IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 22, 2021, accepted April 18, 2021, date of publication May 4, 2021, date of current version May 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3077521

Automated Planning With Invalid States Prediction

CAIO GUSTAVO RODRIGUES DA CRUZ[1], RODRIGO ROCHA SILVA[2,3], MAURICIO GONÇALVES VIEIRA FERREIRA[1], AND JORGE BERNARDINO[3,4], (Member, IEEE)

[1]National Institute for Space Research (INPE), São Jose dos Campos 12227-010, Brazil
[2]FATEC Mogi das Cruzes, São Paulo Technological College, Mogi das Cruzes 08773-600, Brazil
[3]Centre for Informatics and Systems of the University of Coimbra (CISUC), 3030-290 Coimbra, Portugal
[4]Polytechnic of Coimbra, Coimbra Institute of Engineering (ISEC), 3030-199 Coimbra, Portugal

Corresponding author: Caio Gustavo Rodrigues Da Cruz (caio.cruz2@fatec.sp.gov.br)

ABSTRACT The increase of automated systems in space missions raises concerns about safety and reliability in operations carried out by satellites due to performance degradation. There have been several studies about the automatic planning process, but many approaches are generated with invalid states. The invalid state can be understood as a prohibited, degraded or risky scenario for the domain. This paper proposes an automated planning process with restrictions that enables automatic planners to not generate plans with invalid states. We implement a validator method for the planner software which proves that plan generation matches the restrictions imposed on the domain. In the experiments, we test an automatic planning process that is specific to the aerospace area, where a knowledge base with invalid states is available in the context of the operation of a satellite. Our proposal to carry out the verification of invalid states in automatic planning, can contribute to plans being generated with higher quality, ensuring that the goal of a plan is only achieved through valid intermediate states. It is also expected that the generated plans will be executed with better performance and will require less computational resources, since the search space is reduced.

INDEX TERMS Automated planning, domain rule learning, machine learning, PDDL.

I. INTRODUCTION

Plan validation is necessary when automated planning is used to solve complex problems, such as the control of satellites in planning operations that involve aerospace area [1]. Generating plans that are consistent with the current domain situation, has encouraged the development of applications to be responsible for diagnosing and validating automatically generated plans. In addition to safety and reliability, satellite operation requirements have been intensified due to the growing number of automated systems in this domain area [2].

In the validation planning stage, all states through which the domain must go through to reach the objective of the problem are verified in the plan. If there are one or more invalid states that compromise the domain and that cannot be executed, the plan gets considered as rejected [1]. The planning of flight operations for satellites is too complex to be solved solely by classical planning techniques because invalid state restrictions are not considered during the plan's state transition [3].

Several planning domains encounter problems of having a large set of solutions or having a set of goals that cannot be completely achieved. There are still scenarios where plans of expected solutions can be generated to contain invalid states given the characteristics of the domain. In such cases, the way in which the goal is achieved may be more important than actually reaching the goal itself. For this reason, when it is possible, it is important to generate valid plans that reach all the goals of the problem [4].

The objective of this work is to propose an architecture for the automatic generation of plans with the validation of invalid states being done through a process of learning and configuring restrictions on the domain. In the approach, a filter is created for the states that are not included in the planned solution. In this case, valid plans are generated respecting the changes that occur in the domain's operating state. In order to validate the application of the architecture, a case study in the aerospace area was used in the experiments. The group of invalid states classified in the learning process were configured with the respective invalid states and the results of the

The associate editor coordinating the review of this manuscript and approving it for publication was Szidónia Lefkovits.

planning stage were compared starting from each configured group. The validation was carried out by comparing the result of the planning stage with each configured group.

This study contributes a new way to validate the invalid states contained within planning domains. Allowing this approach to be applied to validate emerging restrictions not only in the domain of satellite planning, but also in planning domains where states are often modified and new restrictions may arise.

The rest of this study is structured as follows: Section II introduces the concepts and theoretical fundamentals, such as the relevant concepts of *planning*, *planning with preferences* and the inclusion of new techniques in *planning* with the presentation of related works. Section III describes the proposal for an automated planning architecture that performs the prediction of invalid states. Section IV presents the experiment implemented in the space area. Finally, section V presents the conclusion and future work.

II. BACKGROUND AND RELATED WORK

This section provides context on planning and related work. There are several automated planning methods that exist. In [4]–[8], the authors include preferences in planning. Other authors like [9]–[12], use learning techniques. However, they do not implement an invalid state prediction solution.

A. CLASSICAL PLANNING

Automated planning is a branch of Artificial Intelligence that focuses on problem solving and organizes actions through a system in order to achieve the goal of the problem [13]. A planning domain is defined as a finite set of possible states $S = \{s_0, s_1 \ldots s_k\}$, and a finite set of actions $A = \{a_1, a_2 \ldots a_k\}$, applicable to the domain states [13]. A planning problem can be represented as follows: $P = (\Sigma, s_i, S_g)$, where Σ is the state transition system, s_i is the initial state and S_g is a set of goal states. In the concept of classical planning, a plan is not deterministic and there may be different ways of finding the sequence of actions that transform the initial state s_i into the objective state S_g.

$$s_1 = \gamma(s_0, a_1),\ s_2 = \gamma(s_1, a_2), \ldots,\ s_k = (s_{k-1}, a_k) and\ s_k \in S_g$$

The state transition system represented above describes the function $\gamma(s, a)$ which applies an action to a state, resulting in a state transition system. For each generated state, an action is applied until the generated state corresponds to the goal. A problem can be solved in a number of ways. In other words, a set of different stages can transform the initial state into the goal state [13].

In classical planning, STRIPS (Stanford Research Institute Problem Solver) is a technique used to find solutions from a domain and a problem [14]. STRIPS aims to improve efficiency by reducing the size of the search space going through all possible states after applying the domain actions until it finds the goal state [14]. However, when classical techniques are used to solve more complex problems they can create invalid plans, because the generation phase does not consider existing restrictions in the domain [15].

Automated planning is performed using a specific planning software. The planner must find a sequence of actions that transform the initial state into the goal state using only the actions contained in the domain [16]. The result is a plan containing the sequence of actions required to find the goal of the problem. The most common language in automated planning is PDDL (Planning Domain Definition Language) introduced in 1998 by [16].

PDDL was established as the standard language commonly used in automated planning. PDDL paves the way for stronger collaboration, providing a platform for benchmarking approaches, exchanging different tools, as well as techniques and problems, thanks to a series of international planning competitions that have been associated and extended, into several stages to capture more expressive variants [17].

Recent research includes a demonstration that temporal characteristics can be compiled in polynomial work and be subject to certain restrictions that can appear in the problem [8], while other research examined the compilability of conditional effects [5].

Over time planning approaches have evolved to solve more complex problems with new techniques such as the inclusion of preferences, pre-processing using macros and the application of learning using data mining in different planning contexts.

B. PLANNING WITH PREFERENCES

As seen in literature, there are different studies on AI planning that propose to solve more complex planning problems. Some of the approaching issues are the following: generating quality plans; planning complex problems; planning with uncertainty; user preferences and preferences learning. Most of these works are related to the use of preferences, an interdisciplinary topic that has been extensively studied in recent years and has advanced different techniques and planning forms [18].

