Modeling and Simulation of Task-oriented Multi-Robot Applications with MATLAB/Stateflow

Birger Freymann, Sven Pawletta, Artur Schmidt, Thorsten Pawletta
Hochschule Wismar,
Research Group Computational Engineering and Automation
birger.freymann@hs-wismar.de

Abstract

Robot programming software is mostly proprietary and cannot be used for other manufacturers' robots. Nevertheless, there is a desire to allow interactions between robots being developed by different manufacturers in order to set up a *Multi-Robot System* (MRS). An MRS refers to a team consisting of interacting industrial robots which share skills to increase performance. The mapping of classical control development methods for *Single-Robot Systems* (SRSs) to MRSs is difficult. Based on a classification of interactions within MRSs, a *Task-Oriented Control* (TOC) approach is suggested and examined. Furthermore, with *Simulation Based Control* (SBC) an approach for continuous development of event-driven multi-robot controls according to the *Rapid Control Prototyping* (RCP) is presented. SBC supports TOC and the mapping of interactions via tasks. Based on SBC and MATLAB/Stateflow a prototypical, task-oriented model library for interacting robots is developed and tested.

1 Introduction

New fields for applied robotics are made up continuously, something which increases the need for research. For example, current research focus on mobile robotics, service robotics and human-robot interaction is referred to as *Collaborative Robots* (Cobots) [1]. In manufacturing, industrial robots are often used as flexible handling and production units.

In the context of Industry 4.0, there is a desire for intelligent and networking factories [2]. Among other things, these are characterized by a flexible and versatile production of individual parts up to mass production. A *Single-Robot System* (SRS) mostly does not have sufficient skills to meet these requirements. Therefore, *Multi-Robot Systems* (MRSs) are proposed as a key technology. In an MRS, the robot systems which are involved form a team by sharing skills in order to enhance performance. In this work, an MRS consists of commercially available SRSs with a shared central control computer.

In contrast to an SRS the involved robot systems for an MRS always influence each other, which is called interaction. One motivation for the use of an MRS is to increase the efficiency

of the robot team, thus reducing the production time and costs. Nevertheless, the use of an MRS presents new challenges, because classical industrial robots and their software tools are usually not designed to be used in a team.

For an SRS, different design methods for control development are known [3,4] but the transfer to an MRS is difficult [5]. In particular, the use of robots from different manufactures represents a major challenge. Industrial robots are mostly programmed in proprietary development environments, which do not support third party robots. Some of these environments are already supporting a control development for MRSs, but usually only for small team sizes, a limited number of predefined interactions and exclusively for robots of the particular manufacture. Also the integration of sensors and actuators is usually restricted to vendor specific products.

Proprietary development environments are based on vendor-specific robot programming languages. Attempts to standardize robotic programming languages, such as the *Industrial Robot Language* (IRL) and its successor, the *Programming Language for Robots* (PLR), are usually ignored by robot manufactures. However, the ongoing spread of the *Robot Operating System* (ROS) [6], an open source project to provide cross-vendor robot programming and process visualization, shows that there is an unbroken interest in unified development methods and environments.

In the context of control development, different demands exist [7]. In this work the following issues will be considered in more detail: (i) flexible adaptation to new problems and quick commissioning (ii) testability, maintainability (iii) cross-vendor interactions between robots and (iv) support of vendor-independent hardware and software.

In order to meet these demands, the software and conceptual foundation are presented. At first a non-vendor-specific control of industrial robots is shown, using the *Robotic Control & Visualization* (RCV) Toolbox for MATLAB as middleware with associated interface. Then the concept of *Task-Oriented Control* (TOC) development [4,8] is described for an SRS. For MRS control development, the *Simulation Based Control* (SBC) procedure model is proposed which follows the *Rapid Control Prototyping* (RCP) approach by Abel [9]. SBC enables continuous control development from the early planning phase to the operational control use and defines a framework for practical implementation. In addition, the investigations in [10–12] show that the SBC approach supports TOC. Furthermore a classification of interactions between industrial robots is introduced based on [13]. Based on a case study, the mapping of interactions to tasks is examined. The implementation of the SBC approach is realized in the MATLAB/Stateflow environment.

