Chapter accepted for Informatics in Control, Automation and Robotics 12th International Conference, ICINCO 2015 Colmar, France, July 21-23, 2015 Revised Selected Papers

The final authenticated version is available online at https://doi.org/10.1007/978-3-319-31898-1_12

On Structure and Distribution of Software for Mobile Manipulators

Andreas Schierl, Andreas Angerer, Alwin Hoffmann, Michael Vistein, and Wolfgang Reif

Institute for Software and Systems Engineering, University of Augsburg, Germany
{schierl, angerer, hoffmann, vistein, reif}@isse.de

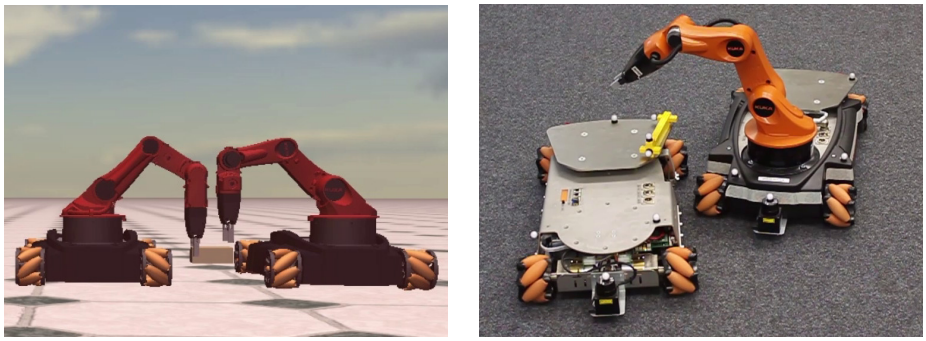
Abstract. Complex robot applications or the cooperation of multiple mobile robots are use cases of increasing popularity where software distribution becomes important. When developing mobile robot systems and applications, software structure and distribution has to be considered on various levels, with effects on the organization and exchange of data. With respect to structure and distribution, this work proposes to distinguish between real-time level, system level and application level. Ways of structuring the software, as well as advantages and disadvantages of distribution on each level are analyzed. Moreover, examples are given how this structure and distribution can be realized in the robotics frameworks OROCOS, ROS and the Robotics API. The results are demonstrated using a case study of two cooperating KUKA youBots handing over a work-piece while in motion, which is shown both in simulation and in a real world setup.

Keywords: Mobile Robots, Cooperative Manipulators, Software Distribution, Robot Architectures

1 Introduction

With service robotics getting more and more important, robot demand has extended from factory automation towards mobile robot systems. Thus, the topic of mobile robotics has become important, and a lot of research has been performed. However, in some cases a single mobile robot is not sufficient to execute a task or deal with all problems [22]. For example, Knepper et al. [20] introduced the *IkeaBot* which is a coordinated furniture assembly system with multiple KUKA youBots. Based on that work, a flexible assembly system with cooperative robots was suggested in [12]. When multiple robots work together, cooperative mobile manipulation becomes important and poses new challenges – especially to the software structure.

Already in the 1990s, Dudek et al. [13] and Cao et al. [9] described a classification for cooperative mobile robotics. Dudek et al. [13] defined a taxonomy for multi-agent mobile robotics, where such a system can be e. g. differentiated by the number of agents, the communication range, topology and bandwidth as well as the reconfigurability and the composition of homogeneous or heterogeneous



(a) Simulated scenario (b) Real world scenario

Fig. 1. Case study of two cooperating youBots

agents. Similarly, Cao et al. [9] defined research axes such as the differentiation, the communication structure or different types of modeling other agents. Moreover, they made a distinction between centralized and decentralized control in the software for mobile robots. Later, Farinelli et al. [14] added a classification based on coordination, where they compared cooperation, knowledge, coordination and organization of agents in multi-agent systems. This classification defines aware multi-robot systems where each robot knows that it is cooperating with other robots.

Concentrating on aware systems, this work analyzes which software structure and distribution can be used. However, there is not only *one* software structure and distribution that is possible in cooperative mobile robotics. From our point of view, there are different levels and aspects in the software architecture of a multi-robot system that can be distributed, ranging from the application level to low-level real-time device control. Typical early software architectures such as 3T [6] include reactive behavior as well as planning and execution, but are limited to single robot systems. The decision to distribute the software solution on one of the mentioned levels may affect the other levels and the complexity of the solution. Every possible solution, i. e. the chosen distributed software architecture, has its advantages but also its shortcomings that must be considered. For example, in current robotics systems component-based software frameworks which facilitate transparent distribution (e. g. ROS) are commonly used (cf. [7]). However, although communication between distributed components is easy using these frameworks, the decision about distribution as well as the assignment of responsibilities to certain components affects the overall capabilities of the solution (e. g. support for hard real-time).

In this work, we introduce a taxonomy for distributed software architectures in cooperative mobile robotics. However, the concepts are not specific to mobile robots, but also apply to other cooperating robots such as teams of industrial robots in automation systems. Hence, we are interested in finding a generalized representation and description of distributed robotics systems that can be used

to classify and compare software architectures of distributed robots. Additionally, we give advantages and disadvantages of applying distribution on different levels. It is important to be able to compare distributed robotics systems as the chosen software architecture often influences or sometimes even determines the complexity of the solution.

For experimental results, a case study is used where two KUKA youBots [3] physically interact with each other to transfer a work piece from one robot to the other. This scenario is inspected in different levels of difficulty. In simulation (cf. Figure 1a), both robots can be coordinated in real-time, exact position information is available and all control inputs and trajectories are exactly followed. Initially, workpiece transfer happens while the first robot is standing still, and then while both robots are moving. As some of the assumptions made for simulation are not valid for real robots, a second scenario with real youBots is analyzed (cf. Figure 1b). There, a youBot platform (left, without arm) is initially carrying a workpiece, which is then picked up by the second youBot (right) while both youBots are moving. The youBots and the workpiece are tracked externally using a Vicon optical tracking system, so precise position information is available.

