Towards CLIPS-based Task Execution and Monitoring with SMT-based Planning and Optimization

Tim Niemueller
Gerhard Lakemeyer
Knowledge-Based Systems Group
RWTH Aachen University

Francesco Leofante
Erika Ábrahám
Theory of Hybrid Systems
RWTH Aachen University

Abstract

In robotics, automated task planning is still the exception rather than the norm. While generating a plan certainly represents a crucial aspect, another often neglected one, in particular relevant to integration in robotics, is the task-level executive. It forms the coherent behavior out of the determined plan to achieve the mission goals. This work presents first results towards the integration of a knowledge-based plan executive. This work is embedded in a project to generate guaranteed-quality plans based on Satisfiability Modulo Theory (SMT) in industrial scenarios. The initial prototype is integrated based on the scenario of logistics robots in simulation.

1 Introduction

The goal of automated task planning is often considered completed as soon as a plan has been generated. That notion is carried, for example, by the International Planning Competition. While this is certainly a crucial aspect, acting according to the plan is another. Certainly so when it comes to robotics scenarios where the goal is to facilitate a certain function in a real (or simulated) environment, achieving goals according to a plan and monitoring that execution are equally important. Furthermore, the integration of planning systems with the many other components that compose a robotic system often proves to be challenging.

In this paper, we present two early developments in our current research: first, we describe how to use a Satisfiability Modulo Theory (SMT) solver with optimization as a plan generator. Doing so, we intend to explore the possibilities and limits leveraging the more expressive power of SMT to solve a real-world scenario. State-of-the-art solvers offer more features than just solving constraints expressed in expressive theories. For instance, some solvers allow incremental solving where constraints about task feasibility can be updated; others instead offer optimization features allowing to state and solve (multiple) optimization problems (it should be noted that SMT solvers can handle problems which cannot be handled by traditional linear optimization tools, as Boolean combinations of constraints can be present). Therefore, a crucial point is to *make use of these optimization capabilities* to not only synthesize feasible plans, but guaranteed-quality solutions (i.e., optimal) for plans involving arbitrary combinations of theories. Second, we build on our existing incremental task-level reasoning system (Niemueller, Lakemeyer, and Ferrein 2013) based on the CLIPS rule-based production system and extend it into a knowledge-based plan executive that supports concurrency, execution monitoring and (limited domain-specific) contingency mitigation. It has been used before to model the full scenario, from world modeling, decision making, and execution. However, the previous system required an extensive modeling covering many situations and it always made the decision for just the next action. No look-ahead strategy was employed to plan for cooperation and anticipate the optimal use of multiple robots on a single task.

The general evaluation scenario is the Planning Competition for Logistics Robots in Simulation (Niemueller et al. 2016a) which is based on the RoboCup Logistics League (RCLL). For this first prototype, we focus on the Exploration Phase, where robots need to identify the positions of machines in the environment in a limited time. The task is then to efficiently distribute the overall task to the group of three robots. While this phase is not part of the simulation competition, it provides a sufficient testbed to demonstrate the overall prototype integration.

This work is embedded into a joint project[1] that strives to explore possibilities to employ SMT for optimizing the logistics of autonomous robots in a smart factory. We build on previous work of both involved research groups to create an integrated system. We model the RCLL natively in SMT in order to explore different encoding strategies in terms of solving efficiency and to leverage its optimization capabilities. Building on and extending our proven CLIPS-based executive allows to off-load some crucial run-time aspects like monitoring and failure recovery, and to reason about subproblems and goals to achieve to reduce the computation requirements of the SMT solving process. The executive is also used in other projects with different planners.

In Section 2 we describe some general aspects concerning plan execution and present the evaluation scenario in Section 3. In Section 4 we describe the SMT-based planning approach before explaining our CLIPS-based executive prototype in Section 5. After describing some initial results in Section 6 we conclude in Section 7.

[1] Optimizing the Performance of Robot Fleets in Production Logistics Scenarios Using SMT Solving: `https://ths.rwth-aachen.de/research/projects/smt4robots/`

2 Plan Execution and Related Work

Following Verma et al. (2005b), a *plan* is specified as a series of actions designed to accomplish a set of goals but not violate any resource limitations, temporal or state constraints, or other operation rules. Desirable characteristics of a plan are that it be valid, complete and optimal (or of high quality). Algorithms that can reason about achieving goals over a future time period and in the face of various constraints are called *planners*. As the authors pointed out, a plan usually needs further specification and a system to execute the plan. An *executive* is then a software component that realizes such plans.

While planning certainly is combinatorially and computationally the more complex problem, execution of plans is a very intricate one. It tends to be more platform- and environment-specific than the planning part because the models are often more detailed. For the planning model, the tendency is to strip details to increase planning efficiency or because it is irrelevant for generating the task pattern achieving the goal. Particular challenges may also be slack during execution, or uncertainty, e.g., in travel times due to other agents in the environment. In the multi-robot case, issues of synchronization and mutual exclusion may be relevant.

