Simulation-Based and Vendor-Independent Multi-Robot Control Development

Birger Freymann, Thorsten Pawletta*, David Jammer

Research Group Computational Engineering and Automation, University of Applied Sciences Wismar,
Philipp-Müller-Straße 14, 23966 Wismar, Germany
*birger-freymann@hotmail.de, *thorsten.pawletta@hs-wismar.de, david.jammer@hs-wismar.de*

SNE 34(4), 2024, 185-194, DOI: 10.11128/sne.34.tn.10711
Submitted: 2024-06-28; Revised: 2024-07-30
Accepted: 2024-10-01
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna
ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. The Simulation-Based Control (SBC) approach and the Task-Oriented Control (TOC) development, which have already been investigated for individual articulated robots, is extended to teams of articulated robots, hereinafter also referred to as Multi-Robot System (MRS).
The basics of SBC, TOC and client-server approach for vendor-independent robot programming are introduced. Possible interactions in an MRS are then analyzed and classified, and a TOC description of interactions is developed. The TOC description is based on reusable, atomic tasks that can be aggregated sequentially and hierarchically via interfaces.
For three interaction classes, the TOC specification and its transformation into executable robot commands using a client-server based robot midleware is discussed. The control dynamics is described with the Discrete Event System Specification (DEVS) and represented using DEVS diagrams.

Introduction

Robots have been established in industry for decades as powerful and flexible tools. With new areas of application, as defined in the context of Industry 4.0, the requirements for the efficient development of robot controls are growing. The methods of Rapid Control Prototyping (RCP) can make a significant contribution here (Abel and Bollig [1]).

RCP requires a consistent process model, an end-to-end tool chain from the design phase through to operation, and the consistent use of modeling and simulation methods to continuously verify development steps. Almost all robot manufacturers offer RCP-based development environments. However, these are manufacturer-specific and only support their own robots.

The methodological and software differences make it difficult to set up Multi-Robot Systems (MRSs), consisting of robots from different manufacturers.

Another method for the efficient development of robot controls are high-level specifications, such as the Task-Oriented Control (TOC) approach (Siciliano [2]). The principle of TOC is to break down complex problems into a set of reusable, atomic tasks that can be aggregated sequentially or hierarchically via interfaces. To execute a TOC, the tasks must be translated into executable robot commands.

This paper proposes a vendor-independent RCP approach for robotic control development. We build on the Simulation-Based Control (SBC) approach introduced by Maletzki [3] for Single-Robot Systems (SRS) and extend its application to MRS.

The SBC approach corresponds to the RCP method and enables manufacturer-indepent end-to-end development from the control design phase through to the operational use. Furthermore, the SBC approach defines a framework for practical implementation. A simulation environment with real-time process interface and client-server based robot midleware form its core.

In contrast to an SRS, the interactions between robots must be described when controlling an MRS. For this purpose, interactions between articulated arm robots are analyzed and classified. We examine how the interactions can be specified and processed using reusable tasks. To do this, we define a case study and discuss for three interaction types the TOC specification and its transformation into executable robot commands.

We use a Discrete Event System Specification (DEVS) based modeling and simulation environment and describe the tasks and their transformation into robot code using DEVS diagrams. Details on the DEVS environment can be found in Freymann [4].

1 Fundamentals

In this section, the basics of the SBC and TOC approach are presented and some methods for manufacturer-independent control programming of MRS are discussed.

1.1 Simulation Based Control Approach

The SBC approach is a methodology for control development and defines a framework for practical control implementations (Maletzki et al. [5]).It supports a continuous simulation-based development process from the early specification of a control logic through to the operational use of a control system on the basis of an integrated tool chain. A schematic representation of the SBC approach is shown in Figure 1.

A Simulation Model (SM) developed in the design phase is extended step by step to a Control Software (CS) and continuously tested by simulation during the development process. This approach eliminates the need to reimplement simulation code in control code. In this way, errors are avoided, development time is saved and overall development costs are reduced. The consistent testing of development steps with SM makes it possible to identify and correct errors at an early stage.

As shown in Figure 1, it is necessary to consistently distinguish between the Control Model (CM) with the control logic and the Process Model (PM) with the image of the real process as early as possible during the development. The operational mode requires an interface to the real process, which records measured sensor values and conversely sends actuator commands to the components of the real process. For this reason, an Interface Model (IM) must be developed for the operational phase. This IM should support simulative testing with a robot simulation and at the same time act as an interface to the real process. In contrast to most other RCP approaches, the PM remains a component in the control system even in the operating mode. This enables the calculation of non-measurable or poorly measurable process variables and the realization of observer concepts.

1.2 Task-oriented Control Design

TOC is an established concept for control design (Siciliano et al. [2]) and is used for programming SRS in many applications [3, 6]. The procedure for creating a TOC corresponds to the human way of thinking when solving complex problems.

