A Formal Approach Combining Event-B and PDDL for Planning Problems

Sabrine Ammar[1] and Mohamed Tahar Bhiri[2]
[1]*MIRACL, Faculty of Economics and Management of Sfax, Sfax, Tunisia*
[2]*MIRACL, Faculty of Science of Sfax, Sfax, Tunisia*

Keywords: Code Generation, Correct by Construction, Event-B, PDDL, Formal Modelling, Refinement Strategy.

Abstract: In artificial intelligence, the goal of automatic planning is to structure actions in the form of a plan to achieve an expressed goal. The PDDL (Planning Domain Definition Language) was designed to allow the common representation of planning problems during ICAPS (International Conference on Automated Planning and Scheduling) competitions. PDDL has many verification and validation tools allowing the description, resolution and validation of planning problems. But they only allow the reliability of PDDL descriptions a posteriori. In this article, we recommend a rigorous approach coupling Event-B and PDDL favoring obtaining PDDL descriptions deemed correct, a priori, from an ultimate Event-B model. The formal Event-B method allows us to obtain, by successive refinements with mathematical proofs, correct by construction formal models of planning problems. A refinement strategy appropriate to planning problems is, then, proposed. The ultimate Event-B model, correct by construction, is automatically translated into PDDL using our MDE Event-B2PDDL tool. The obtained PDDL description is submitted to efficient planners for generation of correct and efficient plan-solutions.

1 INTRODUCTION

The automatic planning community has developed a formal de-facto standard Planning Domain Definition Language (PDDL) (McDermott et. al. 1998) to formally describe planning problems. In addition, this community has developed solvers able to calculate solutions to PDDL-formalized planning problems. In addition, validation tools were developed for verifying whether a given plan-solution can be derived from a PDDL description. In general, PDDL descriptions are difficult to write, read, and evolve. Moreover, the tools associated with the PDDL language, namely planners and validators; do not allow a rigorous a priori analysis of the PDDL descriptions. In fact, these tools are used a posteriori after establishing PDDL descriptions.

In this work, we advocate the opening of the automatic planning community on the formal methods community through Event-B (Abrial, 2010). To achieve this, we suggest a transformation from Event-B to PDDL. This promotes the development of planning problems correct by construction. The ultimate Event-B model, derived from a chain of refinements with mathematical proofs, is translated into PDDL in order to generate quality plans through various planners supporting PDDL.

This article extends our previous short paper presented in (Ammar and Bhiri, 2018). In fact, we detail in particular the strategy of refinement with mathematical proofs recommended for a class of planning problems. Such a class has state change operators with complex preconditions: a logical formula comprising several operands and operators. In addition, a sub-section (Discussion) is added in the "Proposed refinement strategy" section in order to discuss the compatibility of the proposed refinement strategy for the development of certain planning problems with respect to what is recommended by the Event-B method. In addition, the "Related works" section is added to place our work in relation to existing works. Finally, the conclusion is enriched by new research lines pointed out.

This article contains seven sections and one conclusion. The second section presents and evaluates the PDDL language. The third section provides a general overview of modeling in Event-B method. The fourth section proposes an Event-B and PDDL coupling approach. The fifth section provides a refinement strategy for formal modeling of Event-B planning problems. The sixth section describes our

Ammar, S. and Bhiri, M.
A Formal Approach Combining Event-B and PDDL for Planning Problems.
DOI: 10.5220/0010577102610268
In *Proceedings of the 16th International Conference on Software Technologies (ICSOFT 2021)*, pages 261-268
ISBN: 978-989-758-523-4

MDE Event-B2PDDL tool. Finally, the seventh section is devoted to related works. The conclusion draws up the balance sheet of this article and proposes the possible extensions of this work.

2 THE PDDL LANGUAGE

A planning problem formalized using PDDL has two separate parts: **domain** and **problem**. The **domain** construction makes it possible to describe all the common aspects in a class of problems known as generic domain.. A domain described in PDDL includes types, constants, predicates, numeric functions and actions.

