Using GIPO to support learning in knowledge acquisition and automated planning

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
http://scomeng.hud.ac.uk/planform/gipo/

Abstract

The process of how knowledge is acquired and formulated in knowledge-intensive AI is difficult for a student to grasp without practical experience. Often, AI text books and lecture notes contain examples of logic formula or structured knowledge representations which are well refined and bug free. These polished examples are then used to show reasoning mechanisms or the execution of AI search methods. The process of how the knowledge representations themselves are acquired and validated is often neglected. In this paper we describe the use of a tool called GIPO for teaching AI students. GIPO helps students understand and integrate aspects of knowledge acquisition, knowledge engineering, automated planning and machine learning. We show how the tool's features supports teaching and the student's learning experience, and helps integrate the theory and practice in a range of AI and related subject areas.

1. Introduction

Very often AI practical classes, AI text books and AI lecture notes start with examples of logic formula or structured knowledge representations which are well refined and bug free. These polished examples are then used to show reasoning mechanisms or the execution of AI search methods. For example, in AI planning, students are given a set of crafted action representations and are then shown how planning algorithms reason with these structures to generate plans. Both the structured representations employed, and the reasoning mechanisms themselves, are difficult for students to grasp, and they require a set of integrated knowledge from previous courses on logic and computer science.

The peculiar problems to do with acquiring and crafting knowledge bases about actions and change is another factor as to why the teaching of knowledge-intensive AI is difficult. Students may have encoded dynamic knowledge in other parts of the curriculum: they represent dynamic systems informally in object-oriented analysis and design, or formally using a process algebra. However, representing knowledge about actions and change for automated reasoning purposes presents more problems.

The process of how knowledge is acquired and engineered is not easy for a student to grasp without *practical experience* of the process. As is the case with programming and design, it seems that an integrated tools environment that allows the student to effectively apply the theory in a practical scenario is desirable. This should give the student a high level platform from which to learn advanced concepts without the need to worry about editors and syntax. From our experience, a useful tool to help in the teaching of AI within the computing curriculum should integrate a range of theory taught during lectures with the application of the theory during practical classes, supporting a wide range of the AI curriculum. Its interface should have a familiar look and feel, and allw the student to produce non-trivial AI implementations. It is also useful if the tools helps integrate AI with other subject areas taught at undergraduate level. Traditionally AI has been taught within practical sessions by the introduction of declarative programming languages such as Prolog, Lisp and Haskell. While these programming languages can be used for a wide range of AI topics, it is not easy to lead students to build or integrate advanced AI functions from the basis of a programming language. The tutor would implement AI algorithms to expose their workings, but knowledge intensive issues such as domain modelling would be harder to illustrate.

Here we describe the use of a tool called GIPO for teaching AI students. It has been used since its creation in 2001 for knowledge engineering of AI Planning knowledge. We argue that GIPO meets the criteria in the paragraph about and helps students understand and integrate aspects of knowledge acquisition, knowledge engineering, automated planning and machine learning. We show how the tool's features supports teaching and the student 's learning experience, and helps integrate the theory and practice in a range of AI and related subject areas.

2. Overview of GIPO

GIPO [1] the 'Graphical Interface for Planning with Objects' (Simpson *et al.* 2001) (pronounced GeePo) is the name of a family of experimental tools environments providing help for those involved in knowledge acquisition, domain modelling, task description, plan generation and plan execution. GIPO was an output of the PLANFORM project (Planform 1999), and has been demonstrated in several major AI conferences. GIPO won the prize for best general tool at the first international competition for knowledge engineering in AI planning, held at Monterey, USA in June 2005. Three versions of GIPO - GIPO I,II, and III are available for downloading from the website. GIPO integrates a range of planning tools to help the user explore the domain encoding, and determine the kind of planner that may be suitable to use with the domain. In particular it has:

- graphical tools and visual aids for the input/display of objects, object classes (sorts), predicates, constraints, states, operator schema, and tasks. There are familiar point and click, drag and drop functions to help the user build up a new domain or reuse existing components.
- validation checks for consistency across parts of the developing domain model. Once operator schema have been developed GIPO features a 'plan stepper' which helps the user build up their own solutions to problems in a kind of 'mixed-initiative' mode.
- resident plan generation engines, and an API for plugging in to third party AI planners. A plan animator / visualiser displays a planner's solution to a problem in terms of the objects which are effected by the plan. This can be stepped through by the user to see the effects of operators on objects and their properties.