In Section 2, the different levels for structuring and distribution of software for (mobile) robots is introduced. Subsequently, the identified levels (i. e. the real-time, system, and application level) are discussed in Sections 3 to 5. Implications on the world model of robotics software are addressed in Section 6. To show the general validity of the suggested taxonomy, the possibilities of structuring and distribution on each level are explained using three different robotic frameworks in Section 7. Experimental results with different possible solutions of the case study are presented in Section 8. Finally, Section 9 concludes this work and gives an outlook.

2 Different Levels of Structuring Robotics Software

When designing a software architecture for a distributed robot scenario, we propose to group the software components into different layers as illustrated in Figure 2. Each of the hardware devices present in the robot solution is represented and controlled by a *device driver* which is defined as the component that communicates with the hardware device through the vendor-specific interface. Additionally, the device driver is responsible for exchanging data with the surrounding software components. It has to derive control inputs and forward them to the device, as well as receive feedback from the device and make it available to other software components.

Each device driver can belong to a *real-time context* where data transfer and coordination between components occur with given timing guarantees. Depending on the implementation, the real-time context can contain only one device or span over multiple devices. Within a real-time context, reactions to events or the processing of sensor data can be guaranteed to happen before a given time limit. This allows to handle safety-critical situations that require timing

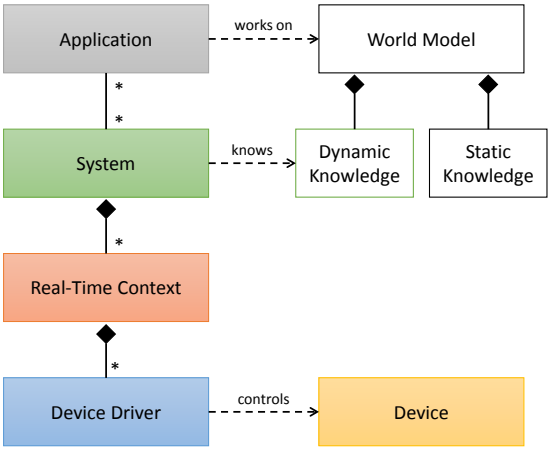


Fig. 2. Software structure for distributed robots

guarantees (e. g. to stop the robot if an obstacle occurs), or to execute precise behaviors (such as actions that happen at a given point on a trajectory).

Above the real-time level, one or multiple real-time contexts belong to a *system* where all knowledge is shared between the components. Hence, all components within one system are allowed to access each other's data, as well as to communicate with and send commands to each other. This allows components to directly include other components' data into planning or computation, however no real-time guarantees are given unless the communication is handled within one real-time context.

To perform a desired task, systems can be controlled from *applications* that coordinate the work flow. Within an application, data is read and commands are sent to the controlled systems, so that the corresponding devices execute the task. However, if data from one system is required for an action in another system, it is the responsibility of the application or the deployment to facilitate the data transfer, as there is no concept of implicit shared data between systems. The overall behavior of cooperating robots results from the interplay of all applications that coordinate the robots.

Each application performs its work based on its *world model*, i. e. the knowledge about the controlled devices and systems, as well as about the environment including other (cooperating) devices. This includes geometric information such as positions and orientations of the relevant objects, as well as also physical data (such as mass and friction), shape data (such as 3D models for visualization or collision checks), semantics and ontologies. Information from the world model can be stored and used in applications, systems or real-time contexts, and can also be shared between different applications and systems. Structurally, the world model data can be differentiated into dynamic and static knowledge, with static knowledge (e. g. maps, shapes and ontologies) being valid everywhere,

while dynamic knowledge (such as positions and sensor data) may be known in only one system or be different in different systems.

Depending on the requirements and technical limitations of the robot solution, the size and distribution of real-time contexts, systems and applications and thus the structure of the software can vary. The following sections discuss different design decisions concerning this structure based on the examples of the case study and using the three popular frameworks OROCOS, ROS and the Robotics API. OROCOS as a component framework mainly targets control systems with real-time guarantees [8]. The main focus of ROS is to be a component framework with transparent distribution, which over time has collected a large amount of algorithms as reusable components [23]. The Robotics API focuses on high-level robot programming using modern programming languages (such as Java) while still providing real-time guarantees [1].

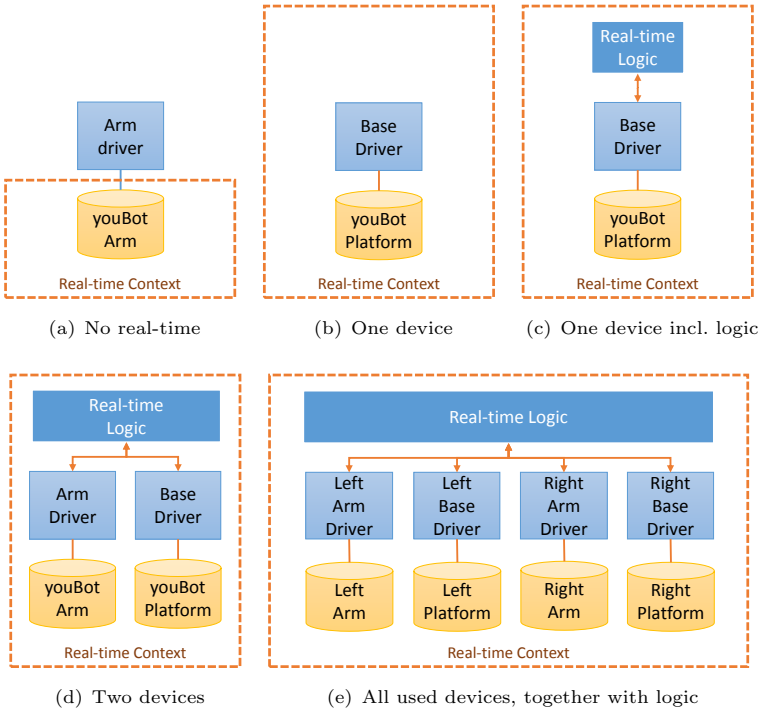
3 Real-time Level

First, the existing hardware devices and device drivers have to be grouped into one or more real-time contexts. Within a real-time context, reactions to events or the processing of sensor data can be guaranteed to happen before a given time limit. Having hard real-time guarantees allows to control precise behaviors or to handle safety-critical situations that need strict timing. In the mobile manipulator example of the case study, the available device drivers have to be grouped into real-time contexts, especially focusing on the two youBot platforms and one or two arms.

