Tool Support for Planning and Plan Analysis within Domains embodying Continuous Change

T. L. McCluskey and R. M.Simpson
School of Computing and Engineering
The University of Huddersfield, Huddersfield HD1 3DH, UK
t.l.mccluskey,r.m.simpson@hud.ac.uk

Abstract

In this paper we describe work in progress that aims to develop a domain-independent tool set which supports the creation and analysis of domain descriptions and plans containing continuously changing processes, instantaneous events, and actions. The tools described are (i) a life history editor that enables a designer to create a domain description diagrammatically, automatically generating much of the textual encoding; (ii) an animator that simulates plans, firing events and processes; (iii) an HTN planner that generates plans in such domains. Tools (i) and (ii) are currently implemented as a subset of the GIPO III environment. We conclude by stating some initial findings and points that might be raised at the workshop.

Introduction

Planning in domains which are represented with rich notations has long been a great challenge for AI (Bresina *et al.* 2002). In particular, many real world phenomena are outside the explicit control of plan execution, yet have to be reasoned with during plan generation. Changes ocurring because of fuel consumption, continuous movement, or environmental conditions may not be adequately modelled through instantaneous or even durative actions; rather these require modelling as continuously changing processes.

Our research is heavily involved with knowledge formulation of domains which involve AI Planning. Even small, classical planning domain descriptions are difficult to encode adequately and correctly. Without tool support, encoding descriptions require the user to understand the planner and the description language in great detail. GIPO (McCluskey, Liu, & Simpson 2003) is a tool which supports the creation of domain descriptions, and the exploration of planning within the domain, using an interface that hides much of the technical details. Users can build a domain description using a diagrammatic interface, explore plan creation using a plan stepper, interface their domain to an external planner, and analyse the resulting plans using an animator. The generality of the graphical tools are enabled by the use of the 'object' as the common semantic thread.

Recently Long and Fox re-visited the issues of planning within domains with events and continuous processes, with the introduction of PDDL+ level 5 (Fox & Long 2001). In this paper we discuss extensions to the GIPO architecture and related tools for domain formulation, plan generation and plan analysis in such rich domains. We have adopted an underlying formalism similar to PDDL+ called *OCLplus*, but one which also supports the use of the GIPO knowledge acquisition tool.

Using *OCLplus*: Planning Domains with Processes and Events

OCLplus is derived from GIPO's object-centred language which is documented in the literature (McCluskey, Liu, & Simpson 2003). This language imposes a structure on the domain in terms of objects and object classes, and structures action representations in terms of object state transitions. Additionally in *OCLplus*, time is modelled explicitly as a real quantity. This enables simulation or planning with the model to adopt an arbitrary approximation to the advancement of time; hence in the domain description time is modelled as a continuous real variable, but in simulation an approximation has to be adopted. This presents a set of serious problems as discussed in the literature (Howey, Long, & Fox 2004). For example, during discrete simulation events may fire or not fire depending on the chosen granularity of time.

State changing operations are divided into three: actions, events and processes. Actions (with Strips-related semantics) bring about instantaneous change to the state of domain objects and may also update the numeric properties of those objects. Actions are the entities that agents must initiate to achieve planning goals, trigger events and processes. Events are specified in the same way as actions, but their semantics differ: whenever their preconditions are made true, the event fires and may change the state of the object(s) involved. In other words, events are automatically triggered as a result of the numeric changes brought about by domain processes, and (possibly) by state changes of other objects. Like actions, events bring about instantaneous change, but unlike actions events are not directly triggered by the plan executive. Processes specify in addition to their start condition how numeric properties of the objects in the domain are updated with the passage of time as a result of the running processes. Hence processes update numeric properties, but do not bring about state change of the objects (in the sense of changing the truth value of relations); their descriptions are

available declaratively, but they cannot be directly activated by the agents acting in the domain, despite the fact that the intention behind performing some action may simply be to start a process running.

Review of GIPO III's Life History Editor

GIPO III's object life history editor (OLHE) is used to formulate domian descriptions. Is is reminiscent of CASE tools that support designers construct diagrammatic representations of systems. These systems are usually based around an 'object model'. The CASE tool's principal function would be to help the user construct a (partial) formal model from the diagrammatic description, thus insulating the designer from the code. Depending on the CASE tool, the code generated by the tool may be some abstract, formal specification (eg tools that generate partial Z specifications), or partial implementations within a programming language.

