A correct-by-construction approach for development of reliable planning problems

Sabrine Ammar
University of Sfax
Taoufik Sakka Rouis
srtaoufik@yahoo.fr

Institut Supérieur d'Informatique et de Mathématiques de Monastir
Mohamed Tahar Bhiri
University of Sfax

Research Article

Keywords: Correct by construction, Event-B, Code generation, PDDL, Automatic planning, Refinement

Posted Date: March 2nd, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2621444/v1

Additional Declarations: No competing interests reported.

A correct-by-construction approach for development of reliable planning problems

Sabrine Ammar[1], Taoufik Sakka Rouis[2*] and Mohamed Tahar Bhiri[1]

[1]Computer Science Department, Miracl Laboratory, Sfax, 3021, Tunisia.
[2*] Computer Science Department, ISIMM, Monastir, 5000, Tunisia.

*Corresponding author(s). E-mail(s): taoufik.sakkarouis@isimm.u-monastir.tn;
Contributing authors: ammar.sabrine@hotmail.fr; tahar_bhiri@yahoo.fr;

Abstract

The automatic planning community has developed a defacto standard planning language called PDDL. Using the PDDL tools, the reliability of PDDL descriptions can only be posteriori examined. However, the Event-B method supports a rich refinement technique that is mathematically proven. This allows the step-by-step correct construction of Event-B models. In order to specify and solve the planning problems, a development process based on the combination of Event-B and PDDL is proposed. Our development process begins with modeling the planning problem by an Event-B abstract model. Through successive refinements, an Event-B ultimate model correct by construction is obtained. Then, using our Event-B2PDDL Eclipce plugin, the Event-B ultimate model can be automatically translated into a PDDL description. Thus, the resulting PDDL description can be considered correct by construction. Finally, using the PDDL planner tool on this generated PDDL description, plan-solutions related to the planning problems initially described by an Event-B model can be produced. Our process is successfully experimented on a set of representative case studies.

Keywords: Correct by construction, Event-B, Code generation, PDDL, Automatic planning, Refinement

1 Introduction

In Artificial Intelligence (AI), planning designates a field of research that aims at automatically generate, via a formalized procedure, a hinged result called plan. This is intended to orientate the action of one or more executors (robots or humans). Such executors are called upon to act in a particular world to achieve a predefined goal. Actually, automatic planning is a full-fledged discipline in AI. It allows to model and solve planning problems in many fields including robotics, crisis management, logistics, web services composition, resource management, assembly of complex objects, storage management, games, etc. In [1], a more or less exhaustive list of planning domains is established.

The automatic planning community has developed a defacto standard planning language called PDDL (Plannig Domain Definition Language) [2]. This standard allows to formally describe planning problems. In addition, this community has been interested in generating plans. Moreover, it has developed validation tools [3] to check whether a given solution plan can be derived from a PDDL description. The formal PDDL language is used to describe the two components of a planning problem: state and state change operators. The state is described by the types and logical predicates. State change operators are described in PDDL by actions with Pre/Post semantics. The pre-condition describes the applicability condition and the post-condition describes the effect of the action. Besides problems related to difficulties of reading, writing and developing of complex PDDL descriptions, these complex descriptions are subject to errors that are difficult to identify in a priori time. This is because the planners and validators tools associated with the PDDL language make it possible, at most, to detect errors a posteriori. In addition, the locating errors in a PDDL description are not easy. This is also true for PDDL error correction. Based on a literature (see Section 2), we can conclude that the PDDL language can be used to describe the states and actions of planning problem. However, it cannot specify all the state's semantic aspects. In particular, it cannot be used for specifying the intra-atomic and inter-atomic semantic properties. In the same context, the Event-B formal method [4] supports the paradigm correct by construction [5]. Indeed, a very rich refinement technique with mathematical proofs is supported by the Event-B method. This allows incremental development through successive refinements with mathematical proof.

In this work, a development process based on the coupling of Event-B method and PDDL language is proposed. Our process begins with modeling the planning problem by an Event-B abstract mode. Through successive refinements, an Event-B ultimate model correct by construction and valid using the Event-B proof/validation tools is obtained. As a second step, using our Event-B2PDDL Eclipce plugin, the ultimate Event-B model described by a subset of Event-B can be easily translated into PDDL. Thus, the resulting PDDL description is considered to be correct by construction. Finally, using the PDDL planner tool on this generated PDDL description, plan-solutions

related to the planning problems initially described by an Event-B model can be produced. The remainder of this paper is structured as follows: Section 2 examines the related work referring to PDDL and Event-B. Section 3 presents the PDDL language and its verification and validation tools. Section 4 presents the Event-B formal method with its modeling, refinement, proof and validation steps. The set representation versus the predicative one is discussed in Section 5. Our formal development process for planning problems is presented in Section 6. Finally, Section 7 presents the conclusion and provides some implications for further studies.