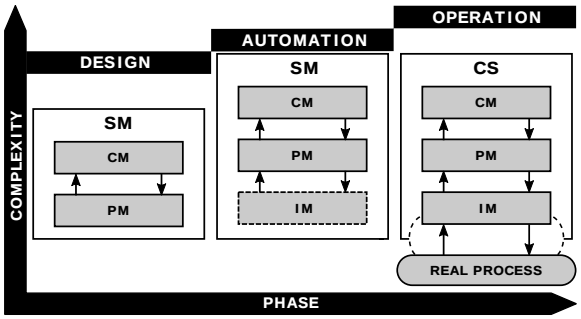


Figure 1: Schematic represenation of the SBC approach.

The basic principle is the representation of control problems by a sequence of tasks. Tasks are usually independent work steps. The tasks should be formulated universally in order to be able to represent different control problems by linking tasks in different ways. Tasks can be linked purely sequentially, conditionally or in loops. Aggregated tasks can be created by links on one level or in hierarchies. The *Closure Under Coupling* principle according to Zeigler et al. [7] should apply to both atomic and aggregated tasks, i.e. a task composed of subtasks cannot be distinguished from an equivalent atomic task. This property is the basis for modularizing task-based descriptions and implementing reusable tasks. In addition, solving a problem may require processing tasks in parallel.

According to Figure 2, a task-based control is specified within the CM. Such a control specification is not directly executable because tasks are an abstract description of work steps. A task only describes the *what* but not the *how* or *with what* something is to be implemented. A *transformation method* is needed to execute tasks. It transforms tasks into control commands for real devices using a *world model*. The result is an executable control code in a language specific to the application. In robotics, for example, this can be a control code in a manufacturer-specific robot programming language. In the SBC approach, the PM corresponds to the world model and the IM to the transformation method.

1.3 Vendor-independent Control Development

In addition to different types of robots, which are optimised for certain areas of application in terms of hardware, the software solutions for programming robots also differ greatly and are often manufacturer-specific.

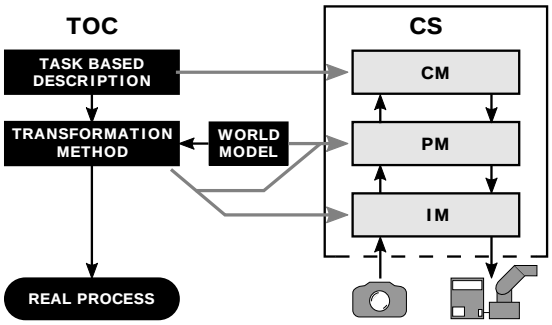


Figure 2: Implementation of TOC within the SBC approach.

Manufacturers are creating their own self-contained ecosystem that exclusively supports the robots, sensors and actuators of their own brand. The robots are mostly programmed using a specific programming language. Therefore, Robot Oriented Middleware from different manufacturers is generally incompatible.

Manufacturer-specific software and hardware systems complicate the development of robot applications, especially for MRS. Once developed, control solutions cannot simply be ported to robot systems from other manufacturers, even if the robots have almost the same functionality. Long-standing efforts to standardize robot programming languages have failed.

An alternative approach for robot programming is based on the client-server model, as shown schematically in Figure 3. Here, the robot controller acts as a server and gives a client access to all manufacturer-specific services of the robot. Client and server communicate via interfaces (Serial, Ethernet, etc.) of the robot controller. Control commands from the client are translated into a data word and then transmitted to an interpreter via a data connection. The interpreter is developed in the robot manufacturer's specific language and executed on the robot controller. The interpreter executes received data words and sends back data words to the client. In this way, sensor data and robot movement commands are communicated and executed.

Based on this concept, Deatcu et. al [8] developed a toolbox in MATLAB (client side) and interpreter (server) for KUKA, Kawasaki and virtual robots. Physical and virtual robots from both manufacturers can be programmed, tested and deployed using a uniform MATLAB-based command set. The integration of physical or virtual robots from other manufacturers only requires the implementation of an interpreter in their specific language.

With the virtual robots, the toolbox comprehensively supports the requirements of the RCP with regard to an end-to-end software chain and the use of simulation models from the early planning phase through to operation.

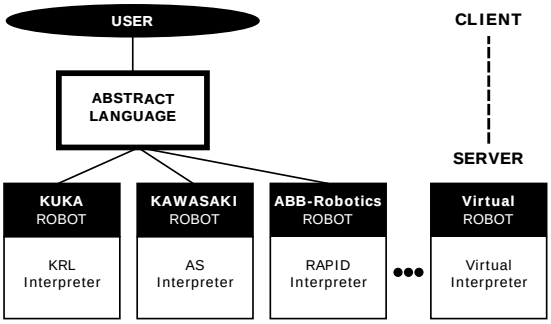


Figure 3: Robot control based on client-server approach.

2 Interaction in MRS and its Implementation Within the SBC Approach

In this section, interactions of joined-arm robots in an MRS are analyzed based on the interaction classes formulated by Lüth [9].

