# Autonomous Parallelization of Resource-Aware Robotic Task Nodes

Sebastian Georg Brunner; Andreas Dömel; Peter Lehner; Michael Beetz; Freek Stulp

Keywords: Software, Middleware and Programming Environments; Autonomous Agents; Mobile Manipulation; Planning, Scheduling and Coordination; Agent-Based Systems

# Autonomous Parallelization of Resource-Aware Robotic Task Nodes

Sebastian G. Brunner[1], Andreas Dömel[1], Peter Lehner[1], Michael Beetz[2], and Freek Stulp[1]

***Abstract*— Robot task programming often leads to inefficient plans, as opportunities for parallelization and precomputation are usually not exploited by the programmer. This inefficiency is often especially obvious in mobile manipulation, where path planning and pose estimation algorithms are time-consuming operations. In this paper, we introduce the concept of Resource-Aware Task Nodes (RATNs), a powerful descriptive action model for robots. Next, we propose an algorithm that executes so-called Concurrent Dataflow Task Networks (CDTNs), robot plans consisting of RATNs. It optimizes programmed plans based on two sources of information: 1) The control flow represented in the original task plan, whose constraints are relaxed to generate opportunities for parallelization and precomputation. 2) Dependencies between actions pertaining to resources, data flows and world model changes, the latter being equivalent to preconditions and effects. CDTNs have been integrated in our open-source task programming framework RAFCON, and we show that it leads to 11-29% improvement in terms of execution time in two simulated mobile manipulation scenarios.**

## I. Introduction

Programming robots that require prolonged phases of autonomy to solve tasks is still very challenging. Robot task programming frameworks facilitate this process, for instance by offering graphical design tools [1, 2]. As such, these frameworks are important components in the *Industrie 4.0* initiative for more flexible production [3]. However, in practice, the execution of such programs is often slow due to high software run-times. We identify two reasons for this.

First, path planning and object pose estimation are usually computationally intensive operations, and only by applying probabilistic and/or learning-based algorithms can sound solutions be found at all. This holds in particular for mobile manipulation, which requires many calls to such expensive operations for perception, navigation, and planning.

Second, actions are often programmed only sequentially, and opportunities for parallelization and precomputation are not exploited. In our experience, users often program *a* task solution that works in principle, but in the space of *all* possible task solutions, is hardly ever the most efficient one.

See for instance the mobile manipulation use case in Fig. 1, where the task is to navigate to a table, and pick up two objects on it. Here, both objects could be detected before manipulating them, rather than detecting the second after the first has been manipulated, and several path plans could be precomputed simultaneously in parallel before executing them, rather than waiting until they are needed.

[1] German Aerospace Center (DLR), Robotics and Mechatronics Center (RMC), Münchner Str. 20, Wessling, Germany.

[2] University of Bremen, Institute for Artificial Intelligence (IAI), Bibliothekstr. 1, Bremen, Germany.

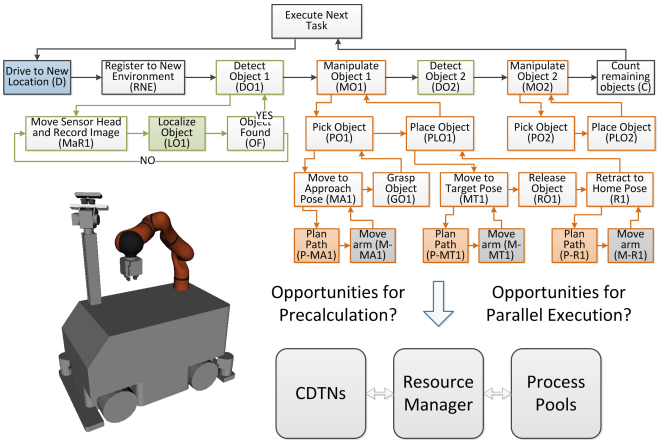


Fig. 1. An exemplary hierarchical state machine (HSM) representation for a mobile manipulation task, in which two objects are manipulated. The scientific question of this work: How can we exploit opportunities for parallelization and precomputation in such overconstrained task descriptions? To do so we propose a conversion of the HSM to a Concurrent Dataflow Task Network (CDTN), whose execution is monitored by a resource manager which, amongst others, manages pools of processes.

We argue that the optimization of programmed tasks should be automated, rather than leaving it as a tedious task for the user. Therefore, this paper addresses the following question: How can we leverage the knowledge and respect the constraints encoded in programmed task descriptions, whilst still automatically determining opportunities for parallel execution, and the precomputation and caching of results?

A second aspect of the example worth highlighting is that all actions are required to solve the task, but not all of them affect the environment state (i.e. the state of the real world). For instance, computing a path with a path planner does not affect the objects in the environment, but this computation is necessary before the arm can be moved to follow this path. In this paper, we propose *resource-aware task nodes*, which extend robot actions with *dependencies*, which represent robot-specific constraints on the ordering of actions that do not affect the environment state.

The main contributions of this paper, which also define its structure, are: Section III: Generalizing and extending the concept of pre- and post-conditions so that they can represent data flows and resources, which is highly relevant for the execution of plans on real robots. Section IV: Proposing a formalism for converting states in hierarchical state machines (HSMs) into *resource-aware task nodes*, which extend the states in a HSM with dependencies. Section V: Introducing a resource-aware algorithm that is automatically able to find opportunities for the parallel execution of task nodes and the pre-computation and caching computations, whilst respecting the defined dependencies between the task nodes.