2 Related Work

Several Integrated Development Environments (IDE) supporting PDDL exist [6] [7]. Such environments provide functionality for editing PDDL descriptions (both Domain and Problem parts), lexical-syntax checking of PDDL text, generating plans using a planner that accepts PDDL descriptions, and viewing the state space associated with the planning problem described by PDDL. The latter 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 explain the behavior of the PDDL planner to find a solution plan. This reduces the opacity of the PDDL planners. In fact, these planners are used as black boxes when there is no visualization.

The automatic planning community has developed many planners that accept PDDL descriptions [2]. These software tools, based on state space and planning graph search and SAT solvers, are rarely or no longer used by other software platforms such as robotic architectures, web architectures and software engineering architectures. The PDDL4J toolkit [8] written in Java is factorizing techniques from automatic planning: planning algorithms, planning heuristics, a number of planners, and the syntactic and semantic facilities of planning domain description languages such as PDDL. The correction of plan-solutions generated by PDDL planners is entrusted to validators (see Section 3). The authors of [9] recommend the use of the proof assistant Isabelle/HOL to formally develop a certified validator. To achieve this, they formalize the PDDL language in HOL.

In more or less completed jobs, automatic scheduling is seen as a model checking problem. For example, the work described in [10] explores the use of two model-checkers ProB and NuSMV for modeling and solving planning problems. It empirically compares these two model-checkers on five planning problems described by B (for ProB) and BDD (Binary Decision Diagrams for NuSMV). Similarly, an approach for translating PDDL to CSP to use the PAT model-checker in the planning domain is proposed in [11].

Unlike the B-method, the Event-B-method does not have a standard code generator. Indeed, the ultimate Event-B model depends on the targeted code: sequential, concurrent, distributed, etc. The work described in [12] allows Event-B to be translated into imperative languages that support sequential programming. The authors of [13] provides an approach for translating Event-B to BIP that supports distributed programming.

In general, even a confirmed modeler has a difficulty in carrying out a mathematically proven refinement process. To meet these limitations, automatic refinement is recommended. The BART tool [14] is used to assist B modeles in the refinement process, especially for B models close to the B0 language. In [15] a process allowing the formal decomposition of a centralized Event-B specification is proposed. Such a process combines manual and automatic refinement.

3 PDDL for automatic planning

PDDL is a formal declarative language that is proposed by the automatic planning community. It has been designed to allow the common representation of planning problems in International Conference on Automated Planning and Scheduling Competitions (ICAPSC). The PDDL language is based on the first-order logic to formalize data and on the Precondition/Post-condition specification to formalize treatments. A PDDL specification consists of two parts: domain and problem (See Table 1). The domain part encompasses the static and dynamic aspects of a planning domain. In the first part we define the set of types, predicates and possible actions. The problem part includes the definition of an initial state and the logical condition of the goal states of a given planning problem. In [16], a formal process based on the PDDL language that favors the obtaining of reliable PDDL descriptions is proposed. This process is successfully experimented on a set of representative case studies such as the Hanoï Towers.

Table 1: PDDL description structure

Construction of domain	**Construction of problem**
define (*domain* < *domainnName* >) (: *requirements* < *requirements list* >) (: *types* < *types list* >) (: *predicates* < *predicates code* >) (: *functions* < *functions code* >) (: *action* < *first action code* >) [...] (: *action* < *last action code* >))	*define* (*problem* < *problem name* >) (: *domain* < *domain name* >) (: *objects* < *objets code* >) (: *init* < *initial state code* >) (: *goal* < *gool code* >) (: *metric* < *metric code* >)

Two complementary tools are associated with the PDDL language. The first, called planner [17] is based on heuristic planning algorithms. A planner accepts as input a PDDL description (both domain and problem parts) and

produces, as output, plan-solutions. In addition, to generate a plan-solution, a planner provides more or less elaborate lexico-syntactic checks. In this work, we used the planners tool provided online by the two platforms: Web Planner [6] and Planning Domains [7]. The second, called validator [3] is used to check whether a given solution plan can be generated from a PDDL description.

