

Plan Validation and Mixed-Initiative Planning in Space Operations

Richard Howey Derek Long

Maria Fox

`richard.howey, derek.long, maria.fox @cis.strath.ac.uk`

Department of Computer and Information Systems

University of Strathclyde, Glasgow, UK

Abstract. This paper describes aspects of our plan validation tool, VAL. The tool was initially developed to support the 3rd International Planning Competition (IPC), but has subsequently been extended in order to exploit its capabilities in plan validation and development. In particular, the tool has been extended to include advanced features of PDDL2.1 that were untested in the 3rd IPC but which have proved important in an application to mixed-initiative planning in a space operations project. The tool has also been extended to keep abreast of developments in PDDL, providing critical support to participants and organisers of the 4th IPC.

1 Introduction

VAL is a plan validation tool for PDDL. It played an important role in the 3rd IPC [7], allowing reliable validation of the several thousand plans produced by the competitors, as well as providing competitors with support for their development and debugging cycles. We have found the capabilities of VAL to be critical in understanding the structures of large plans, providing visualisation and reporting facilities [5]. VAL has continued to play an important part in the 4th IPC [3], which has seen several minor extensions to PDDL and its semantics. In this paper we review this role of VAL, as an encapsulation of the semantics of PDDL in a form that allows developers to compare their own understanding, captured in their planners, with the semantics described in [4] and modelled in VAL.

VAL has also been extended to support features that were included in the definition of PDDL2.1, although not used in the competitions, including the expression of continuous change. We describe a motivating example for the role of this extension, based on a project in space operations planning. VAL has proved to be a valuable resource for this project, supporting the development of a domain description, the validation of plans constructed by humans and, using a new extension to the system, providing advice on the correction of flawed plans. We describe this recent feature and discuss further ways in which the tool can be exploited in plan development, both in mixed-initiative and in fully automated plan construction.

2 A Brief Review of PDDL and its Semantics

When Drew McDermott and the first planning competition committee proposed PDDL as a community standard[10] planning domain description language, they initiated an important

process of development in which the planning research community has seen incremental extensions and modifications as the language has adapted to various goals. In the first instance, the core of PDDL was a STRIPS language, offering an ADL extension. This language has a semantics that is widely accepted, based on a simple state-transition model, with few areas of potential ambiguity. Perhaps the most significant issue for which alternative resolutions exist is concurrency: classical plans are often considered to be *sequences* of steps, representing state transitions, but partial-order planning [9] and Graphplan [2] both offer alternative models in which some form of parallelism is considered.

McDermott developed a simple plan validation tool for PDDL that accepted only sequential plans. However, the question of interpretation for more complex extensions of PDDL is more difficult. There is no prior widely accepted model, so choices must be made that are not necessarily universally accepted. Since the language plays a central role in communication of domains between researchers, it is important that there be a standard by which a common understanding may be developed for the semantics of domains and plans for those domains. A formal semantics is the first component of this. However, a formal semantics is not sufficient by itself, because a formal semantics is notoriously difficult to read. In practice, many formal semantics are read in detail by few and understood in all details by even fewer. To make the semantics accessible, their implementation as a validation tool is an important step. In this form, it is possible to confirm understanding of the semantics by testing various plans and domains with the tool, confirming the behaviour is as expected. VAL supplies a variety of forms of feedback, making it possible to explore quite precisely what might be wrong with a flawed plan and aiding in the interpretation of the more subtle details of the semantics.

3 Semantics of Continuous Effects

Continuous Effects

A continuous effect can only affect metric quantities: it is not possible to change a propositional fluent continuously. A metric variable that can be changed by a continuous effect is called a *Primitive Numerical Expression (PNE)*. A durative action that has a continuous effect on a PNE changes it so that the values taken are described by a continuous function of time.

Semantics

A formal semantics has been developed, based on the semantics of discrete durative actions [4]. The semantics of discrete durative actions can be formulated in terms of discrete state changes at the instants of change, in a familiar state-transition semantic framework, but with the addition of an embedding of the activity into a real time line. This is a straightforward extension, except for two complications: (i) to explain under what circumstances the end points of actions (when instantaneous change occurs) may coincide (see [4] on *mutex actions* for details); (ii) to account for the way in which the interaction between action execution and invariants is handled. This is achieved by observing that it is only necessary to confirm each invariant between the finitely many discrete points of change.

