2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

A Hierarchical Extension of Manipulation Primitives and Its Integration into a Robot Control Architecture

Ingo Weidauer, Daniel Kubus and Friedrich M. Wahl

Abstract— **Manipulation Primitives are state-of-the-art for describing and performing complex manipulation tasks – especially in industrial applications. They combine easy task specification and execution which are key components for the success of this paradigm.**
However, there are some flaws when describing a complex task by decomposing it into Manipulation Primitives. For instance, the relations between the individual Manipulation Primitives cannot be captured by the original definition but have to be defined in additional glue code. This contribution introduces an extended definition of Manipulation Primitives to describe manipulation tasks as hierarchical structures using Manipulation Primitives at each level of the hierarchy.
Additionally, a concept for a generic control architecture is presented which is capable of meeting the requirements of the extended Manipulation Primitive concept and which can be easily „plugged“ together using modular code blocks.

I. INTRODUCTION

Performing manipulation tasks in highly complex or uncertain environments is one of the major goals the robotic community has been trying to achieve for the last decades. Some of the requirements for reaching this goal are the integration of sensors into robot control architectures and the reduction of complexity in describing demanding manipulation tasks using sensors.

Already in the 1980s, Mason introduced concepts for compliance and force control coining the *Task Frame Formalism* (TFF) [1]. Extended by De Schutter et. al [2] in the 1990s, it became a key concept for abstract programming of robotic tasks. Inspired by the latter ideas, the concept of Manipulation Primitives was devised by Mosemann and Wahl [3]. It was augmented by Finkemeyer [4] and further extended in [5]–[7]. During the last decade Manipulation Primitives have been adopted by various groups all over the world.

To describe a manipulation task with this paradigm, the task has to be decomposed into Manipulation Primitives which only comprise the primitive subtasks but *not* the relations between them. Thus, these relations have to be defined in the source code. In this paper an extended definition is introduced which eliminates this flaw.

Task descriptions using place transition nets [8] in robot automation were introduced by Thomas [9]. Milighetti [10]–[12] presented a concept describing complex tasks for humanoid robots as primitive skills in a hierarchical petri net with only one level of hierarchy using several resources (e.g., robot, tool). His extension to Manipulation Primitives uses further elements to allow fuzzy- and/or probability-based switching decisions for humanoid robots. Szynkiewicz [13] defined manipulations skills which are hierarchically structured actions to reach predefined goals. He uses a hierarchical definition of manipulation tasks distinguishing different manipulation classes. These classes describe different subgoals and are divided into smaller manipulations skills until the lowest level of basic skills is reached. Both, Milighetti and Szynkiewicz, share the idea of describing manipulation tasks as a hierarchical decomposition of abstract tasks reaching down to primitive tasks which can be performed with a robot. As hierarchy can facilitate code reuse and effectively hide complexity, a fully hierarchical approach to task description is incorporated into the extended definition of Manipulation Primitives presented here.
Although many types of complex tasks can be tackled with the idea of Manipulation Primitives, the setup of a new robot system including sensors and control algorithms is still a time consuming task. Since these systems become more and more demanding, a way of reducing complexity is needed. As in other areas, the robotics community has begun to focus on building modular code blocks which can be reused in robot control frameworks.

Within the OROCOS project, rFSM [14] is used to specify complex behavior with hierarchical state machines using a scripting language named Lua. The ROS project provides a package called smach [15]. Complex behavior can be described by hierarchical state-machines using a python based language. Both approaches describe tasks only from an abstract point of view. The concrete movement of a robot or tool has to be specified somewhere else.

One of the latest efforts is the BRICS project [16] which tries „to identify and document best practices in the development of robotics systems“. One of these practices is to build modular code blocks which can be reused in several tasks, as for instance presented by Brugali [17], [18].

The primary aim of this paper is to present an improved hierarchical description of tasks by place transition nets using the same concept at each level of the hierarchy from the top-level down to the primitive robot tasks. We demonstrate that the proposed definition of manipulation tasks can describe concurrent tasks with several robots and significantly reduce complexity. Furthermore, the aspect of interchangeability of subtasks is addressed, which enables the user to easily replace parts of the already defined task on different levels of the hierarchy. Manipulation tasks can be defined from an abstract point of view down to primitive tasks known as

The authors are with the Technische Universität Braunschweig, Institut für Robotik und Prozessinformatik, 38106 Braunschweig, Germany {i.weidauer, d.kubus, f.wahl} at tu-bs.de

Manipulation Primitives.

The second aim is to introduce a robot control architecture which is capable of meeting the above presented requirements posed by the extended definition of Manipulation Primitives. Robotic systems with multiple robots can be easily „plugged“ together from already implemented modular software blocks for robots, control algorithms, and sensors.