4 Formal specification in Event-B

The Event-B method [4] encompasses a formal language and a formal development process. The Event-B formal language allows formalizing both data and treatments. On one hand, the data are described using a logico-set language (first-order predicates and set theory). On the other hand, treatments are described using the event concept. The event concept consists of three parts: local parameters, a guard and an action. The first two parts are predicates described using the Event-B logico-set language and the third part is described using a simple Event-B action language. Five actions types are proposed in Event-B: deterministic assignment (:=), two non-deterministic assignments (:∈ and : |), parallel action (||) and (SKIP) action. In Event-B, data and processing are grouped in two syntactic constructions CONTEXT and MACHINE.

The formal development process supported by Event-B via its Rodin platform is based on successive refinements with mathematical proofs. Such process starts with a coherent abstract model formalizing the concerned application. The coherence of this model is obtained by discharging the Proof Obligations (POs) associated with it. These POs are considered as correction criteria defined by the Event-B theory. Technically speaking, these POs are mathematical lemmas, automatically generated by Rodin's Proof Obligation Generator component, to be discharged automatically or interactively using the proofs provided by Rodin. Then, a multi-step refinement strategy is applied. These steps form a refinement chain. Each step takes an abstract model as input and produces a refined or concrete model as output. The refinement relationship between the two abstract and refined models is formally verified by the appropriate POs. The final refinement step produces a correct by construction concrete model with respect to the initial abstract model. This is explained by the fact that the refinement relationship is transitive.

In an Event-B specification, the CONTEXT construction brings together the modeling elements related to the static aspects of the application to be modeled. Such elements concern sets, constants, axioms and theorems. In this context, Axioms are supposedly true properties attached to sets and constants. The theorems must be demonstrated automatically or interactively using the proofs of the Rodin platform. However, an Event-B MACHINE construction brings together modeling elements related to the dynamic aspects of the processed application. Such elements concern variables, invariant, theorems and events. The variables form the state of the machine. The machine invariant includes invariance properties describing intra and inter-variable constraints

of the machine. The theorems must be deduced logically within the machine. Events can act on the state of the machine by preserving its invariant. A particular event called INITIALISATION allows initializing its state by establishing its invariant. Evidently, an Event-B machine can use modeling elements coming out from an Event-B context via the SEES relationship.

4.1 Case study: Sliding Puzzle Game

Sliding puzzle is a solitaire game in the form of a checkerboard created around 1870 in the United States by Sam Loyd. This problem consists of moving numbered tokens on an n × n grid to achieve a given configuration. The constraints imposed on the displacements are as follows:

- A movement can be carried out horizontally or vertically (diagonal movements are prohibited).
- To move a token numbered t_i, the destination location on the grid must be empty.

The 9 square (3×3) Sliding Puzzle Game is modeled in Event-B by a context called problem (See Figure 1) and a machine called T1 (See Figure 2). In this abstract Event_B, the state of this game is modeled by the variable grid of bijective function type whose starting set is a Cartesian product 1..3×1..3 and the ending set is from 0..8. Both initial and goal states are considered as two constants initial_state and goal_state introduced in the problem context. The (T1) machine contains two events INITIALISATION and goal, respectively, to initialize grid to initial_state and to see if grid is a goal state, i.e. coincides with goal_sate. The move event allows calculating all the following situations of the Sliding Puzzle Game starting from the current situation in grid. It is a non-deterministic event with four local parameters:

- **row** and **column** enabling to locate the empty square in grid
- **r** and **c** allowing to designate the chosen neighbor of the empty square in the grid.

CONTEXT *problem*
CONSTANTS *initial_sate*, *goal_state*
AXIOMS
$axm1:\ initial\_state \in 1..3 \times 1..3 \rightarrowtail\!\!\!\!\!\rightarrow TILE$
$axm2:\ goal\_state \in 1..3 \times 1..3 \rightarrowtail\!\!\!\!\!\rightarrow TILE$
$axm3:\ initial\_state = \{1 \mapsto 1 \mapsto 4, 1 \mapsto 2 \mapsto 0, 1 \mapsto 3 \mapsto 8, 2 \mapsto 1 \mapsto 6,$
$2 \mapsto 2 \mapsto 3, 2 \mapsto 3 \mapsto 2, 3 \mapsto 1 \mapsto 1, 3 \mapsto 2 \mapsto 5, 3 \mapsto 3 \mapsto 7\}$
$axm4:\ goal\_state = \{1 \mapsto 1 \mapsto 1, 1 \mapsto 2 \mapsto 2, 1 \mapsto 3 \mapsto 3, 2 \mapsto 1 \mapsto 4,$
$2 \mapsto 2 \mapsto 5, 2 \mapsto 3 \mapsto 6, 3 \mapsto 1 \mapsto 7, 3 \mapsto 2 \mapsto 8, 3 \mapsto 3 \mapsto 0\}$
END