2 Fundamentals

In this section, the necessary basics for the development of a *Multi-Robot System* (MRS) control are explained. The methods described and the tools are already used for *Single-Robot Systems* (SRSs) and will be applied to MRSs.

2.1 Robotic Control & Visualization Toolbox for MATLAB/Simulink

As mentioned before, the programming of robotic systems is affected by vendor dependent languages and proprietary development environments. Existing standards are mostly ignored by manufacturers, which makes control development for multi-robot controls difficult or even impossible. In engineering, control development is often associated with *Scientific and Technical Computing Environments* (SCEs), such as MATLAB. To close the gap between vendor-specific robotic environments and common SCEs, the *Robotic Control & Visualization* (RCV) Toolbox for MATLAB/Simulink was developed [14,15]. The RCV Toolbox consists of three modules: (i) a set of generalized robot-oriented MATLAB functions, (ii) a set of MATLAB functions for process visualization and (iii) robot-specific command interpreters. The basic structure of an RCV based control is shown in Figure 1. The *Control PC* runs a MATLAB instance which processes the control program on the basis of functions of the module (i). The *Visualization PC* runs another instance of MATLAB and visualizes the process based on functions of the module (ii). The communication with real robots is performed by the robot controllers and specific command interpreters according to the module (iii). A detailed description of the RCV Toolbox can be found in [16].

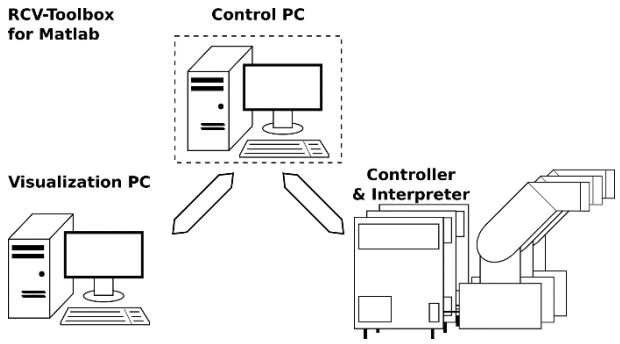

Figure 1: Principle structure of an RCV-based multi-robot application

It should be noted that the use of an SCE, such as MATLAB, simplifies the integration of additional sensors and actuators to the control.

2.2 Task-Oriented Control Design

Task-Oriented Control (TOC) design [4,8] is already known for SRSs [17,18]. The principle is to divide the complex control problems into a set of tasks and their couplings. Tasks are logical, mostly independent, abstract operations. Once identified, tasks are coupled together to map the control problem. Couplings can be carried out sequentially, conditionally or in a loop. Some problem descriptions mean it is necessary to use tasks multiple times or to work them off in parallel. The general procedure corresponds to the human way of thinking in solving complex problems.

Figure 2 shows the TOC approach on the example of a transportation problem. The problem can be divided (decomposition) into the two tasks PickPart and PlacePart. The task *PickPart* specifies the grasping of a component at a defined position and the task *PlacePart* corresponds

to placement at a new location. The task *PickPart* has to be executed before the task *PlacePart*, which corresponds to a serial execution. Furthermore, the figure shows that a task can be composed of other (sub-) tasks. For example, the task *PickPart* can be described by the serial linking of the tasks *Move* (M), *GripperAction* (G) and *Move* (M). According to [10], TOCs can be realized using *Top Down* (decomposition) or *Bottom Up* (composition) principles.

A TOC specification is not directly executable, because tasks are an abstract description of operations. A task describes '*what*' but not '*how*' something has to be solved. To perform tasks, a transformation method is required, as shown schematically in Figure 3. All tasks have to be transformed into control commands in order to execute them. To perform the transformation it is either possible to use a world model or a process model.