Subsequently, it is examined how the interaction type *coordination* can be realized within the SBC approach. According to section 1.2, the task specification, the *WHAT*, is done in the CM layer and the task transformation, the *HOW* and *WHO*, in the PM layer. We start our consideration with the task transformation and then devote ourselves to the task specification.

2.1 Interaction Classes

According to Lüth [9], the interactions of industrial robots in MRS can be divided into six classes. For better illustration, these are discussed below using the example of a transport problem.

Figure 4 shows the structure of the transport problem using an SRS. Parts are transported from an Input Buffer (IB) to an Output Buffer (OB). Based on this basic structure, Figure 5 shows derived system structures for MRS according to the six interaction classes by Lüth [9].

Class 0: The structure in Figure 4 shows an SRS that does not involve any interaction. The robot R1 can be understood as a server S with the capacity of one. It has to move one part at a time from the IB to the OB.

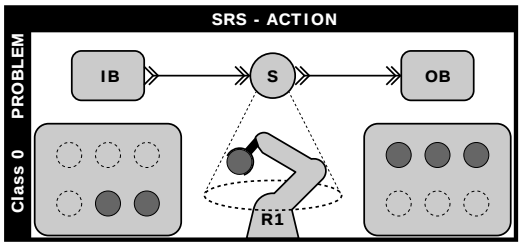


Figure 4: Structure of the transport problem with an SRS (interaction class 0).

Class 1: The MRS consists of two robots (R1, R2) with separate workspaces. Both robots have identical tools and capabilities.

The overall task consists of the transport of parts from the IB to the OB. Due to the separate workspaces and the same identity of the robots, they do not have to interact to solve their partial task.

Class 2: The structure is analogous to class 1, but each workspace contains a different part type, and robot R2 is adapted to the new part type. Due to the separate workspaces, the robots do not have to interact and do not need to be coordinated. In contrast to class 1, the robots are not mutually interchangeable.

Class 3: The robots are no longer spatially separated from each other. Their workspaces overlap, making it necessary to coordinate their movements in order to avoid collisions. *Coordination* requires an exchange of information, a *communication*.

Class 4: Another part type is added, which can only be moved by both robots together. This requires *cooperation* and *coordination* between the robots. The timing of the cooperation must be planned in order to avoid unnecessary waiting times for the robots.

Class 5: Each robot can handle the other robot's part types to support the partner in an overload situation. There follows no principally new requirement regarding communication and coordination.

Class 6: There is a third type of part to be transported that cannot be handled with the standard configuration of the two robots R1 and R2. This results in the need for a new interaction, referred to here as *dynamic function expansion*. This can be done by changing tools or temporarily integrating a specific robot R3.

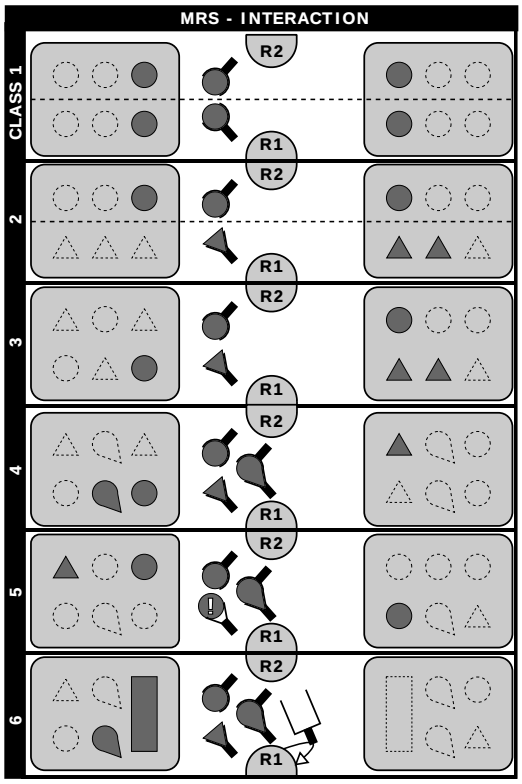
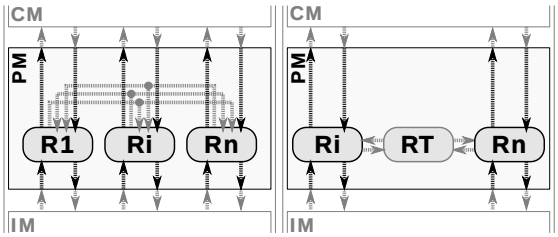


Figure 5: Illustration of the six interaction classes according to Lüth [9] using the transportation problem.

2.2 Task Transformation on the PM Layer

Interaction class 3 requires *temporal coordination* of robots in order to avoid collisions in the shared workspace. Figure 6 shows two variants for bidirectional *communication* of robot model components on the PM layer. In Figure 6a, the communication is solved on the basis of direct couplings between the robot components. The structure created by the couplings resembles a peer-to-peer (P2P) network as is known from the field of network technology. The effort required to couple components with each other in this way increases greatly with their number.