In order to improve the quantity of expressed criterion in complex plans, the PDDL language was extended to a PBP (Preference-based Planning). PDDL3 uses HTN (Hierarchical Task Network) to increase the expressive power over the plan's quality specification [19]. Two new features have been added that allow the user to express strong and soft constraints about the structure of the desired plans, as well as strong and soft problem goals. The first feature is the ability to express goals that apply not only to the final state of the trajectory of states visited by a plan, but also to intermediate states. The second extension is the ability to express soft constraints or mild preferences. Furthermore, the authors make it clear that the extensions are useful for expressing restrictions that manage the plan quality, instead of controlling the knowledge itself [7], [20].

The PBP aims to find the most preferred plans in a planning instance using some criteria to determine when a plan is more suitable than another. The search for the development of ideal plans is a theme addressed in [21], which examines various types of representations to solve planning problems. The theoretical decision planning uses the MDP (Markov Decision Process) to explore the assembly of optimal policies in order to indicate planning problems. The MDP associates a reward function with each state transition, to define the users' preferences. All the possible states are classified quantitatively where the chosen optimal policy returns an action considering the history of actions performed by the agent [21]. In this approach, obstacles can be found when combining methods that explore different structures at the same time, making it a challenge to integrate or develop additional tools [21].

Another approach based on heuristic techniques was created, named PSP (Partial Satisfaction Planning), which offers resources to partially solve problems, and can meet a collection of objectives [6]. In this approach, the time needed to find the best solutions can take very long, as the search for better plans ends up generating a much larger number of research nodes. Another limiting factor is the use of the evaluation function used to trim non-promising states, which can inevitably lose an optimal plan while removing a state from a research space [6].

There are works that compile the planning instance creating a method of controle procedure based on PDDL. [4] allowed the representation of the planning domain as a finite state automaton, adding a predicate for modifying effects and predefined conditions of actions, allowing the procedure to be fulfilled, in a process called PCK (Procedural Control Knowledge) [4]. Although, it is a general approach, the biggest advantage of this technique is the possibility to deal with a variety of domain control specification languages [4].

C. DOMAIN REMODELLING AND LEARNING IN PLANNING

In order to improve the efficiency of planners and making them solve a greater number of problems, modular modules were proposed. The use of reformulation and configuration techniques have contributed to the improvement of domains, such as the domain remodeling to identify irrelevant actions and operators that can be removed or replaced by other actions in the plan. [22] proposed a pre-processing step to remove actions that are generated by a sequence of other actions that do not contain it. This process reduces the number of explored states and improves the search time. Another technique applied to domain remodeling planning is the use of macro operators, which act as shortcuts to deeper states in the branches of the planning search space. For example, [23] adds macro operators in the domain, increasing it to solve problem instances like a new normal operator.

Machine learning has been used in different contexts in order to improve the planning process. [9] proposed the Marvin planner, which evolved the FF (Fast Forward) planner by applying macro learning. During the plan searching process, [10] creates a predictor to choose the best heuristic. [11] proved that specific remodeling of the planning instance using machine learning may improve the performance of plan generation. There is also the implementation of a division process such as automatic actions, which decompose the most complex operators into simpler ones using the type domain-specific learning proposed in [12].

Although macros act as shortcuts in the search space, they increase the branching factor and potentially make some operators redundant [11]. Removing redundant or irrelevant operators, on the other hand, decreases the branching factor. But it is difficult to find operators that are safe to remove for all instances in a domain. Therefore, according to a study in [11] the combination of a macro addition and an operator removal is a promising path that can achieve a significant acceleration during planning.

In approaches using macros, machine learning is applied to learn domain rules. In automated planning, domain models can change over time, with the emergence of new rules or certain situations that may not be available sometimes, resulting in a major problem that generates invalid plans.

An opportunity to also apply data mining is to produce rules or rule mappings for quality measures in the plan. In the work presented by [24], on the use of learning in planning, the planning problems found in space agencies were observed because they require the validation of plans before they are proposed. According to [24] since planning languages are unable to capture the full description of the domain, he suggests that automated planning can incorporate a simulation made in the plan validation process that can be used during plan generation. He also concludes that data mining can be used to improve the quality of planning in several ways. First, the validator can be seen as a function that can be an approximation in order to speed up the planner. In this case, data mining from the validator can produce an efficient and approximate function that can serve as a quick first cut once the automated planning is completed. An even better approach would be one that can be used much earlier in the planning process in a similar way to useless learning as proposed by [24].

D. ANALYSIS AND COMPARISON

In recent studies and research we can find many different planning approaches, which have all been proposed to solve different problems. The difference between one approach and another is the use of different techniques, such as, the inclusion of preferences, remodeling of domains, use of macros, use of machine learning, among other resources that have been incorporated in planning. TABLE 1 shows a comparison of the differences between the planning approaches presented and considers the main resources used by them. It is understood that each approach includes different resources for planning and for solving specific problems.

In this work, similarly to the previously mentioned planning approaches, we propose the inclusion of a new resource for plan validation in automated planning through the

TABLE 1. Comparative table of planning approaches.

Approach	Differential
HTN - Trajectory restrictions	Violated preferences metric
MDP - Reward function	Classification of actions taken
PSP - Heuristics	Planning with objectives subset
PCK - Automata	Additional action predicate
Macro Operators	Shortcuts to deeper states
Marvin Planner	Macro learning application

verification of invalid states, which is different from the rest of the other works found and which contributes to the generation higher quality plans.

III. PROPOSED ARCHITECTURE

In order to create an application that allows a better representation of the domain, whilst considering its changes over time, an architecture was designed using a planner with state validation. In addition to the validation being done during planning time, it significantly reduces the total execution time and the application of a learning process that classifies the domain's current states contributing for plans with higher quality, since invalid states are excluded from the solution.

A. INVALID STATE DEFINITION

An invalid state is formed by current situations that undermine a domain scenario. In order to illustrate the understanding of the invalid states formulation, FIGURE 1 shows an example of the planning problem in the world of blocks. Where three different moments (Instant I, II, III) are presented using the same domain scenario, with three stacked blocks on a table. In this example it is noted that with the variation of time, the domain states planning has changed, as certain circumstances have arisen over time.

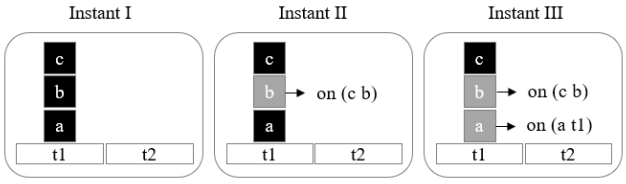

FIGURE 1. Illustration of invalid states.

The first instant represents the scenario where there is no invalid situation. In the second instant, a given situation becomes invalid, assuming that *block b* is affected by an anomaly that prevents it from sustaining *block c*. In this case, we say that the situation *on (c b)* has become invalid. In the third instant, another invalid state is formed by two invalid situations, assuming that the element *block a* cannot be stacked on the table *table t1*, the situation *on (a t1)* also becomes invalid.