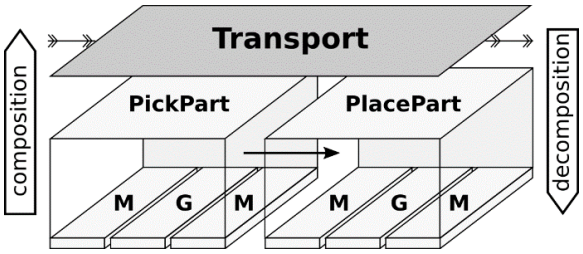

Figure 2: Tasks and subtasks of transport problem

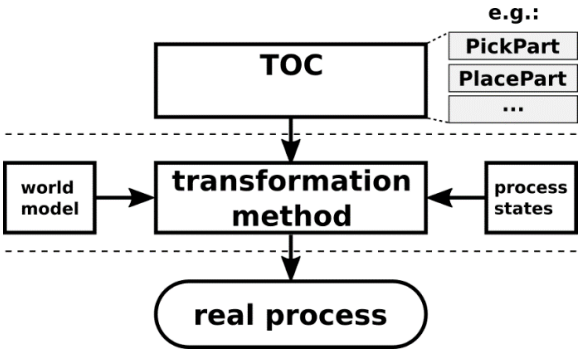

Figure 3: Processing of a TOC according to [8]

Early TOCs used a compilation approach which produces an executable control program in advance. The transformation was carried out before the commissioning by using a world model. More recent works [8,17] understand the transformation step as a function of the operational control. This has the advantage that current process states can be used instead of the world model. Thus, reactive controls can be realized, which can flexibly adapt to current process states and, for example, respond to faults by alternative processing of individual tasks. Here, the task transformation has to keep pace with the process dynamics.

Another extension of classical TOC is parameterized tasks. These allow the grouping of similar tasks into a common task. Thus, the reusability in repositories increases which also reduces development times, costs and errors.

2.3 Simulation Based Control Approach

The *Simulation Based Control* (SBC) approach [11] is a methodology for control development and defines a framework for practical implementation. The main goal of SBC is to provide a systematic way for *Rapid Control Prototyping* (RCP) [9].

Accordingly, a gradual and consistent model-based control development from the early planning phase to real operational use is supported (Figure 4). The control designs can be successively tested by simulation and supplemented by additional requirements. To enable this form of step-wise control development, an appropriate software environment or tool chain is required.

The consequent usage of simulation in all phases of the development process is key for detecting design errors as early as possible. The *Simulation Models* (SMs) are step-wise refined. During the transition from planning to the automation phase, a separation of the SM into a *Control Model* (CM) and a *Process Model* (PM) is introduced. Within the automation phase the SM is completed by an *Interface Model* (IM). In this stage the SM can be used for system simulation (in conjunction with an appropriate physical process model), or for *Software-in-the-Loop* (SiL) simulation.

During the transition from the automation to the operational phase, the SM becomes real *Control Software* (CS). This transition is usually known as *Code Generation*. Depending on the real-time requirements of the control application, the SBC approach distinguishes between explicit and implicit code generation. The first type is the classical method for high real-time requirements in conjunction with mostly embedded controller hardware. In this case, the explicit code generation is done by an appropriate compiler. For applications with rather slow timing, implicit code generation is suitable. That means the last SM of the design process is used as CS without modification. This is possible due to the IM. It provides a process interface enabling SiL but also operational use.

Following the SBC approach the CS always includes a PM. Thus, observer concepts can easily be realized. In the case of poorly or not measurable process states, the PM can provide at least estimated values. In [11,17] the SBC-Framework has been successfully applied for SRS control development for different manufacturers. Furthermore, it is shown that SBC supports a task-oriented control specification (Figure 5).

In this case, the CM contains the task-oriented control specification. The PM still has a component-oriented structure according to the elements of the real process. With a task-oriented CM on top, the PM is also the place of task transformation as described in Section 1.2. The tasks are mapped onto the generalized commands of the RCV Toolbox (Section 1.1). Thus, the PM remains independent according to concrete robots. The vendor-specific mapping is subject of the IM.

