A Multi-Agent Extension of PDDL3.1

Daniel L. Kovacs
Budapest University of Technology and Economics
Budapest, HUNGARY
dkovacs@mit.bme.hu

Abstract

Despite a recent increase of research activity in the field of multi-agent planning there is still no de-facto standard for the description of multi-agent planning problems similarly to the Planning Domain Definition Language (PDDL) in case of deterministic single-agent planning. For this reason, in this paper a multi-agent extension of the currently latest official version of PDDL (3.1) is proposed together with a corresponding multi-agent planning track for the International Planning Competition (IPC). Our aim is to allow for a more direct comparison of planning systems and approaches, a greater reuse of research, and a more coordinated development in the field. Multi-agent planning is fundamentally different from the single-agent case with a broad range of applications (e.g. multi-robot domains). Not only is it inherently harder because of an exponential increase of the number of actions in general, but among others also constructive/destructive synergies of concurrent actions, and agents' different abilities and goals may need to be considered. The proposed multi-agent extension copes with these issues and allows planning both for and by agents even in temporal, numeric domains. It implies minimal changes to the syntax of PDDL3.1 and the related parsers.

1. Introduction

Multi-agent planning (de Weerdt and Clement 2009) is about planning by N planning agents for M executing agents (or actors, actuators, bodies) situated in a multi-agent environment, with a broad range of applications. Planning and executing agents may be the same or separate entities. Executing agents are always situated in the environment, while planning agents may be external to it. However the most typical scenario is either when an external agent is planning for a group of situated agents ($N = 1, M > 1$), or when there is a group of autonomous, situated planning-and-executing agents ($N = M > 1$). In general four cases can be distinguished (cf. Table 1).

In cases where $M > 1$ the control of multiple executing agents may be *centralized* or *decentralized*. A typical case of *distributed* planning is when $N > 1$ agents plan for

N \ ***M***	**=1**	**>1**
=1	single-agent planning for a single agent	single-agent planning for multiple agents
>1	multi-agent planning for a single agent	multi-agent planning for multiple agents

Table 1: a general categorization of multi-agent planning

$M = 1$ agent. Planning can be done *on-line* or *off-line*, and agents and environments can correspond to types mentioned in Chapter 2 in (Russell and Norvig 2010).

Multi-agent planning is inherently harder than single-agent planning because agents may act independently and thus the number of possible actions in general is exponential (combinations of individual actions need to be considered). Moreover agents may be heterogeneous; they may have different abilities, contradicting goals or asymmetric beliefs; they may require coordination of plan execution, communication or synchronization of concurrent actions; constructive/destructive interference of joint actions may arise (joint actions may produce different effects from the union of effects of their parts); cooperation and self-interest, goals of teams and individuals may need to be conciliated; and the level of coupling between agents is also important. Thus single-agent planning can't be directly applied to multi-agent planning problems.

Research in the field of multi-agent planning was focusing recently mainly on the following topics.

- *scaling up the performance* of planners, e.g. (Shah, Conrad, and Williams 2009; Stefanovitch et al. 2011; Jonsson and Rovatsos 2011; Kumar, Zilberstein, and Toussaint 2011; Spaan, Oliehoek, and Amato 2011);
- coping with *more realistic domains*, e.g. (Beaudry, Kabanza, and Michaud 2010; Pajarinen and Peltonen 2011; Zhuo and Li 2011; Wang and Botea 2011; Fox, Long, and Magazzeni 2011);

- *improving solution quality*, e.g. (Yabu, Yokoo, and Iwasaki 2009; Marecki and Tambe 2009);
- *exploiting problem structure* (Brafman and Domshlak 2008; Nissim, Brafman, and Domshlak 2010);
- utilizing *learning*, e.g. (Martins and Demiris 2010; Zhuo, Muñoz-Avila, and Yang 2011);
- reasoning about *agents' knowledge*, e.g. (Baral et al. 2010; Baral and Gelfond 2011);
- and addressing *agents' self-interest*, e.g. (Brafman et al. 2009; Crosby and Rovatsos 2011).

The problem is that despite all this progress there is still no standard description language for multi-agent planning problems allowing a more direct comparison of systems and approaches and a greater reuse of research similarly to the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) in single-agent planning, a base language of the International Planning Competition (IPC).

Naturally there were some previous approaches (cf. Section 2.2), but none of these languages became de-facto standards probably partly because of their limitations. On the other hand PDDL is not enough to describe multi-agent planning problems in general (e.g. possibly different goals and utilities of different agents, synergy of joint-actions).

To address these issues, in this paper a *multi-agent extension of PDDL3.1* is proposed, which is currently the latest official version of PDDL (Helmert 2008), and based on this extension, ideas for a corresponding *multi-agent planning track* are also proposed for the upcoming IPCs.

The structure of the paper is as follows: after Section 1 discusses the motivation behind the proposed approach, Section 2 examines its background; Section 3 presents the main result of the paper, the formal syntax and informal semantics of the proposed multi-agent extension of PDDL3.1 and an example; Section 4 discusses ideas for a multi-agent planning track at the upcoming IPCs based on the proposed extension; finally Section 5 concludes the work and outlines some directions for future research.

2. Background

This section examines some of the considerations and decisions behind the proposed multi-agent extension of PDDL3.1. Namely it discusses **(1)** some minor corrections of PDDL3.1's syntax, **(2)** previously published multi-agent planning problem description languages, and **(3)** requirements of an appropriate multi-agent extension.