As an example, Listing 1 from (Bibai, 2010) describes the domain of the sliding puzzle game in PDDL. The domain of the sliding puzzle game has two types: *position* and *tile*. In PDDL, a type does not have a structure and is designated by a name. The predicates *at* (having two parameters *?tile* of type *tile* and *?position* of type *position*), *neighbor* and *empty* allowing to formalize the concept of state of a sliding puzzle game problem. The *move* action is the only state change operator relating to the sliding puzzle game domain. In PDDL, an action can have parameters typed (parameters clause) and defined by a Pre/Post specification: precondition and effect clauses. An action can be applied in a state if and only if all preconditions are satisfied in this state. The effect of a PDDL action is defined by the additions and withdrawals of atoms in the current state.

```
(define (domain n-sliding-puzzle)
(:types position tile)
(:predicates
(at ?tile - tile ?position - position)
(neighbor ?p1 - position ?p2 -position)
(empty ?position - position))
(:action move
 :parameters (?tile - tile ?from ?to – position)
 :precondition (and (neighbor ?from
  ?to) (at ?tile ?from)(empty ?to))
 :effect (and(at ?tile ?to)(empty ?from
  (not (at ?tile ?from))(not (empty
   ?to)))))
```

Listing 1: State of the application.

The **problem** construction makes it possible to formalize a problem belonging to the domain described by the **domain** construction. A problem includes the domain of this problem, typed objects (**objects**), an initial state (**init**) and a goal state (**goal**). For example, the Sliding Puzzle game containing 8 tiles is provided by Listing 2.

```
(define (problem n-sliding-puzzle-pbl)
(:domain n-sliding-puzzle)
(:objects
 p_1_1 p_1_2 p_1_3 p_2_1 p_2_2 p_2_3
 p_3_1 p_3_2p_3_3 - position
 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 -tile)
(:init (empty p_1_2)
(at t_4 p_1_1)(at t_8 p_1_3)
(at t_6 p_2_1)(at t_3 p_2_2)
(at t_2 p_2_3)(at t_1 p_3_1)
(at t_5 p_3_2)(at t_7 p_3_3)
(neighbor p_1_1 p_1_2)(neighbor p_1_2
p_1_1)(neighbor p_1_2 p_1_3)(neighbor
p_1_3 p_1_2)(neighbor p_2_1 p_2_2)
(neighbor p_2_2 p_2_1)(neighbor p_2_2
p_2_3)(neighbor p_2_3 p_2_2)(neighbor
p_3_1 p_3_2)(neighbor p_3_2 p_3_1)
(neighbor p_3_2 p_3_3)(neighbor p_3_3
p_3_2)(neighbor p_1_1 p_2_1)(neighbor
p_2_1 p_1_1)(neighbor p_1_2 p_2_2)
(neighbor p_2_2 p_1_2)(neighbor p_1_3
p_2_3)(neighbor p_2_3 p_1_3)(neighbor
p_2_1 p_3_1)(neighbor p_3_1 p_2_1)
(neighbor p_2_2 p_3_2)(neighbor p_3_2
p_2_2)(neighbor p_2_3 p_3_3)(neighbor
p_3_3 p_2_3))
(:goal (and (at t_1 p_1_1)
(at t_2 p_1_2)(at t_3 p_1_3)
(at t_4 p_2_1) (at t_5 p_2_2)
(at t_6 p_2_3) (at t_7 p_3_1)
(at t_8 p_3_2))))
```

Listing 2: Sliding puzzle game with 8 tiles.

A planning problem described using PDDL is solved by a software item called planner. A planner combines exploration and logic. In fact, it can be seen either as a program that calculates a solution called plan-solution or as a program that **demonstrates** the existence of a solution. For example, the Sliding Puzzle game planning problem described by both Listings 1 and 2 submitted to the LPG planner (Gerevini, Saetti and Serina, 2003) provides a plan-solution comprising 52 actions. Listing 3 illustrates an extract of this plan solution.

```
0: (move t_8 p_1_3 p_1_2)
1: (move t_2 p_2_3 p_1_3)
2: (move t_3 p_2_2 p_2_3)
3: (move t_8 p_1_2 p_2_2)
4: (move t_2 p_1_3 p_1_2)
```

Listing 3: Plan-solution extract associated with the 8-tile problem of the Sliding Puzzle game.