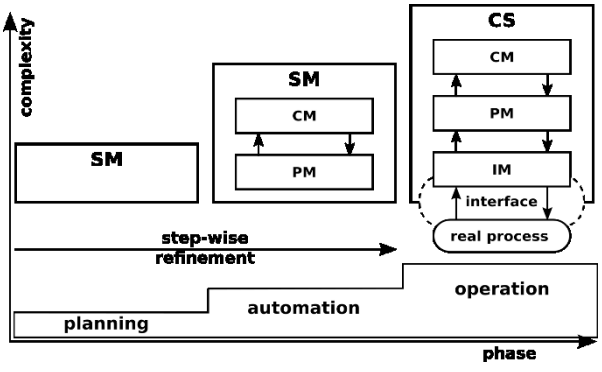

Figure 4: The SBC approach

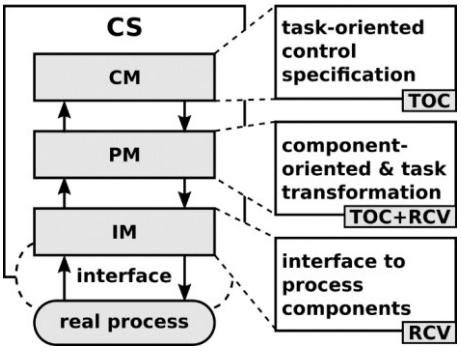

Figure 5: Use of TOC and RCV Toolbox within the SBC Framework

Below, the mapping of interactions in MRS to a task-based control specification using the SBC framework is shown after, the term interaction is discussed.

3 Interactions in MRS

In the previous sections, methods and tools for control design of *Single-Robot Systems* (SRSs) were introduced, which should be applied to *Multi-Robot Systems* (MRSs). A fundamental element of an MRS is interactions. This section treats characteristics and classification of interactions in MRSs.

3.1 Discussion

In MRSs, the robots usually support each other in order to improve the team performance. The term interaction describes the mutual influence of the robots. An interaction often requires the sharing of a limited resource, such as a robot's workspace (Figure 6). For example, if two robots need to pass a workpiece, their workspaces must overlap. This implies that the robots have to coordinate the handover to avoid a collision. One possibility for realizing the coordination is to exchange information between the robot systems.

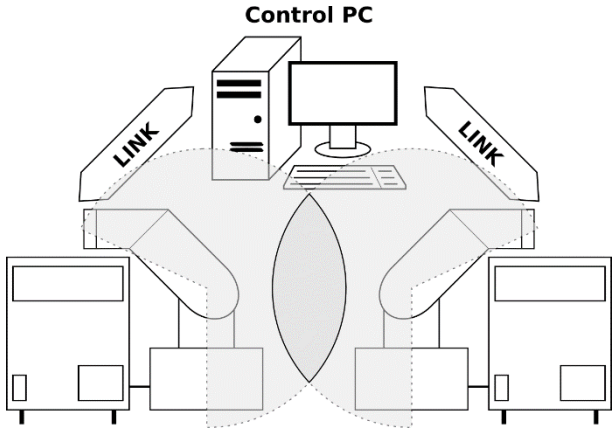

Figure 6: Example of an MRS with two industrial robots

3.2 Classification

Based on the general classification in [13], interactions of industrial robots can be divided into six classes. Figure 7 shows the example of a transportation problem for better illustration. Parts have to be transported from an *Input Buffer* to an *Output Buffer*. *Class 0* starts with an SRS and hence involves no interaction. *Class 1* to *Class 6* solve the same problem with an MRS. The complexity of the transportation problem can increase from one class to another class. For example, new part types are introduced with special requirements. To solve the problem, the necessary level of interaction will increase too.

Class 0. One type of part has to be transported by an SRS. The robot's tool is adapted to the part. The task is completed when all parts have been transported from the *Input Buffer* to the *Output Buffer*. There is no interaction.

Class 1. An MRS consisting of two robots (R_1, R_2) with separate workspaces. Both robots have identical tools. No exchange of information between the robots is required. The interaction refers to the collective solution of a problem by two or more robots.

Class 2. As for *Class 1*, supplemented by a new type of part, which requires the exchange of one robot tool. Regarding the interaction there is no change compared to *Class 1*. The robots still solve one problem and have separate workspaces. Therefore, *Class 1* and *Class 2* have the same degree of interaction.

Class 3. As for *Class 2*, but the robots share a common workspace. To avoid collisions the robots have to coordinate their motions. Therefore, the degree of interaction is increased compared with *Classes 1 & 2*.

Class 4. As for *Class 3*, but another type of part leads to a stronger interaction. The new type can only be moved by the two robots together. Thus, the degree of interaction is increased compared to *Class 3*.

Class 5. One robot supports the other one, even if the tool is not ideal for this purpose. This form of interaction is used to compensate for the overload of one robot by irregular arrival of parts.

Class 6. A new type of part which cannot be handled by the robot team requires the replacement of a robot or the tool. Interaction refers to the modification of team members.