However, these semantics are insufficient for plans with durative actions with continuous effects. Fortunately, the original semantics can be extended to give a suitable account of plans with continuous change. The implementation in VAL is directly related to this extension.

3.1 Implementation of Semantics in VAL: Semantics of a Semi-Simple Plan

The semantics of classical actions in terms of state transitions is familiar. Following [4], we call these actions *simple actions* and plans constructed only from simple actions we call *simple plans*. For lack of space we briefly summarise definitions given in full in [5]: a *simple plan* is a collection of pairs (t, a) , where t is a time and a is an action name. Each distinct time in a simple plan defines a *happening* at which point a set of simple actions in the simple plan occurs. A *plan* extends a simple plan to include durative action instances, each with an associated duration.

Durative actions with discrete effects can be given a semantics in terms of the semantics of simple plans. This is shown by mapping plans containing durative actions to simple plans (details can be found in [4]), in which the end points of the durative action are treated as simple actions in a simple plan. Invariants of durative actions can also be treated as simple actions with preconditions but no effects. These appear at points in a simple plan corresponding to the critical times at which the invariants must be checked during the interval of the corresponding durative action. It is not possible to give the semantics of durative actions with continuous effects in terms of a simple plan, because the values of PNEs may be required at arbitrary points over an interval on which they are continuously changing.

We therefore define an extension of a simple plan, a *semi-simple plan*.

Definition 3.1. Act An act is a happening labelled with an act type. The act type can be one of three values: invariant, continuous update or regular.

Definition 3.2. Semi-Simple Plan A semi-simple plan, *SSP* consists of a finite collection of timed simple actions which are 3-tuples (t, A, a) , where t is a rational-valued time, A is the act type and a is an action name.

Definition 3.3. Act Sequence for a Semi-Simple Plan The act sequence for a semi-simple plan *SSP*, $\{(t, A)_i\}_{i=0\dots k}$, is the lexicographically ordered (by time, then act type), sequence of time and act type pairs appearing in the timed simple actions in *SSP*. The act types are ordered invariant, continuous update, regular. All t_i must be greater than 0.

The definition of ground durative actions [4] is extended to include continuous effects by introducing a simple action to abstract out the continuous effects.

A plan containing actions with continuous effects can be mapped to a semi-simple plan in a straightforward way: end points of a durative action are mapped to regular acts, invariant checks are mapped to invariant acts and continuous effects are mapped to continuous update acts. This is illustrated in figure 1. In (1) we show how the interval of a durative action effecting continuous change can be handled by updating continuously changing PNEs discretely at the end of the interval. Each invariant check is responsible for confirming correctness over the preceding interval of continuous change. In (2) we show that if a simple action occurs between the

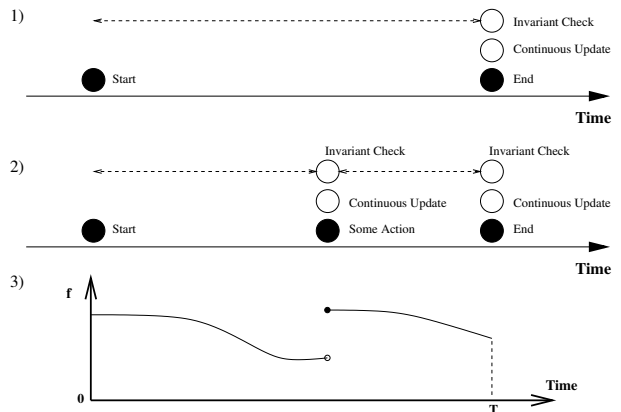


Figure 1: Durative action with continuous effects.

start and end points of a durative action then a continuous update and invariant act for that durative action is placed before this simple action. This mapping of P is called the *induced semi-simple plan*, written $\text{semi-simplify}(P)$. Part (3) shows how discrete effects can arise, due to parallel activity, breaking the continuous change into piece-wise continuous components.