In the case of GIPO III, the user has to create a diagrammatic representation of the planning domain using a state - machine metaphor. This forces the designer to decompose the system into an assembly of machines, where each machine represents the behaviour of a class of objects. Each machine's nodes represent the set of mutually exclusive states that an object of the class can hold. Other attributes of the objects are modelled as 'properties'. The diagrammatic form first undergoes consistency checks, and then is translated into a domain description which can be used for plan generation purposes.

Apart from an adequate diagrammatic description, the user only need specify problem scenarios (including object instances) before a target planner can be invoked. As an example, to construct the Dockworkers - robots world used in the recent AI planning textbook (Ghallab, Nau, & Traverso 2004), one only needs to construct the diagram shown is Figure 2. From this diagram GIPO III generates the dynamics of the domain, and converts this into a planner-friendly form.

The diagrams are based on primitive machines and machine transitions. Machines represent a class of objects that share the same behaviour, nodes represent the state of objects, and arcs change the state that objects go through. In addition to changing state, nodes can have changing properties (for example, locatedness) and these occur as annotated transitions of a different colour to state changes. Actions involving more than one type of object are modelled by co-ordination arcs. To capture preconditions, we constrain objects from different machines to be in certain states or have certain properties when a transition occurs. To capture necessary or conditional changes we constrain two or more transitions of objects from different machines to occur together. For more details, the reader can consult a recent paper in which we gave an informal, translator-independent semantics to these diagrams via the techniques of algebraic specification (McCluskey & Simpson 2005).

As a further aid to construction, the user is given an extensible set of machine primitives with which to work (in Figure 2 the primitives are listed in the left hand pane). This leads to a higher level platform than constructing at the level of the node and arc. For example, the dockworkers - robots world can be constructed by selecting two instances of stacks, and two bistates, and then configuring and combining them together to produce the diagram shown in Figure 2. GIPO III then produces the domain description code from the diagram, ready for input to a planner.

algorithm PlusPlan: *OCLplus* Planner

```
1.store = {node(0,init-state,empty,init-task,init-constraints)}
2.repeat
3.        call expand-plan();
4.        call simulate events-processes();
5.until empty(store) or ∃ N ∈ store:
          solution-node(N);
6.end
procedure expand-plan()
1. repeat
2.          remove node(T,S,PP,Exp,C) from store;
3.          if not methods(start(Exp),S,C) = {}then
4.            ∀M ∈ methods(start(Exp),S,C)
5.              Exp+ = Exp with M expanded;
6.              C+ = C with M's expansion constraints;
7.              store node(T,S,PP,Exp+,C+)
8.          else if not operators(start(Exp),S,C) = {}then
9.            ∀ O ∈ operators(start(Exp),S,C)
10.             store node(T,apply(O,S),add(PP,O),rem(Exp,O),C)
11.         else write node(T,S,PP,Exp,C) to temporary store
12. until empty(store) or ∃ N ∈ store
          solution-node(N);
13. let store := temporary store;
14. end
procedure simulate events-processes()
1. temporary store := store; store = empty;
2. ∀ node(T,S,PP,Exp,C) ∈ temporary store:
3.    ∀ E ∈ Events:
4.      if preconds(E,S) = true then S := apply(E,S)
5.    ∀ P ∈ Processes:
6.      if preconds(E,S) = true then S := apply(P,S)
7.    store node(T+Delta,S,PP,Exp,C);
8. end
```

Figure 1: The PlusPlan Algorithm

GIPO III's Life History Editor for *OCLplus*

The OLHE for continuous domains uses the same philosophy as the standard OLHE discussed above. Colour coding distinguishes actions, events, transitions, processes, states etc. Events are connected to the states that change as a result of the event, and an event may be co-ordinated with objects of other classes in the same way as actions. Additionally, an event will have one or more preconditions which depend on time. The specification of these properties are initiated and changed by clicking on the event's icon.