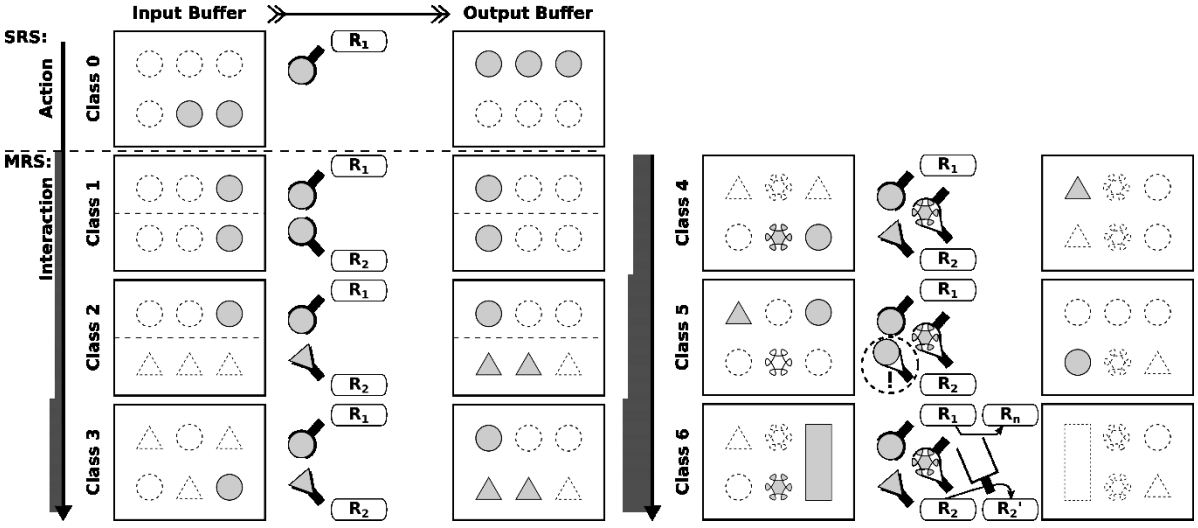

Figure 7: Classification of interacting robots based on [13] – with level of interaction

4 Case Study

In the following, the control development for a *Multi-Robot System* (MRS) is examined by a handling and assembly problem. Different interactions are implemented using a task-oriented control design. Based on the above described methods and tools, the control will be developed in the MATLAB/Stateflow environment.

4.1 Layout and Workflow

The experimental setup is illustrated in Figure 8. It consists of two robots (R_1, R_2) and two cameras (C_1, C_2). Each robot has a separate *Input* (I_1, I_2) and *Output Buffer* (O_1, O_2) in its workspace. Furthermore, both robots share a portion of the workspace by a *Common Output Buffer* O_C. The two *Input Buffers* are externally stocked up with parts of type A or B.

Each robot randomly picks up a part of its respective *Input Buffer*. The position of the parts is assumed to be known. The gripped part can either be of type A or B which makes

identification by a corresponding camera necessary. The result of the identification dictates how to proceed with the part. If the gripped part is of type A, it can be directly placed in the *Output Buffer* of the robot. Then the robot picks up another part from the *Input Buffer* and the procedure is repeated until a part of type B is identified. Two parts of type B always have to be assembled with each other by both robots before they are stored in O_C.

Figure 9 shows the subtasks which have to be done to solve the task *CommonPlacePart* (CPP). If a robot picks up a part of type B, he moves to a mounting position above O_c and is blocked then (field a) until the second robot also picks up a part of type B and reaches its mounting position (field b). Both robots proceed with a synchronous motion (field c) in order to place both parts in O_C (click mounting, field d). Subsequently, a new cycle takes place