To execute a regular act, we apply the state transition corresponding to all of its simple actions using the familiar add and delete effect semantics, together with numeric updates in the obvious way. To execute a continuous update act is also straightforward: the continuously changing PNEs are updated according to the functions of time describing their behaviour on the interval from the preceding regular act. Invariant acts are not straightforward and checking invariants is considered in section 4.2.

Definition 3.4. Executability of a Semi-Simple Plan A semi-simple plan, SSP , is executable if it defines an act sequence, $\{(t, A)_i\}_{i=0\dots k}$ with states, $\{S_i\}_{i=0\dots k+1}$. S_0 is the initial state and for each $i = 0 \dots k$, S_{i+1} is the result of executing the act, Act_i , for $(t, A)_i$:

- If A_i is regular then the preconditions of Act_i must hold in S_i and S_{i+1} is the result of removing delete effects, adding add effects and applying numeric effects.
- If A_i is invariant then the conditions of Act_i must hold over the interval between the preceding regular act and t_i (taking into account any continuous change).
- If A_i is continuous update then the effects of Act_i are applied at time t_i for the continuous effects over the interval between the preceding regular act and t_i .

The state S_{k+1} is called the final state produced by SSP and the state sequence $\{S_i\}_{i=0\dots k+1}$ is called the trace of SSP . Note that an executable plan produces a unique trace.

Definition 3.5. Validity of a Semi-Simple Plan and of a Plan A semi-simple plan is valid if it is executable and produces a final state S , such that the goal specification is satisfied in S . A plan, P is valid if $\text{semi-simplify}(P)$ is valid.

4 VAL and Plans Including Continuous Change

4.1 Interacting Continuous Effects

There may be a number of continuous effects active at one time each of which additively modifies the derivative of a PNE. If a PNE has its derivative modified more than once then the derivative is given by the sum of the contributions. The rate of change of a PNE may also depend on the value of other PNEs which may themselves be continuously changing. The values of all the changing PNEs are thus given by a system of differential equations. For example consider the following continuous effects describing the motion of a car:

```
increase (distance ?c) (* #t (speed ?c))
increase (speed ?c) (* #t (acceln ?c))
```

4.2 Invariants

Continuous effects have their most significant impact on validation of plans when they interact with invariants [6]. An invariant comparison containing PNEs that are continuously changing can always be expressed as a function of time, t , that must be greater than zero (or greater than or equal to zero). For example, $t^4 + 3t + 1 > 0$ may be required to be invariant for $t \in (0, 3)$. If the invariant expression is linear we can simply evaluate the expression at the end points of the interval to confirm the condition holds. However, when checking an invariant

condition with a non-linear expression in time checking end point is insufficient. *The key to the problem of checking invariants that are comparisons with non-linear expressions in time is one of finding the roots of a non-linear function.* This problem is, in general, non-trivial, even in the case of polynomials. There are many algorithms to find the roots of equations but we need to be sure of finding all the roots in a given interval in every possible case. It is therefore necessary to impose certain restrictions on the invariants that may be expressed to guarantee that they may be validated on a given interval.

We initially only consider invariant comparisons which depend on continuously changing PNEs given by polynomials, and later consider other continuous functions by approximating with polynomials. For one-clause invariant comparisons which are given by an inequality that is strict we are in fact only interested in the existence of real roots on a given open interval and not their exact values.

Invariants with disjunctions provide an extra complication when the disjuncts depend on continuously changing PNEs. For example consider the following invariant with disjunction $(t^2 - 9t + 14 \geq 0) \vee ((t - 1 > 0) \wedge (-t + 8 \geq 0))$ for t in $(0, 10)$. We must find the values of t in $(0, 10)$ each disjunct is satisfied on then take the union and see if the result covers $(0, 10)$. In general it is necessary to find all the roots of all the continuous functions involved, these points can be used as the end points of the sub-intervals that each disjunct is satisfied on.

4.3 Implementation

The validation of plans with continuous effects provides two further main challenges, both in theory and in the subsequent implementation. These challenges are: solving a system of differential equations (as seen in section 4.1) and then finding the roots of continuous functions (polynomials in particular) on given intervals. The implemented validation process is presented in [6], along with discussion, following the semantics of a semi-simple plan. The validation process alternates between discrete activity and continuous activity on the time line of the plan.