[1] http://scom.hud.ac.uk/planform/gipo

A key design goal in building the tool's interface has been to allow the creation of a specification in terms of images that describe domain structure at a high level of generality. The tool takes care of the detail of the syntax of the underlying specification, making it impossible to construct a syntactically ill-formed specification. The process of domain model development on which this is based is detailed in the literature, see references (McCluskey & Porteous 1997; Liu & McCluskey 2000) for more details.

3. Capturing Domain Structure

Students learn how to use GIPO in two ways: Firstly, they can investigate and execute one of the several pre-engineered applications domains that is supplied with GIPO. The student can use these at an early stage to see the result of domain building. They are able to bind the models with a planner of choice and use GIPO to solve planning problems and execute the solutions. Secondly, the student can use GIPO's tutorials. These lead the student through a staged method of domain development using appropriate examples. This is a similar to the tutorial material written for the Protege tool (Gennari *et al.* 2003) which introduces the user to Description Logic.

The central conception in domain capture is that planning essentially involves changing properties and relationships of the objects that inhabit the domain. This appeals to computing students' intuition and is consistent with their studies in object-oriented programming and design. Knowledge is structured around object descriptions, their relationships and the changes they undergo as a result of the application of operators during plan execution (in contrast to the traditional literal-based approach used in Planning languages such as PDDL (Ghallab *et al.* 1998)). The student identifies the kinds of objects that characterise the domain, and organises them around distinct collections of objects, which we call *sorts*, into a hierarchy. Object instances for each sort are identified. Each object instance in a sort is assumed to have identical behaviour to any

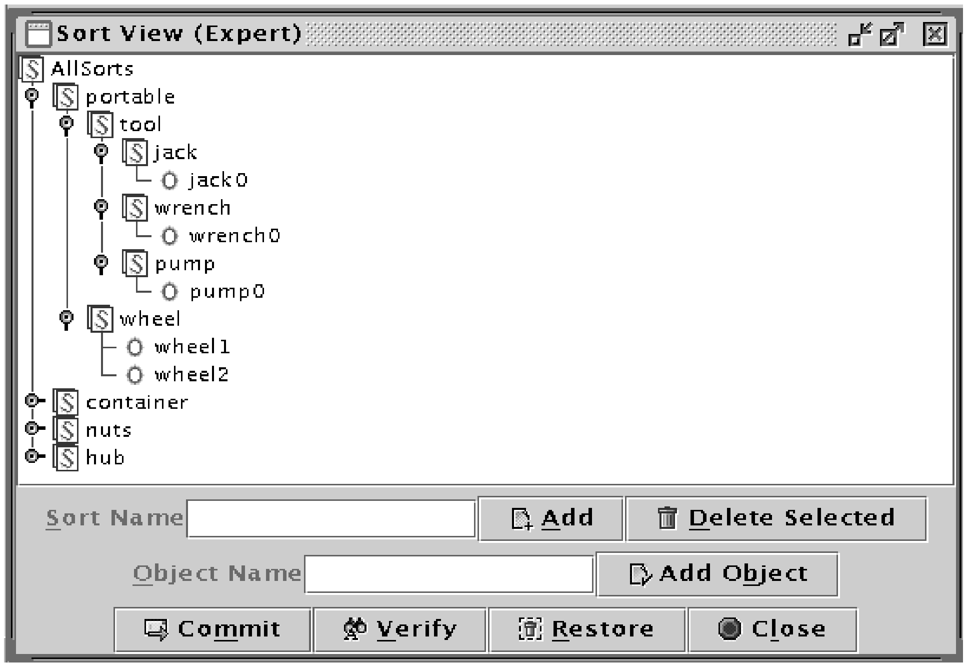


Figure 1: Snapshot of the Sort Editor

other object in the sort. To assist in this element of the conceptualisation GIPO provides a visual tree editor (Figure 1). Domain checking at this initial stage involves enforcing the tree structure and requiring that node names (for sorts and objects) are unique.