Sec. II describes the extension of Manipulation Primitives and Sec. III presents a generic framework for performing manipulation tasks. Two examples demonstrating the advantages of the above presented enhancements are shown in Sec. IV. Sec. V concludes the paper.

II. MANIPULATION TASKS

The Manipulation Primitive concept proposed in [4]–[6] is a powerful specification tool for manipulation tasks. Every task has to be decomposed into elementary subtasks which correspond to Manipulation Primitives. The relations between them cannot be described by this definition and thus have to be defined in source code – leading to glue code which is hard to maintain.

Inspired by the ideas of Milighetti and Szynkiewicz, a fully hierarchical extension of the Manipulation Primitive paradigm is presented. The main idea is to describe a task with the same concept at each level of the hierarchy – starting from an abstract point of view (e.g. grasping an object) and going down to primitive tasks (e.g. moving the robot, closing the gripper, ...) which are directly executed by a robot or tool.
The extension features a Manipulation Task as its central element which is defined as follows:

$$\mathcal{MT} := \{\alpha, \{\mathcal{HM}|\mathcal{PN}|\emptyset\}, \rho, \tau, \omega\} \tag{1}$$

where

1) α is the start condition including sensor values (real sensor values or virtual sensor values, cf. Sec. III), relational operators $(=, \neq, <, >, >=, <=)$, logical operators $(\&, |)$ and brackets (,) for ordering, e.g. "$(force <= 20.0 \ \& \ button = 1) \ | \ force > 20.0$)". If the condition becomes true, the Manipulation Task is started immediately. All available sensor values can be used.
2) $\{\mathcal{HM}|\mathcal{PN}|\emptyset\}$: The first option in this triple is the hybrid move $\mathcal{HM}$ (cf. Sec. II-A), the second one $\mathcal{PN}$ is a modified place transition net where the places are Manipulation Tasks (cf. Sec. II-C) and the third one is the empty set $\emptyset$. Only one of this three options can occur in a Manipulation Task. The empty set $\emptyset$ may, for instance, occur in a subtask when a human movement is to be tracked but no robot movement is desired.
3) ρ is defined as a set of parameters which are valid for the Manipulation Task and all lower levels of the hierarchy.
4) τ is defined as a set of commands, where $\tau = \{device\_name, command\}$. Instead of the original definition in [5], each device can receive commands (manipulation, control and sensor devices; cf. Sec. III). Tools can be implemented as manipulation devices.
5) ω is the stop condition. It is defined the same way as the start condition. If the condition becomes true, the Manipulation Task is stopped in the current control cycle.

A. Hybrid Move $\mathcal{HM}$

The hybrid move comprises several types of movements. Not only trajectory following movements can be performed but also sensor guided and sensor guarded movements. Any kind of control algorithm can be used as pointed out in [5], [6].

The hybrid move is defined as follows

$$\mathcal{HM} := \{\mathcal{M}, \mathcal{TF}, \mathcal{D}\} \tag{2}$$

where

1) $\mathcal{M}$ is the used manipulation device (e.g. robot, gripper) for this hybrid move.
2) $\mathcal{TF}$ is the task frame (cf. Sec. II-B).
3) $\mathcal{D}$ denotes desired values for the control devices [5], [6]. The present definition differs from the original one in only one point. The desired values can be directly taken from sensor values or parameters which are known in the system. This enables the user to implement functional algorithms whose results can be directly used in the *next* Manipulation Task (e.g., detecting an object with a laser range finder and using the pose of this object in the following grasping task).

B. Task Frame $\mathcal{TF}$

The task frame is the key component for defining different manipulation tasks. It determines the coordinate system w.r.t which the task is described. A more detailed description can be found in [4], [5].

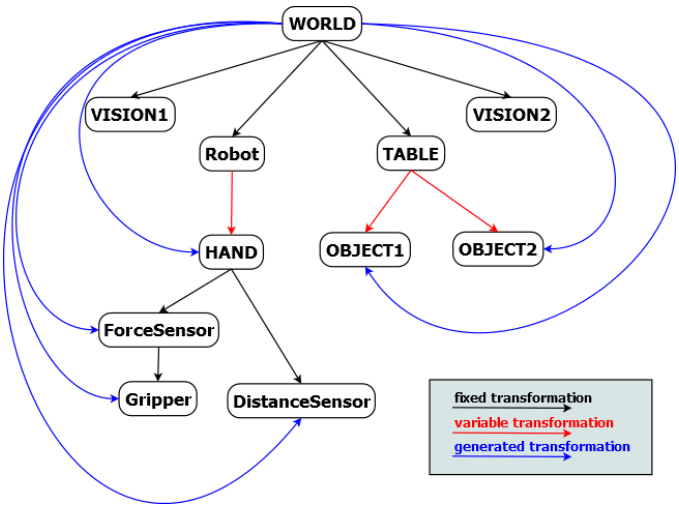

Fig. 1. Internal representation of transformations for the environment of the robot system. All presented coordinate systems are known by name and can be used in the $\mathcal{TF}$.