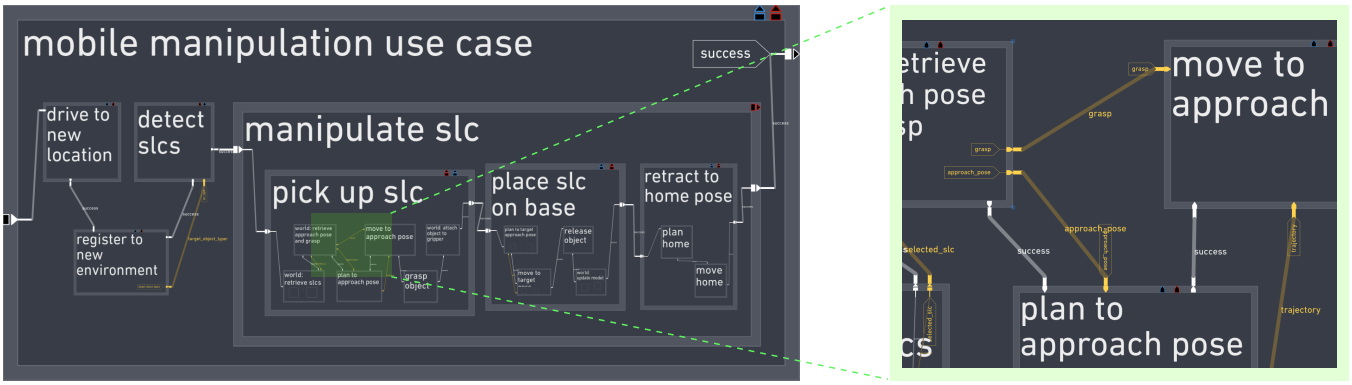


Fig. 2. Screenshot of the Hierarchical State Machine (HSM) in Fig. 1 as implemented in RAFCON. The inset highlights that not only state transitions are defined (white connections labeled success), but also data flows (yellow connections labeled grasp, approach_pose, trajectory, etc).

Section VII: Demonstrating how the above leads to 11-29% faster execution times on mobile manipulation tasks.

## II. Related Work

Our work falls into the area of transformational planning [4], which modifies existing plans – regardless if they have been planned or manually designed – in order to meet different constraints. Transforming a plan in order to reduce the number or probability of errors is the goal of GORDIUS [5], HACKER [6] and CHEF [7]. As in our work, these approaches are able to integrate plan revisions into a partly executed plan, which is necessary to react to exogenous events and to avoid long planning pauses during execution. Moreover, our work is related to the criticize/revise cycle of XFRM [4], the main difference being that our focus is to make the execution of plans more efficient rather than robust. The projection of action effects and the creation of different versions of possible future world states is very similar. On top of the concepts of Concurrent Reactive Plans [4] we provide a powerful resource model, a semantically more distinctive action definition and proper concepts for dataflow modeling.

Concerning highly parallel execution approaches, the concepts of automatic parallelization in e.g. compiler construction [8], dataflow programming [9], software pipelining [10], and speculative multi-threading in the context of processor development [11] are relevant. Our approach shares with these methods the aim to execute operations in parallel and/or in advance as much as possible. To the best of our knowledge, such parallelization and precomputation approaches have not yet been applied to make the task control of robotic systems more efficient, especially in combination with graphically programmed tasks.

Finally, the distribution of executable actions to available resources poses many open scheduling challenges [14], e.g. scheduling under complex constraints, managing change during execution runtime, and the advantages and disadvantages between self-scheduling systems and central schedulers.

The concepts presented in this paper are kept as general as possible. Our implementation of these concepts builds strongly on the open-source[1] tool RAFCON [2]. RAFCON models tasks as hierarchical finite state machines (HSMs), adds functionality for explicit modeling of data flows, and enables concurrent execution of states (similar to [15] for behavior trees). Although the original extended state machine formalism is called HPFDs (hierarchical, parallel, finite state machine with data flows), we will – for sake of brevity – refer to them simply as HSMs in the rest of this paper. Child nodes at the bottom of the hierarchy are called *Execution States*, because they execute a python snippet. All other states define the control flow of the task.

## III. Dependency Model

Graphically programmed HSMs, as illustrated in Fig. 1 and 2 are often overconstrained, and do not exploit opportunities for parallelization. The most extreme approach to automate such parallelization would be to attempt to execute all Execution States in the HSM in parallel. In almost all cases, this will either violate the logical control flow represented in the HSM, or violate data flow and/or resources dependencies. In this section, we present how such dependencies are modeled Section IV will then focus on mapping the control flow represented in the HSM to *resource-aware task nodes* with which opportunities for parallelization can be exploited.

We model three types of dependencies[2]: *Data dependencies*, where an action requires some data structure to be computed before it can be executed. *Resource dependencies*, where a resource must be available before an action can be executed. *Worldmodel dependencies*, where certain conditions in the world state must hold before the action can be executed. Worldmodel dependencies are equivalent to pre- and post-conditions in symbolic plans, so we present them first.

### A. World Model Dependencies

World model dependencies represent action preconditions, whereby actions may only be executed if certain facts in the world state hold, as in the Planning Domain

[1] https://github.com/DLR-RM/RAFCON

[2] A *condition dependency* is a fourth (special) case, which will be presented in Section IV-B.4.