In the scenario presented in Instant III, the two invalid situations – one object stacked on top of another – are similar, but they connect elements with different types. In the *on (c b)* situation, there is a relationship between two elements of the block type, yet in the situation *on (a t1)* one element is of the block type and the other element is of the table type. However, to identify which domain action generates the invalid situation, it is necessary to observe which elements are related to the action's effect. For example, in this domain, the action to stack two blocks, has an effect with the definition *on (block1 block2)* indicating that the effect of applying the action *stack* generates the situation *on (c b)*. The invalid situation *on (a t1)* was generated by the action *mover* that has the definition *on (table block)* in its effect. To summarize, the effect of each action defined in the planning domain is composed by a set of situations. The parameters of each situation assume the values of the objects described in the definition of the planning problem, this way we can formulate the invalid states.

B. ARCHITECTURE OVERVIEW

In this section, we present in FIGURE 2 the architecture that consists of grouping the main components that involve the domain state classification and automated planning processes. The components were grouped into three main layers: the state classification layer, configuration, and planning. The proposed architecture aims to generate plans according to the current changes and situations of the current domain, allowing the prediction of invalid states. Starting with the configuration of preferences included in the configuration layer by a domain expert, it is possible to predict invalid states in the automated planning process. In the next sections, we explain each one of the three layers.

C. CLASSIFICATION LAYER

The classification layer encapsulates the learning process of planning domain. The input data at this layer is obtained through routine operation and the knowledge of experts in the field. The classification criteria is the information that defines all possible classes, in which the domain can be classified. The formulation of these criteria depends exclusively on the domain, as it is the set of characteristics that configure each classification. The state classifier component represents the learning model, which uses the criteria for training the classification algorithms, storing classified data in its history, which provides a prediction for the configuration layer.

D. CONFIGURATION LAYER

The configuration layer is the main component of the architecture and has the objective of integrating the two fundamental processes for generating valid plans: planning and classification. For a plan to be generated while respecting the invalid situations from the domain state at the time of planning, it is necessary that the appropriate changes to the domain are very well represented in the planning process. The prediction of invalid states in the generation of the plan is made from the configuration of a set of invalid situations, corresponding to each of the classes that the domain can

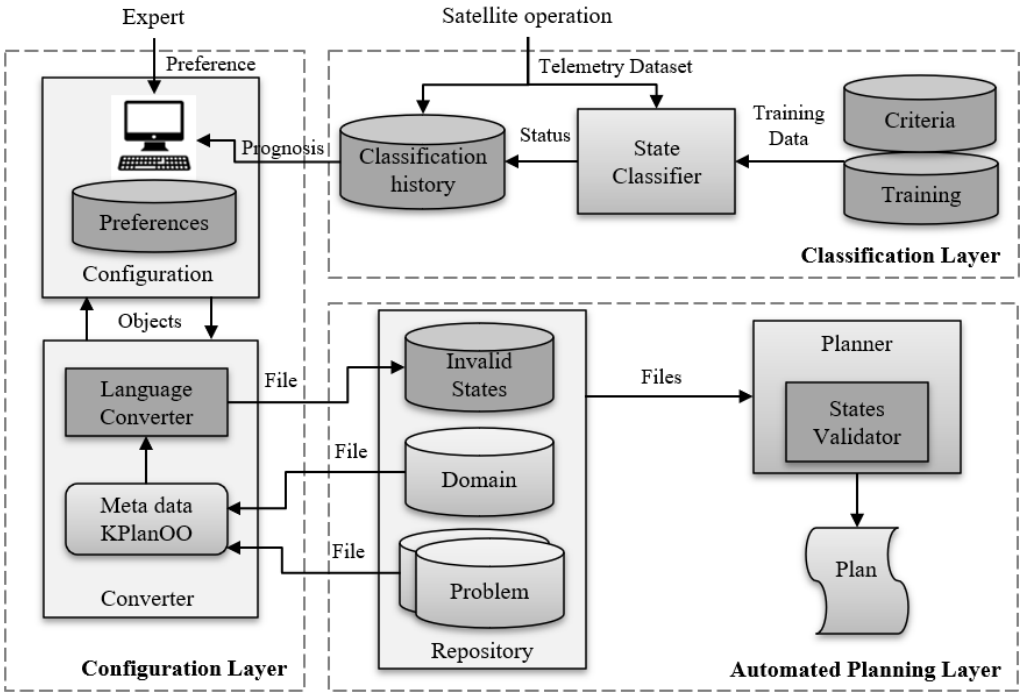

FIGURE 2. Architecture overview.

assume over time. These rules are stored in the configuration database that has been named as *preferences*.

In order to ensure that the knowledge the planning domain representation is correctly mapped between the layers of the approach, we use KPlanOO - An object-oriented Meta-model to describe planning domains [25]. The ontology applied in the KPlanOO domain representation allows the creation of configurations and integrations of the processes in a flexible and standardized way. In addition to being a generic model, it enables the solution to be reused without depending on the planning language that was used in the process.

The following is a brief description of some object types defined in the KPlanOO representation structure, that are used in the architecture of this work, according to [25]:

- **Domain**: represents the planning domain description, composed of a list of actions, that have a list of probable states for problem objects with defined types.
- **Problem**: its purpose is to represent the problem description for a given domain. It is composed of an instance of the *Init class*, an instance of the *Goal class* and also a list of elements that will represent the objects to be used by the planner.
- **Action**: it is a domain's actions abstraction, which contemplates the specification of "Super Action", through a self-relationship. Each action has an effect and a pre-condition, which are objects of the scenario type.
- **Scenario**: entity that represents a scenario composed of the object's states defined in the problem in question. These being of types declared in the domain.
- **Situation**: entity that represents the object's situations that will be manipulated in the planning. These situations are the set formed by the objects state and the conditions linked to this state.
- **Element**: abstracts the objects that will be part of the problem. Element has a type and can have values (Value).
- **Type**: is the abstraction of the types that can be created in a domain. For example: satellite, subsystem, orbit, etc ...

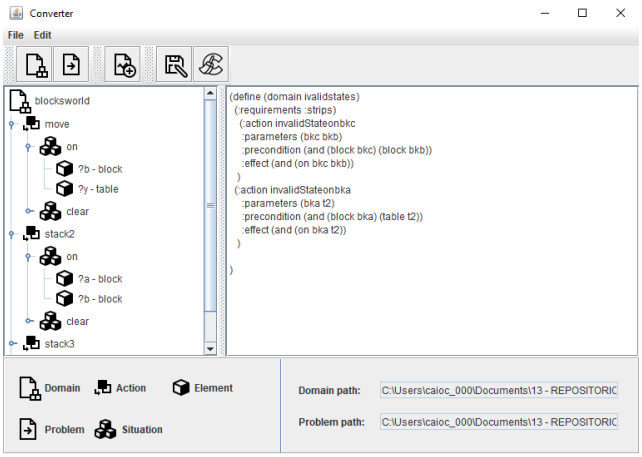

FIGURE 3. Graphical interface of the state converter.