For this reason, the introduction of a new component, called Robot-Team (RT), is proposed in Figure 6b. This serves as a *middleman* between the robot components. The number of input and output ports required for communication is reduced to one input and output port per robot component R. The structure of a robot component R is independent of the number of components communicating with each other. RT stores all information relevant for team coordination. This significantly reduces the complexity of modeling at PM level.



(a) Peer-to-Peer (P2P) (b) Robot Team (RT)

Figure 6: Two variants for modeling communication on the PM layer.

2.3 Task Specification on the CM Layer and Relationship to the PM Layer

Figure 7 shows the basic implementation of a TOC within the SBC for interaction classes 1 to 3 according to the problem illustrated in Figure 5. The task sequences of the two robots (R1, R2) are specified as concurrent processes on the CM layer.

As already described, the task transformation takes place in the PM layer. The notation Ri.STS and Ri.CMD stands for coupling relationships between the CM and PM layers to exchange control commands (CMD) and status information (STS).

The interfaces to the robot middleware are implemented in the IM layer (INTF1 and INTF2). The RTC component in the IM layer stands for *Real Time Clock* and realizes the real-time synchronization of the executing simulator required during the operational phase.

Due to the separate workspaces, the robots can perform their tasks independently of each other in the case of interaction class 1 and 2. There is no need for interaction between the robots. The *Lock* and *UnLock* tasks on the CM layer and the RT component on the PM layer are not required.

A task sequence starts with the part identification task *IdPrT*, which provides position parameters (Pos). The Pos are passed to the task *Move*, which is followed by the task *PickPrt* to pick up a part, and so on.

In the case of interaction class 3, IB is a common workspace for both robots. It must be ensured that the robots do not move into the IB at the same time. To map a mutual exclusion (mutex) during the execution of the *Move(IB)* task for both robots, the tasks *Lock* and *UnLock* are introduced. These specify the start and end of a task coordination.

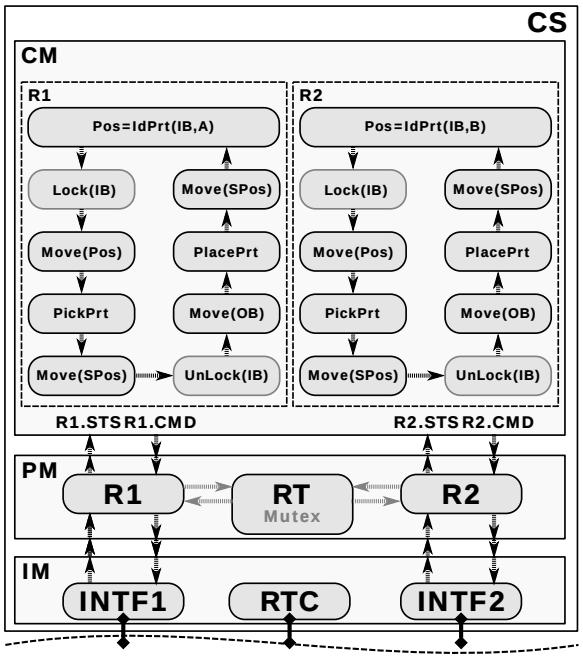


Figure 7: Concept of a TOC within the SBC approach by the example of interaction class 3 (for classes 1 and 2, the Lock and UnLock tasks as well as the RT component are omitted).

The resulting coordination of the two robots is implemented by the RT *Mutex* component. Mutex can be implemented using inter-process communication according to Tanenbaum and Bos [10].

When the task *Lock(IB)* is called, a resource with the name IB is reserved by the respective robot and blocked for others. The information about which robot owns which resource is stored in the RT component and exchanged via the coupling relationships between RT and the robot components R1 and R2. The robot that executes the *Lock* task first becomes the owner. Another robot cannot initially complete the *Lock* task.

However, a robot's wish to own the resource next can already be saved in RT. The reservation only becomes valid when the resource is released again with the *UnLock* task. If a reservation has already been made, ownership is immediately transferred to the reserving robot. Before a robot that is in ownership of a resource releases it again, it should leave the critical workspace. The *Move(SPos)* task is defined for this purpose, which describes the movement to a safe position.

3 DEVS-based Implementation of a TOC Within the SBC Approach

This section discusses the implementation of TOC according within the SBC approach using a DEVS simulation environment with a real-time process interface. It starts with the definition of tasks for the CM layer using DEVS. Then, we consider a DEVS-based transformation of tasks without interactions as well as a DEVS-based implementation of the interface to the robot middleware.

Finally, a DEVS-based transformation of tasks with interactions is described using the example of interaction type *Coordination* (class 3). The DEVS specifications are presented using extended DEVS diagrams. The practical implementation was carried out using DEVS-RCP-V2, a MATLAB-based DEVS environment. Details on the extended DEVS diagrams and on DEVS-RCP-V2 can be found in Freymann [4].