Processes are connected up diagrammatically with states that are required as their preconditions, but expressions involving continuously varying values require the use of a expression editor. We illustrate this idea with the familiar bath - filling application shown in Figure 3. In the diagram there is one process specified (filling), annotated with a

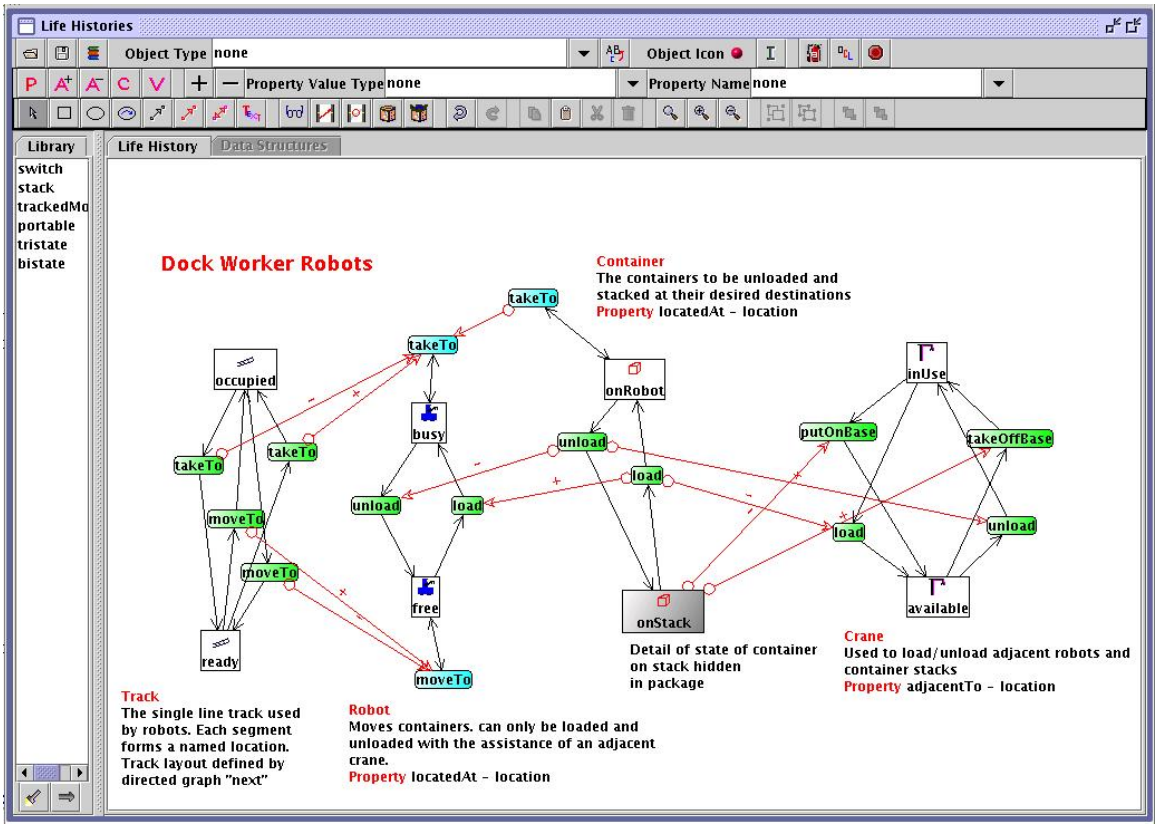

Figure 2: The Dockworkers Domain: Example of GIPO III's OLHE

clock icon to show it is a process. As well as connecting up the process to relevant states/events, the designer must input triggers for the process, i.e.

$level(Bath) \leqslant capacity(Bath)$
$flow(Bath) > 0.0$

as well as the effect of the process:

$level(Bath) = level(Bath) * flow\text{-}rate * time\text{-}increment$

An expression editor is provided for this. There is one event in the diagram - 'flood'. State change information is generated by the connections of the event in the diagram, but, like a Process, an expression editor is needed to input the fluent trigger for the event, in this case:

$level(Bath) \geqslant capacity(Bath) \& flow(Bath) > 0.0$

With this extra information, GIPO III can generate the domain description for use with the Stepper and Planner outlined below.

GIPO III's Plan Stepper

The **Plan Stepper** assumes a complete but not necessarily correct domain description exists and that the description requires dynamic testing. The stepper allows the user to manually develop a graphical representation of a plan to solve a given domain problem by choosing from the defined operators a sequence in which to apply them and by choosing instantiations of the operator parameters that legally allow the operator's application at that stage in the developing plan. We continue the simple bath filling example in Figure 4. Here the object's timelines run along from left to right, in the top pane. In the lower pane the process(es) activation is shown from left to right. Initially a plug is placed in the bath (first dot - in red) then the hot water tap is turned on (affecting the bath, the tap and having the effect of starting the filling process). Next the cold tap is turned on, and finally after several minutes the bath overflows and causes a flood (the flood event is signified by a cross).