The task frame is defined as follows:

$$\mathcal{TF} := \{\mathcal{RF}, \mathcal{MF}, \mathcal{FFC}\} \tag{3}$$

1) $\mathcal{RF}$ is the reference frame. Instead of the classical definition [5] of the reference frame and a numerical description of the offset, the $\mathcal{RF}$ is taken from a set of all coordinate systems known in the robot system. Fig. 1 depicts an example of possible reference frames.
2) $\mathcal{MF}$ is the moving frame known by its name (Fig. 1). As described in [5], only the transformation ${}_{\mathcal{TF}}T_{\mathcal{HAND}}$ was used for a task frame which was anchored in the world. With the definition of a $\mathcal{MF}$, any frame which is attached to the robot can be used instead of the hand frame resulting in the transformation ${}_{\mathcal{RF}}T_{\mathcal{MF}}$. This extension enables the user to define complex tasks without calculating the transformation between tool and hand frame every time it is needed.
3) A definition of the $\mathcal{FFC}$ frame is used to enable feed forward compensation, as described in [5] when the task frame is attached to a moving system (e.g., a second robot or a rotating plate). If this frame is not defined, feed forward compensation is disabled.

C. Place Transition Net $\mathcal{PN}$

The relation between Manipulation Tasks is described in a place transition net. Decisions as well as concurrent execution of manipulation tasks is possible.

The place transition net is defined as follows:

$$\mathcal{PN} := \{P, T, F, C, M_0\} \tag{4}$$

1) P is a set of places. Each place corresponds to a Manipulation Task ($\mathcal{MT}$).
2) T is a set of transitions.
3) F is a set of arcs where an $\mathcal{MT}$ is connected to a transition and a transition is connected to an $\mathcal{MT}$ but two $\mathcal{MT}$s or two transitions cannot be connected to each other,i.e., $F \subseteq (P \times T) \cup (T \times P)$.
4) C is a set of transition firing rules.
   $C \subseteq (T \times \Gamma \times \Pi)$ where Γ is the set of all possible conditions (including the empty condition) and $\Pi \in \mathbb{N}$ describes a priority. Conditions are defined the same way as the start condition α and the stop condition ω of the $\mathcal{MT}$. If defined, the condition has to become true for firing the corresponding transition. Any sensor value which is known in the framework can be used (cf. Sec. III).
5) M_0 is a set of initially marked places $M_0 \subseteq P$.

The place transition net allows only one mark for every place. A transition fires immediately if no condition is defined. If a condition is defined, it has to become true for firing. If one place has arcs pointing to several transitions, only one transition can fire. This is called a decision as only one mark is allowed for every place. If no conditions are defined or conditions do not exclude each other, the transition with the highest priority will fire. Therefore, no two priority values in the respective set of transitions can be the same. If one transition points to several places, it is called a fork and concurrent tasks can be synchronized. If several places point to one transition, it is called a synchronization. The transition will only fire if all incoming Manipulation Tasks have finished.