3.1 Software Structure on the Real-time Level

Generally speaking, there are five different choices to structure these devices (and their device drivers) into real-time contexts. In the first case (i. e. the real-time context in Figure 3a), the device driver software is written without real-time in mind. Here, the real-time context only spans the (possibly real-time capable) firmware or controller present in the device itself. For the youBot arm, this could mean that only the position control mode of the arm motor controllers is used. Thus, it is sufficient to give one joint configuration that the robot is expected to move to. While easy to implement, no synchronization between the joints or support for precise Cartesian space motions is possible. Moreover, no guarantees can be given regarding the interpolation quality of user-defined trajectories or the timing of reactions to events (unless supported directly by the device).

In the next case (cf. Figure 3b), the device driver and the communication with the device is implemented in a real-time capable fashion. This requires to use a real-time operating system (RTOS) and more care when implementing the device driver, but allows to execute precise custom trajectories and handle sensor events with timing guarantees. Besides the device driver, additional real-time logic (cf. Figure 3c) can be present that implements control, trajectory tracking or coordination of the device. For example, a real-time capable driver running at



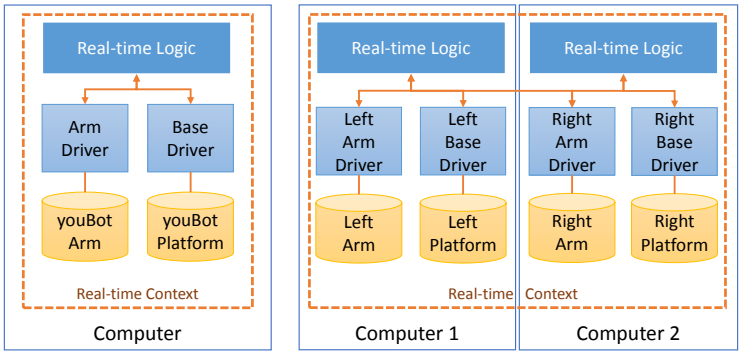
(a) No real-time (b) One device (c) One device incl. logic

(d) Two devices (e) All used devices, together with logic

Fig. 3. Variants of real-time contexts

250 Hz can be implemented for the youBot Platform on a RTOS such as VxWorks or Xenomai. It provides the motor controllers with smooth control set points for velocity or torque control, which allows precise user-defined trajectories or custom feed-forward or feedback control laws. However, as a drawback only information that is provided by the device itself can be included in the control law. For example, only the wheel position can be controlled exactly, but the position of the entire robot in Cartesian space is inaccurate (due to wheel slip and other factors limiting odometry precision), and the platform motion cannot be synchronized with the arm motion.

Increasing the real-time context, multiple devices can be combined up to all devices that are physically connected to the controlling computer (cf. Figure 3d). Both the youBot arm and the platform – connected to the onboard computer via EtherCAT – can be controlled from a real-time capable software on a RTOS. In this way, coordinated motions between platform and arm are possible by combining the five joints of the arm and the three degrees of freedom provided by the omni-directional platform. This allows, for example, to execute Cartesian space motions of the end-effector relative to a point in Cartesian space known to the youBot (such as the position where the youBot started assuming that odometry exactly provides the current position relative to this origin based on



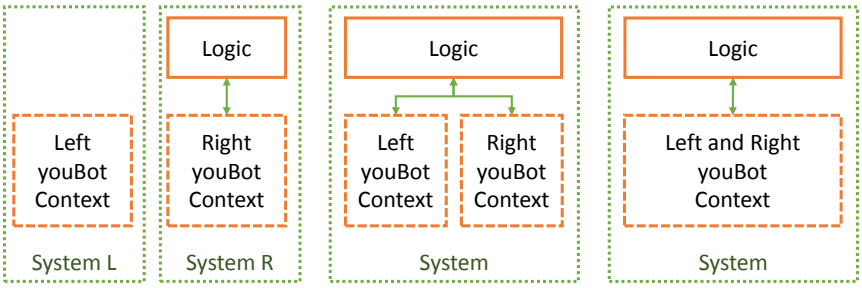
(a) No distribution (b) Distribution over two computers

Fig. 4. Examples for distributing real-time contexts

wheel rotations). Additionally, one device can react to events that occur at other devices or are detected by other sensors. However, in order to be able to specify these reactions, either this part of the real-time logic has to be changed for a specific application, or a flexible specification language is required in the real-time logic [24]. To enable easy cooperation between multiple robots, the devices of all robots could be combined into one real-time context (cf. Figure 3e). However, if the corresponding devices are connected to different PCs, real-time distribution becomes important to establish this kind of real-time context.