3.1 Specification of Tasks for the CM Layer

Figure 7 shows the task sequences of the case study on the transportation problem for interaction classes 1 to 3. With the exception of the task *IdPrt* for part identification, all other tasks can be defined according to the uniform pattern in Figure 8. The ports *STS* and *CMD* form the interface to the PM layer (see Figure 7)), while the ports *BEG* and *NXT* are used to link tasks in the CM layer. The task pattern defines the two phases *Passive* and *Active*.

In DEVS diagrams, phases are values of the state variable *phase*, which are represented with a box. These state values are often referred to as main states. Each state has a dwell time, including zero and infinity. The dwell time is notated with *@Variable* or *@Value*.

In the example in Figure 8, the initial phase is *Passive*. An external event *'next'* at the input port BEG (*BEG?'next'*) causes a change to the phase *Active* and the state transition $\sigma = 0$ schedules an immediate internal event. Internal events can cause state changes and trigger output events. In this case, an output event $(tid, p1, p2)$ at port CMD (*CMD!(tid,p1,p2)*) is generated. Parameter *tid* encodes the ID of the current task and the identifiers *p1, p2* are placeholders for specific task parameters. In the *Move* task, for example, the position to be approached is coded in *p1*.

Due to the state transition $\sigma = \infty$, the task remains in the phase *Active* until the next external event. When an external event *'done'* at the port STS (*STS?'done'*) occurs, the task changes to phase *Passive* and triggers immediately an internal event due to $\sigma = 0$. As a result, an output event *'next'* is generated at port NXT (*NXT!'next'*) and a follow-up task is activated. Due to the state transition $\sigma = \infty$, the current task remains in phase *Passive* until a new external event *BEG?'next'* occurs.

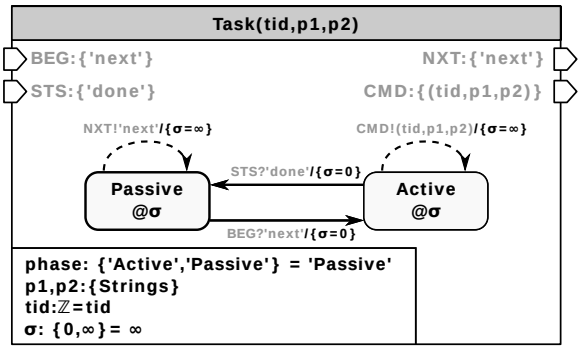


Figure 8: DEVS specification of a general task pattern.

3.2 Transformation of Tasks Without Interactions

The task transformation takes place by the robot components R1 and R2 in the PM layer (see Figure 7). Both components have an identical behavior. Figure 9 shows the DEVS specification for translating the task sequences of the case study for classes 1 and 2. It defines the three phases *Passive, Error* and *Active*, with the initial phase *Passive*. If component R is to execute a task, it must be in the phase *Passive*.

The task to be executed is received as an external event $(tid, p1, p2)$ via the input port CM_CMD (*CM_CMD?(tid,p1,p2)*) and an internal event is immediately scheduled with $\sigma = 0$. The external event codes a task ID in tid and task specific values in $p1$ and $p2$. The task ID decides which state transition, and therefore which task transformation, is executed.

The part identification task *IdPrt* is realized with a table, which defines part types and position coordinates. The part is selected depending on the task parameters $p1$ and $p2$, the coordinates are assigned to the state variable *position*, and the state variable *sts='done'* is set.

If no part can be identified, *sts='none'* is set. As there is no communication with the real process, there is a return to the phase *Passive*. The scheduled internal event triggers an output event (*CM_STS!sts*) that sends the value of the state variable *sts* to the CM layer via the port CM_STS. The state transition $\sigma = \infty$ schedules no further internal event, the component R is back in its initial phase, and ready to execute a new task again.

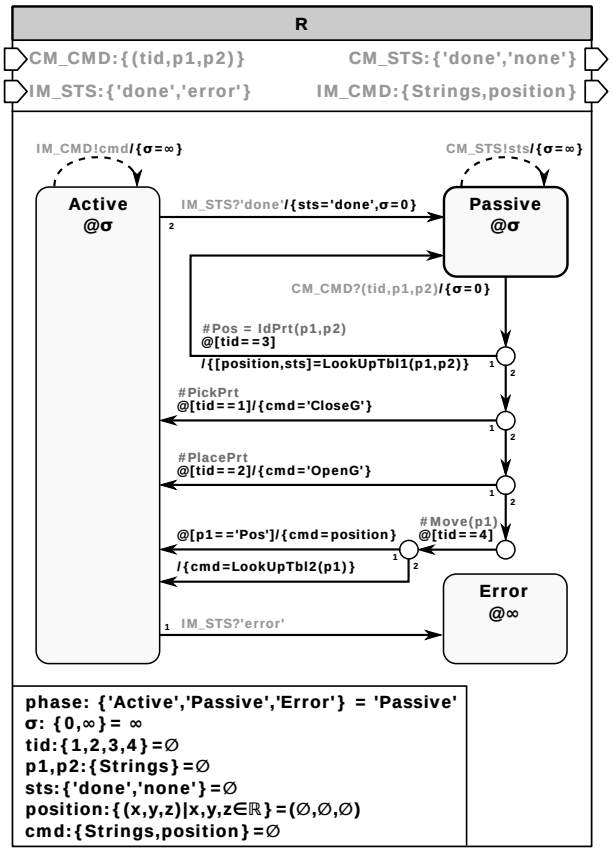


Figure 9: DEVS specification of the robot components R1 and R2 for translating the task sequences of the case study for classes 1 and 2.