The language converter reads the domain and planning problem files. Using the domain representation KPlanOO structure, it loads the domain objects in a graphical interface, as shown in FIGURE 3. In the tree view that presents the planning items, the situations defined in each domain action

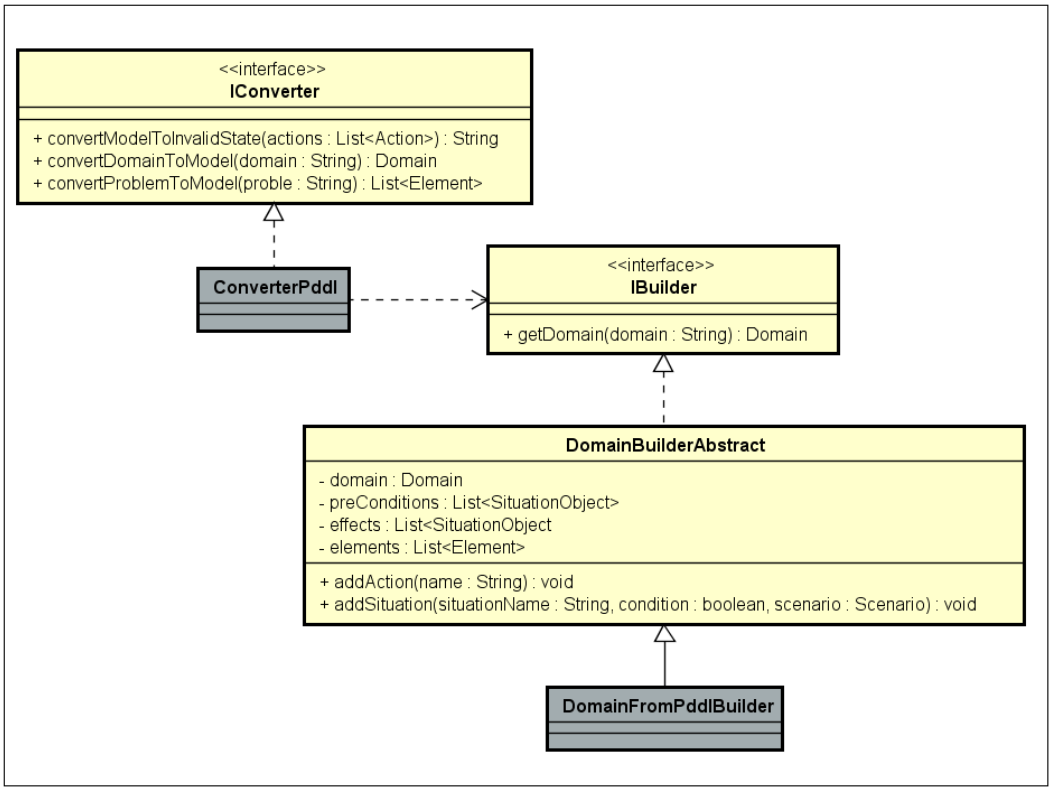

FIGURE 4. Static Structure of classes.

is displayed and listed according to their type: action; situation or element.

For the generation of the invalid states file, it is necessary to first select a situation and then associate it with the elements defined in the problem. In order to assemble the file definition, an expert visualizes the loaded objects and chooses within an action which situation will be invalidated. As described in the section III-A, after choosing a situation for each element, an object defined in the problem must be chosen. Finally, the chosen situation's set will be converted into the language used in the planner code.

FIGURE 4 shows, through the class diagram, the structure of the state converter component. The KPlanOO representation classes are used in the interface. The *IConverter* interface was created to define the following conversion methods:

- **ConvertModelToInvalidState:** is responsible for building the file of invalid states in a planning language. The received parameter depends on a list of objects of type *Action*, which are used to construct the invalid states and return to the definition in an object of type *String*.
- **ConvertDomainToModel:** is responsible for loading the planning domain file from an object of type *String*. The file data is read, instantiated, and returned in an object of type *Domain*.
- **ConvertProblemToModel:** is responsible for loading the planning problem file from an object of type *String*. A list of objects of type *Element* is returned.

The classes highlighted in the diagram represent the specific implementations necessary for the conversion of the planning language. The converter can be implemented using other languages, granted that new classes implement these interfaces and convert the domain to the respective language used. The class that implements the *IConverter* interface is related to a domain object constructor class. Based on the inheritance of the *DomainBuilderAbstract* class, which encapsulates the construction of the domain through a methods group responsible for facilitating the conversion of each object into language code. Using the methods *addAction* and *addSituation* it is possible to create the files with the planning language.

Finally, the expert will associate an invalid state file to each class defined in the classification layer according to the domain's need. Thus, being able to configure groups of invalid situations, as in the example presented in the section III-A, each instant can be considered a domain class, whereas a different situation's group is invalid in each class.

E. AUTOMATED PLANNING LAYER

The automated planning layer represents the planning process with a predicion of invalid states. The planner component was altered with the inclusion of a state validator, which is responsible for permitting invalid states to be validated during the planning time. In addition to receiving the domain and problem files, the planner also receives the file of invalid

```
1 (define (domain invalid-state-settings)
2   (:requirements :strips)
3     (:action invalidState
4      :parameters (?c ?b)
5      :precondition (and (block ?c)
6                         (block ?b))
7      :effect (and (on ?c ?b))
8     )
9 )
```

Listing 1. Setting invalid states.

states which contains the definition of states that cannot make up the plans [15].

In [15] is proposes the creation of an additional function within the planner. Which uses the same planning language that was used by the planner to describe invalid states and represent them in memory. The state is configured as a common action using the same notations as a domain file. Listing 1 presents the example of the definition of invalid states illustrated in *Instance III* of FIGURE 1. The state was configured as a common action, using the same definitions as the PDDL domain file. In this example, in the invalidState's action (Listing 1, line 3), two variables are defined as parameters (Listing 1, line 4), and in the preconditions, the type is associated to each parameter, being two blocks. The positioning and order of the stacked blocks is defined in the effects, such as: *(and (on ?c ?b)* (Listing 1, line 7).

The approach uses the existing implementation in its own planner to read the new file and load it into memory. The function that applies the actions generating new child states, has been adapted to validate the states. In [15] was implemented the States Validator function in the planner. Because during the planning, after the generation of each new state, the function is called to validate if the generated state corresponds to an invalid state.

```
1 function
2 StateValidator(newState, invalidStates)
3  var equalState
4  for each invalidStates.defs do
5    actions <- invalidStates.defs.effect
6    for each actions do
7      equalState <- false
8      for each newState.defs do
9       if newState.defs.operation!= not
10       and newState.defs.action ==
11       actions.action and
12       Equal(newState.defs.params,
13            actions.params) then
14         equalState <- true
15   return equalState;
16 end
```

Listing 2. State validator function.

Listing 2 shows the definition of the State Validator algorithm. The function receives the new generated state and the list of invalid states as parameters. The comparison of the states is based on three conditions (Listing 2, line 9): if the operation is not negative; if the action of both parties is the same; and if the parameters of the parts are the same. The Equal function compares the state's parameters, verifying if the size and values are identical.

The validation is done by comparing all the definitions in the *invalidStates* list (Listing 2, line 4). If any invalid state has all of its parts in the new state, the function returns true (Listing 2, line 14) and the generated state is discarded from the planning. The planner continues to generate and validate other states until it finds the objective state of the problem [15].