The tool in addition to providing a graphical representation of the plan checks that each step can be legally applied assuming the given problem's initial state. In this way the user gains feedback on whether or not the domain description fulfills his/her expectation. Clearly if the plan is not legal then either the plan itself is flawed or the domain specification is at fault. The user is expected to iterate over this process until both sample plans and domain specification appear satisfactory. The plan stepper allows a user to manually create plan descriptions and have them checked against a developing formal description of the problem domain. Tools based on the idea of a plan stepper can be of more use than

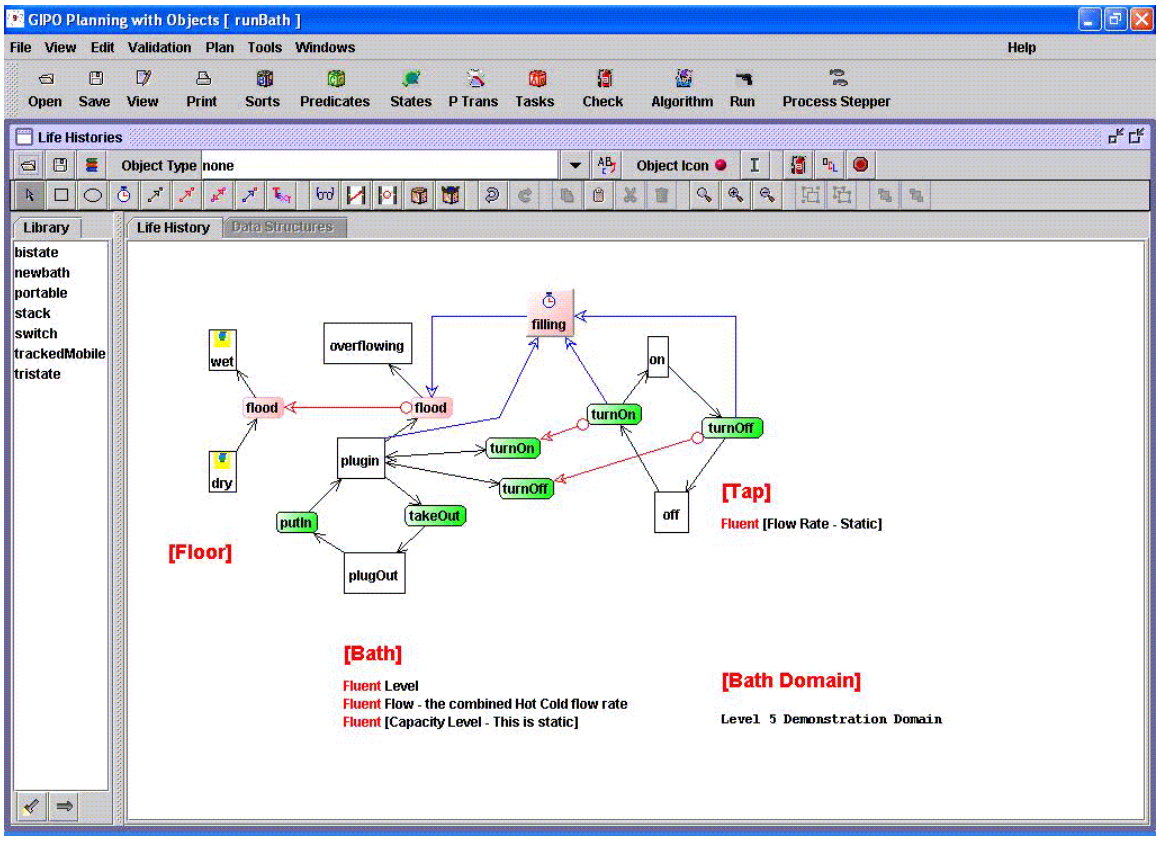

Figure 3: The Bath Filling Domain: Example of GIPO III's OLHE

just as a tool for domain description validation. In many domains users are not looking for nor would be prepared to trust fully automated tools to carry out their planning tasks.