2.1. Corrections of PDDL3.1's syntax definition

Since PDDL3.1 was chosen as the basis of the multi-agent extension, a complete and correct BNF (Backus-Naur Form) definition of its syntax becomes necessary, which was made available in (Kovacs 2011). It makes mainly the following minor corrections to previously published BNF.

- The default type (of objects) in PDDL is `object`, but until now this was not made explicit in the grammar. Accordingly the next rule should be **added** to the BNF.

```
<primitive-type> ::= object
```

- Similarly the definition of the built-in 2-ary = predicate in case of the `:equality` requirement was also left out from previous definitions of PDDL. To correct this, the following rule needs to be **added** to the BNF.

```
<atomic formula(t)> ::=:equality (= t t)
```

- Since PDDL2.1 (Fox and Long 2003) function-expressions in the domain description allowed only 2-argument numeric operators, although in the problem description, in the definition of `metric` they could be also multi-argument. To fix this, the following two production rules should be **added** to the grammar.

```
<f-exp>  ::=:numeric-fluents
        (<multi-op> <f-exp> <f-exp>+)
<f-exp-da> ::=:numeric-fluents
        (<multi-op> <f-exp-da> <f-exp-da>+)
```

- The definition of non-terminals `<name>` and `<number>` was underspecified until now, so it is suggested to **define** them more precisely, for example as shown in the Appendix in (Teichteil-Königsbuch 2008).
- The following rule would allow durative actions to have *non temporally annotated* numeric effects, which would contradict the specification of durative actions in (Fox and Long 2003). Thus it needs to be **deleted**.

```
<da-effect> ::=:numeric-fluents
          (<assign-op> <f-head> <f-exp-da>)
```

- The following rule is present in the BNF of PDDL2.1 and PDDL3.0 (Gerevini and Long 2005). The problem with it is that `<a-effect>` is not defined anywhere, so `<a-effect>` should be **changed** to `<cond-effect>`. Otherwise `<p-effect>` or `<effect>` may also be considered, but the former would not allow conjunctions of propositions, while the latter would overly complicate the syntax and allow semantically ambiguous constructs (e.g. nested conditional effects).

```
<timed-effect> ::=
          (at <time-specifier> <a-effect>)
```

- Production rules for non-terminals `<assign-op-t>` and `<f-exp-t>` are referenced, but missing from the BNF since PDDL3.0 in the form they were given in the BNF of PDDL2.1. They should be **included** again.
- The following rule defines the syntax of derived predicates since PDDL2.2 (Edelkamp and Hoffmann 2004a). The problem with it is that there is no mention of the name of the derived `<predicate>`. Thus instead of `<typed list (variable)>` (Edelkamp and Hoffmann 2004b) would suggest `<atomic formula(term)>`, which is better, since it includes the

name of the derived predicate, but then there are no argument-types in the head of the derived-rule as one might expect in case of `:typing`. To include both the name of the predicate and the type of its arguments `<atomic formula skeleton>` should be **used instead** of `<typed list (variable)>` below.

```
<derived-def> ::=
     (:derived <typed list (variable)> <GD>)
```

- The following rule is present in the initial conditions part of the problem description since PDDL2.1, but it is incorrect, since `<f-head>` may be lifted, although it should be grounded. To fix this `<basic-function-term>` can be **used instead** of `<f-head>` below.

```
<init-el> ::=:numeric-fluents
                     (= <f-head> <number>)
```

- PDDL3.0 introduced plan constraints via modal operators at the 5th IPC in 2006, but they were not allowed to be nested at the time of the competition. Nonetheless this restriction could be lifted by using production rules provided in Section 3 in (Gerevini and Long 2005). The problem with those rules, which are still part of the BNF, is that they do not allow a normal end to the recursive nesting of modal operators. This needs to be **corrected**, e.g. as given in (Kovacs 2011).

2.2. Previous approaches

Previous multi-agent planning problem description languages provided valuable experience and ideas for the design of the proposed multi-agent extension of PDDL3.1. In the following an overview of these languages is given.

2.2.1. Non-deterministic Agent Domain Language

The Non-deterministic Agent Domain Language (NADL) introduced in (Jensen and Veloso 2000) is suitable for describing multi-agent planning domains to a limited extent. It could be seen a predecessor of numeric fluents of PDDL2.1, but its syntax differs significantly from PDDL.

In NADL each explicitly given agent is a collection of actions that have preconditions and effects (numeric and/or propositional formulas). Actions can also refer to state variables they constrain. These constraints are then used in planning time to avoid joint actions that have destructive synergetic effects, i.e. which constrain an overlapping set of state variables (e.g. actions that assign different values to a numeric fluent). Constructive interferences on the other hand are not modeled. This makes for a relatively simple model of interactions among concurrent actions.

However NADL allows a distinction between system and environment agents, the latter being non-controllable and thus responsible for possible non-deterministic effects.

NADL's model of time is discrete. Actions have equal duration and each agent can execute only one action at a time. All agents share the same goal. Later in (Bowling, Jensen, and Veloso 2002) this was extended to multiple agents having possibly different goals, but no accompanying description language was provided, and the model was applicable only to propositional domains.