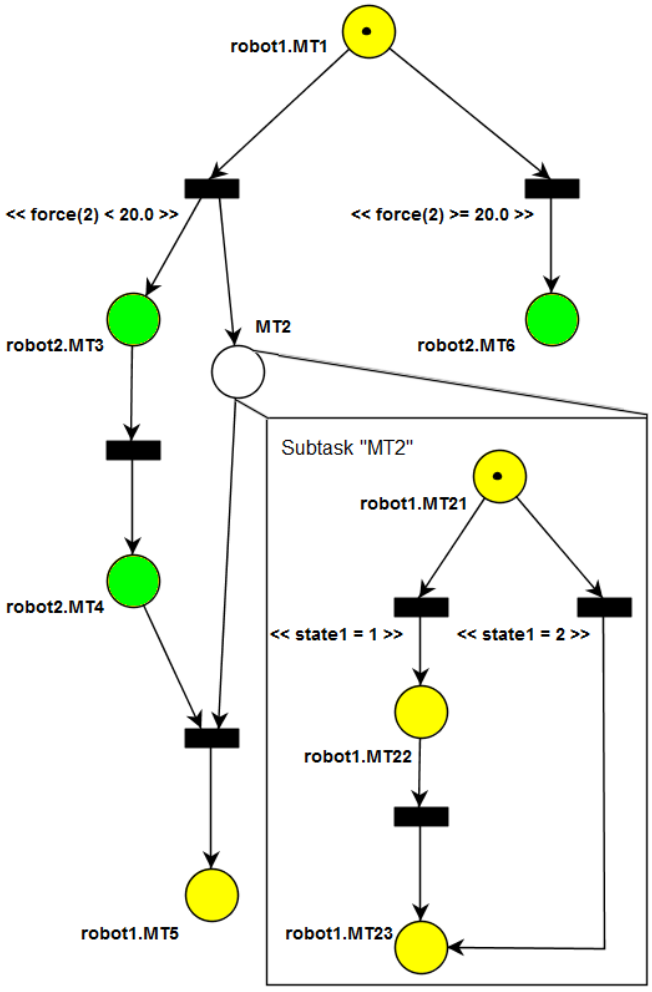

Fig. 2. Example of a hierarchical task description with the introduced notation. Circles show different Manipulation Tasks and different colors depict different robots. The black bars depict transitions. Conditions are shown next to the transitions with „<<“ and „>>“ brackets. The subtask MT2 is shown in the rectangular box.

The initial marking is important to define the start of a task (or subtask) as all Manipulation Tasks which have an initial mark will start at the beginning of the task on this level. A subtask is started when the corresponding Manipulation Task in the place transition net is started. A task or subtask is finished if no more Manipulation Tasks can be executed. If a Manipulation Task contains a place transition net as a subtask, the Manipulation Task is finished either if the subtask is finished or the stop condition ω becomes true.

Fig. 2 depicts an example of a fictitious manipulation task

described with the above concept. Two different robots are used (colored yellow and green) and a subtask is shown. The task starts with MT1 holding a mark. If MT1 is finished, two options for going through the net are available. If the sensor value „force“ is greater or equals 20.0, MT6 is executed. The other branch forks after the transition and MT2 and MT3 are started synchronously. As MT2 includes a subtask, it starts with MT21 according to the initial marking. After both, MT2 and MT4, are finished, the last Manipulation Task MT5 is executed.

Several enhancements result from the above definition. Complex manipulation tasks can be easily described using the hierarchical definition with Manipulation Tasks at each level of the hierarchy. As subtasks are part of the Manipulation Task, easy replacement is possible without changing the structure of the original net. If, for instance, the Manipulation Task „MT2“ in Fig. 2 describes a grasping task, it can be easily replaced with another subtask if the grasped object changes or a different subtask has to be used for reaching the goal. Another advantage is the possibility to parametrize not only a single primitive task but also the whole subtask. Also, concurrent tasks with several robots or manipulation devices (e.g., gripper, linear axis) can be synchronized with the proposed definition.

In a nutshell, this approach to describing complex tasks enables the designer of automation processes to define complex tasks with different Manipulation Tasks starting from an abstract point of view and refining tasks progressively with the same concept.

III. FRAMEWORK

Performing complex manipulation tasks requires a control framework which is capable of executing Manipulation Tasks in real-time. Moreover, it has to guarantee that after finishing a Manipulation Task the next one can be executed in the next control cycle. As Manipulation Tasks can terminate during execution if the stop condition is fulfilled, a task may finish with a velocity $\neq 0$ and no other Manipulation Task can be executed. In this case, a default Manipulation Task has to be executed to bring the robot into a safe condition.

The presented framework has been implemented using the C++ programming language and the QNX real-time operating system. It is designed with a centralized structure featuring a main software block coordinating (cf. MTController in Fig. 3) the execution in every cycle. This approach guarantees that the same sensor values are used for all devices in every cycle. The design not only meets the above described requirements for performing complex manipulation tasks with different robots but it also enables users to build complex robot systems by simply „plugging“ all needed software components and hardware devices together and get rid of the time consuming task of implementing a complete robot system from scratch. The framework is work-in-progress and is currently able to execute Manipulation Tasks with one place transition net (cf. Sec. III-A). To give the reader an impression of the used software design, the framework is described in the following.

Fig. 3 depicts the structure of the proposed modular framework which is an extension of the framework proposed in [5], [6]. The main advantages of this framework are the