PDDL offers interesting ways to represent planning problems. In fact, PDDL supports various representations such as propositional representations, first order logic and both numeric and temporal. This makes it possible to describe the **states** and **actions** of

a planning problem. The tools associated with the PDDL language are: planners and validators. Unlike a planner who performs a plan-solution **production** activity, a validator (Howey, Long and Fox, 2004) performs a **verification** activity. From the functional point of view, a validator accepts as input: a PDDL description (**domain** and **problem** file) and one or more plan-solution files and gives a verdict as an output. 'YES' means that the plan-solution can be obtained from the subject PDDL description. However, 'NO' means a failure. Validators can be used in a profitable way to appreciate PDDL domains by adopting the functional test. In addition, validators allow double verification by checking the plan-solutions generated by various planners. Finally, a validator can be used as a tool to **objectively** compare the abilities of various planners. The **dynamic** analysis tools associated with PDDL, namely planners and validators, are insufficient for the verification and validation of PDDL descriptions. In fact, complex PDDL descriptions involving actions with elaborated preconditions and post conditions are subject to errors that are hard to detect **a priori**. In fact, the dynamic analysis tools associated with PDDL makes it possible to detect errors **a posteriori** by means of a test activity.

3 MODELING IN EVENT-B

An Event-B model can contain contexts (construction CONTEXT) only, machines (construction MACHINE) only or both. Contexts are modeling static properties of the model. Machines (construction MACHINE) are modeling the dynamic behavior of the system. A machine may refine and see another one or more contexts. The state of the machine is defined by variables introduced by the VARIABLES clause. The invariance properties related to these variables are grouped together in the INVARIANT clause. They are considered as predicates in the sense of Event-B as described by its logico-set language. An Event-B machine is grouping events that affect its state. An event consists of two parts: a “guard” that defines the condition according to it the event may or may not be triggered, and an "action" called event body permitting the evolution of state variables. An event may have parameters called local parameters. The Event-B language offers a simple action language to describe the processing (event body): simple assignment (:=), non-deterministic set assignment (:∈), non-deterministic assignment governed by a predicate (:|), parallel assignment (||), and skip that does nothing.

4 FROM EVENT-B TO PDDL

We advocate a rigorous approach combining Event-B and PDDL for automatic planning. Event-B is used for formal modeling by successive refinements with mathematical proofs of planning problems. The refinement of data supported by Event-B can be used in a profitable way in order to refine the notion of state of a planning problem **step-by-step**. In addition, the one-to-many refinement provided by Event-B is very useful for **determining** the state change operators of a planning problem. Finally, the possibility of reinforcing the guard of an Event supported by Event-B during a refinement step is very useful to incrementally identify the applicability conditions of a state change operator of a planning problem. Proof tools associated with Event-B (generator of proof obligations and provers) guarantee in particular the verification of the coherence of a planning problem described by Event-B. The ProB (Leuschel and Butler, 2003) tool that accepts Event-B offers the possibility of checking the dynamics of a planning problem. The use of Event-B coupled to ProB allows to obtain Event-B model correct by construction (thanks to the Event-B theory) and valid (thanks to ProB) describing a planning problem. Then we have to translate this Event-B model into a PDDL. To achieve this, several refinement steps are required in order to have a model described by an **Event-B subset**: the data are described by the language of the first-order predicates of Event-B (the theory of sets is discarded because it is not translatable to PDDL) and the processing are described only through deterministic action (:=).

5 PROPOSED REFINEMENT STRATEGY

Following numerous Event-B modeling of various planning problems, we have established a refinement strategy that could be reused for modeling various planning problems with Event-B in several areas. In fact, all planning problems can be formalized by the concept of state space: initial state, goal states, intermediate states and state change operators. Based on all of its common aspects of planning problems, we propose the refinement strategy that includes the steps as outlined and justified below. The purpose of this problem is to transfer three cannibals and three missionaries from one side of a river to another via a boat. The requirement of this problem is that the

number of missionaries must always be greater than or equal to the number of cannibals.