2.2.2. Concurrent interacting actions in STRIPS

This multi-agent extension of STRIPS (Boutilier and Brafman 2001) provides a more elaborate way to model interactions of concurrent actions than NADL based on the idea of *concurrent action lists*. Essentially the same (but a bit simplified) idea is presented in Section 11.4.1 in (Russell and Norvig 2010). Concurrent action lists refer to state variables and concurrently executed actions in form of separate lists attached to actions' preconditions or to conditional effects' conditions. In Section 2.2 in (Boutilier and Brafman 2001) they are described precisely as follows.

> If an action schemata A' appears in the concurrent action list of an action A then an instance of schema A' must be performed concurrently with action A in order to have the intended effect. If an action schema A' appears negated in the concurrent action list of an action A then no instance of schema A' can be performed concurrently with action A if A is to have the prescribed effect.

This is a generic and intuitive way to model interference of concurrent actions, however the implicit quantifiers over actions are a bit restrictive and the scope of quantification (the whole list) is also a bit broad. This could be improved by having explicit quantification, and including reference to concurrent actions directly in (pre)conditions.

Agents responsible for the execution of actions are referred to in form of variables (always the first parameter of an action). The only issue with this is that there is no typing, and thus no distinction between agents and objects. Effectively every object can be considered an agent (e.g. a planner may try to instantiate a `table` object in the first variable of a `pickup` action, which wouldn't make much sense). Otherwise the language is just like STRIPS: time is discrete, states are propositional. Moreover, each agent can execute only one action at a time (no parallel or partially ordered actions are allowed for one agent), and all share the same goal, i.e. only cooperative agents are modeled.

Despite all these limitations this language shows that a proper multi-agent extension can be achieved with minimal changes to a single-agent base language (STRIPS), and that the changes implied to planners may also be limited.

2.2.3. Multiagent Planning Language

Multiagent Planning Language (MAPL) was presented in (Brenner 2003a; 2003b) after interest in such an extension was coined in the Call for Contributions of the Workshop on PDDL at the ICAPS-03 conference. However there was no multi-agent planning track at any IPC ever since or before. Understanding all the reasons is beyond our scope, but some observations can still be made regarding MAPL.

MAPL builds upon PDDL2.1 and thus it includes PDDL2.1's main features (typing, numeric fluents, and durative actions), but at the same time it also makes quite drastic changes to the base language, which may be partly responsible for MAPL's limited success. Among others it abandons the closed-world assumption and instead of predicates it introduces *n*-ary state variables (which may be even `unknown`). This is done partly to cope with partial observability arising from multiple agents operating in the environment, but it also gives rise to the question, if e.g. actions' preconditions reflect an agent's knowledge necessary to execute the action, or *states of the "physical" environment* in which the action can be executed? In our interpretation the latter is closer to the design philosophy of PDDL, since "PDDL is intended to express the 'physics' of a domain" (McDermott et al. 1998). Moreover object-fluents added in PDDL3.1 allow for a very similar functionality without significant changes (to PDDL3.0).

MAPL also introduces a qualitative model of time, which was introduced in PDDL3.0 in a more concise form (of modal operators). This was necessary to coordinate multiple agents' behavior via *speech acts: fixed meta-actions*, whose definition is not part of the description.

Such coordination was necessary to synchronize actions or events in general with initially *uncertain duration* which is again a novelty of MAPL intended to allow greater realism and flexibility. An additional *control function* (for each agent) decides whether this duration is controlled by the environment or by the agent. Similarly there is also a *responsibility function*, which maps state variables to agents to represent which agent is responsible over a state variable. The mentioned additional functions are not part of the MAPL description, yet the definition of planning problems includes the control function for example, which may be confusing. These additions are effectively *advices* to the planner, which contradicts original intentions again: "We have endeavored to provide no advice at all as part of the PDDL notation" (McDermott et al. 1998).

Despite the above additions MAPL still handles the interaction of simultaneous actions similarly to NADL. "Two events are mutually exclusive (mutex) if one affects a state variable assignment that the other relies on or affects" (Brenner 2003a). This model avoids destructive synergetic effects, but isn't considering constructive ones.

In MAPL every agent may face a different planning problem, but all of them eventually share the same goals. In actions, agents are represented with variables like in Section 2.2.2, but they are handled just like any other parameter, which implies further questions, e.g.: *Can an action have more/no agent-parameters? Should the actor always be the first? Can an agent inherit an action defined for a parent-type? Can actions be redefined for children?*

Such questions may become important when a planner is being implemented and so they should be addressed together with a complete syntax definition at least. The reception of MAPL may have also been influenced by that it is not just a new requirement (as derived predicates or numeric fluents), but it is a new language, which is again not in accordance with some intentions behind PDDL, e.g. as stated in the Preface of the Proceedings of the Workshop on PDDL at ICAPS-03: "how the...development of PDDL can be managed within the community to ensure that it does not...fork into multiple incompatible directions...".

2.2.4. Concurrent STRIPS

Concurrent STRIPS (CSTRIPS) was proposed in (Oglietti and Cesta 2004). It is classic STRIPS with the addition of *concurrent threads*, which are explicitly declared, fixed subsets of action schemas. However they are defined only at model level without exact syntax (not even in examples).

Each agent and controllable environmental process can have a separate thread. The planner should find a sequence of fully instantiated actions for each thread based on their respective action-subsets to produce a joint-plan that achieves a common set of goals. While the simplicity of this approach may be tempting, it is not enough to describe challenging multiagent problems (e.g. action interactions).