Similarly to the approaches described in the related work section, such as, in [19], where a proposition to create a PDDL language extension, in [21], who developed a classification process based on the history of execution; or in [4], which included new definitions in the representation of planning. In this architecture, the creation of the concept of invalid states, which is considered a way of including preferences within the planning process by modifying the automated planning through the validation of plans, allows to generate plans with higher quality in domains with restrictions on the states.

This section described the architecture designed to improve the automated planning process. The application of the data classification process and configuration of invalid states are presented through a case study in the experiment's section.

IV. EXPERIMENTS

This section describes the application of automated planning architecture with the prediction of invalid states in an aerospace area case study. The telemetry's data from a spacecraft, obtained through simulation was used to develop a classification model, configure invalid states and integrate a planning process. The experiment was carried out to achieve the following objectives:

- Create a case study to apply a learning process for invalid states;
- Present the result of the integration of the learning and planning processes through the automatically generated plans;
- Validate the configured invalid states to make sure they were considered when generating the plan, and corresponding to the classification presented by the current state of the domain.

A. MATERIALS AND METHODS

In this experiment we used the Atom SysVAP [26] finite automata validation system. The simulator allows the following: setting the values of environment variables and the machine's state, sending commands to change states and storing data generated by the simulation [26]. We simulated the operation of a satellite to obtain telemetry data and defined a case study to create a learning process for invalid states.

The case study considered the analysis and creation of invalid states related to the power supply subsystem of a satellite. According to the study of [27], 27% of failures in spacecrafts are due to the power supply subsystem. Data mining techniques have been applied to detect these anomalies in satellite telemetry [28]–[30]. The simulation consisted of putting the satellite in a situation of extremely low battery, through the operation with several subsystems connected at the same time, causing greater battery consumption.

The chosen domain was the BR2 nano satellite that is present in the example models of the Atom SysVAP simulator [26]. This domain is composed of five subsystems, three of which are payload (SDATF, SLP, SMDH), the OBC - on-board computer and the transmitter. This domain has the following set of remote controls, which can be executed in the simulator's plan: **get_tm** - generates the telemetry package of the subsystems; **repair** - repairs the alert state of the onboard control subsystem status; **com_t_on** - turns on the transmitter's transmission mode; **com_t_off** - turns off the transmitter's transmission mode; **sub_on** - connects a subsystem and **sub_off** - shuts down a subsystem [26].

TABLE 2. Telemetry data.

Name	Type	Values
Battery	numeric	13% - 19%
Total drop	numeric	0 - 1
Orbit	categorical	Sun, Ecl
OBC	categorical	Alert, On
Communication	categorical	Re/Tr, Receiver
SDATF	categorical	On, Off
SLP	categorical	On, Off
SMDH	categorical	On, Off
Time(s)	numeric	-

The TABLE 2 lists the characteristics of the data set sent by the telemetry package. *Battery* telemetry is a numerical value that indicates the battery level of the satellite (this level varies between 13% and 19%). The *total drop* is an index of the total energy discharge from the battery, calculated using the sum of the energy consumption of each subsystem by the time of operation (the value varies between 0 and 1). *Orbiting* telemetry indicates the reading of the environment value: *Sun* - when the satellite battery is being charged by the sun, *Ecl* - when the satellite is in eclipse (without sunlight) and the battery is not being charged. *OBC* telemetry indicates whether the system is in a normal (On) or an alert state (Alert). Telemetry *communication* indicates the operating mode of the transmitter, *Re / TR* - indicating that the transmit and receive mode is active or *Receiver* - only the active receiver mode. The remaining telemetry data *(SDATF, SLP, SMDH)* corresponds to the payloads that indicate whether they are **on** or **off**.

The data process analysis and the creation of the data classification model was carried out using the Orange Data Mining tool [31]. When performing the satellite operation

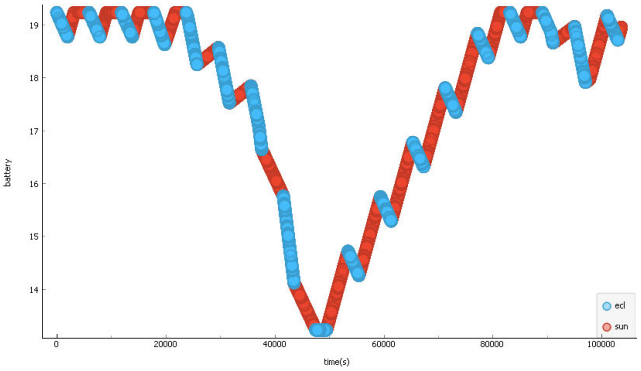

FIGURE 5. Scatter plot of battery level time series.

simulation, the generated telemetry data was loaded and analyzed to form the classification criteria and to train the classification algorithm.

B. CASE STUDY

The telemetry data generated in the simulation with the battery experiment was exported to *.csv* format and loaded into the Orange tool. FIGURE 5 shows the scatter plot of the battery level variation during the simulation. It is observed that at the instant 25,000s the battery level begins to gradually decrease until it reaches 40,000s where even with the presence of the sun (represented by the color red) the battery level continues to discharge. The battery discharge scenario occurs until 50,000s, when the OBC subsystem goes on alert, shutting down the other satellite subsystems, to ensure that the battery is not fully discharged. After this moment, the battery level starts to charge again.

In the next subsections, we describe the execution of the experiment in three stages. In the classification stage, we analyzed the data, developed the classification criteria and the choice of an algorithm to classify the data. During the configuration we chose the groups of invalid states that were configured. During the planning we showed the result of the plans generated using the experiment setup.

1) CLASSIFICATION

One of the ways to automatically identify abnormalities in a data set is through the use of outlier detection algorithms [32]. The Orange Outlier Detection widget [31] was used with the covariance estimator method to visualize the abnormalities present in the simulation data. FIGURE 6 shows the result of the outliers identified in the time series.

Based on the analysis of the abnormal behavior identified in the outliers and the study of classification of battery states presented in [33], the criteria for classification of states presented in TABLE 3 was developed. Considering the battery level, the orbit and the total discharge, three classes were created: Forbidden, Alert, Safe. The *Forbidden* state represents the situation where the battery is no longer charged when illuminated by the sun. The *Alert* state represents the situation

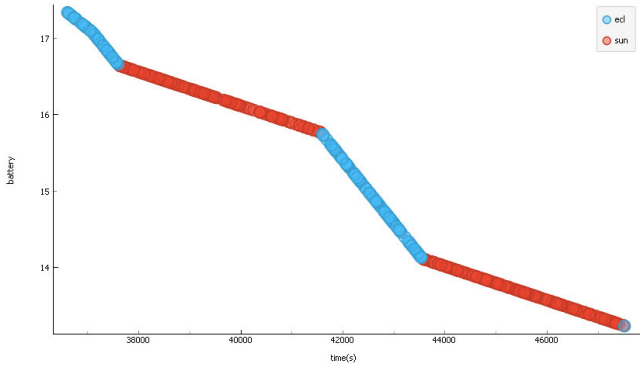

FIGURE 6. Outlier analysis scatter plot.