3.2 Distribution of Real-time Contexts

In the simplest cases, all devices structured into one real-time context are connected to the same computer (cf. Figure 4a), which requires no distribution. Regarding our case study where both youBot platforms are not connected to one computer using a single (wired) EtherCAT bus, the real-time context has to be distributed in order to achieve this structure (cf. Figure 4b). However, to distribute a real-time context, special real-time capable communication is required. For stationary robots such as manipulators or automation systems, as well as for complex mobile robots where different devices are connected to the different on-board computers (such as the PR2 or DLR's Justin), this is possible through Ethernet or a field bus like EtherCAT. In the automation domain, standard equipment such as PLCs are used, while in robot research software frameworks such as aRDx [15] or OROCOS [8] are preferred. But between mobile robots, using a wired connection usually is no option, and standards for general purpose real-time capable wireless connections are not yet common, so providing a single real-time context is not yet widely usable. In summary, while distributing a real-time context over multiple computers can improve the scalability of the solution (w. r. t. processing power or device connectivity), the need for deterministic communication implies special requirements (such as field bus hardware or dedicated networks) that make the solution more complex or expensive.



(a) Two separate systems (b) Two real-time contexts (c) One real-time context

Fig. 5. Variants of system structure

4 System Level

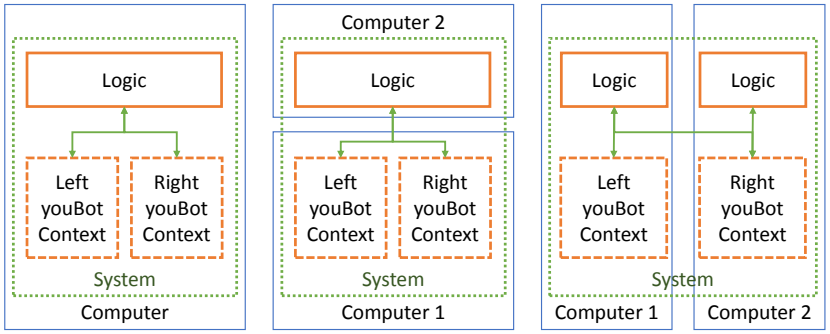
Proceeding to the system level, one or more real-time contexts can be grouped into one system. Within a system, all components are allowed to access each other's data, and to communicate with and send commands to each other. This allows components to directly include other components' data into planning or computation, though no real-time guarantees are given (unless handled within one real-time context).

4.1 Software Structure on the System Level

Working with the two youBots, three different structures of systems are possible: As shown in Figure 5a, each youBot can work in its own system optionally together with further logic used in the system. Then, no implicit data transfer occurs between both youBots. The second option is to group both real-time contexts for the two youBots into one system (cf. Figure 5b), in addition with further computation logic. Then, the system introduces communication between both real-time contexts that however does not provide real-time guarantees. Finally, one real-time context spanning both youBots can be used in a system (cf. Figure 5c), which allows real-time cooperation of the youBots. However, this may require a distribution of the real-time context.

Using a big system spanning all robots (cf. Figures 5b and 5c) has the advantage of simplifying application programming or deployment: All the data that any component might need is made available everywhere in the system. Hence, no manual data transfer is required. This especially covers the world model. Within one system, a consistent world model is possible, because the best knowledge about the world is available to every component. Moreover, every change to the common world model is available to every component immediately.

However, there can be various reasons to use multiple systems: The sheer amount of data present in a big (multi) robot system can be a technical reason. Scalability can be limited by the management overhead induced by the data transfer between a great amount of components, and the addressing or mapping



(a) No distribution (b) Separate logic (c) Distributed devices

Fig. 6. Examples for distributing systems

of data to components can become problematic. Furthermore, network bandwidth or reliability can be a limiting factor. In particular, this can be a problem when multiple robots are used that cooperate in varying teams. While for constant teams the corresponding robots could be joined into one system, varying teams quickly increase the required system size as all robots that might work together in a team at any time have to be within the same system.

But also more political reasons can opt for the separation into multiple systems, if cooperating robots belong to different people or parties. In this situation, not everyone might want to provide access to all of the robot's data, or allow everyone else to control the robot. Then, matters of trust or access control become important that do not fit into the share-everything theme of a system. However, these reasons do not occur between the different devices within one robot (such as the arm and the platform of one youBot), so grouping them into two different systems does not really make sense and has been omitted in the figure.

4.2 Distribution of Systems

Looking at the distribution aspect of a system, some constraints are given by the real-time contexts: As the real-time contexts have to communicate with the corresponding hardware devices, the assignment to a certain computer is usually given. Depending on the devices and their connectivity, this can lead to a solution without distribution (cf. Figure 6a), if all devices are connected to the same computer. In this case however, further logic components can be moved to a different computer as shown in Figure 6b, based on performance deliberations. When the devices are connected to different computers and are not handled within a distributed real-time context, distribution of the system is required. Here, each real-time context's assignment is given, while the further logic components can be assigned to the different computers based on further requirements (cf. Figure 6c). For communication between the different components and real-time contexts, no timing guarantees are required. When data is to be exchanged

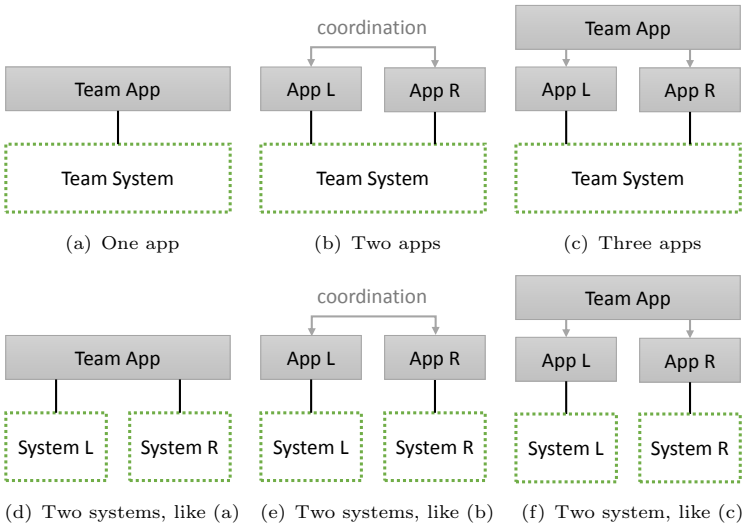


Fig. 7. Variants of application structure

between components on the same computer, communication can be performed locally, while otherwise network connections can be used to transfer the corresponding data and make necessary information available. Therefore, standard communication methods including wireless ones such as WiFi are applicable, however reliability or bandwidth can be limiting factors.