2.2.5. MA-STRIPS

MA-STRIPS (Brafman and Domshlak 2008) is a multi-agent extension of STRIPS. Its idea is similar to CSTRIPS: partition different agents' grounded actions into disjoint subsets (corresponding to threads in Section 2.2.4). This however may raise implementation-level questions like: *Is it possible that different instantiations of the same operator-schema belong to different agents? If yes, then how should we represent this exactly, syntactically?*

Because of the similarity with CSTRIPS, eventually the same conclusions hold here too, but it must be noted, that the work of (Brafman and Domshlak 2008) focused mainly not on the subtleties of describing multi-agent planning problems, but given MA-STRIPS, a simple description language, they rather set out to formalize and efficiently exploit *loosely coupled agents*. They provided formal results to quantify the notion of agents' coupling and a centralized multi-agent planning algorithm that was shown to be polynomial in the size of the planning problem for fixed coupling levels. Their notions of *internal/public atoms/actions* and the *agent interaction digraph* could be extended to more complex descriptions straightforwardly, but the extension of their planning algorithm and the implied complexity results relying on these notions could be less trivial (e.g. extending them to actions with interacting effects, continuous time or competing goals).

2.3. Requirements of a multi-agent extension

Based on the observations made in previous sections the requirements of a multi-agent extension of PDDL3.1 can be summarized as follows. **In general** it should be...

- *Additional*: a new, additional, optional extension (a PDDL-requirement), not a completely new language;
- *Minimalistic*: introduce only minimal changes to the base language and try to minimize the modifications implied to existing planning systems and approaches;
- *Backward compatible*: compatible with every existing extension (PDDL-requirement) in the official language;
- *Forward compatible*: designed to be easily integrated with anticipated future extensions (e.g. partial observability, stochastic effects, events, processes);
- *General*: useful in all four general categories of multi-agent planning shown in Table 1;
- *Conforming*: in accordance with the design philosophy of the language, i.e. neutrally expressing the “physics” of the domain and including no advice for planners;
- *Compact*: the extended problem- and domain-description should include every model-level detail;
- *Well-defined*: has a complete and accessible definition of formal syntax and at least informal semantics;

In particular the multi-agent extension should allow...

- Modeling concurrent actions with interacting effects;
- Modeling competitive, cooperative or mixed domains;
- Agents having possibly different actions/goals/utilities;
- Straightforward association of agents and actions;
- Distinction between agents and non-agent objects;
- Inheritance/polymorphism of actions/goals/utilities;
- Different agents in different problems of a domain;
- Modeling full- and/or partial-observability;
- Optional use of any combination of PDDL3.1 features.

Optional communication of agents can be modeled by PDDL (by defining communicative actions in the domain), so communication can be realized in execution time. Similarly agents’ knowledge can be represented with domain-specific predicates and/or by using a PDDL-extension allowing for partial observability, if necessary. The proposed multi-agent extension should be *modular*, i.e. useable with such other extensions, e.g. partial-observability or probabilistic-effects. But in this paper now *we focus only on* extending PDDL3.1 to *multiple agents*.

3. A multi-agent extension of PDDL3.1

In the following the main result of this paper, a multi-agent extension of PDDL3.1 (MA-PDDL) is presented based on the requirements gathered in Section 2.3. First the syntax and semantics are given, then an example and a discussion of solutions. The section should be best read in conjunction with the BNF of PDDL3.1, e.g. in (Kovacs 2011).

3.1. Domain description

A new `:multi-agent` PDDL-requirement is introduced to indicate the presence of multiple agents in the domain. Agents are considered objects (or constants) that may have associated actions, goals and utilities (metric definitions). I.e. the idea is to associate actions, goals and utilities directly to objects and/or types (say, sets of objects).

The necessary changes implied to the grammar first include the following slight **modification** of the production rule defining *non-durative actions* in the BNF of PDDL3.1.

```
<action-def> ::=
        (:action <action-symbol>
             [:agent <agent-def>]^:multi-agent
             [:parameters (<typed list (variable)>)]
             <action-def body>)
```

The only difference is the addition of an optional part for the associated agent(s). It can be used only if the `:multi-agent` requirement is declared, and also implies **addition** of the following rules for `<agent-def>` to the grammar.

```
<agent-def>   ::= <name>
<agent-def>   ::= <variable>
<agent-def>   ::=^:typing <type>
<agent-def>   ::=^:typing <variable> - <type>
```

This means that agents can be associated to an action in form of constants or variables. If `:typing` is declared, then they can be given in form of types or typed variables too. If a type or a variable is given, then the action-schema is associated to every object whose type *is a subset* of the given type or the type of the variable (since in PDDL types correspond to sets of objects). Without an explicitly declared type the agent-variable is assumed to have type `object` by default (corresponding to the set of every object). In this case every object is eventually considered an agent, since those and only those objects are considered *agents*, which have at least one associated action-schema. If agents are referred to with variables, we suggest to use types to enable distinction between agents and non-agent objects. Furthermore it is required that the names of objects and primitive types are unique and not overlapping, and that every object has only one directly associated type.

If the agent is referenced with a variable in the action-schema, then this variable may appear in the body of the action-schema (in conditions and effects) just like any other action-parameter, and thus the name of the agent variable is required to be different from parameter-names.

If the `:multi-agent` requirement is declared, but the agent-reference is not given in the action-schema, then the schema is associated to the type `object` by default.