Plan Generation

We have designed and implemented a plan generation algorithm ('PlusPlan') for *OCLplus* which was derived from HyHTN (McCluskey, Liu, & Simpson 2003). HyHTN is a hierarchical planner which, like Shop (Nau *et al.* 2000), plans by selecting and expanding method operators, and keeping an advanced state within search nodes. HyHTN's innovation is that it is equally suited to hierarchical, non-hierarchical planning, or a combination of the two. During search, nodes undergo expansion operations which replace names of methods with their decomposition. However, if within a node some condition cannot be achieved in the current state then HyHTN can perform a heuristic state space search (using the plangraph heuristic) to generate a plan to achieve the goal.

Essentially, PlusPlan works using a blend of planning and simulation. The simulation is similar to that carried out in the Plan Stepper described above: an incremental time step is used to simulate the passing of time. Planning involves a state-advancing HTN search at each instant of time. Hence, if the goal can be reached by the effect of instantaneous actions only, it will endeavour to find a plan to achieve this, and apply that plan's actions to an advancing state, in the first instant. More likely, nodes will be processed resulting in the addition of new nodes to the search, but these will not be further processed as some event or process is needed to occur or terminate before further work on a plan can occur.

Figure 1 contains an outline of PlusPlan. It searches through a space of nodes of the form:

$$node(T, S, PP, Exp, C)$$

where T is the elapsed time from the start of the plan, S is the current state at time T, PP is the plan that produces S, Exp is a set of operators representing the developing (future) plan, and C is a set of constraints on Exp. Initially Exp is the abstract task to be achieved, and C an initial set of constraints.

Procedure expand-plan tries to create and apply a plan at one instant. The resulting open nodes in 'store' are either solutions or nodes where operators cannot be executed. In line 3 $start(Exp)$ = set of fringe operators, ie those that could possibly start (temporally) Exp;
$methods(start(Exp), S, C)$ is the set of non-primitive operators in the fringe such that their preconditions are met in S and the constraint C are satisfiable. For all these possible decompositions, new nodes are created and stored. If no non-primitive operators can be expanded, then in lines 8-10 any operators that can be applied are applied to advance the