While there is the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) as a (somewhat) unified planning language, there is no such generally accepted language for execution. Planners do not even output in a common output format. Even if using PDDL as a common input language, and yielding PDDL actions to execute, there is most often additional non-uniform information along with the plan that requires special processing, such as planning times or metrics output.

Several execution systems have been proposed in the past. Most executives mentioned in (Verma et al. 2005b) are associated with a specific modeling language. For example, the Universal Executive (Verma et al. 2006) is a general processor for the PLEXIL (Verma et al. 2005a) language. It allows to describe the execution flow as a number of hierarchically structured nodes consisting of a set of conditions when to execute and a body that describes what to execute. The Universal Executive then ties these descriptions with interfaces to actual actors. While PLEXIL is more of a control language, Procedural Reasoning Systems (Ingrand et al. 1996) lean more towards a knowledge-based representation with an explicit fact base, a notion of goals to achieve or maintain, and activation conditions for procedures. An advantage here is less constrained execution flow, however, this gain in expressiveness may easily come with unintended execution orders without the required caution. A more recent system integrating planning and execution is ROSPlan (Cashmore et al. 2015). It provides a general framework where the individual components can be exchanged (with a varying degree of effort). One of the available dispatchers uses a representation of the plan as an Esterel (Berry and Gonthier 1992) program. There, a plan is described as a set of modules interconnected with signals and receiver slots. However, at this point the translation and execution is opaque and no influence can be exerted on the formulation of the program. There is currently only a limited form of concurrency available. A slightly different approach that has been compared to Esterel is RMPL (Ingham, Ragno, and Williams 2001). Instead of a signal flow, it models the flow more as an evolution of states. Both provide primitives for sequential or parallel execution of code blocks, and conditionals. An earlier system to provide an extensible PDDL-based planning system was TFD/M with semantic attachments (Dornhege et al. 2009). However, the executive was a C++ program which had to be augmented each time for the respective available actions and did not provide a flexible specification language. A more unified approach was recently taken through integrating continual planning in Golog (Hofmann et al. 2016). The overall domain model and execution specifics are encoded in Golog. For planning (sub-)problems the model is translated into PDDL and a planner is called. The specification contains assertions to deal with incomplete knowledge and improve planning efficiency. However, the modeling in Golog can be somewhat tedious and it is often deeply intertwined with its Prolog implementation.

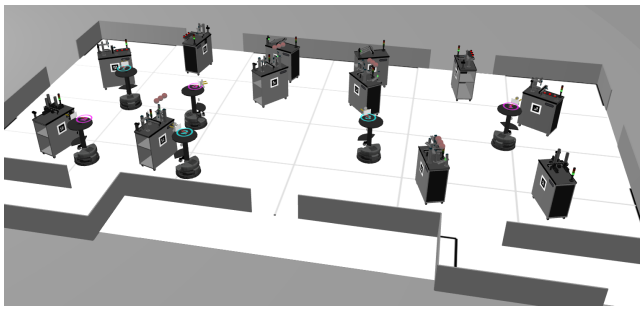

Figure 1: The simulation environment used for our experiments.

In this work, we propose a new formulation of the execution as a rule-based system. With the experience of modeling the decision making, multi-robot coordination, and task execution for the RoboCup Logistics League (Niemueller et al. 2016b), we intend to generalize the framework to be applicable with various planning, reasoning, and decision making components. By this we decouple planning from execution. This comes at the cost of linking two separate models in a consistent way. However, it provides a great flexibility for the executive to choose the appropriate planning system and to add domain-specific interpretations of the plan easily. The planning system we are going to study in more detail in this paper is based on SMT and described in the following.

3 Logistics Robots in Simulation

The example domain chosen for our first integration is based on the Planning Competition for Logistics Robots in Simulation (Niemueller et al. 2016a) and the RoboCup Logistics League (Niemueller, Lakemeyer, and Ferrein 2015). There, the task is to control a group of three robots to transport workpieces among a number of machines. These machines offer processing steps such as mounting a colored ring or a cap. Orders that denote the products which must be constructed with these operations are only posted at run-time and therefore require quick planning and scheduling. Orders come with a delivery time window introducing a temporal

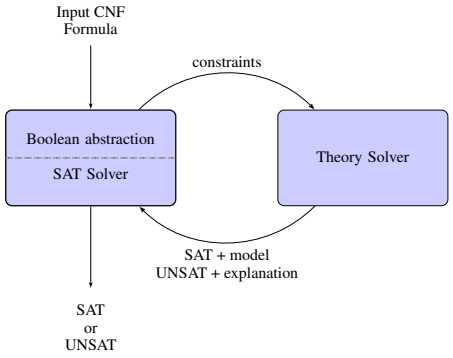

Figure 2: The SMT solving framework.

component into the problem. Furthermore, some machines may undergo maintenance at unknown times at which they may not be used. These aspects make this an integrated planning and execution challenge.