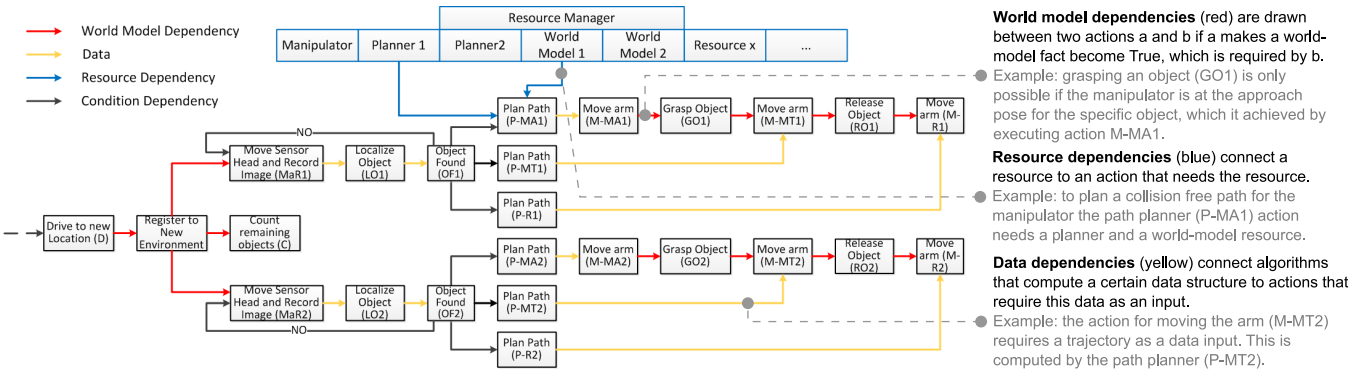


Fig. 3. Exemplary dependency graph for the mobile manipulation task shown in Fig. 1. Different colors model different kinds of dependencies between the actions. Since there are very many resource dependencies, only two exemplary dependencies have been plotted. Condition dependencies and the resource manager will be explained in Section V.

Definition Language (PDDL) [16].[3] Formally, we define world model dependencies as parametrized predicates, which are grounded in ontologies (for information on how we use ontologies in RAFCON we refer to [12]). For instance, the action $grasp\_object(o)$ can only be called if the fact $manipulator\_at\_approach\_position\_of\_object(o)$ is $True$. As can be seen in this example world model facts can be bound to specific objects. Action effects, on the other hand, add, modify or delete facts of the world model and thus satisfy the preconditions of other actions. For information about how we implemented them, see Fig. 8 in the implementation section.

### B. Data Dependencies

In robotic task planning, data, signals and events are often modeled in a global variable structure [4]. Dataflow modeling ensures that data interfaces between processing nodes are consistent. Furthermore, as in our case, the concept strongly facilitates parallelization possibilities. In terms of debugging, this leads to a clear data dependency overview for the whole system. Another advantage is the ability to modularize and reuse functionality [17].

A data dependency represents a necessary datum for an internal computation, but it does not influence the world model for the robot itself. Thus, we model data dependencies as data flows of HSMs. They are formally defined as a quadruple: *(source_action, source_data_port, target_action, target_data_port)* (see [2]). Additionally, they are associated with a defined data type.

The main reason for employing data dependencies in CDTNs is to enable easy projection of action effects and finally the pre-calculation of intermediate task data. This requires a fine-grained modeling, and a separation of the logical structure of actions from the dataflow dependencies between them. As shown in Fig. 2 and 8, data flows are represented as connections between states, and are graphically created and editable in RAFCON.

[3]We use "world model" as synonym for "world state". Both describe the robot's model of the real world it acts it. To refer to the state of the real world we use "environment state".

### C. Resource Dependencies

Planning for robots must take into account the resources that are available to a robot: Is there a processor free to compute a motion plan? Is the planning process running, or has it crashed upon startup? Do I have enough battery power? Is an accurate world model currently available, and is it up-to-date? Such questions are usually not taken into account in planning and scheduling algorithms, but are essential to achieving robustness and autonomy in robotics [18, 4].

The resource manager, presented in detail in Section V-B, is a component which administrates all resources in a resource pool. As a concrete example, if the action Path Plan depends on the path planner process to be running but it has crashed, the resource manager will signal the process manager to create this resource by re-starting the process.

In our resource model, there are five classes of resources: *system, process, module, world* and *task* resources. System resources are created by hardware interfaces (e.g. by monitoring the the battery power), process resources by a *process manager* (e.g. if the object detection process is running), module resources by the modules themselves (e.g. the internal module state of the navigation module is 'unlocalized'), world model resources by the world model pool (i.e. copies of the root world model) and finally task resources by the task control module (e.g. two blue boxes are already delivered). Task resources help to track the progress of the final goal (e.g. if several blue boxes have to be delivered).

Thus, hardware and software resources can be represented in the same model. Almost any component can be modeled as a resource, independent of if it is a software process (e.g. a constraint geometric motion planer), a semantic resource (two objects left to deliver) or e.g. the whole robot (what also makes the model attractive for multi-robot scenarios).

Formally, we define resource dependencies, such as world model dependencies, as parametrized predicates grounded inside an ontology (see [12]). In practice, we model resource dependencies in our Semantic Data Editor, shown in Fig. 8.

### D. Use Case as a Dependency Graph

A CDTN is created by converting all actions of a HSM into task nodes and by defining dependencies between those task nodes (see Section IV). During CDTN execution, we generate a dependency graph (see Fig. 3), in which

the nodes are task nodes, and the edges are task node dependencies. Generating this graph happens in two steps. First, the Condition and Iterator nodes of the CDTN are expanded (see Section IV-B). Then all dependencies of the original and newly created task nodes are inserted as edges into the dependency tree.

Note that several actions are no longer constrained to be executed sequentially (e.g. the path planning actions P-MA1, P-MT1, P-R1), as was the case in the HSM in Fig. 1. Therefore, they can be executed in parallel on different processors. In Section V, we present an algorithm that is able to exploit these potential parallelizations, without violating any of the dependencies.

## IV. From Hierarchical State Machines to Concurrent Dataflow Task Networks

A CDTN is defined as set of task nodes with world model, data, resource and condition dependencies. As illustrated in Fig. 4, a task node extends a HSM state.