Step 1: Initial Abstract Model. The initial abstract model of a planning problem includes elements related to the notion of state, the initial state, and the goal states. These elements are formalized respectfully in Event-B by typed variables. They have invariant properties, INITIALISATION event and an event called goal having a guard to see if the current state is a goal state. The goal event does nothing (skip action). For example, Listing 4 models the notion state, the initial state, and the goal states of the problem of three cannibals and three missionaries. In addition, the initial abstract model of a planning problem shall involve an overly abstract and non-deterministic modeling of the notion of state change operator. This is made possible by the ANTICIPATED status. Listing 5 provides the state change operator related to the problem of three cannibals and three missionaries.

```
VARIABLES
nbc_shore1,nbc_shore2,nbm_shore1,nbm_shore2
INVARIANTS
  inv1: nbc_shore1∈0..3
  inv2: nbc_shore2∈0..3
  inv3: nbm_shore1∈0..3
  inv4: nbm_shore2∈0..3
INITIALISATION=
  nbc_shore1,nbc_shore2,nbm_shore1,
  nbm_shore2:=3,0,3,0
goal =
  WHEN
    grd1: nbc_shore1=0
    grd2: nbc_shore2=3
    grd3: nbm_shore1=0
    grd4: nbm_shore2=3
  THEN
    Skip
END
```

Listing 4: State, Initial state and Goals states.

```
cross = STATUS anticipated
  act1: nbc_shore1:∈0..3
  act2: nbc_shore2:∈0..3
  act3: nbm_shore1:∈0..3
  act4: nbm_shore2:∈0..3
   END
```

Listing 5: Non-deterministic specification of the event cross.

Step 2: Determination of Actions by Successive Refinements. This step includes several successive refinements permitting, ultimately, to get an Event-B model with state change operators having deterministic behaviors. Each operator contains a guard modeling the operator's applicability condition and its action. Refinement techniques supported by Event-B as an event decomposition (one to many) and the reinforcing guards are very useful for implementing this step. For example, Listing 6 shows how to refine the abstract operator to introduce two types of operators. The state change operators are modeled by events in Event-B whose guards indicate the conditions of application of these operators and the actions that are modeling the changes of state: transition from one state to another in state spaces. To list all of the state change operators related to application, we recommend using parameterized non-deterministic events. For example, Listing 7 shows the refinement of the event *cross_shore1_shore2* by introducing two local parameters *c* and *m*. The first is modeling the transfer of cannibals and the second is modeling the transfer of missionaries.

```
VARIABLES
nbc_shore1,nbc_shore2,nbm_shore1,
nbm_shore2,boat
INVARIANTS
  inv1: boat∈BOAT
cross_shore1_shore2 =
 REFFINE cross
   WHEN
     grd1: boat=left
   THEN
     act1: nbc_shore1:∈0..3
     act2: nbc_shore2:∈0..3
     act3: nbm_shore1:∈0..3
     act4: nbm_shore2:∈0..3
     act5: boat:=right
END
cross_shore2_shore1 =
   REFFINE cross
   WHEN
     grd1:boat=right
   THEN
     act1: nbc_shore1:∈0..3
     act2: nbc_shore2:∈0..3
     act3: nbm_shore1:∈0..3
     act4: nbm_shore2:∈0..3
     act5: boat:=left
   END
```

Listing 6: Introduction of two types of state change operators.

```
   cross_shore1_shore2 =
     REFFINE cross_shore1_shore2
     ANY
       m
       c
     WHEN
       grd1: m∈0..2
```

```
    grd2: c∈0..2
    grd3: m+c∈1..2
    grd4: c≤nbc_shore1
    grd5: m≤nbm_shore1
    grd6: boat=left
  THEN
    act1: nbc_shore1:=nbc_shore1- c
    act2: nbm_shore1:=nbm_shore1- m
    act3: nbc_shore2:=nbc_shore2+c
    act4: nbm_shore2:=nbm_shore2+m
    act5: boat:=right
END
```

Listing 7: Introduction of parameters.