The original competition's game is structured in two phases, the exploration and the production phase. While the latter constitutes the main part of the game, in this work we focus on the former as a simpler scenario for this early proof-of-concept work.

In the exploration phase, the robots must roam the environment and determine where the team's own machines are positioned. For this, the playing field is divided into 24 virtual zones, of which 12 belong to each of the two teams (operating at the same time in the environment increasing execution duration uncertainty considerably). In 6 of these zones will be actual machines. Therefore, the task is to efficiently assign the three robots to the 12 zones, identify the zones which contain a machine, and the precisely determine the position of the machine and recognize a industrial light signal for verification.

4 SMT-based Planning and Optimization

Satisfiability Modulo Theory (SMT) is the problem of deciding the satisfiability of a first-order formula with respect to some decidable theory $\mathcal{T}$. In particular, SMT generalizes the Boolean satisfiability problem (SAT) – see (Franco and Martin 2009) for an overview – by adding expressive background theories such as the theory of real numbers, the theory of integers, and the theories of data structures (e.g., lists, arrays and bit vectors). The idea behind SMT is that the satisfiability of a constraint network can be decided in two steps, as shown in Figure 2. Given an input formula φ in Conjunctive Normal Form (CNF), a Boolean abstraction is first built, e.g., the formula

$$x \geq y \wedge (y > 0 \vee x > 0) \wedge y \leq 0$$

is converted as follows

$$A \wedge (B \vee C) \wedge \neg B$$

where $A, B, C \in \{0, 1\}$ and "$\neg$" is the symbol for Boolean negation. A specialized Boolean satisfiability (SAT) solver can now be invoked to compute an assignment that satisfies all the constraints. In the example above, $A = 1, B = 0, C = 1$ satisfies all the constraints. Notice that if no such assignment exists, then the set of arithmetic constraints is trivially unsatisfiable. The second step is to check the consistency of the assignment in the corresponding background theory. In our example, we therefore need to check whether the system of linear inequalities

$$\begin{cases} x \geq y & (A) \\ y \leq 0 & (\neg B) \\ x > 0 & (C) \end{cases}$$

is feasible, which is clearly the case here. Checking that the Boolean assignment is feasible in the underlying mathematical theory can be performed by a specialized reasoning procedure, i.e., SMT solvers like `Z3` (de Moura and Bjørner 2008), `SMT-Rat` (Corzilius et al. 2015), `MathSAT5` (Cimatti et al. 2013) and `Yices2` (Dutertre 2014). If the constraints are consistent in the theory and the SAT solver's assignment is already complete then, a satisfying solution (also called *model*) is found for the input formula. If the consistency check fails the theory solver returns an *explanation* for the conflict (i.e., the infeasible subset of input constraints), then a new Boolean assignment is requested and the SMT solver goes on until either an theory-consistent Boolean assignment is found, or no more Boolean assignments can be found.

Yet, determining whether a set of constraints is satisfiable or not might not be sufficient, especially when planning is involved. This problem being well recognized in the SMT community, state-of-the-art solvers have introduced the possibility to state and solve optimization objective. Back to our working example, we may want to find solutions that, e.g., maximize y. Given our quantifier free formula φ and the objective function f, we first compute and assignment that satisfies φ (say, e.g., $x = 3$ and $y = -2$). Then, use this assignment to compute a local optimum opt for f, which is trivial in our case as our only objective is to maximize y. Now, we can search for the next assignment that improves the current solutions by feeding the solver with φ updated as follows

$$\varphi \leftarrow \varphi \wedge f > opt \wedge \neg(x = 3 \wedge y = -2)$$

Repeating this procedure discarding all previously computed assignments will eventually lead to the assignment $y = 0$.

Planning with SMT Solving

Over the last decade, SMT solvers have emerged as a core technology in many areas that require analysis of large state spaces – see (Ábrahám and Kremer 2016) for an overview. This is because large, even infinite, sets of system states can be compactly represented as formulas in first-order logic, which may lead to improvements in the scalability of state space analysis. Moreover, compared to traditional SAT solvers, SMT solvers provide a more expressive interface with useful features for expressing constraints in robotics domains. For instance, Nedunuri et al. (2014) and Wang et al. (2016) use an SMT solver to generate task and motion plans from a static roadmap, employing plan outlines to

guide the planning process. Dantam et al. (2016) propose an algorithm to perform task and motion planning which leverages incremental solving in `Z3` to update constraints about motion feasibility. (Saha et al. 2014) present a motion planning framework where SMT solving is used to combine motion primitives so that they satisfy safety requirements specified in linear temporal logic (LTL). SMTPlan (Cashmore et al. 2016) takes a PDDL-based model for a hybrid task and translates it into a SMT problem through a generic transformation. In contrast, we aim to directly model problems using SMT. Especially when using optimization, more direct control and possibly other ways of formulation that are not action-based might allow for more tuning and better results. SMTPlan could be used as a pre-processor to generate a SMT formulation which can then be modified and improved , e.g., enriching it with more (potentially domain-specific) constraints to increase efficiency.