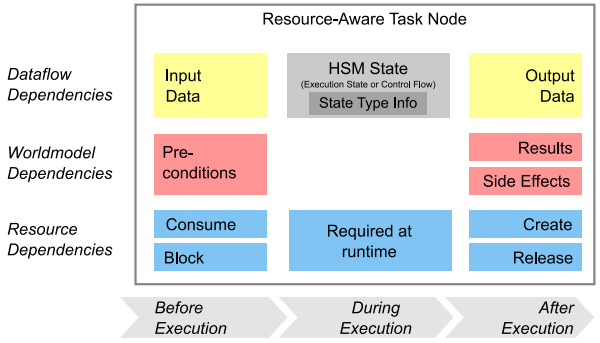


Fig. 4. Overview of the components of resource-aware task nodes

### A. Resource Aware Task Node Definition

*Resource-Aware Task Nodes* (or simply *task nodes*) are states of the HSM formalization (HPFD to be precise, see Section II) with an additional set of properties (see Fig. 4). A task node may have different *dependencies* associated with it. The dependency type can be: *data*, *resource*, *world model* and *condition* (see Section III). Apart from the HSM state type such as *hierarchy* or *concurrency*, task nodes can have an explicit control flow type, which can either be *condition, loop* or *routine* (for their use we refer to the CDTN algorithm V). A task node is not necessarily one HSM state, but can contain several HSM states and other task nodes, which are then called child task nodes. This is achieved by using the hierarchy concept of HSMs. Only condition, loop and child task nodes of routines can have transitions, as defined in HSMs. No transitions are allowed otherwise.

### B. Converting HSM to CDTN control flow

Each control flow element inside a HSM must be converted into a corresponding set of Resource-aware Task Nodes, which together form a CDTN. Control flow elements in the HSM often overconstrain the order of the execution of the task nodes. Whilst translating them into a CDTN, most of these constraints are dropped to enable parallelization. Which parallelizations are actually possible in practice given the dependencies is determined during the run-time execution of a CDTN in Section V.

The HSM control flow elements and their conversion to a CDTN are illustrated in Fig. 5, and now explained in detail.

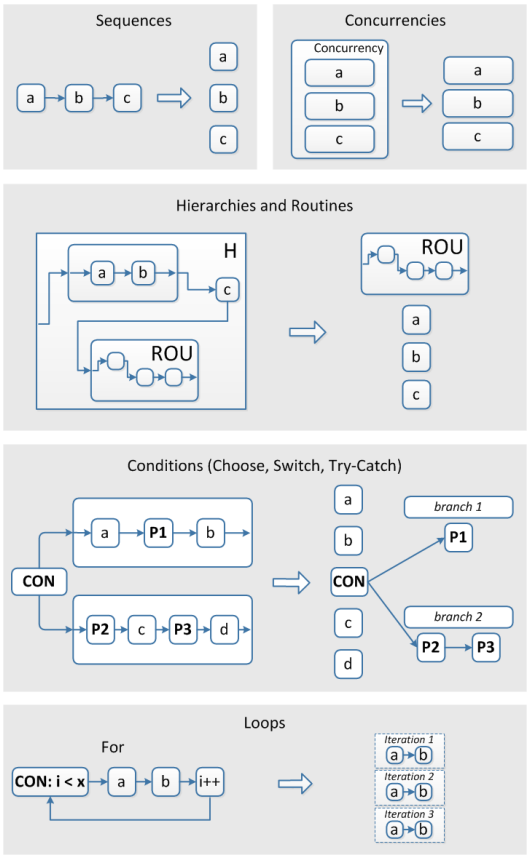


Fig. 5. Basic control flow building blocks in the HSM (left of the arrow) and the corresponding CDTN representation (right of the arrow). Only for-loops are shown; do-while and while-do loops can be rolled out analogously Nodes marked with **Px** are PIAs (see IV-B.4), nodes marked with **ROU** (see IV-B.2) are routines and nodes marked with **CON** are conditions (see IV-B.4).

*1) Sequences and Concurrencies:* All the Execution States in a sequence are Task Nodes. This means that, in principle, they can be executed in parallel. In practice, this may not be the case due to dependencies (including preconditions); the algorithm explained in Section V ensures that no dependencies are violated during execution. If concurrencies are defined in the HSM, they are treated the same as sequences.

*2) Routines (ROU):* The user can define a task node to be a *Routine*. This indicates that the child states in that task node should never be executed in parallel, as they are tightly coupled. Routines are treated differently from other task nodes in the execution algorithm in Section V.

*3) Hierarchies:* Hierarchies are only used by the programmer to define abstraction layers in CDTNs. Before execution the hierarchies are removed and all child nodes are executed in parallel (as long as no dependencies constrain the nodes in another way). All dependencies defined for hierarchies are copied to the child nodes i.e. the child nodes inherit all dependencies from their parents.

*4) Conditions (CON) and Condition Dependencies:* Condition dependencies are used to model conditions in a CDTN, i.e. two mutually exclusive branches of execution. Before a condition is reached, idle processing power can be used to precompute results required in both of the branches. Upon execution, results are then discarded for branches that are not reached. In many cases, this is more efficient (from the point of view of execution time, not computational time) than waiting until it is known which branch will be entered.

Fig. 3 shows an example condition dependency. The Object Found task node, which is a condition node, is connected to subsequent task nodes with condition dependencies (black arrows). If the detection did not find the target object, the sensor head moves and records a new image. Otherwise the condition dependency to the planning task node is satisfied, which can then generate a collision free path for the manipulator.