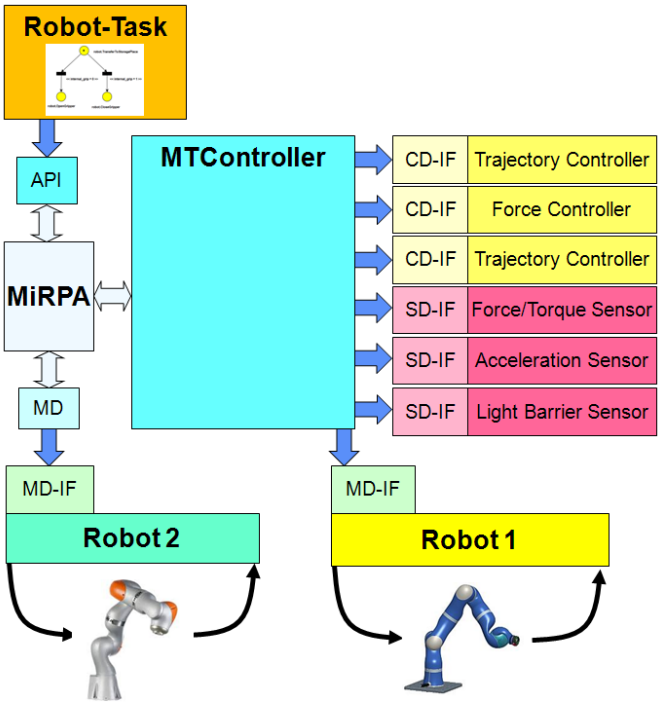

Fig. 3. Application example with two robots. One robot is connected directly to the MTController framework and the other robot uses the real-time middleware MiRPA [19] for the connection to the MTController. MD-IF is a manipulation device interface and MD a module for connecting a robot via MiRPA [19]. Several sensor devices are implemented using the sensor device interface SD-IF. The control devices use the control device interface CD-IF. Additional sensor or control devices may be connected directly or via MiRPA.

reduced complexity when setting up new robot systems with the above requirements and the capability to execute Manipulation Tasks as defined in Sec. II. The user is not limited to the middleware MiRPA [19] as any control loop which is closed over a middleware leads to additional delays which can severely deteriorate performance if several components are daisy-chained. Therefore, any component of the system may be connected directly to the core (MTController). The core components (MTController, API) are implemented as a core library. Apart from this core library, there exist three different types of software devices which have to be implemented by the creator of the robotic system.

- **Manipulation device**: hardware component which can manipulate the environment (e.g. robots, grippers, linear axes, and even simulation environments).
- **Control device**: implementation of a control algorithm, e.g., an online trajectory generator as presented by Kröger [20] or a force control algorithm, etc.
- **Sensor device**: implementation of a driver for sensor hardware which acquires data or a virtual sensor, e.g., a sensor data fusion algorithm, a state estimation algorithm, etc.

To reduce the complexity of the system, interfaces have be-

en defined for each type of device. Such interfaces also define the information which is exchanged at the start and during a control cycle between the device and the MTController. Every type of device registers and provides the following information to the MTController framework:

- Provided sensor values (name, type, dimension, attached to a coordinate system as presented in Fig. 1)
- Requested sensor values (name, optional/mandatory)
- Requested parameters (name)

Manipulation devices provide some more information, e.g., their number of degrees of freedom. Providing this information enables the framework to automatically find dependencies between devices. The provision or request for sensor values is not limited to sensor devices because control or manipulation devices may also provide important information such as sensor values or need further sensor values for successful execution, e.g., an impedance control algorithm. Another example could be a sensor data fusion or state estimation algorithm which requests further sensor values required for its execution.

Every device has to implement the method `runCycle(input_values, output_values)` defined by the corresponding interface which is executed in each control cycle by the MTController. A device can access required information from the input values and has to provide information by writing it to the output values. This method is identical among all types of devices. Thus, no undesired dependencies other than to the interface have to be satisfied, enabling the user to create new devices on-the-fly, test them offline (e.g. control algorithm) and to set up new robot systems very fast.

Using this kind of framework is a fast way to set up demanding robot systems by simply „plugging“ components together and thus reducing the implementation time. Already implemented algorithms can be reused which furthers the idea of code reuse as demanded in [16]–[18]. Another advantage is the possibility to use state-machines, implemented as sensor devices, which can control the flow through the net by changing system states represented as sensor values.

A. Description and Execution of Manipulations Tasks

As the definition of complex tasks in source code is very unhandy, Manipulation Tasks can be described using the place transition net editor Pipe+ [21] which is based on Pipe2 [22]. The editor has been extended to fulfill the above described place transition net definition. As mentioned earlier, the framework is work-in-progress. Currently, tasks can be described as follows:

1) Primitive tasks using the hybrid move $\mathcal{HM}$ can currently be defined only in source code using a unique id.
2) A corresponding place transition net can be described graphically in Pipe+ with a unique id corresponding to the primitive task.

With the Pipe+ editor, a name, a unique id for each place, initial marks for several places, conditions for transitions, and the relation between places and transitions can be edited.

For execution, primitive tasks can be send via MiRPA or they can be passed directly to the MTController. Depending on the MTController working mode („Execute direct“, „Execute net“) the primitive task a) will be executed directly or b) is only loaded and for execution, the place transition has to be sent to the MTController. For the execution of the place transition net, the XML file which has been saved with Pipe+ is parsed ignoring the graphical information included by the editor.

IV. APPLICATIONS

A. Bin Picking

To show the advantages of the proposed concept, an industrial bin picking task [23] is described. In Fig. 4, the structure of the bin picking task is depicted using the notation introduced in Sec. II.