The key differences between the works listed above and ours are that *i.* we do not use additional knowledge (i.e., motion primitive, plan outlines) to seed the search performed by the SMT solver *ii.* we exploit the optimization features offered by solver to synthesize plans that are not only *feasible*, but also *optimal* with respect to some criteria.

Encoding

Formal Description The planning task introduced in Section 3 can be formalized as follows. For each robot $i \in \{1, 2, 3\}$ and each step $j \in \{-3, M\}$ belonging to the path of robot i, $pos_{i,j} = k$ states that robot i visits a potential machine k in the $j-$th step of his plan. Note that locations $-3, -2, -1$ are used to represent the initial location of each robot and 0 the start position. Furthermore, we introduce a stop location -4 to represent the moment when a robot does not move anymore, and $d_{i,j} \in \mathbb{R}$ to store the distance traveled by each robot until step j. Finally, let m_i be a Boolean variable used to represent the maximum over distances $d_{i,M} \quad \forall i$. As our task is to identify all machines in the environment within a time limit, we start by specifying the optimization objectives as

$$\text{minimize} \quad \sum_{i=1}^{3} m_i * d_{i,M}$$

$$\text{minimize} \quad \sum_{i=1}^{3} d_{i,M}$$

so as to minimize the total traveled distance, while minimizing the maximum distance traveled by each robot at the same time. Our problem is then initialized according to the following constraints:

$$\begin{aligned} pos_{i,0} &= 0 \quad \forall i = 1, 2, 3 \\ pos_{i,-1} &= -1 \quad \forall i = 1, 2, 3 \\ pos_{1,-3} &= pos_{1,-2} = pos_{1,-1} \\ pos_{2,-2} &= pos_{2,-3} = pos_{3,-2} = -2 \\ pos_{3,-3} &= -3 \\ d_{i,0} &= 0 \quad \forall i = 1, 2, 3 \end{aligned}$$

The above constraints simply fix an order on the way robots can start moving considering physical constraints coming from their initial locations. Also, the cost of reaching the start location is assumed to be 0. Once reached the start location, each robot will start the exploration phase. The choice of the first machined to be visited is modeled according to

$$\forall i = 1, 2, 3 \quad \bigvee_{k=1}^{M} \Big[pos_{i,1} = k \wedge d_{i,1} = dist(0, k) \Big]$$

meaning that for all robots i, the destination of the first visit can be chosen among all k machines and the individual cumulative distance needs to be updates accordingly. Successive steps are modeled as follows

$$\begin{aligned} &\forall i = 1, 2, 3 \quad \forall j = 2, ..., M \\ &\Big[\bigvee_{k=1}^{M} \bigvee_{\substack{l=1 \\ l \neq k}}^{M} pos_{i,j-1} = k \wedge pos_{i,j} = l \wedge d_{i,j} = d_{i,j-1} + dist(k, l) \Big] \\ &\vee \Big[pos_{i,j} = -4 \wedge d_{i,M} = d_{i,j-1} \Big] \end{aligned}$$

where for all robots i and planning steps j (starting from the second) a robot can either decide to move from machine k to machine l and update the cumulative distance accordingly, or to stop moving at all. In the latter case, the robot is set to stop and the cumulative is not increased anymore.

In order to speed up the search for an optimal solution, we enforce that each machine can be visited only once using

$$\begin{aligned} &\forall k = 1, ..., M \\ &\bigvee_{i=1}^{3} \bigvee_{j=1}^{M} \Big[pos_{i,j} = k \wedge \bigwedge_{u=1}^{3} \bigwedge_{v=1}^{M} ((i = u \wedge j = v) \vee pos_{u,v} \neq k) \Big] \end{aligned}$$

In short, this constraint encodes that if a robot i is visiting a machine k in its $j-$th step, then for all other robots $u \neq i$ and respective planning steps $v \neq j$, machine k cannot be visited. In order to encode $\max_i d_{i,M}$ we specify the following constraints

$$\forall i = 1, 2, 3 \quad \Big[m_i = 0 \vee (m_i = 1 \wedge \bigwedge_{\substack{l=1 \\ l \neq i}}^{3} d_{l,M} < d_{i,M}) \Big]$$

which enforces that variable m_i be set to `true` only if the corresponding distance $d_{i,M}$ is greater than other robots'.