Fig. 1: Event_B abstract context of the Sliding Puzzle Game

MACHINE $T1$
SEES *problem*
VARIABLES *grid*
INVARIANTS
$inv1: \ grid \in 1..3 \times 1..3 \rightarrowtail\!\!\!\!\!\rightarrow 0..8$
$DLF: \ \exists r, c, row, column \cdot (r \in 1..3 \wedge c \in 1..3 \wedge row \in 1..3 \wedge column \in 1..3 \wedge$
$((r = row + 1 \wedge c = column) \vee (r = row - 1 \wedge c = column) \vee$
$(r = row \wedge c = column + 1) \vee (r = row \wedge c = column - 1)) \wedge$
$grid(row \mapsto column) = 0)$
EVENTS
$INITIALISATION \triangleq$
$act1: grid \ \in 1..3 \times 1..3 \rightarrowtail\!\!\!\!\!\rightarrow 0..8$
END
$move \triangleq$
ANY $r, \ c, \ row, \ column$
WHEN
$grd1: r \in 1..3$
$grd2: c \in 1..3$
$grd3: row \in 1..3$
$grd4: column \in 1..3$
$grd5: (r = row + 1 \wedge c = column) \vee (r = row - 1 \wedge c = column) \vee$
$(r = row \wedge c = column + 1) \vee (r = row \wedge c = column - 1)$
$grd6: grid(row \mapsto column) = 0$
THEN
$act1: \ grid := grid \Leftarrow row \mapsto column \mapsto grid(r \mapsto c),$
$r \mapsto c \mapsto grid(row \mapsto column)$
END
$goal \triangleq$
$grd1: grid = goal\_state$
THEN
$skip$
END
END

Fig. 2: Event_B abstract machine of the Sliding Puzzle Game

4.2 Set representation versus predicative representation

In Event-B, the data are formalized due to its logic-set language: first-order predicate calculus augmented by set theory (sets, relations and functions in the mathematical sense). The set data include constants and variables of a set type (abstract sets introduced in the SETS clause of an Event-B context and predefined set), function (partial, total, injective, surjective, bijective and lambda), relation and surjective relation. While predictive data include constants and variables of type BOOL (predefined set with FALSE and TRUE)

and total function. The latter has as starting set either an abstract set or the cartesian product of several abstract sets. Its ending set is imperatively BOOL. In the formal Event-B development process based on successive refinements with mathematical proofs, the set data are generally used in the beginning of the refinement chain and the predicative data are used in the end of the said chain.

5 Combining Event-B and PDDL for automatic planning

In this Section, a development process that couples Event-B and PDDL to specify and solve planning problems is proposed (See Figure 3). The Event-B method is used to develop an Event-B model for a planning problem. Through successive refinements, an Event-B ultimate model correct by construction and valid using the Event-B proof/validation tools is obtained. As a second step, the Event-B ultimate model can be easly translated into a PDDL description. Finally, using the PDDL planner tool on this generated PDDL description, plan-solutions related to the planning problems initially described by Event-B can be produced. Our process combines manual refinement and automatic refinement. The manual refinement ultimately produces the ultimate set-model. Depending on the planning problems, this manual refinement could involve, the decomposition of an event, the reinforcement of guards and the enrichment of the state (addition of variables and/or invariant properties). As for automatic refinement, it concerns the refinement of data allowing going from a set representation to a predicative representation. In particular, the automatic data refinement tool shall solve the problem of rewriting events following the introduction of a predictive variable and the gluing invariant. For reasons related to the automatic discharging proof obligations, this tool must work step by step taking into account the types of variables and set constants.

5.1 Event-B2PDDL translation rules

In this section, a set of intuitive rules allowing the systematic translation of Event-B elements to PDDL elements is proposed. Regarding the Event-B domain elements, the following structural rules is proposed:

- **Rule 1:** An Event-B abstract set can be specified by a PDDL type.
- **Rule 2:** An Event-B constant can be specified by a PDDL constant.
- **Rule 3:** An Event-B constant or variable of type BOOL can be specified by a PDDL predicate which does not contain parameters. A predicate can be true or false.
- **Rule 4:** An Event-B function can be defined in a context as a constant or in a machine as a variable. An Event-B total function having the type "BOOL" as an end set can be specified by a PDDL predicate.
- **Rule 5:** An Event-B event can be specified by a PDDL action. This requires the translation of the Event-B formulas into PDDL.