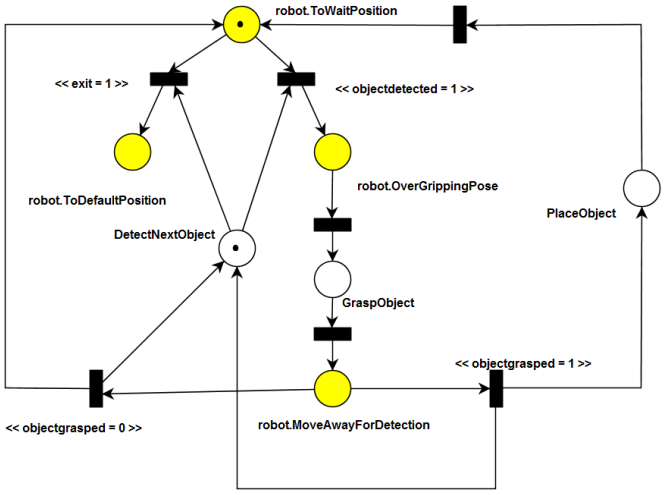

Fig. 4. Example of an industrial bin picking task according to the above defined hierarchical Manipulation Task definition. Fig. 5 shows a potential implementation for the subtask „PlaceObject“, Fig. 6 for the subtask „GraspObject“ and Fig. 7 for the subtask „DetectNextObject“.

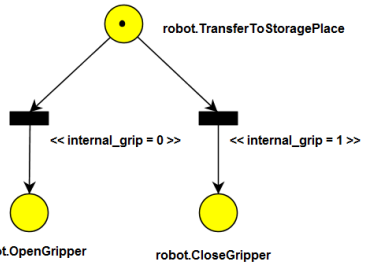

Fig. 5. Potential implementation of subtask „PlaceObject“ of Fig. 4.

The task requires one robot and one linear axis which have to be moved partially synchronously. A gripper, which is capable of grasping piston rods with an internal and an

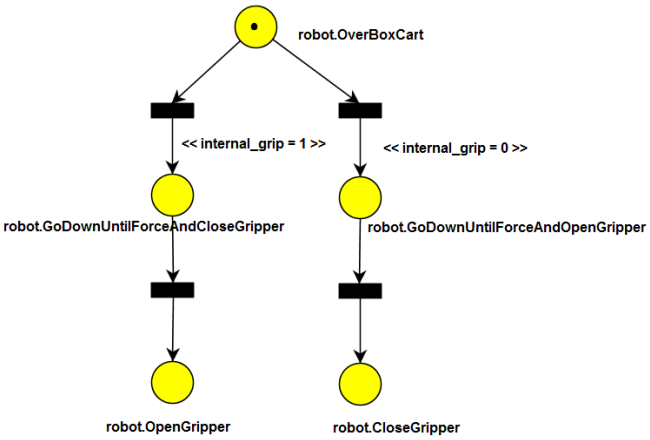

Fig. 6. Potential implementation of subtask „GraspObject“ of Fig. 4.

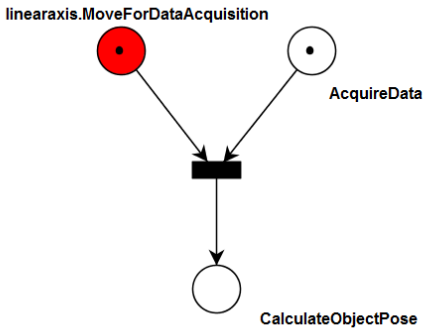

Fig. 7. Potential implementation of subtask „DetectNextObject“ of Fig. 4. The white colored Manipulation Tasks do not use a robot. Only functional algorithms are executed.

external grip, is attached to the robot hand. The task can be easily realized by implementing the robot and the linear axis as a manipulation device. Manipulation Tasks using the robot are colored yellow and Manipulation Tasks using the linear axis are colored red. The white colored Manipulation Tasks are subtasks which are shown in Fig. 5- 7. The place transition net in Fig. 4 has two initial marks displaying the Manipulation Tasks which are executed at the beginning.

The sensor values „objectdetected“, „exit“ „and internal_grip“ are global state variables which can be modified by an external user using the API (e.g. „exit“) or the algorithm employed to detect the next object (MT „DetectNextObject“) implemented as a sensor device.

In the context of bin picking the manipulation task presented in Fig. 4 can be easily adapted if a different object is to be handled or a new sensor for detecting the object is to be used.

B. Estimating the Pose of Grasped Objects

This example is not dedicated to the task itself but the way dependencies between sensor devices can be described in the MTController. It has been implemented for the bin picking approach proposed by Buchholz [24] with only one place transition net.