When parallelizing states with condition dependencies, care must be taken to not execute task nodes that cannot be undone. It is easy to undo the pre-computation of a result, i.e. it can be disparagingly discarded from memory, and an outside observer would not notice. But grasping a cup changes the robot's environment state, and can potentially fail. We call such task nodes Potentially Irreversible Action (PIA). As they cannot be undone, PIAs should only be executed if they are in a branch that is actually executed. This is implemented by automatically adding a condition dependency from a PIA to all its parental task nodes, as illustrated in Fig. 5.

*5) Loops:* Loops, such as for, do-while and while-do loops are rolled out, whereby each iteration becomes a Task Node. If dependencies allow it, the different iterations of the loop can thus be executed in parallel. As for conditions, the unrolling is not performed for the whole robotic task, as they often consist of several thousand task nodes [12] and the computational overhead to do this for every possible logical branch would be too high. Therefore, a task horizon number limits the maximum possible length of each logical branch.

## V. CDTN Execution Algorithm

So far, we have "dissolved" the rigid, overconstrained structure of the HSM into a set of resource-aware task nodes (along with their dependencies) that represent the same task. In this section, we present an algorithm for executing these task nodes. It is able to exploit parallelization and pre-computation opportunities, but also ensures that all dependencies are met.

The inputs for the algorithm are: a CDTN (which can either be auto-generated or manually created) and an initial world model. The CDTN can, assuming the initial world model having a certain structure, reach a goal using the plan encoded inside its task nodes. The goal is explicitly encoded as effects of the task nodes of the plan.

At the beginning of the main function the CDTN and the initial world model are loaded. In line 3 a separate thread is started, which calls the *handle_task_node_termination* function. The following while loop is the main part of the main function. The loop is executed until the whole CDTN is executed. Inside the loop, the current dependency graph (see Fig. 3) is calculated before all executable task nodes are extracted thereafter and finally started in their own thread. A task node is executable if all its dependencies are satisfied.

```
1: function MAIN
2:     load_cdtn_and_world()
3:     do_in_separate_thread(handle_termination())
4:     while ¬task_finished do
5:         dep_graph ← generate_dependency_graph()
6:         execs ← extract_executable_task_nodes(dep_graph)
7:         for all task_node t in execs do
8:             do_in_separate_thread(execute_task_node(t))
9:         end for
10:    end while
11: end function
12:
13: function EXECUTE_TASK_NODE(t)
14:    request_resources_of_task_node(t)
15:    if is_loop(t) then
16:        roll_out_loop(t)
17:    else if is_routine(t) then
18:        execute_with_default_execution_algorithm(t)
19:    else if is_PIA(t) then
20:        execute_PIA(t)
21:    else
22:        execute_task_node(t)
23:    end if
24: end function
25:
26: function HANDLE_TASK_NODE_TERMINATION
27:    while ¬task_finished() do
28:        t ← wait_for_next_task_node_to_finish()
29:        forward_data_of_task_node(t)
30:        propagate_effects_of_task_node(t)
31:        release_and_create_resources_of_task_node(t)
32:        if condition_task_node(t) then
33:            enable_PIAs_of_branch(get_active_branch(t))
34:        end if
35:    end while
36: end function
```

Fig. 6. The CDTN execution algorithm in pseudo code.

The *execute_task_node* function defines how task nodes are executed. In line 14 all resources of the target task node *t* are requested (and either blocked or consumed). If *t* requires a resource of type world model, then a dedicated world model has to be prepared by means of world model projection (see subsection V-A). If the task node is a loop state the loop is unrolled. This means that for each loop iteration the loop body is copied, all input data of the loop is copied as well and distributed to each iteration (for more information about loop unrolling see IV-B). If the task node has the control flow type *routine*, then the task node is executed with the default HSM execution algorithm [2]. Therefore, transitions inside routines are allowed (see Section IV-A). If the task node is potentially irreversible, then a dedicated function to execute the PIA is triggered. A PIA may never be executed before all condition dependencies are satisfied. All other task nodes are just executed as if they were simple HSM states.

The last function *handle_task_node_termination* defined in line 26 was triggered in the main function inside a separate thread. Until the CDTN is not executed it waits for the next task node *t* to be finished, forwards its data outputs to the other states and propagates the task node effects into the world model. A task node has been executed (terminated) if

all its child nodes have been executed. The CDTN releases all resources of the terminated task node and creates all resources defined in $t$. If $t$ is a condition, the CDTN enables all PIAs in the branch that was chosen during execution.

### A. World Model Projection

During execution, a pool of world models is maintained. One world model represents the robot's current belief about the actual state of the robot's environment. The other world models are copies, which are used to project the effects of task nodes into the future. They represent possible future states arising from the execution of task nodes.

The world model is a set of facts, as in PDDL [16]. As shown in Fig. 7, world models are also resources and are managed in the *world model pool*. The initial world model is classified as the *root* world model. The root world model is always the model of the robot's real environment, that resembles it as close as possible. All task nodes which modify the world model are called *world model task nodes*.

Many task nodes require a world model as a resource e.g. a path planning node needs the geometric representation of the world state to plan a collision free path to some goal pose. If a task node $t$ requires a world model, a copy of the root world model is created and all *intermediate* world model task nodes are executed on the world model copy. Intermediate nodes are all nodes, which are on the path from $t$ to the last executed PIA inside the dependency graph. This implies that all future events that would affect the world state before $t$ is executed are applied to the world model copy. The modified world model copy is then passed to $t$. This is called the *world model projection* of resource dependency satisfaction.