Now in case of `:typing`, given a type-hierarchy, an agent-object with a given type is associated with actions that are either associated to it directly, or directly to its type, or directly to an ancestor-type (superset) of its type. This is called *inheritance* of actions. *Polymorphism* on the other hand works as follows: an action with the same name and arity (number of arguments) can be redefined for descendants, i.e. an action directly associated to a type redefines any action directly associated to its ancestor type (superset), if the name and arity of the actions are equal.

An action associated directly to an object (constant) redefines any actions with the same name and arity directly associated to the type of the object, or which are inherited. An agent-object or type cannot have two or more directly associated action-schemas with the same name and arity. Therefore association of actions to agents is unambiguous.

Beyond typing an even more important aspect of a proper multi-agent extension is how interaction of concurrent (joint) actions works. For the proposed extension a *generalization of concurrent action lists* presented in Section 2.2 in (Boutilier and Brafman 2001) is proposed. The idea is simply to allow references to concurrent actions not only in a special construct, such as the concurrent action list, but also among the preconditions of actions and the conditions of actions' conditional effects. Interweaving the content of concurrent action lists with conditions this way allows for a more convenient and compact design. The proposed idea is similar to "progressive predicates" suggested in Section 2.1 in (Bacchus 2003), except that we now refer directly to ongoing actions, and not to facts. This also implies that the name-arity pairs of fluents (predicates, numeric and object fluents) need to be unique, and cannot overlap with the name-arity pairs of actions (durative or non-durative).

The above considerations imply the **addition** of the following 4 new production rules to the grammar.

```
<GD>     ::=:multi-agent <action formula(term)>
<GD>     ::=:multi-agent + :negative-preconditions
                         (not <action formula(term)>)
```

This means that if the `:multi-agent` requirement is declared, a goal description, `<GD>` is allowed to refer also to ongoing actions (similarly to state fluents). The exact form of reference to concurrent actions is the following.

```
<action formula(t)>::= (<action-symbol> t t*)
<action formula(t)>::=:durative-actions (<da-symbol> t t*)
```

The first argument (term) should be always the agent executing the referenced action, while further arguments should be the actual parameters of that action in their respective order. If during execution a grounded reference to an action *A* needs to be positive for the conditions of a grounded action *B* to hold, then this means that *A* needs to be executed in parallel with *B* for *B*'s respective effects to take place. Otherwise, if the grounded reference to *A* needs to be negative for conditions of *B* to hold at a given time during execution, then *A* should not be executed in parallel with *B* for *B*'s respective effects to take place at that time.

The exact time(interval) when *A* should or should not be under parallel execution with *B* is the same time(interval), when a unique "progressive" proposition P_A corresponding to *A* should or should not be true respectively for *B*'s conditions to hold were all occurrences of *A* replaced with P_A in the grounded PDDL-description and thus in the plan. Now this depends only on the semantics of PDDL3.1.

It must be noted though that the *consistency of a joint-action* (where all member actions either refer to other members in their conditions, or they are referred to by at least one of them) should be more relaxed than the traditional definition of mutex actions. For *non-durative actions*, similarly to Definition 2 and 3 in (Boutilier and Brafman 2001) we only require that the members of the joint-action have consistent joint-(pre)conditions and joint-effects in a given state, i.e. interference among effects and (pre)conditions is not considered. In case of *durative actions* the consistency check should focus on the exact time instants and intervals when (pre)conditions need to hold, and when effects take place during scheduled execution. That is in this case it may happen that the (pre)conditions of a member of a durative joint-action are inconsistent with the effects of another member, which may imply a re-scheduling of these actions by the planner. The way this is achieved is beyond the scope of this paper. *A consistent joint-plan consists of consistent joint-actions*.

Durative actions of multiple agents are defined similarly to non-durative actions. The rule for `<durative-action-def>` needs to be slightly **modified** as follows.

```
<durative-action-def>::=
        (:durative-action <da-symbol>
             [:agent <agent-def>]:multi-agent
             [:parameters (<typed list (variable)>)]
             <da-def body>)
```

This means that we can now associate agents to durative actions just like we did to non-durative actions.

Many further **additions** could be made to the grammar of the domain. A minimal, but useful one is the following.

```
<f-exp> ::=:numeric-fluents + :multi-agent
           (num (<typed list (variable)>)
                <emptyOr (GD)>
                <action formula(term)>)
```

The built-in function `num` calculates the number of actions under execution (during execution) for every instantiation of a list of variables where given conditions hold. This small extension adds great expressivity.

3.2. Problem description

The problem description needs to be extended slightly to cope with possibly different goals and utilities of agents. This requires **modification** of the problem definition first.

```
<problem>    ::= (define (problem <name>)
                     (:domain <name>)
                     [<require-def>]
                     [<object declaration>]
                     <init>
                     <goal>+
                     [<constraints>]:constraints
                     <metric-spec>* :numeric-fluents
                     [<length-spec>])
```

The only change here is that now at least one `<goal>` and zero or more `<metric-spec>` structures are required.

Goals can be empty (always true), while utilities don't need to be present when `:numeric-fluents` is declared.