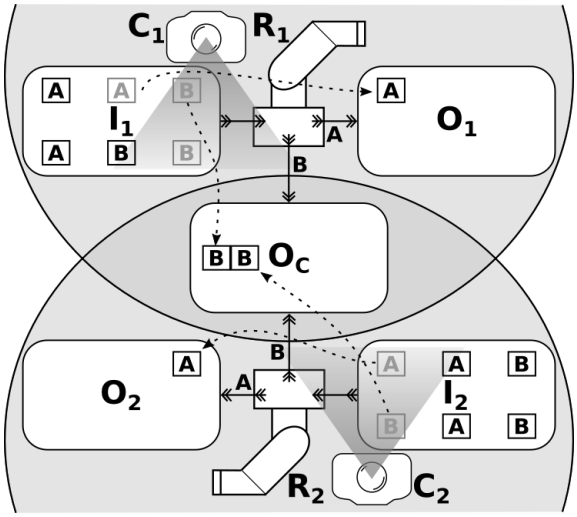

Figure 8: Experimental setup of the case study

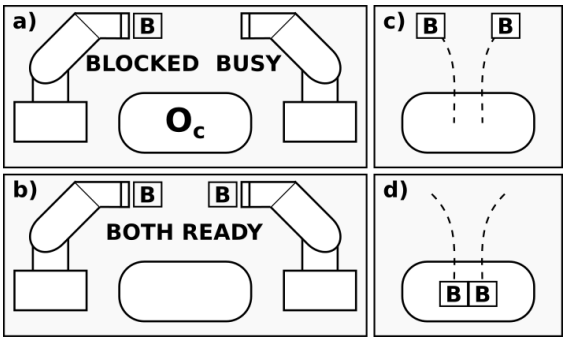

Figure 9: Workflow of two robots doing task *CommonPlacePart* (CPP)

4.2 Task-oriented Analysis

A task-oriented description of the workflow is shown in Figure 10. The notation is based on *Harel statecharts*. The figure shows the first step of the task-oriented control design. The problem description is done using the tasks *PickPart*, *IndentPart*, *PlacePart*, *MoveToPos* and *CommonPlacePart* (CPP). The control logic is identical for both robots and can be mapped by two identical parallel states (AND states), drawn as dashed lines.

As long as no part of type B is identified, both robots work independently and in parallel. The task sequence is: (i) grip a part (*PickPart*), (ii) identify the gripped part with the camera (*IndentPart*) and (iii) place the part in the *Output Buffer* (*PlacePart*).

If a part of type B is identified, the task sequence changes. Instead of *PlacePart*, the task *MoveToPos* will be executed. In consequence, the robot moves to its mounting position above O_C. The subsequent task *CPP* blocks the robot (Figure 9a) until the second robot also executes the task *CPP* (Figure 9b). Both robots coordinate the motions until the task *CPP* ends (Figure 9c, d).

The independent handling of type A parts by a sequence of tasks corresponds to a *Class 1* interaction. The interaction can be modeled by parallel execution of task sequences. The

mapping of the assembly process of type B parts corresponds to a *Class 4* interaction because it necessitates the coordination of both robots. The necessary time synchronization and coordination is mapped in the form of a separate task (*CPP*).

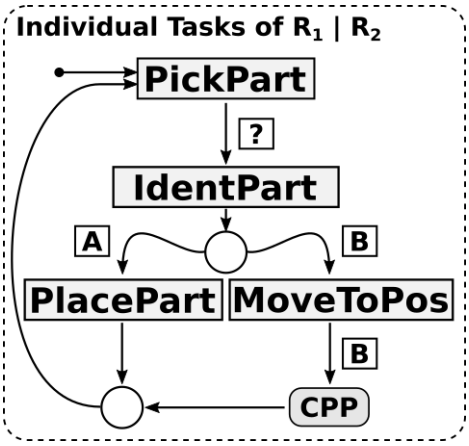

Figure 10: Task composition for both robots

4.3 Implementation Aspects

The implementation of the control is based on the *Simulation Based Control* (SBC) approach and realized in the MATLAB/Stateflow environment. In accordance with SBC, the control consists of a *Control Model* (CM), *Process Model* (PM) and *Interface Model* (IM) (Figure 4, 5). The CM shown in Figure 11 implements the *Task-Oriented Control* (TOC) logic according to Figure 10. The task sequences of both robots are implemented as parallel states. The task *CPP* in both parallel states (tasksR1, tasksR2) is signaling a standby for assembling. The implementation of the necessary interaction Figure 9c, d) was modeled as a separate task *CPP* in another parallel state. Thus, the interaction principle is mapped to a common, reusable task.