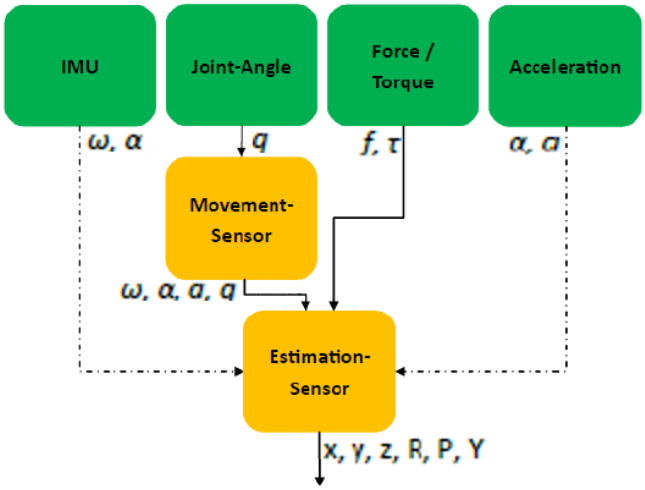

Fig. 8. Dependencies between sensor devices in an implemented application for estimating poses of grasped objects. The continuous lines show mandatory requests for sensor values and the dashed lines show optional requests as the sensor values provided by the acceleration sensor and the inertial measurement unit can also be obtained (with lower quality) from joint angle measurements.

Fig. 8 shows the dependencies of several sensor devices in an application that estimates poses of grasped objects based on inertial features [25]. The object (e.g., a piston rod) is grasped by the robot but due to an uncertain grasping pose the exact transformation relative to the gripper has to be estimated from the mentioned inertial features. This example has been implemented using several sensor devices which partially request sensor values from other sensor devices. In Fig. 8 the dependencies between the sensor devices are shown. The first level (green sensor devices) depicts sensor devices which acquire data from sensor hardware.

- The IMU Sensor acquires linear acceleration and angular velocity measured with an inertial measurement unit attached to the wrist of the robot.
- The JointAngle Sensor acquires joint angles from the robot based on encoder measurements.
- The Force/Torque Sensor acquires forces and torques measured with a force/torque sensor attached to the wrist of the robot.
- The Acceleration Sensor acquires linear and angular accelerations measured with an acceleration sensor attached to the wrist of the robot.

In the second level, the „Movement Sensor“ requests joint angles from the first level of sensors and uses them to calculate the linear acceleration, angular velocity, and angular acceleration. In the third level the „Estimation Sensor“ uses this information to calculate the inertial features and finally the pose of the object. It provides these values to the MTController framework as a sensor value. All dependencies between the sensor devices are modeled using the information (requested sensor values, provided sensor values) which each device provides the framework with.

V. CONCLUSION AND OUTLOOK

In this paper we have demonstrated how complex manipulation tasks can be specified using a hierarchical structure extending the concept of Manipulation Primitives. Tasks can be described starting from an abstract point of view and going down to more and more fine-grained subtasks which finally specify the hybrid move of the robot, formerly defined by Manipulation Primitives. Apart from the extended notion of Manipulation Primitives, concurrent task execution, easy interchangeability of subtasks, and the reduction of complexity are the main advantages of this concept.

The proposed control architecture facilitates the set up of demanding robot systems by simply „plugging“ components together and thus reducing the complexity and the setup time. Already implemented components can be reused which furthers the idea of modular building blocks. Furthermore, it enables the user to use functional algorithms for controlling and adapting the task execution. This opens up new options for the execution of demanding manipulation tasks.

Compared to the concepts by Milighetti and Szynkiewicz, the presented extension of Manipulation Primitives is more flexible owing to its fully hierarchical structure with an unlimited number of levels and the open implementation of every kind of sensor, control algorithm, and manipulation device. In contrast to the popular OROCOS rFSM [14] and ROS smach [15], the proposed extension allows us to specify *and* execute complex manipulation tasks using the same unified concept.

Future work will focus on describing cooperative tasks performed by robots and human workers based on the presented paradigm. To this end, further attention is needed regarding stability and dead-locks which may occur due to functional algorithms which can be directly used to adapt task execution on-line. Moreover, pitfalls due to the semantics of the place transition nets have to be tackled. Since different tasks may try to use the same robot at the same time, conflicts in resource allocation may occur. Fortunately, established techniques to solve these issues exist.

The implementation of a debug interface as used in ROS smach [15] may give the user a handy interface for monitoring complex tasks. Moreover, the extension of the place transition net editor to describe a task with all its subtasks is necessary to improve usability.

REFERENCES

[1] Matthew T. Mason. Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, 11(6):418–432, 1981.

[2] Herman Bruyninckx and Joris De Schutter. Specification of force-controlled actions in the ttask frame formalisma synthesis. *IEEE Transactions on Robotics and Automation*, 12(4):581–589, 1996.

[3] Heiko Mosemann and Friedrich M. Wahl. Automatic decomposition of planned assembly sequences into skill primitives. *IEEE Transactions on Robotics and Automation*, 17(5):709–718, 2001.