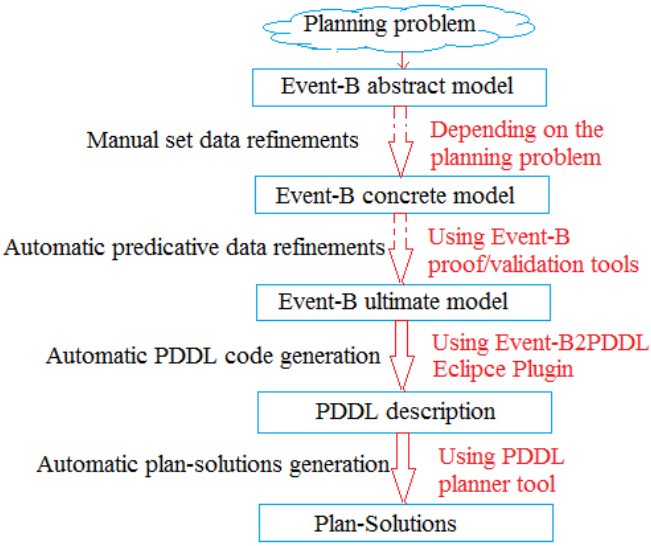

Fig. 3: Process coupling Event-B and PDDL

- **Rule 6:** An Event-B formula can be translated into PDDL by a predicate or by an expression. The relevant predicates are used at the level of event guards. Indeed, the other predicates expressing theorems, axioms and invariants are ignored. In the ultimate Event-B model to be translated into PDDL, the predicates are built on total functions whose starting set is the Cartesian product of zero or more abstract or enumerated sets and the ending set is BOOL. The Event-B expressions to be translated into PDDL are either Boolean constants (TRUE, FALSE), or functional expressions using the overload operator.

  Tables 2 and 3 illustrate these translation rules.

5.2 Event-B2PDDL automatisation

In order to automate our Event-B to PDDL translation rules, we have modelled, implemented and tested an Eclipce plugin called Event-B2PDDL [18]. It accepts as input an Event-B concert model and produces a PDDL description acceptable by the PDDL planner tool. To do so, a set of Eclipce APi and Xtext and Xpand MDE languages [19] are used. The Xtext language is used to create an integrated development environment specific to an Xtext grammar of the Event-B source language. However, the Xpand tool proposes a template language specialized in code generation. These two languages are powerful complementary MDE languages. Indeed, the Xtend language is based on the Java language. This favors the use of adequate Java libraries. In addition, the Xpand language supports the main language features like syntax

Table 2: From Event-B to PDDL translation rules

Event-B formulas	**PDDL formulas**
Sets translation	
$SETS\ <TYPE1>\ <TYPE2>$ ...	(: *types* $<TYPE1>\ <TYPE2>$)
Constants translation	
CONSTANTS $cst1\ cst2$ ... AXIOMS $axm1$: *partition* ($TYPE1$, $cst1$, $cst2$, ...)	(: *constants* $<cst1>\ <cst2>$ ... $-<TYPE1>$)
Bool Variables translation	
CONSTANTS $name1$ AXIOMS $name1 \in BOOL$ VARIABLES $name2$ INVARIANTS $name2 \in$BOOL	(: *predicates* ($name1$) ($name2$) ...)
Formulas translation	
$P \wedge Q$ $P \vee Q$ $P \Longrightarrow Q$ $\neg P$ $\forall\ z \bullet P \Longrightarrow Q$ $\exists\ z \bullet P \wedge Q$ $E = F$ $E /= F$ $b := TRUE$ $b := FALSE$ $f\ (\ x\ ) := TRUE$ $f\ (\ x\ ) := FALSE$ $f := f \Leftarrow \{\ x \mapsto TRUE,\ y \mapsto FALSE\}$	(*and* $P\ Q$) (*or* $P\ Q$) (*imply* $P\ Q$) (*not* P) (*forall* (? z) (*imply* $P\ Q$)) (*exists* (? z) (*and* $P\ Q$)) ($=\ E\ F$) (*not*($=\ E\ F$)) (b) (*not* (b)) ($f?x$) (*not* ($f?x$)) (*and* ($f?x$) (*not* ($f?y$)))
Functions translation	
VARIABLES $name\_function1$ INVARIANTS $inv\_name\_function1$: $name\_function1$ $\in$TYPE1$\times$TYPE2 ... $\rightarrow$BOOL $name\_function2$ ...	(: *predicates* ($name\_function1$ $?var1$ $-$TYPE1 $?var2$ $-$TYPE2 ...) ($name\_function2$ ...))

colouring, error highlighting, navigation, refactoring, code completion, pluggable type system and dynamic dispatch of functions. These features enable an easy implementation of Xpand programs. In our tool, the Xtend tool is used to implement our translation tool and produce the PDDL models in a top-down way.