In case of multiple agents, *goals* can be captured by the **addition** of the following production rule to the grammar.

```
<goal> ::=:multi-agent (:goal
                          [:agent <agent-def>]
                          :condition <emptyOr (pre-GD)>)
```

The only essential change here compared to PDDL3.1 is the addition of the agent-reference. If it refers to the agent with a variable, then the variable may appear in the goal formula. Goal conditions are prefixed with `:condition` to emphasize them more. *Utilities* need a similar **addition**.

```
<metric-spec> ::=:multi-agent + :numeric-fluents
            (:metric
               [:agent <agent-def>]
               :utility <optimization> <metric-f-exp>)
```

The declaration of the `:multi-agent` requirement is necessary for the use of the above two structures, but we can also use default PDDL3.1 goals and metric structures, which would mean – similarly to the case of actions – that goals and utilities are associated with the `object` type, i.e. with every agent-object. Above we see that goal and utility schemas can be associated directly to objects (or constants) or types similarly to actions, although one object or type can have only one directly associated goal or utility schema in contrary to actions. But inheritance and polymorphism are the same as in case of actions. Therefore the assignment of goals/utilities to agents is unambiguous.

One last **addition** is necessary to the grammar to allow agent-variables in hitherto grounded metric expressions.

```
<metric-f-exp>::=:multi-agent (<function-symbol> <term>*)
```

This way now we can include *fluents* in connection with agents in the definition of their utility, but naturally the value of metric needs to remain numeric. Not all agents have to have a utility though, but all of them need to have (at least an inherited) goal, which may be the same for all of them or different depending on the problem at hand.

An important topic is still left untouched: *For which agents is a planner planning? Which object(s) represent(s) the planner in the description? Should it be represented?*

The answer depends on how the MA-PDDL description is used: whether the planner is external or situated; whether planning is centralized or decentralized, whether it is distributed; or whether planners share the same MA-PDDL description. The association of the planner and agent(s) can vary from run-to-run (similarly to how an agent may assume different players' role during different plays of the same game (von Neumann and Morgenstern 1944)). This meta-information, the connection of planners and agent-objects is therefore not included in the description. It is the responsibility of the planner to know *for whom* it plans, and possibly *which object(s) represent(s) it* in the problem. So it is suggested to planner applications to have 1-2 more inputs carrying this information beside other parameters.

3.3. Example

The following simple example aims to give a basic idea of how the proposed multi-agent extension works. A minimal set of PDDL features is used to illustrate important aspects, such as cooperation, joint-actions, constructive synergy.

The only action-schema in the domain (`lift`) is associated with type `agent`, which is a direct descendant (subset) of `object`; `lift` allows an agent to lift the `table` (the only domain-constant of type `object`), but only if it is not yet lifted, if the agent is at the `table`, and if there is at least one other agent at the `table` lifting it simultaneously.

```
(define (domain ma-lift-table)
(:requirements:equality :negative-preconditions
               :existential-preconditions :typing :multi-agent)
(:types agent)(:constants table)
(:predicates  (lifted ?x - object) (at ?a - agent ?o - object))
(:action lift :agent ?a - agent :parameters ()
:precondition (and (not (lifted table)) (at ?a table)
               (exists  (?b - agent)
                (and (not (= ?a ?b)) (at ?b table) (lift ?b))))
:effect (lifted table)))
```

The related problem description defines 2 agents: `a` and `b`, both being at the `table`, and having the same goal: the `table` being lifted. Their goal is defined for type `agent`.

```
(define (problem ma-lift-table-1)
(:domain ma-lift-table)
(:objects a b - agent)
(:init (at a table) (at b table))
(:goal :agent agent :condition (lifted table)))
```

The solution requires cooperation from `a` and `b`, since they have the same goal and the only way for them to achieve it is to coordinate their actions. The only, trivially simple solution is when both `lift` the `table` starting at time 0: `[0:(lift a) 0:(lift b)]`. Because of the lack of options the same plan should arise in case one or more external *rational* planners plan for `a` and `b`.

When a planner chooses a grounded `lift` action for execution in a given state, it can assert a corresponding unique `(lift ·)` fact to the state, and check what implications this has on the applicability of other chosen actions. If they remain executable, it may continue, otherwise it may retract `(lift ·)` from the state, and choose different actions. This should work also in general.

In case of decentralized planning, i.e. when different planners plan for different agent-objects, but all planners share the same MA-PDDL description (which is *common knowledge* among them), then solution-plans should not be fully ordered sequences of temporally annotated actions anymore. They should be rather *strategies* (for each agent) that prescribe actions to observation-histories of the agent. A joint plan in this case is a *combination of such strategies*.

In our case, since partial observability isn't introduced yet as another extension, the planning environment is *fully observable*, i.e. observations are complete descriptions of new states and action-combinations that produced them.

We should also note that though an MA-PDDL description may be converted to *an extensive- or normal-form game*, it *would be a much less detailed description*.

Two issues arise in the decentralized case: **(1)** coping with durative strategies; and **(2)** both in durative and non-durative case it is not trivial how to compactly represent strategies, especially in case of large state-spaces.

However both issues can be solved **(1)** by *reasonably restricting the scheduling* of durative actions (e.g. an agent could schedule its next actions only when an other action starts/ends); and **(2)** by using a *client-server architecture* with planners as clients. Planners could receive new observations for relevant time-instants (see previous issue) from the Server and answer with their actual actions.

4. A multi-agent planning track at the IPC

In this section a short proposal is made for a multi-agent planning track at the forthcoming IPCs based on the multi-agent extension of PDDL3.1 presented in Section 3.

There are 3 organizational steps (similarly to current IPCs): **(1)** preparation; **(2)** competition; and **(3)** evaluation. During the *preparation* phase the following should be made public: a Call for Submissions; detailed rules of the competition/evaluation; any source-code and additional applications with documentation; a detailed manual/article about MA-PDDL; and domains/problems for participants.