Step 3: Determination of Parameters by Successive Refinements. This step aims to remove the non-determinism related to the parameters introduced in the clause ANY of each states change operator. Eventually, we obtain events without parameters. Technically in this step, the "one to many" refinement technique and the WITH clause are used in a profitable way. The problem of three cannibals and three missionaries has 10 state change operators with deterministic actions and no parameters (see Listing 8).

```
cross_one_cannibal_shore1_shore2 =
  REFFINE cross_shore1_shore2
  WHEN
    grd1: boat=left
    grd2: 1≤nbc_shore1
  WITH
    c: c=1
    m: m=0
  THEN
    act1: nbc_shore1:=nbc_shore1-1
    act2: boat:=right
END
```

Listing 8: State change operator without parameters.

Step 4: Reinforcing Applicability Conditions. This step consists in reinforcing applicability conditions of the state change operators (WHERE clause) introduced in the previous step. The ultimate model from this step must have well-defined state change operators. Technically, this step introduces new invariant properties (reinforcement of the invariant) and guards (reinforcement of guards). For example, the problem of three cannibals and three missionaries shall make it inaccessible for the states where the number of cannibals is greater than that of missionaries (see Listing 9).

```
INVARIANTS
inv:nbm_shore1≠0⇒nbm_shore1≥nbc_shore1
inv2:3-nbm_shore1≠0⇒3-nbm_shore1≥
     3-nbc_shore1
cross_one_cannibal_shore1_shore2 =
REFFINE cross_one_cannibal_shore1_shore2
 WHEN
 grd1: boat=left
 grd2: 1≤nbc_shore1
 grd3: (3-nbm_shore1=0)∨(3-nbm_shore1≥
        3-nbc_shore1+1)
  WITH
    c: c=1
    m: m=0
  THEN
    act1: nbc_shore1:=nbc_shore1-1
    act2: boat:=right
END
```

Listing 9: Applicability conditions of cross_one_cannibal_shore1_shore2.

Step 5: Data Concretization. The purpose of this step is to eventually provide an Event-B model translatable into PDDL. All Event-B set constructions must be realized using the Event-B predicative constructions. To achieve this, data refinement is used via Event-B gluing invariant. Note in passing that this step is not applicable to the problem of three cannibals and three missionaries. Recently, we have successfully applied this step on the MICONIC domain (Haslum et. al. 2019).

Step 6: Translating a Reduced Event-B into PDDL. The reduced Event-B model from Step 4 is translated using our Event-B2PDDL tool introduced later. Listing 9 gives an extract of the ultimate Event-B model translatable into PDDL. Listing 10 provides an excerpt from the translation of the problem of three cannibals and three missionaries into PDDL.

```
(:types SHORE CANNIBAL MISSIONARY)
(:constants left right - SHORE)
(:predicates (boat ?x - SHORE)
(pos_cann ?x - CANNIBAL ?y - SHORE)
(pos_miss ?x -MISSIONARY ?y -SHORE)
(diffc ?x ?y - CANNIBAL)
(diffm ?x ?y - MISSIONARY))
(:functions nbc_shore1) (nbm_shore1))
(:action cross_one_cannibal_shore1_shore2
:parameters (?cc - CANNIBAL)
:precondition (and (boat left)(<= 1
(nbc_shore1))(or (= (- 3 (nbm_shore1))
0)(>= (- 3 (nbm_shore1))(+ (- 3
(nbc_shore1)) 1)))(pos_cann ?cc left))
:effect (and (boat right)(not(boat
left))(decrease (nbc_shore1)1)(pos_cann
?cc right)(not (pos_cann ?cc left))))
```

Listing 10: PDDL translation excerpt.