5.3 Event-B2PDDL validation

Our proposed process based on the coupling of Event-B and PDDL is validated on a set of representative case studies (available at [18]). The choice of these case studies is based on the verification of planning-specific problems, and properties common to all applications. In particular, our process is validated on the Hanoï Towers planning problem and on the Sliding Puzzle Game (See Section 4.3).

Table 3: Events translation

Event-B formulas	**PDDL formulas**
$evt\_name = \ STATUS\ ordinary$ ANY $var1\ var2$ $WHERE$ $grd1: \ var1 \in TYPE1$ $grd2: \ var2 \in TYPE2$ $grd3: \ < GD1 >$ $grd4: \ < GD2 > \ ....$ $THEN\ act1 :< ACT1 >$ $act2: \ < ACT2 > \ ....END$	$(: action \ < evt_name >$ $: parameters\ (?var1 \ - TYPE1$ $?var2 - TYPE2$ $...)$ $: precondition$ $(and\ (< GD1 >)$ $(< GD2 >)$ $.....)$ $: effect$ $(and\ (< ACT1 >)\ (< ACT2 >)$ $.....))$

The ProB tool [20] allows the animation of the Event-B models proven by the Rodin platform proofs. Such animation is considered as a certain "execution" of the Event-B models. Indeed, the ProB animator is equipped with a constraint solver capable of establishing solutions for event guards. The triggering of a triggerable event chosen by the modeler leads to the execution of the action associated with this event by ProB. Knowing that a triggerable event is an event whose guard is satisfied. Thus, we have successfully used the ProB's animator on the different models of these case studies. In particulier in the Sliding Puzzle Game application, eight Event-B models are used: an initial abstract model (T1); an ultimate model (T8) and six intermediate models (T2, T3, T4, T5, T6 and T7). The ProB's animator is successfully used on these different models. Table 4 summarizes the POs linked to the different Event-B models of the Sliding Puzzle Game application. We have used the external provers of the RODIN platform, in particular the SMT provers and the ProB counter-prover, to discharge the POs in an interactive way. Moreover, we introduced lemmas in order to discharge the DLF (DeadLock Free) theorems associated to the different machines. Thus, all the machines forming the Sliding Puzzle Game application are non-blocking. This allows access to all the attainable states of the state space related to the Sliding Puzzle Game. The cyclic character of the Sliding puzzle state space (for example, move_left followed by move_right) fully justifies the absence of blocking of Event-B models with Sliding Puzzle Game. The static predicates neighbor_left, neighbor_right, neighbor_up and neighbor_down are modeled by Event-B constants. The initialization of these constants is the respon-sibility of the modeler. In order to reduce the risk of error, we have established properties relative to these constants considered as theorems. The eight Event-B models are available at [18].

Thus, applying our Event-B2PDDL Eclipce plugin on the (T8) ultimate Event-B model, we obtained the PDDL description. The Domain (respectively Problem) construct of this PDDL genrated description is presented on the Figure 4 (respectively on Figure 5). Finally, we submitted the PDDL generated description to the WEB PLANNER [6]. This planner established a solution plan including a sequence, of length 25 trips. Such plan-solution explains the

Table 4: POs associated to Event-B models of Sliding Puzzle Game

Machine	POs	Automatic POs	Interactive POs	Not discharged POs
T1	6	1	5	0
T2	10	4	6	0
T3	45	36	9	0
T4	86	19	67	0
T5	16	1	15	0
T6	29	10	19	0
T7	54	8	46	0
T8	42	9	33	0

direction of the legal movements of the tiles via the names of the operators executed, i.e. move_right, move_left, move_up and move_down.

(**define** (**domain** $n - sliding - puzzle$)
(**:types** *position tile*)
(**:predicates** (*at ?position − position ?tile − tile*)
(*empty ?position − position*)
(*neighbor_left ?p1 − position ?p2 − position*)
(*neighbor_right ?p1 − position ?p2 − position*)
(*neighbor_up ?p1 − position ?p2 − position*)
(*neighbor_down ?p1 − position ?p2 − position*))
(**:action** $move_left$
:parameters (?*from ?to − position ?tile − tile*)
:precondition (*and* (*neighbor_left ?from ?to*) (*at ?from ?tile*) (*empty ?to*))
:effect (*and*
(*at ?to ?tile*) (*empty ?from*) (*not* (*at ?from ?tile*)) (*not* (*empty ?to*))))
(**:action** *move_right*
:parameters (?*from ?to − position ?tile − tile*)
:precondition (*and* (*neighbor_right ?from ?to*) (*at ?from ?tile*) (*empty ?to*))
:effect (*and*
(*at ?to ?tile*) (*empty ?from*) (*not* (*at ?from ?tile*)) (*not* (*empty ?to*))))
(**:action** *move_up*
:parameters (?*from ?to − position ?tile − tile*)
:precondition (*and* (*neighbor_up ?from ?to*) (*at ?from ?tile*) (*empty ?to*))
:effect (*and*
(*at ?to ?tile*) (*empty ?from*) (*not* (*at ?from ?tile*)) (*not* (*empty ?to*))))
(**:action** *move_down*
:parameters (?*from ?to − position ?tile − tile*)
:precondition (*and* (*neighbor_down ?from ?to*) (*at ?from ?tile*) (*empty ?to*))
:effect (*and*
(*at ?to ?tile*) (*empty ?from*) (*not* (*at ?from ?tile*)) (*not* (*empty ?to*)))))