For the *competition* the participants would need to submit planners (sources, binaries) and papers about them. The competition itself could consist of 2 fully-observable sub-tracks at first: **(2a)** when $N = 1$ external planner plans *for* $M > 1$ situated agents, and **(2b)** when planning is done *by* $N = M > 1$ situated planners. In both cases problems can be categorized according to 3 properties: whether **(i)** all agents' goals/utilities are the same; **(ii)** if there are utilities at all; and **(iii)** if durative actions are allowed. If in **(2b)** we do not allow durative actions, then altogether *12 categories* of multi-agent competition emerge.

When **(i)** holds, problems are *cooperative*. Otherwise they are *competitive*. The latter case can be divided into sub-cases, where each agent has different goals/utilities, and where only agents in *teams* have the same goal/utility.

It should be noted that the easiest category is sub-track **(2a)** when **(i)** holds, but this is still harder than single-agent planning e.g. because of possible constructive synergies.

In each of the 12 categories approx. 12-14 domains could be present each with around 20 related problems. The *evaluation* in case of sub-track **(2a)** could measure normalized **quality** of joint-plans and planning-time per problem, and the number of solved problems per domain for each planner. The sum of these scores could decide the winner of sub-track **(2a)**. But it should be added, that the quality of plans depends mostly on **(i)**. If **(i)** and **(ii)** hold, then quality is defined by utility, but if **(ii)** is not true, then quality can be the makespan of plans. If **(i)** is false, then the number of agents whose goal was achieved, or the sum of achieved sub-goals or of plans' makespan can be used.

The evaluation of planners in sub-track **(2b)** could be similar to evaluation at the probabilistic track at IPC-2011. As mentioned in Section 3.3, a client/server architecture could be used with planner-clients receiving observations from a server and replying to it with their actions. The server could wait for planners' actions at each step for a given time. In case of time-out (e.g. after 30 seconds/step) the `no-op` action could be chosen for late planners.

Initially the server should broadcast the MA-PDDL description, and then for each problem and permutation of planner-agent assignments it could execute e.g. 30 runs to determine planners' average **fitness** for each assignment (since some may make non-deterministic decisions). The sum of these averages over assignments could be the score of a planner for a problem, and thus the sum of scores over problems and domains could determine the winner of sub-track **(2b)**. If **(ii)** holds, then planners' fitness could be the individual utility of their agent. Otherwise it could be the maximum of its simultaneously achieved sub-goals.

5. Conclusions

A *multi-agent extension* of PDDL3.1 was proposed with a corresponding *multi-agent planning track* for the IPC to enable more direct comparison of multi-agent planners and approaches and a greater reuse of research. Planning *by* and *for* agents is both possible. The syntax and semantics of the extension were provided together with an example. A few corrections to the BNF of PDDL3.1 were also listed and an overview of current research in the field was given.

Future research could focus on providing more detailed, possibly formal semantics; planning algorithms; more application domains (e.g. multi-robots, such as RoboCup, or networking problems, such as efficient routing with limited resources). The addition of *partial observability* (in a separate, modular PDDL-requirement) would be primary, but *probabilistic effects* and *events/processes* may also be considered to allow treatment of more realistic problems. The corresponding multi-agent IPC track could also be developed further to narrow the gap between theory and practice and to advance the field of multi-agent planning.

Acknowledgments

This work was partially supported by the ARTEMIS JU and the Hungarian National Development Agency (NFÜ) in frame of the R3-COP (Robust & Safe Mobile Co-operative Systems) project. The author wishes also to thank SUZUKI Foundation Japan, Derek Long, Carlos L. López, Gabriele Röger, Ron Alford, Éric Jacopin, Takashi Watanabe, Naoki Fukuta and Tadeusz P. Dobrowiecki.

References

Bacchus, F. 2003. The Power of Modeling - a Response to PDDL2.1 (Commentary). *J. of AI Res.* 20:125-132.

Baral, C.; Gelfond, G.; Son, T. C.; and Pontelli, E. 2010. Using Answer Set Programming to model multi-agent scenarios involving agents' knowledge about other's knowledge, In *Proc. of AAMAS-2010*, 259-266. IFAAMAS.

Baral, C.; and Gelfond, G. 2011. On representing actions in multi-agent domains, In Engelmore, R., and Morgan, A. eds. *Logic programming, knowledge representation, and nonmonotonic reasoning*. 213-232. Springer.

Beaudry, E.; Kabanza, F.; and Michaud, F. 2010. Planning for Concurrent Action Executions Under Action Duration Uncertainty Using Dynamically Generated Bayesian Networks. In *Proc. of ICAPS-10*, 10-17. AAAI Press.

Boutilier, C.; and Brafman, R. I. 2001. Partial-order planning with concurrent interacting actions. *J. of AI Res.* 14(1):105-136.

Bowling, M.; Jensen, R.; and Veloso, M. 2002. A formalization of equilibria for multiagent planning. In *Proc. of the Workshop on Planning with and for Multiagent Systems, AAAI-02*, 1-6.

Brafman, R. I.; and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proc. of ICAPS-08*, 28-35. AAAI Press.

Brafman, R. I.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2009. Planning Games, In *Proc. of IJCAI-09*, 73-78. AAAI Press.

Brenner, M. 2003a. A Multiagent Planning Language. In *Proc. of the Workshop on PDDL, ICAPS-03*, 33-38.

Brenner, M. 2003b. Multiagent Planning with Partially Ordered Temporal Plans, Technical Report No. 190, Institut für Informatik, Universität Freiburg, Germany.