TABLE 3. Classification criteria definition table.

Battery	Total Drop	Orbit	Operating State
$> 17.5\%$	< 0.000533	ECL	SAFE
$> 18\%$	< 0.000533	SUN	SAFE
$< 17.5\%$	< 0.000533	ECL	ALERT
$< 18\%$	< 0.000533	SUN	ALERT
$< 16.65\%$	> 0.000533	ECL/SUN	FORBIDDEN

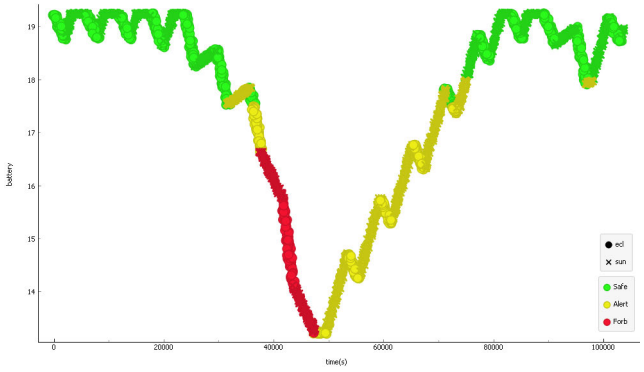

FIGURE 7. Scatter plot data classified.

where the battery level is below 18% in the sun and 17.5% in eclipse. The *Safe* state represents the common situation when the battery level is higher than the values of 18% in sun and 17.5% in eclipse.

The Orange tool provides the Feature Constructor widget used for creating new features in a data set. The classification parameters defined in TABLE 3 were inserted in the widget to create a new feature with the value of the class established from the data characteristic. FIGURE 7 shows the result of the classification through a scatter chart where the classes are identified by colors.

With the classified data, a data prediction model was created using the Predictions widget connected to the Orange Naive Bayes classifier model [31]. We use Naive Bayesian algorithm for fault detection in satellite batteries because it presents good results as demonstrated in [29]. The classifier Naive Bayesian presents a good result as a classifier for fault detection in satellite batteries. The model uses the classified data to make the predictions in the newly received telemetry observations.

2) CONFIGURATION

The files with the domain definitions and planning problem to be used in the validation of the automated planning layer were created based on the commands present in the simulation, as described in the section IV-A. The objective of the planning problem is to send the telemetry data obtained in the subsystems. However, the *OBC* subsystem must obtain the telemetry data from the payload subsystems and generate a packet to be sent by the transmitter.

In the domain actions, the following preconditions were defined to achieve the objective of the problem: i) the transmitter must be configured in Re/Tr mode to send the telemetry data; ii) the subsystem must be turned on for the *OBC* to obtain the telemetry data; iii) and the package must be generated with a pair of telemetry data corresponding to a maximum of two payload subsystems. The initial state of the planning problem was defined with all payload subsystems turned **off** and the transmitter configured in Receiver mode. The main objective of the problem is to send telemetry data from the satellite.

TABLE 4. Invalid state configuration table.

Group	Operating State	Invalid Situations
G1	SAFE	ON - SLP
G2	ALERT	ON - SLP and SDATF
G3	FORBIDDEN	ON - (SLP and SDATF and SMDH) and (SLP or SDATF or SMDH)

In order to configure the invalid states in the automated planning process, for each operating state defined in TABLE 3, a group of invalid states has been assigned as shown in TABLE 4. In the first group it was defined that in the SAFE operating state, the scenario where the SLP subsystem is on represents an invalid situation. In the second group in the ALERT operating state, the scenario where the SLP and SMDH subsystems are on simultaneously is invalid. Lastly, in the third group if both SLP, SMDH or SDATF subsystems are on individually or simultaneously, it is invalid.

The invalid state file was generated using the corresponding configuration layer interface, where the files generated in each of the groups configured in this experiment are listed below. Listing 3 shows the invalid state's file configured for the G1 group. The invalid situation is configured from the definition of the action effect *subsystemon*, where the expression *(poweron ?slp)* (Listing 3, line 6) represents the situation where the SLP subsystem is on.

Listing 4 shows the definition of the G2 group invalid state's file. In order to represent the simultaneous turn on of two subsystems, the effect was configured as shown in the

```
1 (define (domain invalidstatesG1)
2   (:requirements :strips)
3     (:action invalidState
4      :parameters (?slp)
5      :precondition (and
6        (subsystem ?slp))
7      :effect (and (poweron ?slp))
8     )
9 )
```

Listing 3. Setting group G1 invalid states.

```
1 (define (domain invalidstatesG2)
2  (:requirements :strips)
3    (:action invalidState
4     :parameters (?slp ?sdatf)
5     :precondition (and (subsystem ?slp)
6                 (subsystem ?sdatf))
7     :effect (and (poweron ?slp)
8                 (poweron ?sdatf))
9    )
10 )
```

Listing 4. Setting group G2 invalid states.

expression *(poweron ?slp) (poweron ?sdatf)* combining the SLP and SMDH subsystems being turned on simultaneously.

The configuration of the third invalid states group is shown in Listing 5. Each subsystem is defined in a different action, representing that the subsystems cannot be turned on individually or simultaneously.

In Listing 5 the action *simultaneouslyStates* (Listing 5, line 3) defines an invalid state when all three subsystems are linked together. The rest of the definitions *invalidSlp* (Listing 5, line 14), *invalidSdatf* (Listing 5, line 20) and *invalidSmdh* (Listing 5, line 26) define that the subsystems cannot be linked individually.

3) PLANNING

The classification model defined and the planning domain's invalid states are configured according to the architecture's definition in section III-B. The invalid state planner proposed by [15] was used to execute the planning problem, operating the same domain and problem with the different groups of invalid states. Listing 6 shows the plan generated using the G1 group of invalid state's file.

As shown in Listing 6 in step 1 - the transmitter was changed to Re/Tr mode; in step 2 - the SDATF subsystem was turned on; In step 3 - the OBC obtained the telemetry from the SDATF; In step 4 - the SMDH subsystem was turned on; In step 5 - the OBC obtained SMDH telemetry; In steps 6 and 7 - the subsystems were turned off; In step 8 - the telemetry package was generated; and in step 9 - the transmitter sends the telemetry package generated by the OBC. Note that the SLP subsystem defined in group G1, was

```
1(define (domain invalidstatesG3)
2  (:requirements :strips)
3    (:action simultaneouslyStates
4     :parameters (?slp ?sdatf ?smdh)
5     :precondition (and
6        (subsystem ?slp)
7        (subsystem ?sdatf)
8        (subsystem ?smdh))
9     :effect (and
10       (poweron ?slp)
11       (poweron ?sdatf)
12       (poweron ?smdh))
13   )
14   (:action invalidSlp
15    :parameters (?slp)
16    :precondition (and
17                (subsystem ?slp))
18    :effect (and (poweron ?slp))
19   )
20   (:action invalidSdatf
21    :parameters (?sdatf)
22    :precondition (and
23                (subsystem ?sdatf))
24    :effect (and (poweron ?sdatf))
25   )
26   (:action invalidSmdh
27    :parameters (?smdh)
28    :precondition (and
29                (subsystem ?smdh))
30    :effect (and (poweron ?smdh))
31   )
32)
```

Listing 5. Setting group G3 invalid states.

```
1. changetransmitter transmissor rectr
2. subsystemon sdatf
3. gettm sat obc sdatf
4. subsystemon smdh
5. gettm sat obc smdh
6. subsystemoff sdatf
7. subsystemoff smdh
8. generatepkg obc sdatf smdh
9. sendtelemetry transmissor rectr obc

- Solution found in 9 steps!
Depth: 9, 134883 child states.
Runtime: 476579.710ms
```

Listing 6. Plan executed with G1 group.

not linked by the action *subsystemon* during the steps of the generated plan.