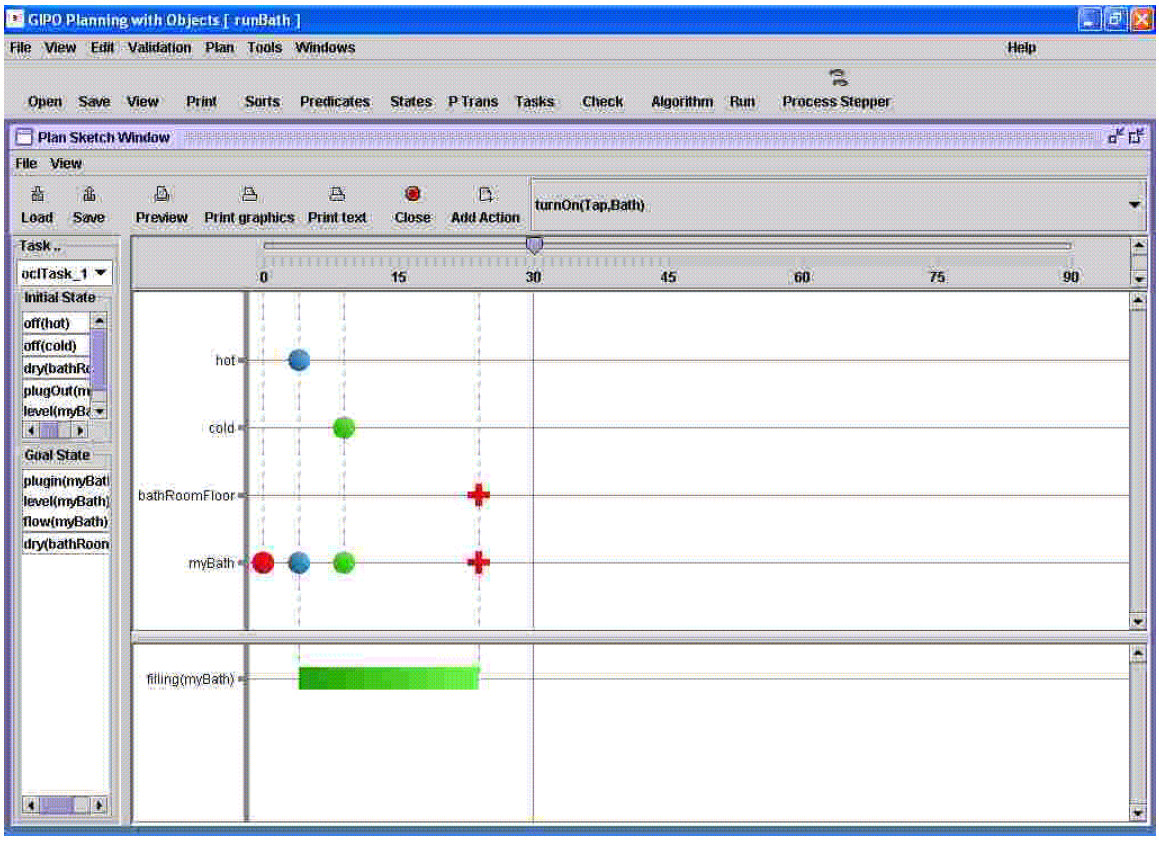

Figure 4: The Bath Filling Domain: Example of GIPO III's Plan Stepper

state. When a node cannot be expanded, it is put into a temporary store, and when all nodes are processed (or a solution is found) then the temporary store is copied back to the main store and control is given back to the main algorithm.

The procedure for simulating events and processes retracts all of the stored nodes, and with each in turn, it applies all applicable events and processes. If > 1 events occur at the same time then the order that they fire is assumed unimportant - the object states that any two firing events refer to must be independent. Similarly, if events occur during process execution, they are assumed not to interfere with the processes, that is the 'order' of simulation is not important. Finally, it is assumed that two processes that are continuously running do not directly interfere. These assumptions avoid the problem of interfering events "occurring at the same time" - which can lead to a paradox. After all the effects of applicable events and processes have been applied to a node, the time of its state is moved on an instant (signified by "Delta") and the node is stored.

The main loop progresses until expand-plan() is in a position to fully expand the abstract task into a plan containing primitive instantiated operators.

Results and Open Questions

Planning in domains with continuous processes, events and actions has long been a great challenge for AI. Real world phenomena are often outside the explicit control of plan execution and yet have to be reasoned with during plan generation. In this position paper we have outlined three tools used to formulate domain descriptions, analyse plans through simulation, and generate plans respectively, in domains with actions, events, and processes. An important aim of this work to allow the expressiveness of the *OCLplus* description to be explored and to investigate additional knowledge engineering features that may be added to the language to support the modelling and domain validation processes.

We have evaluated the tools on toy problems including an air traffic control scenario (specified in (Simpson & McCluskey 2003)), a bath domain, and an autonomous vehicle domain. The first tools (formualation and simulation) are encoded into GIPO III with *OCLplus*, and can be downloaded from http://scom.hud.ac.uk/planform/gipo. For the future we are aiming to apply the tools to a more realistic flood simulation domain. The initial work shows that (a) the GIPO tools originally designed for classical domains generalise naturally to domain descriptions in *OCLplus* (b) problems to do with the simulation are extremely complex. Depending on the choice of the time step "Delta" in both simulation and plan generation, events may fire or not fire, and processes may not run accurately or at all. The work has already raised certain fundamental questions such as: is the development of a domain-independent toolset for such

expressive domains feasible or even desirable? While continous change is expressible in domain description languages such as *OCLplus*, how can this be adequately simulated in animators and planners?

References

Bresina, J. L.; Dearden, R.; Meuleau, N.; Ramkrishnan, S.; Smith, D. E.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for ai. In *UAI*, 77–84.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains . In *Technical Report, Dept of Computer Science, University of Durham*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann ISBN 1-55860-856-7.

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *ICTAI*, 294–301.

McCluskey, T. L., and Simpson, R. M. 2005. Towards An Algebraic Formulation of Domain Definitions using Parameterised Machines. In *Proceedings of the 24th UK Planning and Scheduling Workshop (PLANSIG-2005), City University, London*.

McCluskey, T. L.; Liu, D.; and Simpson, R. M. 2003. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. In *The Thirteenth International Conference on Automated Planning and Scheduling*.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 2000. SHOP and M-SHOP: Planning with Ordered Task Decomposition. Tech. Report CS TR 4157, University of Maryland, College Park, MD.

Simpson, R. M., and McCluskey, T. L. 2003. Plan Authoring with Continuous Effects. In *Proceedings of the 22nd UK Planning and Scheduling Workshop (PLANSIG-2003), Glasgow, Scotland*.