergencyStop/bb	*outputManipState/pf*						•	
	outputMotorCon/pf							•
	calculatePosition/pf		•	•				
	emergencyStop/pf				•			

Transition Functions act in the following way. First, they compute the Partial Transition Functions from the *pfc* set, and then they compute the Partial Transition Functions from the *pfo* set. The fulfilment of Equations (7) and (8) makes it possible to run Partial Transition Functions being members of *pfc* in parallel in the first stage of the Transition Function execution, and then run the Partial Transition Functions being members of *pfo* set in parallel in the second stage of Transition Function execution. To illustrate the above considerations, Figure 15 shows the definition of $jointMove/bb.tf.execute()$ operation – practical realization of Partial Transition Functions execution for $jointMove$ Basic Behaviour.

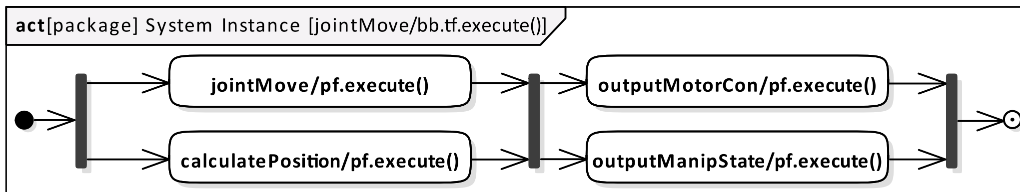


Figure 15: $jointMove/bb.tf.execute()$ operation definition.

References

[1] A. Caliciotti, G. Fasano, S. G. Nash, and M. Roma. "An adaptive truncation criterion, for linesearch-based truncated Newton methods in large scale nonconvex optimization". In: *Operations Research Letters* 46.1 (2018), pages 7–12. ISSN: 0167-6377. DOI: 10.1016/j.orl.2017.10.014.

[2] A. Caliciotti, G. Fasano, S. G. Nash, and M. Roma. "Data and performance profiles applying an adaptive truncation criterion, within linesearch-based truncated Newton methods, in large scale nonconvex optimization". In: *Data in Brief* 17 (2018), pages 246–255. ISSN: 2352-3409. DOI: 10.1016/j.dib.2018.01.012.

[3] S. Dennis, L. Alex, L. Matthias, and S. Christian. "The SmartMDSD Toolchain: An Integrated MDSD Workflow and Integrated Development Environment (IDE) for Robotics Softwaree". In: *Journal of Software Engineering in Robotics* 7.1 (2016), pages 3–19.

[4] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane. "RobotML, a Domain-Specific Language to Design, Simulate and Deploy Robotic Applications". In: *International Conference on Simulation, Modeling, and Programming for Autonomous Robots.* Nov. 2012, pages 149–160. DOI: 10.1007/978-3-642-34327-8_16.

[5] M. dos Santos Soares and J. Vrancken. "Requirements specification and modeling through SysML". In: *IEEE International Conference on Systems, Man and Cybernetics.* 2007, pages 1735–1740. DOI: 10.1109/ICSMC.2007.4413936.

[6] W. Dudek, W. Szynkiewicz, and T. Winiarski. "Nao Robot Navigation System Structure Development in an Agent-Based Architecture of the RAPP Platform". In: *Recent Advances in Automation, Robotics and Measuring Techniques.* Edited by R. Szewczyk, C. Zieliński, and M. Kaliczyńska. Volume 440. Advances in Intelligent Systems and Computing (AISC). Springer, 2016, pages 623–633. DOI: 10.1007/978-3-319-29357-8_54.

[7] S. Friedenthal, A. Moore, and R. Steiner. *A practical guide to SysML: The systems modeling language. 3rd ed.* Elsevier, Morgan Kaufmann, 2015.

[8] T. Kornuta and C. Zieliński. "Robot control system design exemplified by multi-camera visual servoing". In: *Journal of Intelligent and Robotic Systems* 77.3–4 (2013), pages 499–524. DOI: 10.1007/s10846-013-9883-x.

[9] *OMG Systems Modeling Language - Version 1.6.* accessed on 4 April 2020. Open Management Group. Dec. 2019. URL: https://www.omg.org/spec/SysML/1.6/.

[10] A. Salado and P. Wach. "Constructing True Model-Based Requirements in SysML". In: *Systems* 7.2 (2019). ISSN: 2079-8954. DOI: 10.3390/systems7020019.

[11] M. Soares, J. Vrancken, and A. Verbraeck. "User requirements modeling and analysis of software-intensive systems". In: *Journal of Systems and Software* 84 (Feb. 2011), pages 328–339. DOI: 10.1016/j.jss.2010.10.020.

[12] P. Trojanek. "Design and implementation of robot control systems reacting to asynchronous events". PhD thesis. Warsaw University of Technology, 2012.

[13] T. Winiarski, K. Banachowicz, M. Walęcki, and J. Bohren. "Multibehavioral position–force manipulator controller". In: *21th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR'2016*. IEEE, 2016, pages 651–656. DOI: 10.1109/MMAR.2016.7575213.

[14] T. Winiarski, M. Węgierek, D. Seredyński, W. Dudek, K. Banachowicz, and C. Zieliński. "EARL – Embodied Agent-Based Robot Control Systems Modelling Language". In: *Electronics* 9.2 (2020), page 379. DOI: 10.3390/electronics9020379.

[15] C. Zieliński, T. Winiarski, and T. Kornuta. "Agent-Based Structures of Robot Systems". In: *Trends in Advanced Intelligent Control, Optimization and Automation.* Edited by J. Kacprzyk and et al. Volume 577. Advances in Intelligent Systems and Computing. 2017, pages 493–502. DOI: 10.1007/978-3-319-60699-6_48.

[16] C. Zieliński. "Transition-Function Based Approach to Structuring Robot Control Software". In: *Robot Motion and Control.* Edited by K. Kozłowski. Volume 335. Lecture Notes in Control and Information Sciences. Springer-Verlag, 2006, pages 265–286.

[17] C. Zieliński, M. Figat, and R. Hexel. "Communication Within Multi-FSM Based Robotic Systems". In: *Journal of Intelligent & Robotic Systems* 93.3 (2019), pages 787–805. ISSN: 1573-0409. DOI: 10.1007/s10846-018-0869-6.

[18] C. Zieliński, T. Kornuta, and T. Winiarski. "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*. IEEE, 2014, pages 1–14. DOI: 10.1109/MMAR.2014.6957317.

[19] C. Zieliński and P. Trojanek. "Stigmergic cooperation of autonomous robots". In: *Journal of Mechanism and Machine Theory* 44 (Apr. 2009), pages 656–670.