The execution of the tasks *PickPrt, PlacePrt* and *Move* always leads to a transition to the phase *Active*. However, before this, the state transition caused by the external event (*CM_CMD?(tid,p1,p2)*) sets the state variable *cmd* to the task-specific value and schedules an internal event with σ=0.

For the task *Move*, a distinction is made between two transition variants. In the first case, the coordinates in the state variable *position* are assigned to the variable *cmd*. In the second case, the coordinates are determined via a lookup table. The internal event is then triggered in the phase *Active*.

This results in an output event *cmd* at port IM_CMD (*IM_CMD!cmd*), which is sent to the IM layer (see Figure 7). The R component remains in phase *Active* due to $\sigma = \infty$ until an external event, a status message from the IM layer, occurs at port IM_STS.

If a status event *'error'* (*IM_STS?'error'*) is received, the phase *Error* is entered, and an error handling routine is called, which is omitted in Figure 9. In the case of receiving a status event *'done'* (*IM_STS?'done'*), a transition to the phase *Passive* takes place, the state variable *sts='done'* is set, and with $\sigma = 0$ an internal event is scheduled. This triggers an output event *CM_STS!sts* to send the status information to the CM layer and sets $\sigma = \infty$. Now, the component R is ready to execute a new task. The transformation of the tasks *PlacePrt* and *PickPrt* are carried out in the same way.

3.3 Interface to the Robot Middleware

DEVS models require an interface to interact with real processes. In the DEVS-RCP-V2 formalism according to Freymann [4], activities are introduced for this purpose. According to Zeigler et al. [7], activities are Function Specified System (FNSS). An activity is characterized by a start and end event and has a duration. Permissible time windows can be defined for the execution times of activities. During the execution of activities, status queries can be made about the process.

In DEVS diagrams, activities are defined in the bottom right-hand field. For the interface component, an evaluation order of the internal state transitions (dashed lines) must be specified. The order is set with priorities, with value 1 as the highest priority. Otherwise, the notation is identical to the previous diagrams. The following explanation of the DEVS specification focuses on the special features with regard to activities.

Figure 10 shows the DEVS specification of the component INTF, which acts as interface between a robot component R in the PM layer and the robot midleware (see Figure 7). Like R, INTF defines the three phases *Passive, Active* and *Error*, with the initial phase *Passive*. In the phase *Passive*, INTF can receive input events from R and converts these into process interactions. This means that INTF must start and monitor activities and sends status messages back to R as output events.

INTF defines three activities. Two activities are based on application programming interface functions of the robot middleware.

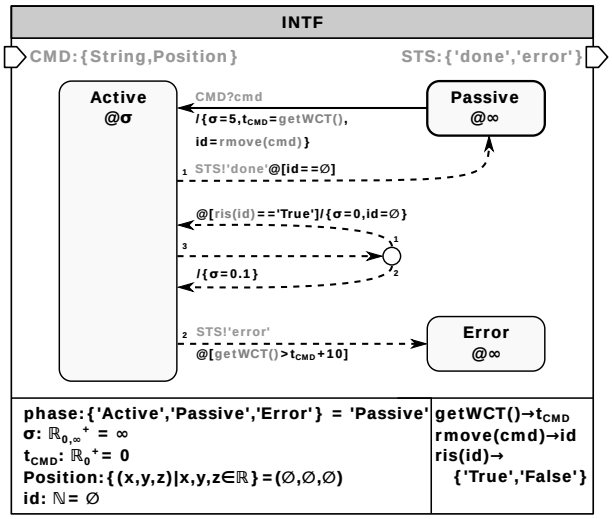


Figure 10: DEVS specification for the robot interface components INTF1 and INTF2.

The activity *rmove* starts a robot or gripper movement and returns an *id* as an identifyer. This *id* can be used with the activity *ris* to determine whether the movement has been completed or not. The activity *getWCT* provides the wall-clock-time (WCT) and is used to monitor the execution time of a movement activity.

When a movement activity starts, i.e. a state transition from *Passive* to *Active* is triggered by an external event *CMD?cmd*, the WCT is written to the state variable t_{CMD}. It is assumed that a movement takes at least 5 seconds (σ=5) and is completed after a maximum of 10 seconds, which is defined by the condition $@[getWCT > t_{CMD} + 10]$. If the upper time limit is exceeded, a transition from phase *Active* to *Error* occurs and an output event *'error'* is sent to the PM layer via port STS (*STS!'error'*).