5 Space Operations Planning for the Beagle 2 Lander

On December 25th, 2003, a small lander, travelling with the Mars Express orbiter, was expected to land on Mars surface. Unfortunately, as with a high proportion of Mars landers, the Beagle 2 lander was unsuccessful. Various explanations of its failure have been proposed, including the possibility that the density of the Martian atmosphere is not as high as had been thought and, as a result, the parachute-brake failed to slow the lander sufficiently before impact. Despite this major setback, the design of the lander is considered so innovative and efficient (both in terms of cost and in terms of science-to-mass) that a future repeat attempt is being seriously considered. Funded by the European Space Agency, the authors have explored the possible roles of planning technology in space operations [11]. The project includes examining interactions between human operations planners and automated technology, initially in the context of operations plans for Beagle 2. Considerable expertise was built up around the Beagle 2 systems, including a partial domain model for human planning operations, and this has formed the core of a project to exploit planning technology to support mixed-initiative and partially automated planning for the lander operations.

Beagle 2 is a static lander, equipped with a jointed arm carrying an array of scientific instruments in a “paw” at its end. Included in the paw is a mole capable of drilling into soil around the lander to a distance of more than 2 meters, to retrieve soil samples for gas analysis on board the lander. Beagle 2 is essentially a geological survey system, capable of performing an array of geological and environmental measurements in its immediate surroundings.

All lander operations are constrained by power availability, provided by solar energy with a battery for storage, and by temperatures. In addition, the lander is equipped with storage buffers for instrument data. These are too small to store all the data generated by the succession of experiments performed by the lander. Data must therefore be uplinked from the lander not only to return it to the scientists, but also to free up space for storage of succeeding measurements. Uplink windows are constrained, both in duration and in bandwidth, so that it is not possible to empty all the buffers in a window. Thus, there is a further problem of scheduling the uplinking of data into the communication windows in order to allow buffers to be emptied in time for experiments to be performed. These constraints make the management of data and communication a critical element of the planning problem, in common with other deep space missions.

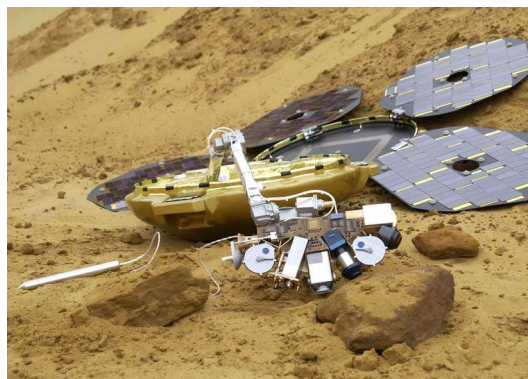


Figure 2: Beagle 2

Therefore, planning lander operations involves managing constraints on continuously changing quantities (the generated power levels and temperatures), scheduling the use of resources, planning the movement sequences of the arm and use of the instruments. Human operations planners were to have carried out the planning for the lander in a complex process involving scientists, providing mission goals and the operations to achieve them, and lander operations personnel, concerned with lander security and, therefore, the power resources and internal lander monitoring systems. When we became involved in the project we discovered that the existing partial domain description was in a form that closely resembled PDDL2.1 durative action descriptions. It was possible to translate the description into PDDL automatically using a simple automatic translator. The domain encoding began with over 50 actions and this has increased to nearly 70 actions following further domain analysis.

Power is the most important continuous factor in the operations of the lander. The lander operates close to margins and the model of the solar generation and the battery charging profiles are vital in determining when operations can be planned. The management of battery and solar power is sufficiently close to the margins of operational envelopes that it plans must interact with the continuous changes involved in the physical system rather than with abstractions into coarse-grained simple step-function changes. Of course, with sufficiently small time-steps a step-function model can approximate the continuous change adequately, but it is infeasible to attempt to model this level of granularity explicitly in the planning domain description. Therefore, this problem demands that the planner has access to a sufficiently detailed model of the continuous changes that affect the power systems.