5.1 Discussion

In this section we take a look at the development of

our refinement strategy applied to the problem of three missionaries and three cannibals as a planning problem. Invariant properties other than typing properties are not introduced into the initial abstract model. In fact, the state change operator: cross is defined in a non-deterministic way. The determination of the state operator: cross is carried out progressively by introducing constraints linking the variables which forms the state of the machine and decomposes it. Such constraints define the invariant properties to be preserved by the state change operators.

The ultimate Event-B model includes semantically well-defined state change operators (events): applicability condition and effect. The properties that transcend these operators are factorized in the invariant of the ultimate model. The proof obligations discharged from the ultimate model are ensuring its static correction with respect to the invariant properties. The dynamics of the ultimate model is formally verified with the ProB toolbox: animation and model-checking. Both INITIALISATION and *goal* events modeling the initial state and the logical condition of the goal states in a planning problem remain unchanged during the application of the refinement strategy. We can see that the number of events is increasing as we apply our refinement strategy: it goes from 3 to 13.

6 THE EVENT-B2PDDL TOOL

Our Event-B2PDDL tool takes as input a reduced Event-B model that is translatable into PDDL and gives as output a PDDL description acceptable to planners. Event-B2PDDL is based on simple intuitive rules allowing the systematic translation of Event-B elements to PDDL elements. Event-B2PDDL is made according to MDE technology.

6.1 Event-B to PDDL Transformation Rules

The PDDL description coming from the Event-B2PDDL tool has two domain and problem constructions (see section Event-B overview). Thus, in (Fourati, Bhiri and Robbana 2016), we have respectively established rules allowing the translation of Event-B elements related to the planning domain and the planning problem. The translation rules for Event-B elements related to the planning **domain** concern: translation of abstract sets (see Table 1), constants (see table 2), Boolean constants or variables (see Table 3), Boolean functions (see Table 4), events

Table 1: Schemes of translations of abstract set.

Event-B	PDDL
SETS TYPE1 TYPE2 … TYPEn	(:types type1 type2 … typen)

Table 2: Schemes of translations of constants.

Event-B	PDDL
CONSTANTS cst1, cst2, …, cstn AXIOMS axm1:partition(TYPE,{cst1},{cst2}, …, {cstn})	(:constants cst1 cst2 … cstn -TYPE)

Table 3: Schemes of translations of Boolean constants or variables.

Event-B	PDDL
CONSTANTS cst AXIOMS axm1: cst∈BOOL VARIABLES var INVARIANTS inv1: var∈BOOL	(:predicates (cst) (var))

Table 4: Schemes of translations of Boolean functions.

Event-B	PDDL
VARIABLES fnc INVARIANTS inv_fnc: fnc∈TYPE1 × TYPE2 × . . . × TYPEn → BOOL	(:predicates (fnc ?var1 - TYPE1 ?var2 - TYPE2 ... ?varn - TYPEn))

Table 5: Schemes of translations of Event-B Events.

Event-B	PDDL
evt_name = STATUS ordinary ANY var1 var2 … WHERE grd1: var1∈TYPE1 grd2: var2∈TYPE2 … grdn: <GDn> grdn1: <GDn1> .... THEN act1: <ACT1> act2: <ACT2> .... END	(:action <evt_name> :parameters (?var1 - TYPE1 ?var2 - TYPE2 ...) :precondition (and (<GDn>) (<GDn1>) .....) :effect (and (<ACT1>) (<ACT2>) .....)))

(see Table 5) and formulas (see Table 6). The translation rules of Event-B elements related to the planning **problem** are about translation of constants linked to the sets defined by enumeration (see Table 7), translation of axioms of initialization (see Table 8) and translation of both INITIALISATION and GOAL events (see Table 9 and Table 10).

Table 6: Schemes of translation of Event-B formulas retained in PDDL.

Event-B	PDDL
P ∧ Q	(and P Q)
P ∨ Q	(or P Q)
P ⇒ Q	(imply P Q)
¬P	(not P)
∀z.P⇒ Q	(forall (?z) (imply P Q))
∃z.P∧ Q	(exists (?z) (and P Q))
E = F	(= E F)
E ≠ F	(not (= E F))
b:=TRUE	(b)
b:=FALSE	(not(b))
f(x):=TRUE	(f ?x)
f(x):=FALSE	(not (f ?x))
f:=f<+{ x|->TRUE, y|->FALSE}	(and (f ?x) (not (f ?y)))

Table 7: Schemes of translations of constants.