Crosby, M.; and Rovatsos, M. 2011. Heuristic Multiagent Planning with Self-Interested Agents, In *Proc. of AAMAS-2011*, 1213-1214. IFAAMAS.

Edelkamp, S.; and Hoffmann, J. 2004a. PDDL2.2: The Language for the Classical Part of the 4th International planning Competition, Technical Report No. 195, Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Germany.

Edelkamp, S.; and Hoffmann, J. 2004b. PDDL2.2: The Language for the Classical Part of IPC-4. In *IPC-4 Booklet, ICAPS-04*, 1-5.

Fikes, R. E.; and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 5(2):189-208.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to pddl for Expressing Temporal Planning Domains. *J. of AI Res.* 20: 61-124.

Fox, M.; and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *J. of AI Res.* 27:235-297.

Fox, M.; Long, D.; and Magazzeni, D. 2011. Automatic Construction of Efficient Multiple Battery Usage Policies. In *Proc. of ICAPS-11*, 2620-2625. AAAI Press.

Gerevini, A.; and Long D. 2005. BNF Description of PDDL3.0. *Unpublished manuscript from the IPC-5 website*. http://cs-www.cs.yale.edu/homes/dvm/papers/pddl-bnf.pdf

Helmert, M. 2008. Changes in PDDL 3.1. *Unpublished summary from the IPC-2008 website*. http://ipc.informatik.uni-freiburg.de/PddlExtension

Jensen, R. M.; and Veloso, M. M. 2000. OBDD-based universal planning for synchronized agents in non-deterministic domains. *J. of AI Res.* 13(1):189-226.

Jonsson, A.; and Rovatsos, M. 2011. Scaling Up Multiagent Planning: A Best-Response Approach. In *Proc. of ICAPS-11*, 114-121. AAAI Press.

Kovacs, D. L. 2011. BNF definition of PDDL 3.1. *Unpublished manuscript from the IPC-2011 website*. http://www.plg.inf.uc3m.es/ipc2011-deterministic/Resources

Kumar, A.; Zilberstein, S.; and Toussaint, M. 2011. Scalable Multiagent Planning Using Probabilistic Inference, In *Proc. of IJCAI-11*, 2140-2146. AAAI Press.

Marecki, J.; and Tambe, M. 2009. Planning with Continuous Resources for Agent Teams, In *Proc. of AAMAS-09*, 1089-1096.

Martins, M. F.; and Demiris, Y. 2010. Learning Multirobot Joint Action Plans from Simultaneous Task Execution Demonstrations, In *Proc. of AAMAS-2010*, 931-938. IFAAMAS.

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, CVC TR-98-003/DCS TR-1165, Yale Center for CVC, NH, CT.

von Neumann, J.; and Morgenstern, O. 1944. *Theory of games and economic behavior*. Princeton University Press.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A General, Fully Distributed Multi-Agent Planning Algorithm, In *Proc. of AAMAS-2010*, 1323-1330. IFAAMAS.

Oglietti, M.; and Cesta, A. 2004. CSTRIPS: Towards Explicit Concurrent Planning. In *Proc. of the 3rd Italian WS on Plan. and Sched., 9th Nat. Symp. of Assoc. Italiana per l'Int. Artif.*, 1-13.

Pajarinen, J.; and Peltonen, J. 2011. Efficient Planning for Factored Infinite-Horizon DEC-POMDPs, In *Proc. of IJCAI-11*, 325-331. AAAI Press.

Russell, S.; and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach.* Prentice Hall.

Shah, J. A.; Conrad, P. R.; and Williams, B. C. 2009. Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *Proc. of ICAPS-09*, 289-296.

Spaan, M. T. J.; Oliehoek, F. A.; and Amato, C. 2011. Scaling Up Optimal Heuristic Search in Dec-POMDPs via Incremental Expansion, In *Proc. of IJCAI-11*, 2027-2032. AAAI Press.

Stefanovitch, N.; Farinelli, A.; Rogers, A.; and Jennings, N. R. 2011. Resource-Aware Junction Trees for Efficient Multi-Agent Coordination, In *Proc. of AAMAS-2011*, 363-370. IFAAMAS.

Teichteil-Königsbuch, F. 2008. Extending PPDDL1.0 to Model Hybrid Markov Decision Processes. In *Proc. of the WS on A Reality Check for Plan. and Sched. Under Unc., ICAPS-08*, 1-8.

Wang, K. C.; and Botea, A. 2011. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *J. of AI Res.* 42:55-90.

de Weerdt, M.; and Clement, B. 2009. Introduction to planning in multiagent systems. *Multiagent Grid Systems* 5(4):345-355.

Yabu, Y.; Yokoo, M.; and Iwasaki, A. 2009. Multiagent Planning with Trembling-Hand Perfect Equilibrium in Multiagent POMDPs, In Ghose, A.; Governatori, G.; and Sadananda, R. eds. *Agent Computing and Multi-Agent Systems*. 13-24. Springer.

Zhuo, H. H.; and Li, L. 2011. Multi-Agent Plan Recognition with Partial Team Traces and Plan Libraries, In *Proc. of IJCAI-11*, 484-489. AAAI Press.

Zhuo, H. H.; Muñoz-Avila, H.; and Yang, Q. 2011. Learning Action Models for Multi-Agent Planning, In *Proc. of AAMAS-2011*, 217-224. IFAAMAS.