### B. Resource Manager

A resource manager holds all resources of the robotic system, and manages which of them are available. It cares about concurrent resource requests and resource assignments to task nodes. The following operations are defined on resources:

- create: adds a new resource to the resource pool
- consume: consumes a resource for the rest of the task
- delete: deletes a resource
- block: blocks a resource for a certain amount of time; if a resource is blocked no other task node has access
- release: unblocks a resource
- exists: asks for the existence of a resource

### C. Process Pools

*Process Pools* are used to manage several software processes of the same type. They consist of a set of identical processes that only differ by their configuration and allocated task, and are maintained by a process manager. To get a new process of a certain type, the process manager can be asked to create a new process by providing some specifications for the process. The manager adds the newly created process to the pool and returns a handle to the process. In Fig. 7 an example of two process pools are shown: one pool for path planning processes and one pool for object detector processes. Being able to spawn arbitrary many processes of each type is a key aspect of executing a CDTN.

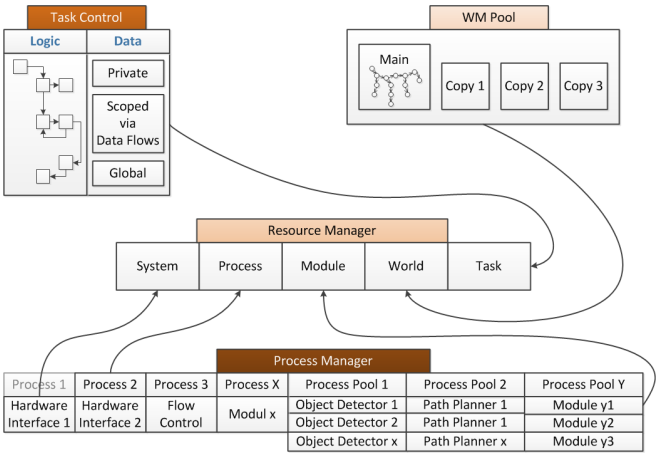


Fig. 7. Overview of different resource classes and their relationships: The 'Resource Manager' in the middle holds all resources, which are created and forwarded to the 'Resource Manager' by the 'Process Manager', the modules administrated by the 'Process Manager', the 'World Model Pool' and the 'Task Control' module.

## VI. Implementation

We implemented CDTNs in the task control framework RAFCON [2] (see Fig. 8).

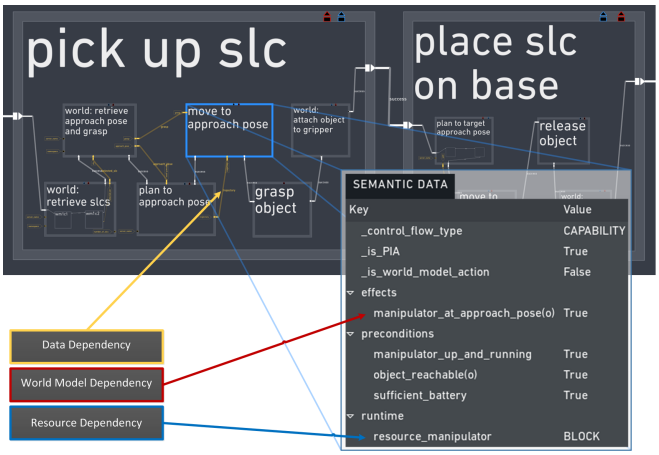


Fig. 8. Implementation of all dependency types in RAFCON. The top image shows a screenshot of RAFCON. Data flows (yellow arrows) represent the data dependencies e.g. the "move to approach pose" task node needs a valid trajectory before it can move the arm. The *Semantic Data Editor* of RAFCON is used to implement world model (red) and resource (blue) dependencies.

The ontology plugin already presented in [12] was used and extended to enhance each state with semantic annotations to make a resource-aware task node out of each state. Resource and world model dependencies are held as strings inside python dictionaries and stored as key-value pairs on disk (json). For data dependencies the data flow information is directly used. The default execution engine was replaced via the plugin concept [2] with the CDTN execution engine described in Section V. The visualization was enhanced to highlight task nodes that are currently waiting for resources.

For the world model we use the graph-database based approach presented in [19]. Finally, the logging mechanism of RAFCON was extended to generate Gantt charts for CDTNs and to analyze task executions.

## VII. Experimental Evaluation

The following application scenario is based on the abstract example use case introduced in Section I. It demonstrates the efficiency benefit of CDTNs in a constraint object manipulation task of our *Advanced Industrial Mobile Manipulator* (AIMM [20]) running in a Gazebo [21] simulator. AIMM consists of a mobile base, a 7-DOF LWR with a two finger gripper, a pan-tilt unit, four cameras and two laser scanners. Fig. 9 shows our robot during object manipulation.

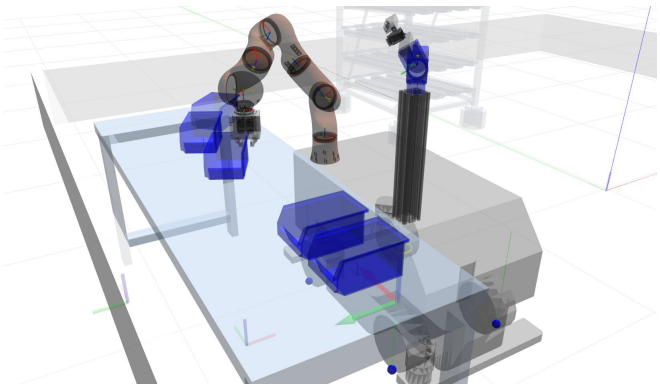

Fig. 9. The AIMM platform in the Gazebo simulation environment with transparent objects in order to visualize frames and contacts.

For simulation we use a setup with three computers. One computer hosts the Gazebo simulator with the robot controllers and sensor data providers. The second computer runs the ROS navigation stack, the path planner process pool with several instances, the world model pool with at least the same number of instances as the path planners, the object detection process pool and several interfacing nodes. For world modeling we use the knowledge representation based on graph databases [19]. The third computer runs RAFCON as the task control framework, together with Rviz and a Gazebo client for visualization and debugging. *Links and Nodes Manager* [22] controls the distribution and management of all processes.