Event-B	PDDL
CONSTANTS cst1, cst2, …, cstn AXIOMS axm1:partition(TYPE,{cst1},{cst2}, …, {cstn})	(:objects cst1 cst2 … cstn - TYPE)

Table 8: Schemes of translations of Axioms of Initialization.

Event-B	PDDL
axm1:fnc1:=((TYPE1×TYPE2×…×TYPEn)×{FALSE})<+({cst1 ›→cst2›→...cstn, …}×{TRUE})	(:init (fnc1 cst1 cst2... cstn) ...)

Table 9: Schemes of translations of the event INITIALISATION.

Event-B	PDDL
INITIALISATION= STATUS ordinary act1:fnc2:=((TYPE1×TYPE2×. . .×TYPEn)×{FALSE})<+({cst1›→cst2›→... cstn,…}×{TRUE})... END	(:init (fnc2 cst1 cst2 ... cstn) ...)

Table 10: Schemes of translations of the event GOAL.

Event-B	PDDL
GOAL = STATUS ordinary WHEN grd1:fnc2=fnc2<+ {cst1›→cst2›→...›→cstn ›→TRUE, ...} ... THEN skip END	(:goal (and (fnc2 cst1 cst2 ... cstn) ...))

7 RELATED WORKS

There exist several “Integrated Development Environments” supporting PDDL (Magnaguagno et. al. 2017; Strobel and Kirsch, 2014; Muise and Lipovetzky 2020). Such environments offer functionalities allowing the editing of PDDL descriptions, lexico-syntactic verification of PDDL text, generation of plans-solution and visualization of states space associated with the planning problem. This last functionality allows, among other things, to provide information to the user, related to the "execution" of PDDL actions. This allows the user to detect errors related to the specification of a PDDL action such as incorrect precondition, incorrect post condition and incorrect precondition and post condition. In addition, the visualization of the state space makes it possible to make the behavior of the PDDL planner explicit in order to find a plan solution. This reduces the opacity of PDDL planners. In fact, without visualization, these planners are used as black boxes. In our opinion, this work supports the analysis of PDDL descriptions a posteriori without correction certification. From among Platforms covering the design of planning domains and problems, we note itSIMPLE (Vaquero et. al. 2009) and GIPO (Simpson, Kitchin, and McCluskey, 2002). On one hand, itSIMPLE platform uses UML (class and object diagram, OCL constraint language, and time diagram) to describe planning domains and problems. In addition, this platform provides a tool for transforming UML to PDDL. On the other hand, GIPO platform uses Object-Centred-Language (OCL) to describe planning domains and problems. It provides a translation tool from OCL to PDDL. In addition, it provides static and dynamic validation capabilities for planning domains.

The work described in (West, Kitchin and McCluskey, 2002) explores the use of formal model-oriented methods based on successive refinements

with mathematical proofs and supporting the paradigm correct by construction. Such work is the closest to our concerns but it does not lead to a PDDL code generation. Moreover, it uses B which has a notion of refinement less rich than that of Event-B.

8 CONCLUSIONS

PDDL is a declarative language. It offers quite significant means for domain modeling areas and planning problems. In addition, PDDL is endowed with powerful software tools - called planners – permitting the automatic generation of plan-solutions from a PDDL description. But PDDL descriptions are often difficult to write, read and evolve. Also, they are subject to several types of errors: data typing, initialization of static and dynamic predicates and pre-condition / post-condition specification of action schemes. To deal with these faults; in this work, we proposed an Event-B to PDDL coupling approach. The transition from Event-B to PDDL makes it possible to model correct by construction and efficient planning problems. Event-B ensures the correct by construction of the states change operators. Whereas PDDL ensures the effectiveness of the plan-solutions obtained thanks to the planners associated with PDDL. We proposed, in addition, a refinement strategy which may be appropriate for a class of planning problems whose actions have complex preconditions. Technically, a complex precondition is a big logical formula comprising atoms connected by logical operators such as: not, and, or, imply, exists and forall. The ultimate Event-B model from our refinement strategy is translated into PDDL using our MDE Event-B2PDDL tool.