6 Mixed-Initiative Planning and Plan Repair using VAL

When VAL is used in its simplest form, without any parameters, in the case of plan failure it reports only that the plan has failed. An option is available for verbose output in which the system generates a report explaining which action in a plan has failed. However, this is still of limited use since no indication is given of how an action precondition might have failed to be satisfied. The action precondition might be very complex, but only failed due to one literal with the incorrect truth value. For example, a large factory machine may have an action for starting processing with a complicated precondition, but an instance of the action in a plan might fail simply because the machine is not switched on prior to planned execution of the start action. Feedback from the plan validation reporting that the machine needs to be switched on would be invaluable advice on how to fix the plan. In complex plans identifying even simple failures such as this can be difficult due to the obscuring effects of the actions surrounding the failure.

With the intention of supplying more informed feedback we have developed in VAL a detailed advice sub-system indicating how to satisfy unsatisfied preconditions in an invalid plan. The advice can be used in a *mixed-initiative planning* cycle in which the human planner firstly produces a plan either by hand or with the aid of software before VAL simulates execution of the plan giving detailed advice on how to repair the plan for each unsatisfied action precondition (or invariant condition or goal). The advice can then be used by the human planner to produce a new plan to correcting the errors, or at least some of them. The new plan can then be executed using VAL which produces new plan repair advice, and so on.

In general, the advice offered by VAL indicates why a given plan failed and what conditions must be achieved in order to repair it. It does not indicate which actions might be applied to achieve those conditions or explore the interactions they might introduce into the plan if they are added to it. Therefore, the advice from VAL must be seen as the first stage in the repair or reconstruction of a flawed plan: other components are necessary to decide how best to act on the advice if this decision is to be made automatically.

6.1 Structure of Plan Repair Advice

The advice given for a failed precondition is derived from a PDDL precondition expression and stored in a structure called an *advice proposition*.

Definition 6.1. Advice Proposition *For a given PDDL precondition of an action in a plan the advice proposition provides instructions on how the state, S , must be altered at this point in the plan in order to satisfy the precondition. An advice proposition (AP) is one of the following:*

- *Instructions to set A to true, for some literal A .*
- *Instructions to set A to false, for some literal A .*
- *Instructions to satisfy a comparison consisting of numerical expressions where each PNE has its current value reported.*
- *A list of APs where all must be followed (conjunction AP).*
- *A list of APs where at least one must be followed (disjunction AP).*
- *No advice (the empty advice case).*

VAL produces advice for each unsatisfied precondition by mapping the precondition and state to an advice proposition.

Definition 6.2. Let ϕ be the mapping from a PDDL precondition, P , and a state, S , to an advice proposition defined as follows if P is a literal, comparison or connective respectively.

$$\begin{aligned}\phi(P, S) &:= \text{if } S \models P \text{ then no advice else set } P \text{ to true} \\ \phi(P, S) &:= \text{if } P \text{ is an unsatisfied comparison then satisfy } P \\ \phi(\wedge_i X_i, S) &:= \wedge_i \phi(X_i, S), \text{ for each unsatisfied } X_i \text{ in } S \\ \phi(\vee_i X_i, S) &:= \vee_i \phi(X_i, S), \text{ for each unsatisfied } X_i \text{ in } S \\ \phi(X \rightarrow Y, S) &:= \phi(\neg X \vee Y, S)\end{aligned}$$

If P is a negation, $P = \neg Q$, then $\phi(P) = \psi(Q)$ where ψ is defined as below if Q is a predicate, comparison or connective respectively.

$$\begin{aligned}\psi(Q, S) &:= \text{if } Q \not\models S \text{ then no advice else set } Q \text{ to false} \\ \psi(Q, S) &:= \text{if } Q \text{ is an unsatisfied comparison then satisfy } Q \\ \psi(\wedge_i X_i, S) &:= \vee_i \phi(\neg X_i, S), \text{ for each satisfied } X_i \text{ in } S \\ \psi(\vee_i X_i, S) &:= \wedge_i \phi(\neg X_i, S), \text{ for each satisfied } X_i \text{ in } S \\ \psi(X \rightarrow Y, S) &:= \phi(X \wedge \neg Y, S) \\ \psi(\neg Q', S) &:= \phi(Q', S)\end{aligned}$$

The map ϕ is well defined since PDDL preconditions and states are finite. Starting from a PDDL precondition that is not satisfied always yields a non-empty advice proposition. The advice takes the form of lists of APs, conjoined or disjoined according to context. Further advice lists may then be nested. The actual conditions that need to be changed in the state will be the truth value of predicates and the numerical values of PNEs.