Practical notes on the encoding The problem as described above can be written in `SMT-LIB` format (Barrett, Stump, and Tinelli 2010) and later fed to a solver. Variables can be declared as follows

```
(declare-fun d_1_1 () Real)
(declare-fun pos_1_1 () Int)
```

while constraints can be asserted using prefix notation as

```
(assert (= (* pos_1_1 d_1_1) 3))
```

The optimization objectives introduced before can be specified as

```
(minimize (+ (* m_1 d_1_12) ...))
(minimize (+ d_1_12 d_2_12 d_3_12))
```

Notice that multiple optimization objectives are handled differently by different solvers. `Z3`, the solver we used, solves objectives in lexicographical order by default (other strategies are allowed). This means that objectives are solved in the order they are specified in the `SMT-LIB` file. We can finally ask the solver to check whether the constraints are satisfiable by stating `(check-sat)`.

If a model can be found, we can traverse it so as to collect all variable assignments relevant to our plan. In our case, the solution obtained will have elements of the form `(pos_1_1 5)`. Converting such solution into a plan is now an easy task as a result of how the problem was originally formulated. Recall that the assignment $pos_{i,j} = k$ states that robot i visits machine k at the $j-$th step of its plan. Given that the only action considered in this problem is `move`, the sample solution above can be read as "the $j-$th action executed by robot i is `move` to machine k".

5 CLIPS-based Execution and Monitoring

The goal of our executive is to allow for a flexible formulation of the plan execution that can work completely automatic given a plan, but can be extended for monitoring the execution and listening for events. The overall architecture is depicted in Figure 3. The CLIPS executive controls the overall execution. At a suitable time (when the game enters the appropriate phase indicated by the referee box), it triggers the SMT solving process to come up with a plan to explore the machines and encodes the available knowledge in a message. The solver queries travel times of the position for exploration from a navigation graph and uses a domain model phrased suitably for SMT. The result is then represented as a plan and sent to the CLIPS executive, which must translate it into a native representation for execution. This is then synchronized with all robots as part of the world model. The robots then execute their respective partial plans by invoking the appropriate basic behaviors through the behavioral and functional components of the Fawkes software framework (BE represents the Lua-based Behavior Engine that provides the basic skills to execute the plans, for details see below). Only through this framework do the reasoning systems interact with the simulation (which would make it easy to replace this with actual robots later).

In the following, we are going to describe the CLIPS rules engine which is used to implement the executive before describing the plan translation and execution in more detail.

CLIPS Rules Engine

CLIPS (Wygant 1989) is a rule-based production system using forward chaining inference based on the Rete algorithm (Forgy 1982). It has been used before for autonomous robot reasoning (Niemueller, Lakemeyer, and Ferrein 2013), knowledge-based instrumentation and control (Niemueller et al. 2016c), and inspired other systems such as Jess (Friedman-Hill 1999). CLIPS consists of three building blocks (Giarratano 2007): a fact base or working memory, the knowledge base, and an inference engine. *Facts* are basic forms representing pieces of information in the fact base. They usually adhere to structured types. The

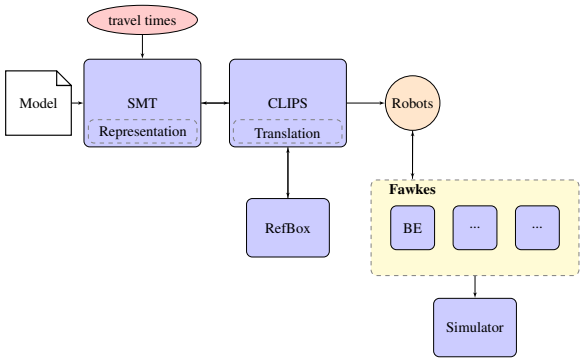

Figure 3: The overall architecture.

knowledge base comprises heuristic knowledge in the form of rules, and procedural knowledge in the form of functions. *Rules* are a core part of the production system. They are composed of an antecedent and consequent. The antecedent is a set of conditions, typically patterns which are a set of restrictions that determine which facts satisfy the condition. If all conditions are satisfied based on the existence, nonexistence, or content of facts in the fact base the rule is activated and added to the agenda. The consequent is a series of actions which are executed for the currently selected rule on the agenda, for example to modify the fact base. *Functions* carry procedural knowledge and may have side effects. They can also be implemented in C++. In our framework, we use them to utilize the underlying robot software, for instance to communicate with the reactive behavior layer described below. CLIPS' *inference engine* combines working memory and knowledge base performing fact updates, rule activation, and agenda execution until stability is reached and no more rules are activated. Modifications of the fact base are evaluated if they activate (or deactivate) rules from the knowledge base. Activated rules are put onto the agenda. As there might be multiple active rules at a time, a conflict resolution strategy is required to decide which rule's actions to execute first. In our case, we order rules by their salience, a numeric value where higher value means higher priority. If rules with the same salience are active at a time, they are executed in the order of their activation.