The next step if for the student to specify the sorts in the domain by identifying predicates that characterise the properties of a typical object of each sort and relationships that hold between objects. GIPO provides an editor to define predicates by a process of drag and drop from the sort tree previously defined. Next, the student specifies *domain invariants*. The most important invariants are those characterising the range of states that an object of each sort can occupy. These form the basis of the static validity checks that can be carried out on the completed domain specification. For example, if the sort representing the physical entity is 'door', and the predicates are closed, locked and unlocked, then the student would use the tool to state that the only possible interpretations that can be true are:

locked and closed;
unlocked and open;
unlocked and closed

Any other combinations (eg open and locked; or locked, closed and open) are excluded. The collection of all such states for the object will be such that at any instance in time exactly one such description will be true of the object (see Figure 2).

When specifying object states, the possible unifications of variables of the same sort or between variables belonging to the same path in the sort tree hierarchy can be restricted using a visual indication of unifying variables as shown in the figure. The student selects from a popup menu how an individual variable is to unify with a target variable and if the decision is that they must be distinct then a *not_equals* clause is generated. This strategy for dealing with the unification of variables is pervasive in the GIPO tool set.

4. Capturing Domain Operators

The next stage of the knowledge acquisition method, and most difficult task for the student, is to specify operators representing domain actions. Operators in GIPO are conceptualised as sets of parameterised object transitions, *LHS* $\Rightarrow$ *RHS*, where the LHS and RHS are the legal state descriptions of the sort of the object parameter. An object transition can have different modalities in an operator - normally it is *necessary*, which means the *LHS* is a precondition of the operator, and after the operator is exe-

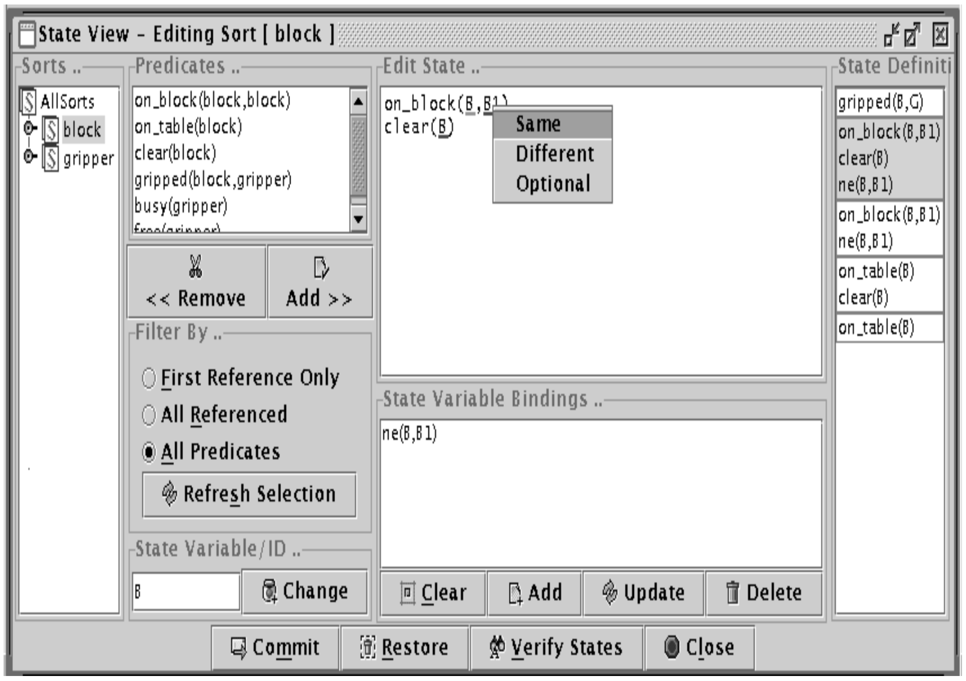


Figure 2: Editor for Specifying Object State Invariants

cuted the object affected will be in the situation specified by a fully instantiated *RHS*. The GIPO operator editor helps the student create a graph representation of an operator where the nodes are the LHS and RHS states of the object sorts involved in the operator. Each such node contains an editable state definition(see Figure 3).

While the use of the 'Operator Editor' is adequate to define operators, students have difficulty primarily due to the possible codesignation of variables across the different nodes presented to the user (although the underlining and right click mechanism described in the state editor is used). Using this manual operator tool illustrates to the student the difficulty of knowledge formulation, particularly to do with actions.