Listing 7 shows the plan generated using the G2 group invalid state's file. The restriction imposed on the G2 group

```
1. changetransmitter transmissor rectr
2. subsystemon sdatf
3. gettm sat obc sdatf
4. subsystemoff sdatf
5. subsystemon slp
6. gettm sat obc slp
7. subsystemoff slp
8. generatepkg obc sdatf slp
9. sendtelemetry transmissor rectr obc

- Solution found in 9 steps!
Depth: 9, 2388 child states.
Runtime: 18887.102ms
```

Listing 7. **Plan executed with G2 group.**

defined that the SLP and SDATF subsystems were not turned on simultaneously. Therefore after obtaining the telemetry of the SDATF subsystem in step 3, the subsystem was turned off in step 4 before the SLP subsystem was turned on, thus respecting the restriction imposed on the invalid states. The restrictions were followed by the planner in both groups, where we can see that the subsystems were linked respecting the configured rules.

In terms of the performance of plan generation, even though the depth of the search tree is the same, we can observe that with the increase of invalid situations in the G2 group, the number of child states generated in the search space is 50 times less, as well as the execution time which is 25 times shorter when compared to the G1 group plan. Due to the fact that the planner ignores invalid states, it generated fewer branches.

In the planning of the G3 group, a plan was not generated, since the planner did not find a solution to send the telemetry from the satellite because at least two payload subsystems must be connected, as described in the section IV-B2. It is concluded that the three groups of invalid states were met in the executed plans and therefore the objectives of the experiment that were defined at the beginning of this section were achieved.

Finally, the case study allows the learning process to be applied based on the classification of operating states. The processes integration result was proven through the plans generated in an automated way and through the different verification steps generated in each of the plans, respecting invalid states configured in each group. In the example, the satellite situation battery was used to define when the subsystems can be turned on, thus preventing the satellite from going on alert and its battery being completely discharged.

V. CONCLUSION

Plan validation is not a trivial task in the automated planning of complex domains, since the generated plans can pose risks to the domain security without a validation step. Based on the creation of a new definition that models the invalid states during the planning stage and allows the plans to be generated following restrictions according to the current state of the domain, the approach presented in this study guarantees the planning of valid plans. With the state's validation being done during planning time, the need to execute a plan validation stage after the planning process was eliminated, contributing to the reduction of the total planning time and avoiding replanning.

To the best of our knowledge, there is no other approach that uses a planner modified to validate invalid states, including planning restrictions obtained from domain learning, it was possible through the use of an object-oriented planning meta-model, to join the planning and learning processes in a standard way. Our proposal enables the solution to be reused and allows the configuration to be easily implemented with other planning languages.

In summary, the architecture implementation allows to generate the plans in an independent mode. Because the changes in the domain's behavior could be considered in the automated planning, without having the domain definitions being modified or a planning problem.

A. CONTRIBUTIONS

The main contribution of this study is the proposal to validate plans during the planning time, presented as a new way to validate invalid states within the planner itself. As with other planning approaches, data mining is used to improve the process as well as learning new rules. In this study, we also contribute with the use of learning to configure groups of invalid states and integrate a viable solution to the automated planning process.

The architecture presented also helps to update the planning of changes that can occur in the domain, because identifying the changes that occur in some domains can take a long time. However, problems such as invalid plans being executed can be eliminated with a process that identifies such changes immediately.

B. LIMITATIONS

- Even though the language used in the planner is reused to represent invalid states, modifying the planner's implementation to include the state validator method might require more complex coding, depending on the complexity or technique used;
- As the restrictions increase, the number of plans not found can also increase. As seen in the result of the last experiment of this study, when the invalid state prevented the planner from finding the solution to the problem. If this type of situation occurs, mitigative solutions can be used, such as operating the domain with its reduced functions or creating contingency plans that take actions to achieve solutions in another way;
- The validation of the invalid states during the planning stage is performed in the known states of the planning

domain used. In other words, the domain must contemplate the invalid state in its instance so that it is possible to be validated. However, any invalid state that may appear in the domain must be contained in the domain definition;

C. FUTURE WORK

As the restrictions increase, the number of unfound plans may also increase. As shown in the last experiment result of this work, when the invalid state prevents the planner from finding the solution to the problem. When this type of situation occurs, mitigative solutions or contingency plans must be created.

As future work, we intend to use techniques for meeting partial plan objectives combined with the validation of invalid states. Using the invalid states feature to improve performance and planning time, as well as learning, can contribute to improve performance because the planner eliminates search paths as it finds invalid states.