Planning and Plan Translation

To start planning, the information relevant to the planning procedure must be encoded in a way accessible to the planning system. In this work, we have used Google Protocol Buffers[2] (protobuf) to represent both, the initial planning situation as well as the resulting plan. Protobuf defines a schema for messages (for an example cf. Listing 2, explained in more detail below) for which code can be generated for a variety of languages. It provides a convenient transport, exchange, and storage representation that is easy to create and read. It also has powerful introspection capabilities which are particularly useful for generic access from typical reasoning systems, for example, the CLIPS-based access requires only the definition files and not any

[2] `https://developers.google.com/protocol-buffers/`

```
1  (defrule production-call-clips-smt
2    (phase PRODUCTION)
3    (team-color ?team-color&CYAN|MAGENTA)
4    (state IDLE)
5    (not (plan-requested))
6    (test (eq ?*ROBOT-NAME* "R-1"))
7  =>
8    (bind ?p
9      (smt-create-data
10       (smt-create-robots ?team-color)
11       (smt-create-machines ?team-color)
12       (smt-create-orders ?team-color)
13     )
14   )
15   (smt-request "explore-zones" ?p)
16   (assert (plan-requested))
17 )
```

Listing 1: CLIPS rule to trigger planning.

```
1  message ActorGroupPlan {
2    repeated ActorSpecificPlan plans = 1;
3  }
4  message ActorSpecificPlan {
5    required string  actor_name = 1;
6
7    oneof plan {
8      SequentialPlan sequential_plan = 2;
9      TemporalPlan   temporal_plan = 3;
10   }
11 }
12 message SequentialPlan {
13   repeated PlanAction actions = 1;
14 }
15 message PlanAction {
16   required string name = 1;
17   repeated PlanActionParameter params = 2;
18 }
19 message PlanActionParameter {
20   required string key = 1;
21   required string value = 2;
22 }
```

Listing 2: Plan representation in protobuf.

pre-generated code. The rule to initiate the planning process in shown in Listing 1. Here, once the game is started (ll. 2–4), the first robot (l. 6) will create the data structure with the relevant information (ll. 8–14, the `smt-create-*` calls are functions) and request a plan from the SMT solver (l. 15).

The SMT side notifies the executive once the plan is ready for retrieval. An excerpt of the message specifications for plan representation is shown in Listing 2. First, a list of plans that name a specific actor (robot) is defined in ll. 1–3. Lines 5–12 define such a plan. It names the actor for the plan and either a sequential or a temporal plan (`oneof` clause). For reasons of brevity, we focus on the sequential plan consisting of a series of actions (ll. 14–16). An action is defined by an operator name and a number of key-value pairs for the arguments (ll. 18–26). An example plan for two robots with two movements commands in shown in Listing 3.

Once the plan has been retrieved, it must be translated into the native CLIPS representation, that is as facts in the working memory. This translation in general is a straight-forward one. But it may require a domain-specific mapping from action names and parameters in the planner's representation to actions understood by the underlying base system. For example, in our system the "move" skill is called "drive_to". This conversion must be performed during translation of the plan. Likewise, differences in action parameters might have to be translated. Example facts for the translation of the plan is shown in Listing 4.

```
1  plans[0]:ActorSpecificPlan {
2    actor_name: "R-1"
3    sequential_plan:SequentialPlan {
4      actions[0]:PlanAction {
5        name: "move"
6        params[0]:PlanActionParameter {
7          key: "to"
8          value: "C-BS-I"
9        }
10     }
11     actions[1]:PlanAction {
12       name: "move"
13       params[0]:PlanActionParameter {
14         key: "to"
15         value: "C-DS-I"
16       }
17     }
18   }
19 }
20 plans[1]:ActorSpecificPlan {
21   actor_name: "R-2"
22   sequential_plan:SequentialPlan {
23     actions[0]:PlanAction {
24       name: "move"
25       params[0]:PlanActionParameter {
26         key: "to"
27         value: "C-CS1-I"
28       }
29     }
30     actions[1]:PlanAction {
31       name: "move"
32       params[0]:PlanActionParameter {
33         key: "to"
34         value: "C-RS2-I"
35       }
36     }
37   }
38 }
```

Listing 3: Plan represented through the messages from Listing 2 (shown in augmented JavaScript Object Notation).

The CLIPS representation denotes tasks as a number of consecutive steps that must be executed sequentially. There may be only a single task active per robot at a time. A step then triggers the execution of a basic behavior. If it completes successfully, the next step is executed, until all have been processed and the task is complete. If any of the steps fails the task is considered to have failed.

Plan Distribution and Execution

Once the plan has been added to the working memory, it has to be distributed to the robots for execution. In the current

```
(task (task-id 1910) (robot "R-1") (name explore)
      (state proposed) (steps 1911 1912))
(step (id 1911) (name drive-to) (state inactive)
      (machine C-BS) (side INPUT)
      (sync-id (next-sync-id)))
(step (id 1912) (name drive-to) (state inactive)
      (machine C-DS) (side INPUT)
      (sync-id (next-sync-id)))
```

Listing 4: Task representation in CLIPS.

version, we rely on the communication infrastructure otherwise used to share world model updates among the robots. It encapsulates fact base updates in protobuf messages and broadcasts them to the other robots. A (dynamically elected) master generates a consistent view and distributes it to the robots. We deem the plans as being part of the world model.