Fig. 4: PDDL domain construct related to the SPG

(**define**(*problem p − sliding − puzzle*)
(**: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_2 p_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 p_1_1 t_4*) (*at p_1_3 t_8*) (*at p_2_1 t_6*) (*at p_2_2 t_3*)
(*at p_2_3 t_2*) (*at p_3_1 t_1*) (*at p_3_2 t_5*) (*at p_3_3 t_7*)
(*neighbor_left p_1_1 p_1_2*) (*neighbor_right p_1_2 p_1_1*)
(*neighbor_left p_1_2 p_1_3*) (*neighbor_right p_1_3 p_1_2*)
(*neighbor_left p_2_1 p_2_2*) (*neighbor_right p_2_2 p_2_1*)
(*neighbor_left p_2_2 p_2_3*) (*neighbor_right p_2_3 p_2_2*)
(*neighbor_left p_3_1 p_3_2*) (*neighbor_right p_3_2 p_3_1*)
(*neighbor_left p_3_2 p_3_3*) (*neighbor_right p_3_3 p_3_2*)
(*neighbor_up p_1_1 p_2_1*) (*neighbor_down p_2_1 p_1_1*)
(*neighbor_up p_1_2 p_2_2*) (*neighbor_down p_2_2 p_1_2*)
(*neighbor_up p_1_3 p_2_3*) (*neighbor_down p_2_3 p_1_3*)
(*neighbor_up p_2_1 p_3_1*) (*neighbor_down p_3_1 p_2_1*)
(*neighbor_up p_2_2 p_3_2*) (*neighbor_down p_3_2 p_2_2*)
(*neighbor_up p_2_3 p_3_3*) (*neighbor_down p_3_3 p_2_3*))
(**:goal** (*and*
(*at p_1_1 t_1*) (*at p_1_2 t_2*) (*at p_1_3 t_3*) (*at p_2_1 t_4*)
(*at p_2_2 t_5*) (*at p_2_3 t_6*) (*at p_3_1 t_7*) (*at p_3_2 t_8*))))

Fig. 5: PDDL problem construct related to the SPG

6 Conclusion

The planning community has developed languages (in this case PDDL), planners and validators for the description of planning problems and the generation and validation of plan-solutions. However, the reliability of PDDL descriptions is dealt with a posteriori. In this work, we proposed a process based on the combining of Event-B and PDDL for the development of planning problems. The Event-B formal method is used upstream and PDDL is generated from the ultimate Event-B model, which is the result of a chain of Event-B models linked by the formal refinement relationship in the sense of the Event-B theory. Our process coupling Event-B and PDDL has been successfully applied to several test cases such as the Hanoï Towers planning problem and the Sliding Puzzle Game.

Currently, we intend to revisit our Event-B2PDDL plugin to make it a software engineering tool worthy of its name. To achieve this, it is necessary to identify and justify the subset of Event-B translatable into PDDL. Like the formal method B which proposes a subset of B called B0 translatable in an imperative programming language like C and Ada, the subset of Event-B translatable in PDDL will be called Event-B0. This is being finalized. In addition, the Event-B0 is designed to allow the bilateral passage between Event-B and

PDDL. The transformation of PDDL to Event-B0 (and therefore the Event-B) promotes formal verification of the PDDL descriptions. Finally, our Event-B and PDDL coupling process is reusing known and recognized tools in the field of automatic planning, namely planners accepting PDDL descriptions.

7 Declarations

7.1 Ethical Approval

We declare that this paper does not contain any human or animal studies.

7.2 Competing interests

We declare that we have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.

7.3 Authors' contributions

Professor Bhiri M. T. and Phd Ammar S. provide a first Word version of this article. Afterwards, Dr. Sakka Rouis T. extends, improves and prepares a LaTeX version for this first paper.