5 Application Level

To perform a requested task, one or multiple systems have to execute actions that are controlled and coordinated by one or multiple applications. There are various ways to specify applications: Mainly sequential workflows can well be expressed as a programming language control flow. For reactive behavior, model-based approaches such as statecharts (e. g. [1], [19] and [27]) or Petri nets (e. g. [11]) can be more appropriate. Solutions offered by different robotics frameworks are discussed in Section 7.

5.1 Software Structure on the Application Level

Figure 7 gives the different possibilities to structure the application(s) for controlling two youBots. One way is to control all robots from one application as shown in Figure 7a. This defines all the interaction present in the solution in one place and thus makes it easier to understand. However, if varying teams are needed in a certain scenario, the corresponding application has to coordinate all robots at the same time. This can become confusing if the concurrent execution of multiple subtasks is encoded in one control flow or sequential state machine.

Thus, separating concerns into subtasks, one for each team, should be considered within the application.

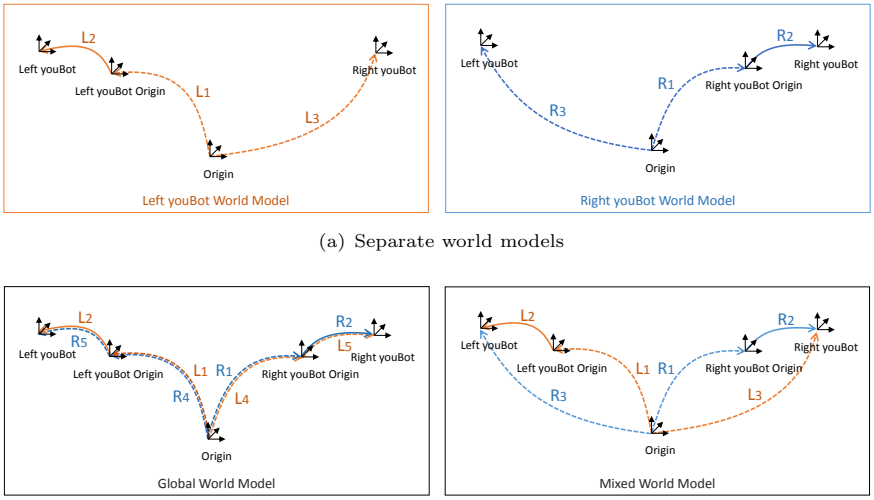
Another way is to use multiple applications, e. g. one for each controlled robot (cf. Figure 7b). In this way, team behavior can be implemented by only locally and independently describing the behavior of each robot. However, the applications have to coordinate, either through explicit communication or through observation of the environment or other robots. As a third way, a further application coordinating both applications is a possible solution (cf. Figure 7c), which however leads to a coordination application similar to the one in Figure 7a. Using separate applications can also be required for political reasons, as described in Section 4.

In a multi-application cooperation scheme, however, the resulting behavior is not easily understandable by looking at one place, but only by examining the interaction of all different applications involved. In multi-agent robot systems (e. g. [2] and [10]) every agent is controlled by one application. The overall behavior is given by the agents' interaction. As shown by Hoffmann et al. [17], this can lead to self-organizing properties making the overall solution more robust. It is even possible that the application for each robot only implements low-level behaviors, and the resulting behavior completely emerges from the interaction [21].

Another structuring approach looks at the relation between applications and systems. It is possible to control (different devices of) one system from multiple applications (cf. Figures 7b and 7c), one system from one application (cf. Figures 7a and 7e) or multiple systems from one application (cf. Figure 7d). The software framework should allow an application that was intended for use with multiple systems (e. g. Figure 7d) to also work when all corresponding devices exist in one system, while the distribution aspect on the system and real-time level is handled by deployment (e. g. through configuration). This may however not be possible the other way round, if the application relies on having one system (or even real-time context).

5.2 Distribution of Applications

When looking at the application level, different possible variants of distribution can occur. An application can run on another computer than the controlled system(s). In this case, network communication is required to transfer data between the system(s) and the application. However, this kind of distribution is equivalent to distribution on the system level, when seeing the application as a component in the system. When multiple applications are used, they can be executed on different computers. Then, coordination between the applications has to be implemented in a way supporting remote communication. Finally, a single application can be distributed onto multiple computers. Therefore, programming language concepts such as remote procedure calls or service-oriented architectures can be used, which can however be used in a standard fashion and do not require any special, robotics-related treatment.



(a) Separate world models

(b) Global world model

(c) Mixed world model

Fig. 8. Different options to represent the world model of multiple systems.

6 World Model

If a robot is expected to cooperate with another robot, it needs the relevant data about that other robot. In the easiest case, both robots belong to the same system, thus the required information is already accessible. Then we refer to both robots as *controlled robots* within this system. Once robots from multiple systems are to be coordinated from one application, the world model becomes more complex. The information about robots in other systems can originate either from communication or observation. If the other system provides (maybe read-only) access to the information, then we term the other robot a *remote robot*. However, if the information is only available through observation, we define the other robot as an *observed robot*. Thus, the world model has to keep (a descending amount of) information about controlled, remote and observed robots.