On each robot, the CLIPS executive has rules that automatically start new tasks. This happens once the plans have been received as world model updates. Since the actor is named in these plans, a robot will only ever execute its own plan. Updates to the tasks (e.g., that a certain task is currently in progress) is again distributed in the world model and hence the planning and execution instance remains informed of the state of the plan execution.

Basic behaviors in the current framework are provided by the Lua-based Behavior Engine (Niemueller, Ferrein, and Lakemeyer 2009), but could principally be provided by other sources, for example through ROS actions or the ROSPlan action dispatching mechanism. A step is executed by triggering the start of the execution of a behavior, i.e., executing a skill is an instant non-blocking operation. Then, information about the execution of the state is read and asserted in the working memory. Once it completes, a rule fires that denotes a step to be finished.

Execution Monitoring

Especially in robotics, execution often does not go as planned. Some actions may (and eventually some will) fail or not entirely give the expected result, e.g., by misplacing an object, or slack during execution could render a plan invalid, for example if a specified deadline cannot be made.

As described earlier, plans are translated into tasks and actions into steps of the task. Steps are invoked non-blocking, i.e., rule evaluation continues normally. This can be used to implement execution monitoring. Rule can be phrased identifying specific situations of interest where a step should be skipped or a task being aborted.

6 Preliminary Results

Even though this work is in an early stage of development, we have an initial fully integrated prototype. It is based on the planning competition reference architecture and extends it by a plugin to integrate the SMT-based planning functionality. Then, the existing code has been modified to call an SMT component with the relevant information. This component will generate a formula (according to Section 4), call the Z3 solver, and translate and execute the resulting plan with multiple robots in simulation. So far, we have used only the generic capabilities of the executive and do not have, for example, specific execution monitoring rules to react to unexpected (yet foreseen) situations.

We have also integrated the executive with other reasoning system based on Answer Set Programming and PDDL-based planning systems. However, work there is still in an earlier stage preventing a comparison between the approaches at this time.

7 Conclusion

We have shown two bodies of work and their integration: first, the CLIPS rules engine has been used to create a flexible and expressive executive for plans. Second, an SMT-based approach has been used to model and execute the task planning and optimization in an explorative setting. These two elements have been combined by exchanging information through well-structured protobuf messages. The work is in an early stage, but a fully working prototype has been developed and the full system has been run and tested. The executive is in control of the overall system and triggers the planning process as necessary. Then, the resulting plan is translated into a native representation, distributed among the robots, and executed concurrently by the individual robots.

Future work will deal with extending the executive model to be more flexible, complete and thoroughly test the integration with other reasoning approaches, and optimize and compare the SMT-based planning approach with these.

Acknowledgments. This work is supported by the ICT project house Foundations of a Digitized Industry, Economy, and Society of RWTH Aachen University.

T. Niemueller is supported by the German National Science Foundation (DFG) research unit *FOR 1513* on Hybrid Reasoning for Intelligent Systems.

References

[Ábrahám and Kremer 2016] Ábrahám, E., and Kremer, G. 2016. Satisfiability checking: Theory and applications. In *Int. Conf. on Software Engineering and Formal Methods*.

[Barrett, Stump, and Tinelli 2010] Barrett, C.; Stump, A.; and Tinelli, C. 2010. The SMT-LIB Standard: Version 2.0. In *8th Int. Workshop on Satisfiability Modulo Theories*.

[Berry and Gonthier 1992] Berry, G., and Gonthier, G. 1992. The Esterel synchronous programming language: design, semantics, implementation. *Science of computer programming* 19(2).

[Cashmore et al. 2015] Cashmore, M.; Fox, M.; Magazzeni, D. L. D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *25th International Conference on Automated Planning and Scheduling (ICAPS)*.

[Cashmore et al. 2016] Cashmore, M.; Fox, M.; Long, D.; and Magazzeni, D. 2016. A Compilation of the Full PDDL+ Language into SMT. In *26th International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*.

[Cimatti et al. 2013] Cimatti, A.; Griggio, A.; Schaafsma, B. J.; and Sebastiani, R. 2013. The MathSAT5 SMT Solver. In *19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS*.

[Corzilius et al. 2015] Corzilius, F.; Kremer, G.; Junges, S.; Schupp, S.; and Ábrahám, E. 2015. SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving. In *18th International Conference on Theory and Applications of Satisfiability Testing SAT*.

[Dantam et al. 2016] Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki, L. E. 2016. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems XII*.