The sampling time for status queries to the process with the activity *ris* is set to 0.1 seconds (σ=0.1). If the *ris* activity returns the status *'True'* in the permitted time interval, the condition $@[ris(id) == \text{'True'}]$ applies.

An immediate internal event is scheduled with σ=0. This means that the output event *STS!done* is triggered immediately in phase *Active* and INTF switches to phase *Passive* with $\sigma = \infty$.

For real-time synchronization, DEVS-RCP-V2 defines an RTC component on the IM layer (Figure 7). The DEVS specification of RTC can be found in Freymann [4].

3.4 Transformation of Tasks with Interactions

The interaction class 3 in Figure 5 defines that the robots move into a common workspace. Access to the IB is critical and must be *coordinated* by mutual exclusion (mutex). Mutex has been discussed in Subsection 2.3 and the concept of a related TOC structure is shown in Figure 7. The coordination of the robots via Mutex is implemented in the RT component on the PM layer.

The robot components R on the PM layer must be adapted to communicate with RT. The tasks *Lock* and *UnLock* on the CM layer can be specified as described in Section 3.1.

Figure 11 shows the DEVS specification of the adapted component R. Communication with RT takes place via the input/output ports RT_IN and RT_OUT. In addition, a new phase *PassiveRT* is introduced. An external event *CM_CMD?(tid,p1,p2)* with *tid=-1* or *tid = -2* encodes the execution of the newly introduced tasks *Lock* or *UnLock*. $\sigma = 0$ schedules an internal event and R switches to phase *PassiveRT*. In *PassiveRT* an output event *RT_OUT!(tid,rid,p1)* is triggered and send to the RT component.

The event contains the task ID in *tid*, the robot ID in *rid* and the identifier of the resource to be locked or unlocked in the parameter *p1*. Due to $\sigma = \infty$, R remains in the phase *PassiveRT* until a *'done'* event is received from RT, which causes the transition of R to the phase *Passive*.

Here, R schedules an internal event ($\sigma = 0$) that triggers an output event *CM_STS!sts* with *sts='done'* to the parent CM layer. R waits in phase *Passive* ($\sigma = \infty$).

Figure 12 shows the specification of component RT.[1] It processes incoming events from the robots R (R1_IN?(tid1,rid1,p1) or R2_IN?(tid2,rid2,p2)) and responds to them with a *'done'* event if a task can be executed.

Robots R can simultaneously request the reservation of a resource from RT. In this case, robot R1 is given priority because its reservation request is checked first. The reservation is made by means of a resource identifier, such as *IB* in the transportation problem.

The information about which robot reserves which resource is stored in a resource list *RcsList*. A resource can only be added to the list if it does not already exist in it.

[1] The specification of RT is incorrect in Freymann [4].

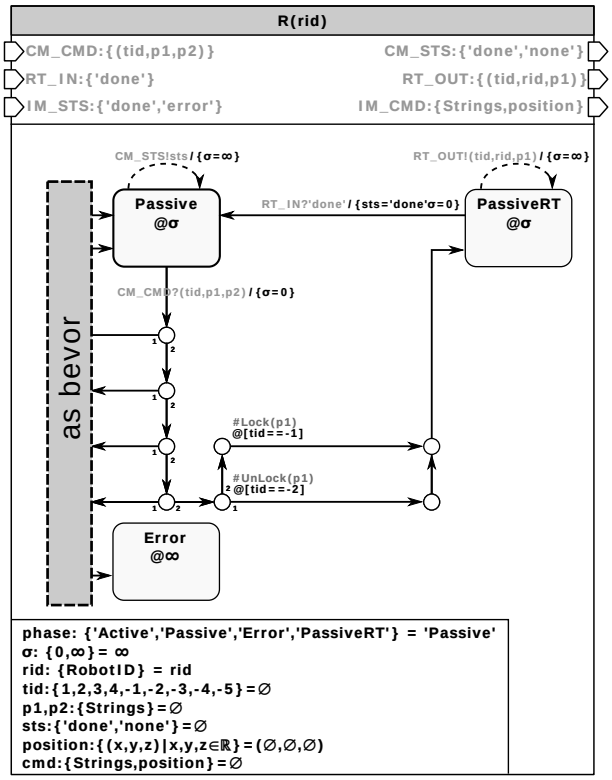


Figure 11: Extension of the DEVS specification for R to support coordination (interaction class 3)