7.4 Funding

We declare that the authors have no funding.

7.5 Availability of data and materials

We declare that all data and material belongs to the authors and no permission is required.

References

[1] Haslum, P., Lipovetzky, N., Magazzeni, D., Muise, C.: An introduction to the planning domain definition language. Synthesis Lectures on Artificial Intelligence and Machine Learning **13**, 1–187 (2019)

[2] Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: Pddl - the planning domain definition language. Technical report, Yale Center for Computational Vision and Control, (1998)

[3] Howey, R., Long, D., Fox, M.: Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In: ICTAI, IEEE Computer Society, pp. 294–301 (2004)

[4] Cataño, N.: Teaching formal methods: Lessons learnt from using event-b. In: Formal Methods Teaching - FMTea 2019, Held as Part of the Third World Congress on Formal Methods, FM 2019, Porto, Portugal, October 7, 2019, Proceedings, vol. 11758, pp. 212–227 (2019)

[5] Afendi, M.: A correct by construction approach for the modeling and the verification of cyber-physical systems in event-b. In: Rigorous State-Based Methods - 7th International Conference, ABZ 2020, Ulm, Germany, May 27-29, 2020, Proceedings, pp. 401–404 (2020)

[6] Magnaguagno, M.C., Pereira, R.F., Móre, M.D., Meneguzzi, F.: Web planner: A tool to develop classical planning domains and visualize heuristic state-space search. In: Proceedings of the Workshop on User Interfaces and Scheduling and Planning, UISP, pp. 32–38 (2017)

[7] Muise, C., Lipovetzky, N.: KEPS Book: Planning.Domains, pp. 91–105 (2020)

[8] Pellier, D., Fiorino, H.: PDDL4J: a planning domain description library for java. Journal of Experimental and Theoretical Artificial Intelligence **30(1)**, 143–176 (2018)

[9] Abdulaziz, M., Lammich, P.: A formally verified validator for classical planning problems and solutions. In: IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018, 5-7 November 2018, Volos, Greece, pp. 474–479 (2018)

[10] Hörne, T., van der Poll, J.A.: Planning as model checking: the performance of prob vs nusmv. In: ACM International Conference Proceeding Series, in Developing Countries, SAICSIT 2008, 6-8 October 2008, Wilderness, South Africa, vol. 338, pp. 114–123 (2008)

[11] Li, Y., Sun, J., Dong, J.S., Liu, Y., Sun, J.: Translating PDDL into csp# - the PAT approach. In: 17th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2012, Paris, France, July 18-20, 2012, pp. 240–249 (2012)

[12] Méry, D., Singh, N.K.: Automatic Code Generation from Event-B Models. In: SoICT 2011, pp. 179–188. ACM ICPS, Hanoi, Vietnam (2011)

[13] Siala, B., Bhiri, M.T., Bodeveix, J.-p., Filali, M.: An event-b development process for the distributed bip framework. In: Proceedings of the ICFEM International Conference, Tokyo, pp. 313–328 (2016)

[14] Requet, A.: BART: A tool for automatic refinement. In: Abstract State Machines, B and Z, First International Conference, ABZ 2008, London, UK, September 16-18, 2008. Proceedings, vol. 5238, p. 345 (2008)

[15] Siala, B., Bodeveix, J., Filali, M., Bhiri, M.T.: Automatic refinement for event-b through annotated patterns. In: 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing, PDP 2017, St. Petersburg, Russia, March 6-8, 2017, pp. 287–290 (2017)

[16] Ammar, S., Sakka Rouis, T., Bhiri, M.T., Gaaloul, W.: Automatic processing of planning problems: Application on representative case studies. In: Advances in Computational Collective Intelligence - 14th International Conference, ICCCI 2022, Hammamet, Tunisia, September 28-30, 2022, vol. 1653, pp. 436–445 (2022)

[17] Roberts, M., Howe, A.E.: Learning from planner performance. Artif. Intell. Journal **173**(5-6), 536–561 (2009)

[18] Ammar, S., Bhiri, M.T., Sakka Rouis, T.: Event-B2PDDL's Inline Repository. (accessed Dec. 25, 2022). https://sourceforge.net/projects/event-b2pddl-tool/

[19] Sakka Rouis, T., Bhiri, M.T., Sliman, L., Kmimech, M.: An mde-based tool for early analysis of UML2.0/PSM atomic and composite components, vol. 14, pp. 1647–1657 (2020)

[20] Krings, S., Bendisposto, J., Leuschel, M.: From failure to proof: The prob disprover for B and event-b. In: 13th SEFM International Conference,York, UK, vol. 9276, pp. 199–214 (2015)