[de Moura and Bjørner 2008] de Moura, L. M., and Bjørner, N. 2008. Z3: an efficient SMT solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

[Dornhege et al. 2009] Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic Attachments for Domain-Independent Planning Systems. In *19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Dutertre 2014] Dutertre, B. 2014. Yices 2.2. In *26th International Conference on Computer Aided Verification (CAV)*.

[Forgy 1982] Forgy, C. L. 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence* 19(1).

[Franco and Martin 2009] Franco, J., and Martin, J. 2009. *Handbook of Satisfiability*. IOS Press. chapter A History of Satisfiability.

[Friedman-Hill 1999] Friedman-Hill, E. J. 1999. Jess, the java expert system shell. Technical report, Sandia National Laboratories.

[Giarratano 2007] Giarratano, J. C. 2007. *CLIPS Reference Manuals*. `http://clipsrules.sf.net/OnlineDocs.html`.

[Hofmann et al. 2016] Hofmann, T.; Niemueller, T.; Claßen, J.; and Lakemeyer, G. 2016. Continual Planning in Golog. In *30th Conference on Artificial Intelligence (AAAI)*.

[Ingham, Ragno, and Williams 2001] Ingham, M.; Ragno, R.; and Williams, B. 2001. A Reactive Model-based Programming Language for Robotic Space Explorers. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*.

[Ingrand et al. 1996] Ingrand, F.; Chatila, R.; Alami, R.; and Robert, F. 1996. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, volume 1.

[McDermott et al. 1998] McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. Technical report, AIPS-98 Planning Competition Committee.

[Nedunuri et al. 2014] Nedunuri, S.; Prabhu, S.; Moll, M.; Chaudhuri, S.; and Kavraki, L. E. 2014. Smt-based synthesis of integrated task and motion plans from plan outlines. In *IEEE Int. Conference on Robotics and Automation (ICRA)*.

[Niemueller et al. 2016a] Niemueller, T.; Karpas, E.; Vaquero, T.; and Timmons, E. 2016a. Planning Competition for Logistics Robots in Simulation. In *WS on Planning and Robotics (PlanRob) at Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Niemueller et al. 2016b] Niemueller, T.; Neumann, T.; Henke, C.; Schönitz, S.; Reuter, S.; Ferrein, A.; Jeschke, S.; and Lakemeyer, G. 2016b. Improvements for a Robust Production in the RoboCup Logistics League 2016. In *RoboCup Symposium – Champion Teams Track*.

[Niemueller et al. 2016c] Niemueller, T.; Zug, S.; Schneider, S.; and Karras, U. 2016c. Knowledge-Based Instrumentation and Control for Competitive Industry-Inspired Robotic Domains. *KI - Künstliche Intelligenz* 30(3).

[Niemueller, Ferrein, and Lakemeyer 2009] Niemueller, T.; Ferrein, A.; and Lakemeyer, G. 2009. A Lua-based Behavior Engine for Controlling the Humanoid Robot Nao. In *RoboCup Symposium 2009*.

[Niemueller, Lakemeyer, and Ferrein 2013] Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2013. Incremental Task-level Reasoning in a Competitive Factory Automation Scenario. In *AAAI Spring Symposium 2013 - Designing Intelligent Robots: Reintegrating AI*.

[Niemueller, Lakemeyer, and Ferrein 2015] Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2015. The RoboCup Logistics League as a Benchmark for Planning in Robotics. In *25th International Conference on Automated Planning and Scheduling (ICAPS) – WS on Planning in Robotics*.

[Saha et al. 2014] Saha, I.; Ramaithitima, R.; Kumar, V.; Pappas, G. J.; and Seshia, S. A. 2014. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Verma et al. 2005a] Verma, V.; Estlin, T.; Jónsson, A.; Pasareanu, C.; Simmons, R.; and Tso, K. 2005a. Plan Execution Interchange Language (PLEXIL) for Executable Plans and Command Sequences. In *9th Int. Symposium on Artificial Intelligence, Robotics and Automation in Space*.

[Verma et al. 2005b] Verma, V.; Jónsson, A.; Simmons, R.; Estlin, T.; and Levinson, R. 2005b. Survey of Command Execution Systems for NASA Spacecraft and Robots. In *Int. Conference on Automated Planning and Scheduling (ICAPS) – Workshop "Plan Execution: A Reality Check"*.

[Verma et al. 2006] Verma, V.; Jónsson, A.; Pasareanu, C.; and Iatauro, M. 2006. Universal Executive and PLEXIL: Engine and Language for Robust Spacecraft Control and Operations. In *American Institute of Aeronautics and Astronautics – Space 2006*.

[Wang et al. 2016] Wang, Y.; Dantam, N. T.; Chaudhuri, S.; and Kavraki, L. E. 2016. Task and motion policy synthesis as liveness games. In *26th International Conference on Automated Planning and Scheduling (ICAPS)*.

[Wygant 1989] Wygant, R. M. 1989. CLIPS: A powerful development and delivery expert system tool. *Computers & Industrial Engineering* 17(1–4).