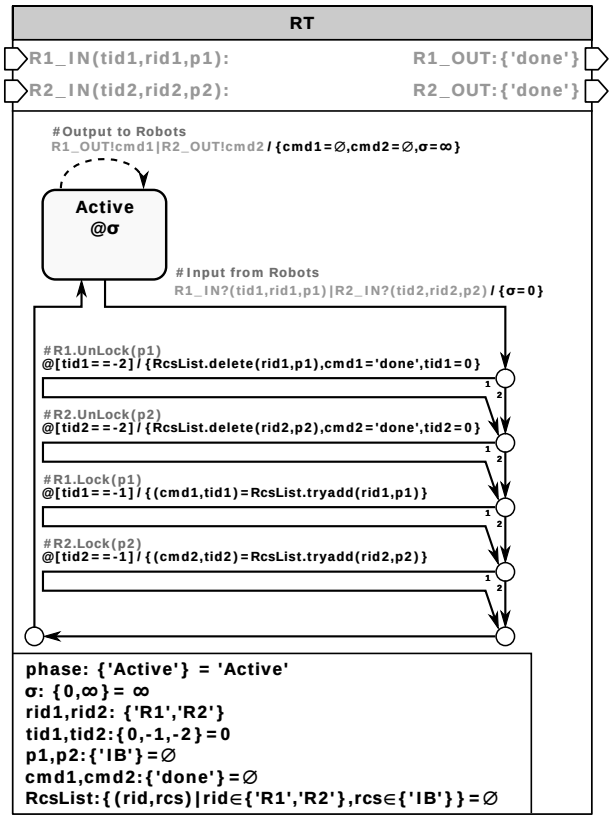


Figure 12: DEVS specification of RT component for interaction class 3

(cmd,tid) = **RcsList.tryadd**(rid,p)
if p is not in RcsList
    cmd = 'done'; tid = 0; RcsList.add(rid,p);
else
    cmd=∅; tid = -1;
with:
    cmd ∈ {'done'}, tid ∈ {0,-1}
    rid ∈ {'R1','R2'}, p ∈ {'IB'}
    logging of rid within RcsList is just for debugging

A successful reservation is communicated to the robot components R via a *'done'* event. If a resource is released again its ownership can immediately pass to another robot. It follows from this that, due to concurrency, the task *UnLock* must be evaluated first. Only then is the task *Lock* evaluated.

4 Summary

It has been shown that the classification of interactions in MRS enables their systematic mapping in the form of reusable tasks. In this way, controls for MRS can be specified in a completely task-oriented manner, analogous to SRS. The task transformation takes place analogously to SRS, with the complexity increasing when interactions are added.

Using an application example, it has been shown that the SBC approach can be used to develop manufacturer-independent and consistent simulation-based controls from the early planning phase to the operational phase. In the course of implementation, generic components were derived that were consistently reused and gradually developed further. The prerequisite for the approach is appropriate robot midleware.

The DEVS formalism has been used as the basis for the SBC approach. This enabled the modular task concept to be implemented step by step and consistently. With the help of DEVS diagrams, even partially complex dynamics could be clearly represented. For the practical tests, the diagrams were implemented one-to-one in a DEVS based MATLAB environment.

Interactions between robots and humans and with the environment were not taken into account. However, the work can serve as a blueprint. The implementation of the other interaction classes will be presented in a follow-up article.

References

[1] Abel D, Bollig A. *Rapid Control Prototyping, Methoden und Anwendung*. Springer. 2006.

[2] Siciliano B, ed. *Springer handbook of robotics*. Berlin: Springer. 2008.

[3] Maletzki G. Rapid control prototyping of complex and flexible robot controls based on the SBC approach (in German). Dissertation, Univ. of Rostock & Wismar Univ. of Appl. Sciences. 2014.
Advances in Simulation Vol. 25.
DOI: 10.11128/fbs.25

[4] Freymann B. Task-oriented multi-robot controls based on the SBC framework and DEVS (in German). Dissertation, Technical Univ. Clausthal & Wismar Univ. of Appl. Sciences. 2022.
Advances in Simulation Vol. 40.
DOI: 10.11128/fbs.40

[5] Maletzki G, Pawletta T, Dünow P, Manemann P. Simulation-based control of a robot cell (in German). *Frontiers in Simulation - Simulationstechnique 18th Symposium, Erlangen, Germany*. 2005;pp. 305–310.

[6] Schwatinski T, Pawletta T, Pawletta S. Flexible Task Oriented Robot Controls Using the System Entity Structure and Model Base Approach. *SNE Simulation Notes Europe*. 2012;22(2):107–114.
DOI 10.11128/sne.22.tn.10135

[7] Zeigler BP, Muzy A, Kofman E. *Theory of Modeling and Simulation. 3rd Edition*. Academic Press. 2019.

[8] Deatcu C, Freymann B, Schmidt A, Pawletta T. MATLAB/Simulink Based Rapid Control Prototyping for Multivendor Robot Applications. *SNE - Simulation Notes Europe Journa*. 2015;25(2):69–78.
DOI 10.11128/sne.25.tn.10293

[9] Lüth T, Längle T. Multi-agent systems in robotics and artificial life (in German). In: *GMD Workshop Artificial Life*. Sankt Augustin, Germany. 1995; pp. 1–10.

[10] Tanenbaum AS, Bos H. *Modern Operating Systems*. Pearson, 5th ed. 2022.