When working with two youBots, each of which belongs to its own system, some information (e. g. the robot positions) is available in different systems with varying precision, and has to be organized. One way is to keep one *world model per system*. Figure 8a shows an example of two resulting world models. The figure concentrates on geometric information, with coordinate systems (frames) indicating positions of objects in Cartesian space, while arrows represent positions or transformations between these frames. Solid arrows indicate controllable positions, while dashed arrows indicate positions retrieved through measurement or communication. Orange arrows (i. e. L1 to L3) belong to the system of the left youBot, while blue arrows (i. e. R1 to R3) belong to the right youBot system. The frames could be augmented by additional information such as shape data, which is omitted here for clarity. In this case, the system of the left youBot

knows information about the left youBot's start position together with the distance traveled, as well as current position information about the right youBot. On the right youBot system, the opposite situation occurs.

Using separate models has the advantage that multiple systems can just use different instances of the same world model, which allows to re-use models designed for single systems to a large extent. However, for cooperation a lot of information has to be duplicated, such as robot models of robots that occur as controlled robot in one and as a remote robot in another system. Additionally, the different world models have to be kept consistent. For example, a workpiece that is grasped and moved in one system also has to appear as grasped and moved in the other system.

The second way is to keep one *global world model* with all the objects and relations, and to provide access to the different transformations or sensor values for each system (cf. Figure 8b). This has the advantage that the world model is always consistent (as far as topology and static information is concerned, however different systems can still disagree about object transformations), and structural changes performed in one system are automatically present in other systems. However, this scheme lacks flexibility when dealing with observed robots: While a mobile robot can keep track of its movement since the start through odometry measurements, an observer has no way to achieve this information from outside. Thus, the frame graph contains two transformations for controlled and remote robots (cf. R1 and R2 in Figure 8a), while for observed robots only one transformation is available (cf. R3 in Figure 8a).

To solve this problem, we propose a *mixed world model* scheme (cf. Figure 8c). In a mixed world model, the static data is shared between all systems, while dynamic data can be different for each system. For example, information about physical objects (such as youBot geometry) as well as static connections (such as the position of the youBot arm relative to the youBot platform) are shared. Dynamic connections (such as the position of the youBot relative to the origin, or the fact that the youBot is positioned relative to the world origin or youBot origin) can be different for each system. Still, in both systems it should be possible to compute the transformation of the left youBot to the right youBot, using the data and topology present in each system (and to use it for planning and execution). This combines the advantages of a shared world model with the flexibility to include limited observations, while still allowing the application to address one youBot in a uniform way.

7 Support in Software Frameworks

When looking at the different software frameworks, a different focus becomes obvious. ROS itself puts no emphasis on real-time guarantees, so typically a real-time context only spans one device (cf. Figures 3a to 3c), where a single device is encapsulated into a ROS node, providing an interface to execute the required local commands. Sometimes multiple devices (such as a youBot arm and platform) are combined into one node, however this leads to higher coupling. On

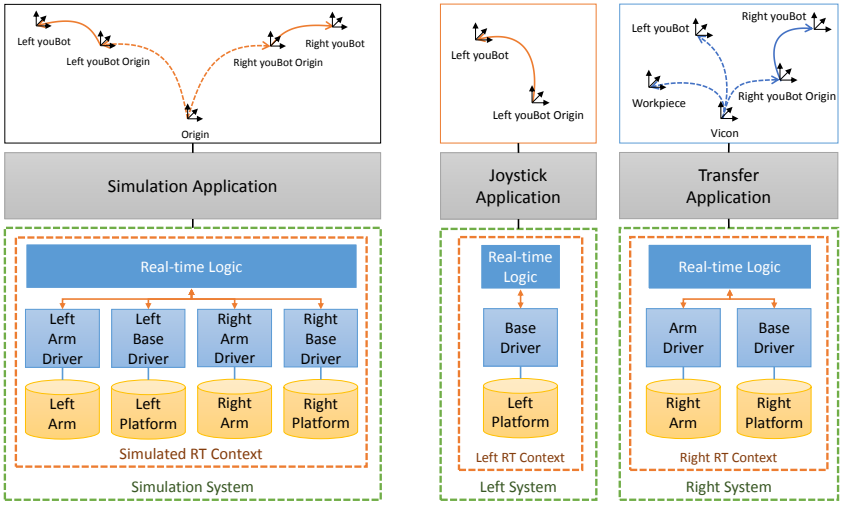
the system level, all nodes that run using the same ROS master are seen as a system. In this situation, all these nodes can subscribe to any data published by other nodes, and post messages or actions, thus providing the transparent data exchange required for a system.

To implement applications in ROS, Python scripts can be used for sequential workflows, while SMACH state machines introduced by Bohren and Cousins [5] allow reactive behavior. However, communication with multiple ROS masters from one application is not natively supported. To share data between different systems (i. e. ROS masters), concepts like *multimaster* or *foreign_relay* [25] can be used to forward topics between multiple masters. The forwarded topics have to be configured during deployment. Additionally, working with multiple masters is one of the new use cases motivating ROS 2[1]. Within one ROS system, the world model is managed through data periodically published by different nodes. It includes transformation data as provided by the *tf service*, as well as robot models and data (e. g. sensor values) from other nodes.

In OROCOS usually most devices are combined into one real-time context (cf. Figures 3b to 3e), because the framework targets real-time capable component systems with device drivers implemented in C++ on a RTOS. When using multiple robots, wired connections are used to ensure one real-time context, sometimes even for mobile systems [18]. Looking at the system level, OROCOS as a framework does not contain features for non-real-time communication, however often ROS is used to combine multiple real-time contexts into one common system [26] that allows for non-real-time communication and data sharing. Consequently, OROCOS does not provide direct support for access to multiple systems. Control flow can be expressed in LUA scripts, while complex coordination is possible using *rFSM statecharts* as suggested by Klotzbücher and Bruyninckx [19]. World models are usually implemented as components in an application dependent manner within the real-time context, which can include geometry, semantics and history [4].