The goal of the task is to pick up a different amount of boxes (*SLCs*, which is short for "small load carriers") from a table and bring them to a shelf. Although the results in this paper have been gathered in simulation, this same task has been executed over 900 times on the real AIMM system at the Automatica trade fair in 2018 [23].

At the beginning of the experiment the robot only knows its initial position and the positions of the table and the shelf. For deeper investigation we focus on the execution of the first part of the task after the robot just reached the table. There, it detects the objects and plans collision free plans to grasp the boxes to place them onto the mobile base.

The chart in Fig. 10 shows the execution as a HSM (top) and the execution of the same actions converted to a CDTN (bottom). On the *y*-axis, several resources of interest are

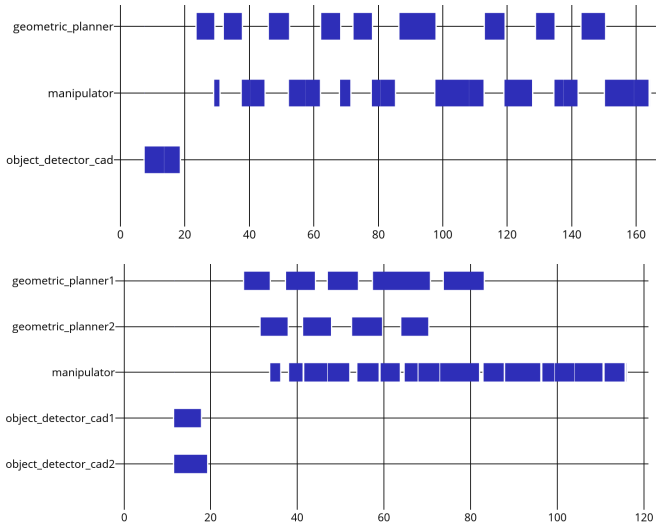


Fig. 10. Gantt charts of the execution of the mobile manipulation task (with three boxes to grasp). Top: execution of the HSM. Bottom: execution of the corresponding CDTN. The x-axis shows the time in seconds and the y-axis the different resources used.

TABLE I

Runtime statistics for fetching boxes from a table with HSMs and the corresponding CDTNs. Execution times are averages of five runs. MP=Motion Planner.

| # boxes | HSM | CDTN (1 MP) | CDTN (2 MPs) | Improvement (2 MPs) |
|---|---|---|---|---|
| 1 | 81.3s | 73.5s | 72.3s | 11.1% |
| 2 | 123.7s | 107.3s | 97.3s | 21.3% |
| 3 | 164.2s | 141.5s | 116.5s | 29.0% |

shown. The HSM execution is purely sequential and there are thus no overlaps between task nodes in the Gantt chart. In the CDTN execution many task nodes are parallelized, such as the object detection calls, the planning tasks themselves and the planning and manipulation tasks. For parallelizing the planning task nodes $p$, the world model state of $p$ is projected (as described in the algorithm in Section V) and then passed to $p$. Table I shows the execution of different task scenarios with both the HSM and the CDTN. Depending on the number of boxes, CDTNs reaches a speedup between 11-29% compared to HSMs (column marked 'Improvement').

In another experiment, our robot sets a table in a living room (Fig. 11). Various dishes and cutlery have to be manipulated to set the table for different amount of persons. Plates, bowls and cups are filled, and the robot thus has to decide if it needs to hold the target objects upright during manipulation (e.g. bowls) or not (e.g. forks). Table II shows the subtask of fetching bowls (B) and forks (F) for 2 and 3 persons.

TABLE II

Runtime statistics for parts of the table setting scenario of Fig. 11 with HSMs and the corresponding CDTNs. Execution times are averages of five runs. MP=Motion Planner

| # objects | HSM | CDTN (2 MPs) | Improvement (2 MPs) |
|---|---|---|---|
| 2 B + 2 F | 169.2s | 122.9s | 27.4% |
| 3 B + 3 F | 255.4s | 207.2s | 18.9% |

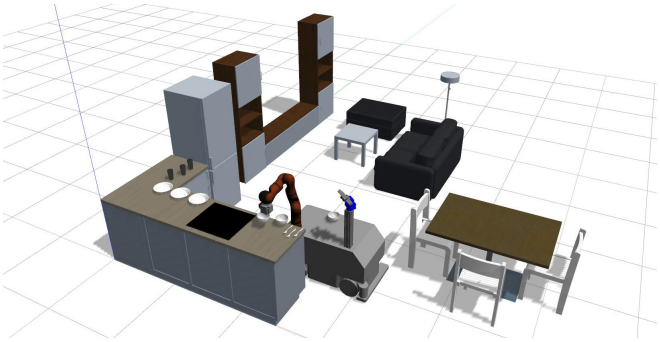

Fig. 11. The AIMM platform in the Gazebo simulation environment manipulating various dishes and cutlery.

## VIII. Conclusion

In this work we presented an uniform approach for automatic pipelining of robotic task nodes by means of Concurrent Dataflow Task Networks. Based on different classes of constraints and dependencies, they allow for increased performance of robotic tasks in different domains. We showcased the efficiency benefits of CDTNs in a constraint manipulation application scenario in which our simulated AIMM platform with thirteen degrees of freedom manipulated several objects.

The application of CDTNs to the given use cases provided us several interesting insights. First of all – and this is what CDTNs where primarily designed for – we have a significant speedup when executing CDTNs compared to a purely sequential execution. Our method will only lead to a performance increase if parallelization is possible; there is little to no room for improvement in inherently sequential tasks, or tasks that can already be executed very quickly. On the other hand, for very complex tasks, graphical representations of data dependencies may become cluttered in practice, and we are developing further visualization tools for assisting and facilitating this process.