6.2 Advice on Invariants depending on Continuous Effects

The introduction of continuous effects into a plan further complicates the validation of an invariant over a given interval, as discussed in section 4.2. There is a natural extension to the plan repair advice given by ϕ to invariant conditions depending on continuously changing PNEs. An invariant condition must hold for all values on a given interval, this further consideration only changes the advice given by ϕ for comparisons that depend on continuously changing PNEs. Instead of considering just one state the advice for satisfying an invariant must consider: one logical state (for the predicates), and a continuously changing numerical state on the interval in question for comparisons depending on continuous effects. The advice for such a comparison is that it needs to be satisfied on the interval, together with a report of the subset of values of the interval that the comparison is satisfied on.

For a disjunctive advice proposition which states that one of the following must be satisfied the meaning should be interpreted appropriately when referring to invariant conditions. That is, for each time value in the invariant interval one of the advice propositions must be followed. The advice proposition that is followed need not be the same advice proposition for each time value. See [6] section 7.2 for more details on disjunctive invariants.

6.3 Plan Repair Example Using VAL: Beagle 2 Plan

We briefly consider an example to illustrate the role of VAL in a mixed-initiative setting for the Beagle 2 planning problem. The problem has been simplified and details omitted in order to keep the example short.

<pre> Initial plan 1: (generate-solar-power) [43200] 1: (PAW-move STOWED WIND_HIGH_MODIFIED) [100] 500: (PAW-move WIND_HIGH_MODIFIED closeup_peanuts) [800] 1400: (SEQ-SCS-CLOSEUP closeup_peanuts) [130] 2400: (SEQ-MIC-FULL_SET_COMPRESS_EACH peanuts) [17300] Plan 2 ...First three actions as initial plan 1400: (SEQ-SCS-CLOSEUP closeup_peanuts) [130] 1366: (PAW-move closeup_peanuts peanuts) [200] 1567: (SEQ-MIC-FIND_FOCUS_RANGE peanuts) [1982] 3550: (SEQ-MIC-FULL_SET_COMPRESS_EACH peanuts) [17300] </pre>	<pre> Advice for initial plan Invariant for (seq-mic-full_set_compress_each peanuts) has its condition unsatisfied between time 2400 to 19700: set (seq-mic-focus_ranged peanuts) true. Condition requires: (seq-mic-find_focus_range peanuts) Plan 3 ...First two actions as initial plan 500: (PAW-move WIND_HIGH_MODIFIED closeup_peanuts) [800] 1400: (SEQ-SCS-CLOSEUP closeup_peanuts) [130] 1531: (PAW-move closeup_peanuts peanuts) [200] 1732: (SEQ-MIC-FIND_FOCUS_RANGE peanuts) [1982] 3714: (SEQ-MIC-FULL_SET_COMPRESS_EACH peanuts) [17300] </pre>
--	---

Figure 3: Mixed initiative planning using VAL for the Beagle 2

In this scenario, starting shortly after dawn, the Beagle begins with its arm stowed and its battery at a low state of charge (due to overnight maintenance). The operations planner wishes to construct a plan that will allow examination of a rock that has been named “peanuts”. The examination procedure consists of taking a close up image with the stereoscopic camera, then a closer image with the microscope, grinding a core sample, transferring it to the ovens and processing it in the gas analysis system (GAP). We will examine only the first few actions. An initial plan is constructed (through a GUI that does not concern us here) and then the validator reports failure for this plan with advice as shown in figure 3.

This implies an action must be added: 1567: (seq-mic-find.focus_range peanuts) [1982] An error is then reported advising that the paw needs to be moved into position for the microscope to be focussed on “peanuts”. This requires the action 1366: (PAW-move closeup_peanuts peanuts) [200]. The plan is modified to become plan 2 as in figure 3.

Unfortunately, the new movement activity is timed to execute prior to the capture of the stereo-camera image, undermining the precondition for that action. One possible solution to this would be to consider moving the image capture image earlier, along with the paw movement that supports it. However, in this case, attempting to move the second arm move and closeup image action earlier leads to plan failure because it exceeds available power. The advice offered is to push the actions after the first move forward until the first point at which there is sufficient power to execute them, for example as in figure 3 plan 3.