Using the Robotics API, the underlying *Robot Control Core* [28] is implemented in C++ for Xenomai and includes real-time capable drivers for devices connected to the corresponding computer. Additionally, the *Realtime Primitives Interface* [28] allows for the flexible definition of real-time logic to execute user-defined tasks. In this way, all devices physically connected to the computer can form a real-time context (cf. Figures 3c to 3e). The system term here refers to the concept of a *RoboticsRuntime*, which represents one real-time context and makes the data available to applications in a non-real-time way as well as to other devices in the same context for real-time reactions. Control flow can be expressed directly in Java applications, as well as through statecharts, e. g. by using Java libraries for SCXML. It is easily possible to use multiple systems in one application (as in the Factory 2020 case study [1]) and to share limited amount of data between different systems or applications – either through common data sources (such as a Vicon system connected to both youBots through WiFi) or through explicit direct transfer. The frame graph [16] contains semantic

[1] http://design.ros2.org/articles/why_ros2.html



(a) Simulation application (b) Real application

Fig. 9. Software structure for the example application

information such as the relation between frames (e. g. if they are static, temporary or have a transformation that can change during execution). With further information about physical objects (e. g. their physical properties or shape), it serves as a world model in an application and can be used for planning as well as task execution.

8 Experimental Results

The simulation experiments described in the case study have been performed using the Robotics API as a framework and its corresponding simulation and visualization engine. A single application as shown in Figure 9a was used to control one system (cf. Figure 5c) containing a single real-time context (cf. Figure 3e). In this real-time context, both youBot arms and platforms were simulated, as well as the youBot grippers. The application was programmed in an object-oriented fashion, referring to the robots and work pieces as software objects. Moreover, the application used a single world model and expressed all interaction in the control flow of a Java method. Its initial version, where the first robot was not moving during transfer, was easily extended into the second version where both platforms were moving. This extension mainly consisted of adding commands to move the first platform, and to make the second platform synchronize its motion to the position and movement of the first platform. This showed the simplicity of programming and synchronizing robots in the idealized simulation case where every device belongs to the same real-time context and can be synchronized, and where exact positioning is available.

Transferring the results from simulation to reality, various changes had to be done. The tests were conducted on two KUKA youBots using a Vicon optical tracking system for external localization. A straightforward approach would be to combine both youBots into one real-time context. Then, the same distribution scheme as in simulation could be re-used, as well as large parts of the implementation. However, lacking real-time communication over wireless networks (cf. Section 3.2), this was not easily possible. On the other hand, while for stationary manipulators adding a common real-time context greatly simplifies and improves the precision of physical cooperation, in the mobile case the gains are less clear. This is because precise cooperation does not only require exact timing synchronization, but also exact spatial synchronization. For stationary manipulators, this can be achieved by appropriate calibration procedures. For mobile systems, this is in general more problematic due to sensor inaccuracies. External positioning systems – e. g. the Vicon system used here – can mitigate this problem. However, wireless real-time communication becomes a problem again when it comes to transmitting the position information to the youBots.

Thus, we decided to choose an alternative distribution approach as shown in Figure 9b. On each internal computer an instance of the *Robot Control Core* was running, which formed the real-time context (cf. Figure 3d) as well as the system for the corresponding youBot. On the system level, this structure corresponds to Figure 5a. Vicon tracking data for both youBots and the workpiece was streamed to both youBot systems through a WiFi connection from an external PC running the Vicon software. Looking at the application level, each youBot was controlled from a separate application (cf. Figure 7e). The motion of the platform carrying the workpiece was controlled through teleoperation. The other youBot was controlled by a Java method similar to the one in the simulation case.

However, both applications used separate world models (cf. Figure 8a). The workpiece and the other youBot platform were not modeled as Java objects, but only the Vicon position data was used to synchronize the motion and find the grasp position. The youBot arm used joint impedance control to mitigate position inaccuracies. Still, the experiment succeeded and the work piece could be transferred. Instead of separate world models, a single application with a mixed world model could have been used, which would be closer to the (single) world model in the simulation case. This way, changes to the world model topology (e. g. the information that the object has been grasped) would have automatically been transferred to the second youBot's system and static position data would be known to both youBots.

9 Conclusion and Outlook

In this paper, we introduced different levels for structuring the software for distributed robot applications: real-time, system, application level. Moreover, implications on the world model have been discussed. The structure on the different levels can be used and combined independently, motivated by technical as well as political constraints. The different options for structuring and distribution

have been explained based on a case study of cooperating mobile manipulators and various robot frameworks, and evaluated both in simulation and in real world setup with two KUKA youBots. In the example applications, different ways to distribute the software on different levels have been introduced, and the advantages and drawbacks for the given scenario have been shown.

Overall, it became clear that there is not a single optimal way of structuring and distributing the software. The levels presented in this work will hopefully be a starting point that can help developers in designing and discussing their software architecture. Based on non-functional requirements to the developed solution (e. g. reactiveness, synchronization quality, data privacy, trust), the choice of the appropriate distribution scheme and framework(s) for implementation should become easier. As next steps, we plan to implement the mentioned other ways of distribution and to evaluate the gains for the given scenario. This especially includes the use of a mixed world model, as well as ways to share a world model between multiple applications or to synchronize relevant structural changes.