Currently, we are working in two directions: The first direction consists of the experimentation of the refinement strategy proposed on various more or less complex planning problems. Recently, we have successfully applied our refinement strategy to the MICONIC planning domain (Haslum et. al. 2019). This domain describes the operation of an elevator in a building. Passengers of various categories are waiting on the different floors and the goal is to transport each passenger to his/her floor of destination. The second direction is about the development of refinement schemes allowing the translation of Event-B data into PDDL: from set representations to predictive representations. Eventually, such schemes could be automated by adopting the technique of automatic refinement like the BART tool (Requet, 2008) associated with the formal method B.

REFERENCES

Abrial, J. R. (2010). *Modeling in Event-B: Systems and Software Engineering*. Cambridge University Press.

Ammar, S., Bhiri, M. T. (2018). Automatic Planning: From Event-B to PDDL. *International Conference on. Model and Data Engineering*. Marrakesh. Morocco.

Bibai, J. (2010). Segmentation et Evolution pour la Planification : Le Système DivideAndEvolve. THALES Research and Technology France. Paris Sud University, Paris XI.

Fourati, F., Bhiri, M. T., Robbana, R. (2016). Verification and validation of PDDL descriptions using Event-B formal method. International Conference on Multimedia Computing and Systems, pp. 770-776.

Gerevini, A., Saetti, A., Serina, I. (2003). Planning through Stochastic Local Search and Temporal Action Graphs in lpg. *J Artif Intell Res* 20:239-290.

Haslum, P., Lipovetzky, N., Magazzeni, D., and Muise, C. (2019). *An Introduction to the Planning Domain Definition Language*. In S*ynthesis Lectures on Artificial Intelligence and Machine Learning*.

Howey, R., Long, D., Fox, M. (2004). VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning using PDDL. In *Tools with Artificial Intelligence*, ICTAI. doi: 10.1109/ICTAI.2004.120.

Leuschel, M., Butler, M. (2003). ProB: A Model Checker for B. In International Symposium of Formal Methods.

Magnaguagno, M. C., Pereira, R. F., More, M. D., and Meneguzzi, F. (2017). WEB PLANNER: A Tool to Develop Classical Planning Domains and Visualize Heuristic State-Space Search. ICAPS 2017. User Interfaces for Scheduling & Planning (UISP).

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D. (1998). PDDL-The Planning Domain Definition Language. Yale Center for Computational Vision and Control. Technical Report CVC TR- 98-003/DCS TR-1165. New Haven, CI, USA.

Muise, C., Lipovetzky, N. (2020). KEPS BOOK: Planning.Domains. I*n book: Knowledge Engineering Tools and Techniques for AI Planning*, pp.91-105.

Requet, A. (2008). BART: A tool for Automatic Refinement. In ABZ. doi: 10.1007/978-3-540-87603-8_33

Simpson, R. M., Kitchin, D.E., McCluskey, T.L. (2002). Planning Domain Definition Using GIPO. *The Knowledge Engineering Review* 22(02): 117 – 134.

Strobel, V., Kirsch, A. (2014). Planning in the Wild: Modeling Tools for PDDL. *37th German Conference on Artificial Intelligence*, Stuttgart. Germany.

Vaquero, T.S., Silva, J.R., Ferreira, M., Tonid, F., Beck, J. c. (2009). From Requirements and Analysis to PDDL in itSIMPLE3.0. In Proceedings of the 3rd International Competition on, 2009.

West, M.M., Kitchin, D.E., McCluskey, T.L. (2002). Validating Planning Domain Models using B-AMN. In: PlanSIG 2002, 21st/22nd Nov 2002, Delft University of Technology, Holland.