In future work, we will evaluate the representation of dependencies in different planning languages, such as PDDL. This will strengthen the formalization of our approach, and also expose the advantages and limitations of different modeling languages for the aims that we have set. This formalization will also allow for direct comparisons with, for instance, the Madagascar planner [24].

In industrial applications, as the one we have presented on our mobile manipulation platform at the Automatica 2018 trade fair, it is essential that robots run continually and robustly. This has been one of the original motivations behind this work: combining graphical programming of HSMs to develop a solution that is *known to work* (but perhaps inefficient) with symbolic planning methods to make this plan more efficient (whilst ensuring that it still works).

In summary, we see our work as lying between artificial intelligence planning methods and parallelization algorithms (as used in software pipelining and dataflow programming) but with a particular emphasis on grounding high-level (symbolic) representations in the concrete data-flows and resources that are typical for autonomous robotic systems.


## Acknowledgment

This work has been supported by the Helmholtz project ARCHES (ZT-0033), and by the Collaborative Research Center EASE (SFB 1320) funded by the German Research Foundation (DFG).


## References


[1] P. Schillinger, S. Kohlbrecher, and O. von Stryk, "Human-Robot Collaborative High-Level Control with an Application to Rescue Robotics," in *IEEE Int'l Conf. on Robotics and Automation*, 2016.

[2] S. G. Brunner, F. Steinmetz, R. Belder, and A. Doemel, "RAFCON: A Graphical Tool for Engineering Complex, Robotic Tasks," in *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[3] Industrie 4.0 Working Group, sponsored by the German Federal Ministry of Education and Research (BMBF), "Recommendations for implementing the strategic initiative industrie 4.0," April 2013.

[4] M. Beetz, *Concurrent Reactive Plans: Anticipating and Forestalling Execution Failures*. Berlin, Heidelberg: Springer-Verlag, 2000.

[5] R. G. Simmons, "A theory of debugging plans and interpretations." in *AAAI*, 1988, pp. 94–99.

[6] G. J. Sussman, "The virtuous nature of bugs," in *Proceedings of the 1st Summer Conference on Artificial Intelligence and Simulation of Behaviour*. IOS Press, 1974, pp. 224–237.

[7] K. J. Hammond, "Explaining and repairing plans that fail," *Artificial intelligence*, vol. 45, no. 1-2, pp. 173–228, 1990.

[8] G. C. Fox, R. D. Williams, and G. C. Messina, *Parallel computing works!* Elsevier, 2014.

[9] J. B. Dennis, "Data flow supercomputers," *Computer*, no. 11, 1980.

[10] J. Ruttenberg et al., "Software pipelining showdown: Optimal vs. heuristic methods in a production compiler," in *ACM SIGPLAN Notices*, vol. 31, no. 5. ACM, 1996, pp. 1–11.

[11] P. Yiapanis, G. Brown, and M. Luján, "Compiler-driven software speculation for thread-level parallelism," *ACM Trans. Program. Lang. Syst.*, vol. 38, no. 2, pp. 5:1–5:45, 2015.

[12] S. G. Brunner, P. Lehner, M. J. Schuster, S. Riedel, R. Belder, A. Wedler, D. Leidner, M. Beetz, and F. Stulp, "Design, execution, and postmortem analysis of prolonged autonomous robot operations," *IEEE Robotics and Automation Letters*, vol. 3, pp. 1056–1063, 2018.

[13] R. T. Effinger, B. C. Williams, and A. G. Hofmann, "Dynamic execution of temporally and spatially flexible reactive programs." in *Bridging the Gap Between Task and Motion Planning*, 2010, p. 121.

[14] S. F. Smith, "Is scheduling a solved problem?" in *Multidisciplinary Scheduling: Theory and Applications*. Springer, 2005, pp. 3–17.

[15] M. Colledanchise and L. Natale, "Improving the parallel execution of behavior trees," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[16] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL - The Planning Domain Definition Language," Yale Center for Computational Vision and Control,, Tech. Rep. TR-98-003, 1998.

[17] W. Wulf and M. Shaw, "Global variable considered harmful," *ACM Sigplan notices*, vol. 8, no. 2, pp. 28–34, 1973.

[18] M. J. Schuster et al., "Towards Autonomous Planetary Exploration: The Lightweight Rover Unit, its Success in the SpaceBotCamp Challenge, and Beyond," *Journal of Intelligent & Robotic Systems*, 2017.

[19] P. Lehner, S. Brunner, A. Dömel, H. Gmeiner, S. Riedel, B. Vodermayer, and A. Wedler, "Mobile manipulation for planetary exploration," in *Aerospace Conference*. Montana, USA: IEEE, 2018.

[20] A. Dömel, S. Kriegel, M. Kassecker, M. Brucker, T. Bodenmüller, and M. Suppa, "Toward fully autonomous mobile manipulation for industrial environments," *International Journal of Advanced Robotic Systems*, vol. 14, no. 4, 2017.

[21] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004, pp. 2149–2154.

[22] F. Schmidt et al., "How we deal with software complexity in robotics: 'links and nodes' and the 'robotkernel'," DLR, Tech. Rep., 2014.

[23] A. Dömel, S. G. Brunner, S. Riedel, and M. Suppa, "Increasing Dependability of Autonomous Robots by Fault Tolerance on Skill Level," in *submitted to IEEE/RSJ Int'l Conf. on Robotics and Automation*, 2019.

[24] J. Rintanen, "Madagascar: Scalable planning with SAT," *Proceedings of the 8th International Planning Competition (IPC-2014)*, vol. 21, 2014.