References

1. Angerer, A., Hoffmann, A., Schierl, A., Vistein, M., Reif, W.: Robotics API: Object-Oriented Software Development for Industrial Robots. J. of Software Engineering for Robotics 4(1), 1–22 (2013)
2. Barbulescu, L., Rubinstein, Z.B., Smith, S.F., Zimmerman, T.L.: Distributed coordination of mobile agent teams: the advantage of planning ahead. In: Proc. 9th Intl. Conf. on Autonomous Agents and Multiagent Systems, Toronto, Canada. vol. 1, pp. 1331–1338 (2010)
3. Bischoff, R., Huggenberger, U., Prassler, E.: KUKA youBot - a mobile manipulator for research and education. In: Proc. 2011 IEEE Intl. Conf. on Robot. & Autom., Shanghai, China. pp. 1–4. IEEE (May 2011)
4. Blumenthal, S., Bruyninckx, H., Nowak, W., Prassler, E.: A scene graph based shared 3D world model for robotic applications. In: Proc. 2013 IEEE Intl. Conf. on Robot. & Autom., Karlsruhe, Germany. pp. 453–460. IEEE (2013)
5. Bohren, J., Cousins, S.: The SMACH high-level executive. IEEE Robot. & Autom. Mag. 17(4), 18–20 (2010)
6. Bonasso, R.P., Kortenkamp, D., Miller, D.P., Slack, M.: Experiences with an architecture for intelligent, reactive agents. J. of Experimental and Theoretical Artificial Intelligence 9, 237–256 (1995)
7. Brugali, D., Shakhimardanov, A.: Component-based robotic engineering (Part II). IEEE Robot. & Autom. Mag. 20(1), 100–112 (Mar 2010)
8. Bruyninckx, H.: Open robot control software: the OROCOS project. In: Proc. 2001 IEEE Intl. Conf. on Robot. & Autom., Seoul, Korea. pp. 2523–2528. IEEE (May 2001)
9. Cao, Y.U., Fukunaga, A., Kahng, A.: Cooperative mobile robotics: Antecedents and directions. Autonomous Robots 4, 7–27 (1997)
10. Chaimowicz, L., Kumar, V., Campos, M.: A paradigm for dynamic coordination of multiple robots. Autonomous Robots 17(1), 7–21 (2004)
11. Costelha, H., Lima, P.: Modelling, analysis and execution of multi-robot tasks using petri nets. In: Proc. 7th Intl. Joint Conf. on Autonomous Agents and Multiagent Systems, Estoril, Portugal. vol. 3, pp. 1187–1190 (2008)

12. Dogar, M., Knepper, R.A., Spielberg, A., Choi, C., Christensen, H.I., Rus, D.: Towards coordinated precision assembly with robot teams. In: Proc. 14th Intl. Symposium of Experimental Robotics, Marrakech, Morocco. IFRR (2014)
13. Dudek, G., Jenkin, M.R.M., Milios, E., Wilkes, D.: A taxonomy for multi-agent robotics. Autonomous Robots 3, 375–397 (1996)
14. Farinelli, A., Iocchi, L., Nardi, D.: Multirobot systems: a classification focused on coordination. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 34(5), 2015–2028 (2004)
15. Hammer, T., Bauml, B.: The highly performant and realtime deterministic communication layer of the aRDx software framework. In: Proc. 16th Intl. Conf. on Adv. Robotics, Montevideo, Uruguay. pp. 1–8. IEEE (Nov 2013)
16. Hoffmann, A., Angerer, A., Schierl, A., Vistein, M., Reif, W.: Service-oriented robotics manufacturing by reasoning about the scene graph of a robotics cell. In: Proc. 41st Intl. Symp. on Robotics, Munich, Germany. pp. 1–8. VDE (June 2014)
17. Hoffmann, A., Nafz, F., Seebach, H., Schierl, A., Reif, W.: Developing self-organizing robotic cells using organic computing principles. In: Meng, Y., Jin, Y. (eds.) Bio-Inspired Self-Organizing Robotic Systems, Studies in Computational Intelligence, vol. 355, pp. 253–274. Springer, Heidelberg (2011)
18. Klotzbücher, M., Biggs, G., Bruyninckx, H.: Pure coordination using the coordinator–configurator pattern. CoRR abs/1303.0066 (2013)
19. Klotzbücher, M., Bruyninckx, H.: Coordinating robotic tasks and systems with rFSM statecharts. J. of Software Engineering for Robotics 3(1), 28–/56 (2012)
20. Knepper, R., Layton, T., Romanishin, J., Rus, D.: Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In: Proc 2013 IEEE Intl. Conf. on Robotics and Automation (ICRA), Karlsruhe, Germany. pp. 855–862. IEEE (2013)
21. Mataric, M.J.: Designing emergent behaviors: From local interactions to collective intelligence. In: Meyer, J.A., Roitblat, H.L., Wilson, S.W. (eds.) From Animals to Animats 2: Proc. of the 2nd Intl. Conf. on Simulation of Adaptive Behavior. pp. 432–441. MIT Press (1993)
22. Parker, L.E.: Heterogeneous multi-robot cooperation. Ph.D. thesis, Massachusetts Institute of Technology Cambridge, MA, USA (1994)
23. Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: Workshop on Open Source Software, IEEE Intl. Conf. on Robot. & Autom., Kobe, Japan. IEEE (2009)
24. Schierl, A., Hoffmann, A., Angerer, A., Vistein, M., Reif, W.: Towards realtime robot reactions: Patterns for modular device driver interfaces. In: Workshop on Softw. Developm. & Integr. in Robotics. IEEE Intl. Conf. on Robot. & Autom., Karlsruhe, Germany. IEEE (2013)
25. Schneider, T.: Distributed Networks Using ROS – Cross-Network Middleware Communication Using IPv6. Master's thesis, Technische Universität München (2012)
26. Smits, R., Bruyninckx, H.: Composition of complex robot applications via data flow integration. In: Proc. 2011 IEEE Intl. Conf. on Robot. & Autom., Shanghai, China. pp. 5576–5580. IEEE (May 2011)
27. Stampfer, D., Schlegel, C.: Dynamic state charts: composition and coordination of complex robot behavior and reuse of action plots. Intelligent Service Robotics 7(2), 53–65 (2014)
28. Vistein, M., Angerer, A., Hoffmann, A., Schierl, A., Reif, W.: Interfacing industrial robots using realtime primitives. In: Proc. 2010 IEEE Intl. Conf. on Autom. and Logistics, Hong Kong, China. pp. 468–473. IEEE (Aug 2010)