[4] Bernd Finkemeyer, Torsten Kröger, and Friedrich M. Wahl. Executing assembly tasks specified by manipulation primitive nets. *Advanced Robotics*, 19(5):591–611, 2005.

[5] Bernd Finkemeyer, Torsten Kröger, and Friedrich M. Wahl. The adaptive selection matrix-a key component for sensor-based control of robotic manipulators. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 3855–3862. IEEE, 2010.

[6] Torsten Kröger, Bernd Finkemeyer, and Friedrich M. Wahl. Manipulation primitives-a universal interface between sensor-based motion control and robot programming. In *Robotic Systems for Handling and Assembly*, pages 293–313. Springer, 2011.

[7] Torsten Kröger and Friedrich M Wahl. Stabilizing hybrid switched motion control systems with an on-line trajectory generator. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 4009–4015. IEEE, 2010.

[8] Jörg Desel and Wolfgang Reisig. Place/transition petri nets. In *Lectures on Petri Nets I: Basic Models*, pages 122–173. Springer, 1998.

[9] Ulrike Thomas and Friedrich M. Wahl. Planning sensor feedback for assembly skills by using sensor state space graphs. In *Intelligent Robotics and Applications*, pages 696–707. Springer, 2012.

[10] Giulio Milighetti. *Multisensorielle diskret-kontinuierliche Überwachung und Regelung humanoider Roboter (in German)*. KIT Scientific Publishing, 2010.

[11] Giulio Milighetti, HB Kuntze, CW Frey, B Diestel-Feddersen, and J Balzer. On a primitive skill-based supervisory robot control architecture. In *Proceedings of the 12th International Conference on Advanced Robotics (ICAR)*, pages 141–147. IEEE, 2005.

[12] Giulio Milighetti and Helge-Björn Kuntze. On the discrete-continuous control of basic skills for humanoid robots. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 3474–3479. IEEE/RSJ, 2006.

[13] Wojciech Szynkiewicz. Skill-Based Bimanual Manipulation Planning. *Journal of Telecommunications and Information Technology*, 4:54–62, 2012.

[14] Markus Klotzbuecher. Orocos rsfm. http://people.mech.kuleuven.be/ mklotzbucher/rfsm/README.html, 2013. [Online; accessed 9-Febuary-2014].

[15] Jonathan Bohren. Ros smach. http://http://wiki.ros.org/smach, 2010. [Online; accessed 9-Febuary-2014].

[16] Rainer Bischoff, Tim Guhl, Erwin Prassler, Walter Nowak, Gerhard Kraetzschmar, Herman Bruyninckx, Peter Soetens, Martin Haegele, Andreas Pott, Peter Breedveld, Jan Broenink, Davide Brugali, and Nicola Tomatis. Brics - best practice in robotics. *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1 –8, june 2010.

[17] Davide Brugali and Patrizia Scandurra. Component-based robotic engineering. part i: Reusable building blocks. *Robotics Automation Magazine, IEEE*, 16(4):84 –96, december 2009.

[18] Davide Brugali and Azamat Shakhimardanov. Component-based robotic engineering. part ii: Models and systems. *Robotics Automation Magazine, IEEE*, 17(1):100 –112, march 2010.

[19] Bernd Finkemeyer, Torsten Kröger, Markus Olschewski, and Friedrich M. Wahl. MiRPA: Middleware for Robotic and Process Control Applications. In *Proceedings of the Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware at the International Conference on Intelligent Robots and Systems (IROS)*, pages 76–90. IEEE/RSJ, 2007.

[20] Torsten Kröger and Friedrich M. Wahl. Online trajectory generation: basic concepts for instantaneous reactions to unforeseen events. *IEEE Transactions on Robotics*, 26(1):94–111, 2010.

[21] Su Liu, Reng Zeng, and Xudong He. Pipe+ - a modeling tool for high level petri nets. In *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE)*, pages 115–121, 2011.

[22] Nadeem Akharware. Pipe2: Platform independent petri net editor. Master's thesis, Department of Computing, Imperial College London, London, 2005.

[23] Dirk Buchholz, Simon Winkelbach, and Friedrich M. Wahl. Ransam for industrial bin-picking. In *Proceedings of the 41st International Symposium on Robotics and 6th German Conference on Robotics (ROBOTIK)*, pages 1–6. ISR/VDE, 2010.

[24] Dirk Buchholz, Daniel Kubus, Ingo Weidauer, Alexander Scholz, and Friedrich M. Wahl. Combining visual and inertial features for efficient grasping and bin-picking. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*. IEEE, 2014 (Accepted).

[25] Daniel Kubus, Torsten Kröger, and Friedrich M. Wahl. On-line rigid object recognition and pose estimation based on inertial parameters. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 1402–1408. IEEE/RSJ, 2007.