REFERENCES

[1] P. Souza, "A mathematical model to predict operating states of satellites," in *Proc. SpaceOps Conf.*, Jun. 2012, Art. no. 1272995.
[2] J. Tominaga, M. Ferreira, and J. Silva, "A rule-based satellite simulator for use in flight operations planning," *J. Comput. Interdiscipl. Sci.*, vol. 2, no. 2, pp. 111–121, 2011.
[3] L. Cardoso, M. Ferreira, and V. Orlando, "An intelligent system for generation of automatic flight operation plans for the satellite control activities at INPE," in *Proc. SpaceOps Conf.*, Jun. 2006, p. 5575.
[4] J. A. Baier, C. Fritz, M. Bienvenu, and S. A. McIlraith, "Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners," in *Proc. AAAI*, 2008, pp. 1509–1512.
[5] B. Nebel, "On the compilability and expressive power of propositional planning formalisms," *J. Artif. Intell. Res.*, vol. 12, pp. 271–315, May 2000.
[6] M. Van Den Briel, R. Sanchez, M. B. Do, and S. Kambhampati, "Effective approaches for partial satisfaction (over-subscription) planning," in *Proc. 19th Nat. Conf. Artif. Intell.*, 2004, pp. 562–569.
[7] A. Gerevini and D. Long, "Preferences and soft constraints in PDDL3," in *Proc. ICAPS Workshop Planning Preferences Soft Constraints*, 2006, pp. 46–53.
[8] J. Rintanen, "Complexity of concurrent temporal planning," in *Proc. ICAPS*, vol. 7, 2007, pp. 280–287.
[9] A. Coles and A. Smith. (2004). *Marvin: Macro-Actions From Reduced Versions of the Instance*. [Online]. Available: https://nms.kcl.ac.uk/andrew.coles/publications/publication84.pdf
[10] C. Domshlak, E. Karpas, and S. Markovitch, "To max or not to max: Online learning for speeding up optimal planning," in *Proc. 24th AAAI Conf. Artif. Intell.*, 2010, pp. 1–6.
[11] M. Alhossaini and J. C. Beck, "Instance-specific remodelling of planning domains by adding macros and removing operators," in *Proc. 10th Symp. Abstraction, Reformulation, Approximation*, 2013, pp. 1–9.
[12] C. Areces, F. Bustos, M. A. Domínguez, and J. Hoffmann, "Optimizing planning domains by automatic action schema splitting," in *Proc. ICAPS*, 2014, pp. 1–9.
[13] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Amsterdam, The Netherlands: Elsevier, 2004.
[14] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artif. Intell.*, vol. 2, nos. 3–4, pp. 189–208, Dec. 1971.
[15] C. Rodrigues da Cruz, M. Ferreira, and R. Silva, "State validation in automated planning," in *Proc. ICEIS*, 2020, pp. 396–406.
[16] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL—The planning domain definition language–version 1.2," Yale Center Comput. Vis. Control, Lenoir City, TN, USA, Tech. Rep. CVC TR-98-003/DCS TR-1165, 1998. [Online]. Available: https://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf
[17] M. Vallati, L. Chrpa, M. Grześ, T. L. McCluskey, M. Roberts, and S. Sanner, "The 2014 international planning competition: Progress and trends," *AI Mag.*, vol. 36, no. 3, pp. 90–98, Sep. 2015.
[18] C. Domshlak, E. Hüllermeier, S. Kaci, and H. Prade, "Preferences in AI: An overview," *Artif. Intell.*, vol. 175, nos. 7–8, pp. 1037–1052, May 2011.
[19] A. Jorge and S. A. McIlraith, "Planning with preferences," *AI Mag.*, vol. 29, no. 4, p. 25, 2008.
[20] A. Gerevini and D. Long, "Plan constraints and preferences in PDDL3," Dept. Electron. Automat., Univ. Brescia, Brescia, Italy, Tech. Rep. 2005-08-07, 2005.
[21] C. Boutilier, T. Dean, and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage," *J. Artif. Intell. Res.*, vol. 11, pp. 1–94, Jul. 1999.
[22] P. Haslum and P. Jonsson, "Planning with reduced operator sets," in *Proc. AIPS*, 2000, pp. 150–158.
[23] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer, "Macro-FF: Improving AI planning with automatically learned macro-operators," *J. Artif. Intell. Res.*, vol. 24, pp. 581–621, Oct. 2005.
[24] J. Frank, "Using data mining to enhance automated planning and scheduling," in *Proc. IEEE Symp. Comput. Intell. Data Mining*, Mar./Apr. 2007, pp. 251–260.
[25] R. Silva, M. G. Ferreira, and N. Vijaykumar, "An object-oriented metamodel as ontology for describing domains and problems for planning space applications planning," in *Proc. SpaceOps Conf.*, Apr. 2010, p. 2172.
[26] A. A. S. Ivo, "A tool for evaluating satellite flight plans using state models," M.S. thesis, Eng. Manage. Space Syst., Nat. Inst. Space Res., São José dos Campos, Brazil, 2013.
[27] M. Tafazoli, "A study of on-orbit spacecraft failures," *Acta Astronautica*, vol. 64, nos. 2–3, pp. 195–205, Jan. 2009.
[28] D. R. Azevedo, A. M. Ambrósio, and M. Vieira, "Applying data mining for detecting anomalies in satellites," in *Proc. 9th Eur. Dependable Comput. Conf.*, May 2012, pp. 212–217.
[29] M. A. Galal, W. M. Hussein, E. E.-D. A. Kawy, and M. M. Sayed, "Satellite battery fault detection using Naïve Bayesian classifier," in *Proc. IEEE Aerosp. Conf.*, Mar. 2019, pp. 1–11.
[30] S. Abdelghafar, A. Darwish, A. E. Hassanien, M. Yahia, and A. Zaghrout, "Anomaly detection of satellite telemetry based on optimized extreme learning machine," *J. Space Saf. Eng.*, vol. 6, no. 4, pp. 291–298, Dec. 2019.
[31] J. Demšar, T. Curk, A. Erjavec, C. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Župan, "Orange: Data mining toolbox in Python," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 2349–2353, 2013.
[32] Orange Data Mining. (2020). *Orange Visual Programming Documentation*. [Online]. Available: https://orange3.readthedocs.io/en/latest/
[33] P. B. de Souza, M. G. V. Ferreira, and J. D. S. da Silva, "A tool for prediction of satellite future states," *J. Aerosp. Comput., Inf., Commun.*, vol. 7, no. 12, pp. 406–414, Dec. 2010.

CAIO GUSTAVO RODRIGUES DA CRUZ graduated in system analysis and development from the FATEC Mogi das Cruzes, São Paulo Technological College, Mogi das Cruzes, São Paulo, Brazil, in 2016. He received the master's degree in space systems engineering and management from the National Institute for Space Research—INPE, São José dos Campos, São Paulo, in 2021.

He currently works as a Software Developer with Muralis Technology. He has an article published in Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS 2020).

RODRIGO ROCHA SILVA graduated in computer science from the University of Mogi das Cruzes, Brazil. He received the master's degree in applied computing from the National Institute for Space Research—INPE, and the Ph.D. degree in computing from the Technological Institute of Aeronautics—ITA.

He is currently a Full Professor with the Paula Souza Center, São Paulo. He is also a member of the Center for Informatics and Systems, Department of Informatics Engineering, University of Coimbra. He is the author of more than 30 articles in international journals and conferences, he develops, and guides works in big data and data mining in academic and business contexts.

MAURICIO GONÇALVES VIEIRA FERREIRA received the master's and Ph.D. degrees in applied computing from the National Institute for Space Research (INPE), in 1996 and 2001, respectively.

He is currently a Full Professor with the Postgraduate Course in Space Engineering and Technology (ETE): concentration area Engineering and Management of Space Systems. He is also a Researcher and a Coordinator of the Satellite Control Center, INPE. He produced more than 170 articles, supervised 16 Ph.D., 24 master's, and two postdoctorates. He supervises four Ph.D. students and seven master's students. He works in the area of research and development of software for satellite control. He is a Scientific Advisor to FAPESP in the area of software engineering. He was already a Productivity Scholarship Developer at Technology and Innovative Extension Level 2. He is a member of the International Committee for Standardization of Software in the Space Area (CCSDS). He is a member of the Organizing Committee of the International Space Congress—SPACEOPS.

JORGE BERNARDINO (Member, IEEE) received the Ph.D. degree from the University of Coimbra, in 2002.

From 2005 to 2010, he was the President of ISEC, Portugal. From 2017 to 2019, he was also the President of ISEC Scientific Council. He was a Visiting Professor with CMU, in 2014. He is currently a Coordinator Professor with the Polytechnic of Coimbra—ISEC. He is also the Director of the Applied Research Institute (i2A), Polytechnic of Coimbra, Portugal. He participated in several national and international projects. He has authored more than 200 publications in refereed conference papers and journal articles. His main research interests include big data, NoSQL, data warehousing, dependability, and software engineering. He is a member of ACM.