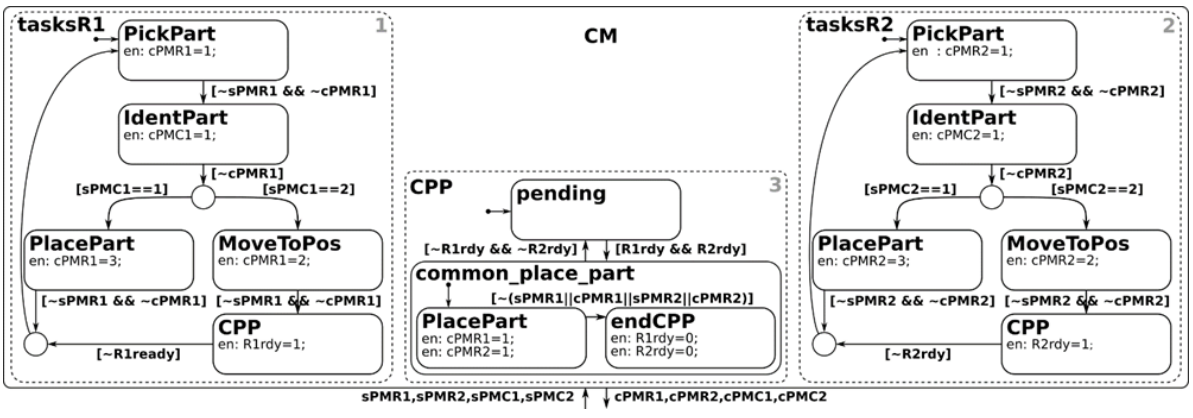

Figure 11: Statechart of CM based on SBC Framework

The PM is modeled analogously to fulfill the requirements of the SBC and the TOC and is illustrated in Figure 12. Each process component is represented as a parallel state and with two sub-states, *idle* and *execTask*. The state *idle* indicates that the component is on standby. The state *execTask* implements the process depending on task transformation. Because

the task transformation is more clearly implemented procedurally it has been transferred to a MATLAB function call. The input parameter is a numeric value and represents the decoded task.

The IM, shown in Figure 13, is realized analogously to the PM. It defines a parallel state with two sub-states for each process component, which needs to be controlled. Communication with the real process components is implemented, in each case, as a MATLAB function. To control the robot, the function *execTask( )* is used which is based on the *Robotic Control & Visualization* (RCV) Toolbox (Section 1.1). For the control of other peripherals, such as the cameras, MATLABs *Instrumental Control Toolbox* [19] can be used.

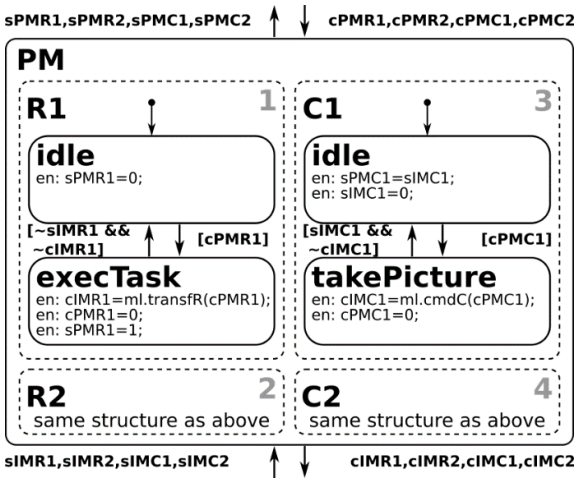

Figure 12: Statechart of PM based on SBC Framework

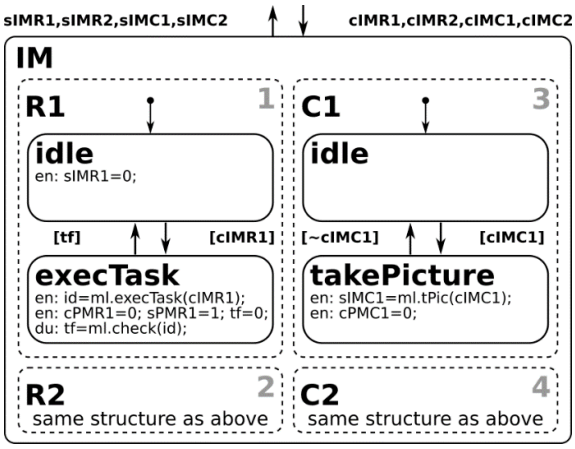

Figure 13: Statechart of IM based on SBC Framework

5 Summary and Outlook

Based on methods and tools, which are established in the control development for *Single-Robot Systems* (SRSs), an approach for *Multi-Robot Systems* (MRSs) was developed. In contrast to SRSs, interactions must be considered in MRSs. For this purpose, a characterization and classification of interactions in MRSs was introduced. An example demonstrating two interaction classes showed how interactions can be modeled as a general, reusable task, according to the task-oriented robot control paradigm.