7 Further Work and Concluding Remarks

Polynomials were the first continuous effects to be handled by VAL [6]. The approach can be extended to continuous effects described by more complex functions, by using polynomials to approximate the functions. This approach has been implemented for exponential functions in VAL. This extension has been an important step in the exploitation of VAL in the context of the Beagle 2 operations, since the power models are sufficiently complex that they cannot easily be modelled using simple polynomials alone. In fact, to model them it has been necessary to numerically solve certain classes of differential equations: we have used the Runge-Kutta-Fehlberg [8] method with very encouraging results.

We are investigating ways to improve the advice given for repairing a plan, for example which actions should be added, removed or moved within the plan. In particular how a durative action with an invariant condition which depends on continuous activity, may be moved within the plan to satisfy the invariant. We are aiming for the least human input in the mixed-initiative planning process, so that most plan repair activity can be automated by VAL. Ultimately, VAL should support a complete plan repair strategy.

This work is part of a larger project exploring the transfer of planning technology into

space operations planning in European space missions. This is in the context of world-wide efforts in space exploration and the NASA mobile robots missions currently being pursued on Mars. These missions also make extensive use of mixed-initiative planning aids, including the MAPGEN[1] tool, used successfully in the Mars Exploratory Rover missions.

Applying planning technology to real problems highlights the need to provide solutions to problems that are often not considered in the pure planning research community. Mixed-initiative planning has long been considered an important bridging technology to help human planners to become familiar with and more trusting of automatic planning technologies, while also solving some of the difficulties faced by planners in complex and realistic domains.

In this paper we have discussed the use of VAL, our plan validation technology, as a tool in mixed-initiative planning for space operations. An important aspect of this tool is that it is directly linked to our development of the semantics of PDDL and therefore provides a basis for formal validation of the domain descriptions we are constructing. The importance of continuous change in this domain is an added complication. We have advocated the role of continuous change in planning domain models for some time and this domain is an illustration that it is of key importance in real domain problems.

In the context of the space operations planning we have integrated VAL into the tool that intended for Beagle 2 operations planning. We anticipate that this work will continue to inspire and motivate further developments in our planning technology.

References

- [1] M. Ai-Change, J. Bresina, L. Charest, J. Hsu, A.K. Jónsson, R. Kanefsy, P. Maldegue, P. Morris, K. Rajan, and J. Yglesias. MAPGEN: Mixed initiative activity planning for the Mars Exploratory Rover mission. In *Proceedings of Demonstration Systems Track, ICAPS'03*, 2003.
- [2] A. Blum and M. Furst. Fast Planning through Plan-graph Analysis. In *Proceedings of IJCAI*, 1995.
- [3] S. Edelkamp, J. Hoffmann, and committee. The 4th International Planning Competition 2004, www.informatik.uni-freiburg.de/~hoffmann/ipc-4/.
- [4] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of AI Research*, 20, 2003.
- [5] R. Howey and D. Long. VAL's progress: The automatic validation tool for PDDL2.1 used in the international planning competition. In *Proc. of ICAPS Workshop on the IPC*, 2003.
- [6] R. Howey and D. Long. Validating plans with continuous effects. In *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, pages 115–124, December 2003.
- [7] D. Long and M. Fox. The 3rd International Planning Competition: Results and analysis. *Journal of AI Research*, 20, 2003.
- [8] J. H. Mathews and K. K. Fink. *Numerical Methods Using Matlab*. Prentice-Hall Inc., New Jersey, USA, 4th edition, 2004.
- [9] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 634–639, Anaheim, California, USA, 1991. AAAI Press/MIT Press.
- [10] D. McDermott and AIPS'98 IPC Committee. PDDL—the planning domain definition language. Technical report, Available at: www.cs.yale.edu/homes/dvm, 1998.
- [11] M.J. Woods, R.S. Aylett, D. Long, M. Fox, and R. Ward. Assessing planning and scheduling technologies for deep space exploration. In *Proceedings of 4th British Conference on (Mobile) Robotics: Towards Intelligent Mobile Robots*, 2003.