The prototypical implementation was based on the *Simulation Based Control* (SBC) approach in the MATLAB/Stateflow environment and used the *Robotic Control & Visualization* (RCV) Toolbox for MATLAB. It was demonstrated that the control development of an MRS can be realized vendor independently, analog to an SRS and with simulative tests throughout the control development cycle.

At the beginning, defined requirements are largely fulfilled by prototypical implementation. The next step will be to examine if other identified interaction classes can be mapped to reusable tasks. Additionally, a proof of concept for a more advanced application will be provided in the future.

References

[1] Bélanger-Barrette, M. *Robotiq: Collaborative Robot EBook*, [internet] http://blog.robotiq.com/collaborative-robot-ebook (cited April 19, 2016).

[2] Hermanm M, Pentek T, Otto B. *Design Prinziples for Industie 4.0 Scenarios: A Literature Review*, [internet] http://www.snom.mb.tu-dortmund.de/cms/de/forschung/Arbeitsberichte/Design-Principles-for-Industrie-4_0-Scenarios.pdf (cited April 14, 2016).

[3] Jacak W. *Intelligent robotic systems: Design, planning and control*, Kluwer Academic, New York, London, 1999.

[4] Siciliano B. *Springer handbook of robotics*, Springer, Berlin, 2008.

[5] Makris S, Michalos G, Eytan A, Chryssolouris G. *Cooperating Robots for Reconfigurable Assembly Operations: Review and Challenges*, Procedia CIRP 3 (2012) 346–351.

[6] Open Source Robotics Foundation. *Robot Operating System*, [internet] http://www.ros.org/ (cited April 14, 2016).

[7] Meier H. *Verteilte kooperative Steuerung maschinennaher Abläufe* [dissertation]. TU München; Germany 2001

[8] Weber W. *Industrieroboter: Methoden der Steuerung und Regelung*, 2nd ed., Fachbuchverl. Leipzig im Carl-Hanser-Verl., München, 2009.

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

[10] Maletzki G. *Rapid Control Prototyping komplexer und flexibler Robotersteuerungen auf Basis des SBC-Ansatzes*. [dissertation], Rostock University; Germany. 2014.

[11] Pawletta T. Pawletta S, Maletzki G, *Integrated Modeling, Simulation and Operation of High Flexible Discrete Event Controls*, MATHMOD 09 6th Vienna Conference on Mathematical Modelling (2009).

[12] Schwatinski T, Pawletta T, Pawletta S, Kaiser C. *Simulation-based development and operation of controls on the basis of the DEVS formalism*, Prag, Czech, 2010.

[13] Lüth T, Längle T. *Multi-Agenten-Systeme in der Robotik und Artificial-Life*, in: GMD Workshop, Sankt Augustin, 1995.

[14] Christern M, Schmidt A, Schwatinski T, Pawletta T. *KUKA-KAWASAKI-Robotic Toolbox for Matlab*, 2011, [internet] https://www.mb.hs-wismar.de/cea/sw_projects.html (cited May 15, 2015).

[15] Otto J, Schwatinski T, Pawletta T. *KUKA-KAWASAKI-Visualization Toolbox for Matlab*, 2011, [internet] https://www.mb.hs-wismar.de/cea/sw_projects.html (cited May 15, 2015).

[16] Deatcu C, Freymann B, Schmidt A, Pawletta T. *MATLAB/Simulink Based Rapid Control Prototyping for Multivendor Robot Applications*, 25th ed., SNE - Simulation Notes Europe, 2015; 25(2): 69–78.

[17] 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.

[18] Pawletta T, Pawletta S, Maletzki G. *Integrated Modeling, Simulation and Operation of High Flexible Discrete Event Controls*, Argesim Report, 2010.

[19] The MathWorks. *Instrument Control Toolbox*, [internet] http://de.mathworks.com/products/instrument/ (cited April 22, 2016).