5. Capturing Domain Operators using Induction from Examples

A semi-automated knowledge acquisition tool in GIPO is 'OpMaker (McCluskey, Richardson, & Simpson 2002): this helps the user to create an operator set simply by providing example solution sequences. The exercise illustrates some of the concepts of 'Learning from Examples' in machine learning - in particular inductive generalisation.

To help explain OpMaker, we use a planning domain that is supplied with GIPO - the 'Lazy Hikers' domain. Two people (hikers) go hiking and driving around regions of the Lake District, with objects such as *tents*, *cars*, *regions*, and actions such as *putdown*, *load*, *getin*, *getout*, *drive*, *unload*, *putup*, *walk*, *sleepintent*. They do one 'leg' of a long circular track each day, as they get tired and have to sleep in their tent to recover for the next leg. Their equipment is heavy, so they have two cars which can be used to carry their tent and themselves to the start/end of a leg. To use OpMaker, the student must first create a 'partial' domain model, containing objects, sorts, predicates and state invariants describing the problem domain. The student then constructs (via a drag and drop process) a solution to a predefined task - for instance the following is a solution to the task of doing one leg of the circular track and being ready for the next leg in the morning:

```
putdown
  tent1 fred keswick;
load
  fred tent1 car1 keswick;
getin
  sue keswick car1;
drive
  sue car1 keswick buttermere;
```

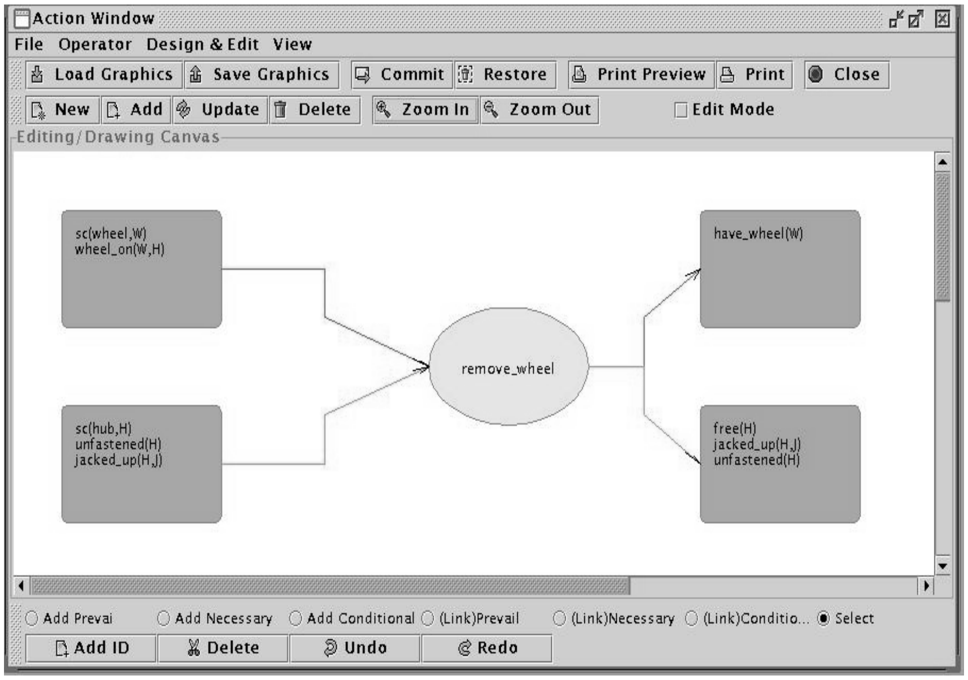


Figure 3: Operator Editor Tool

```
getout
  sue buttermere car1;
unload
  sue tent1 car1 buttermere;
putup
  tent1 sue buttermere;
getin
  sue buttermere car1;
drive
  sue car1 buttermere keswick;
getout
  sue keswick car1;
walk
  sue fred keswick buttermere;
sleepintent
  sue fred tent1 buttermere
```

The student is encouraged to think of each action in terms of a sentence describing what happens. For example in the last action we think of this as 'Sue and fred sleep in their tent in Buttermere'. Each 'action' consists of an action identifier followed by a sequence of objects that the action depends on or changes. From the input of a plan such as the example above, and a partial domain model, a full operator set can be induced with the tool (see Figure 4).

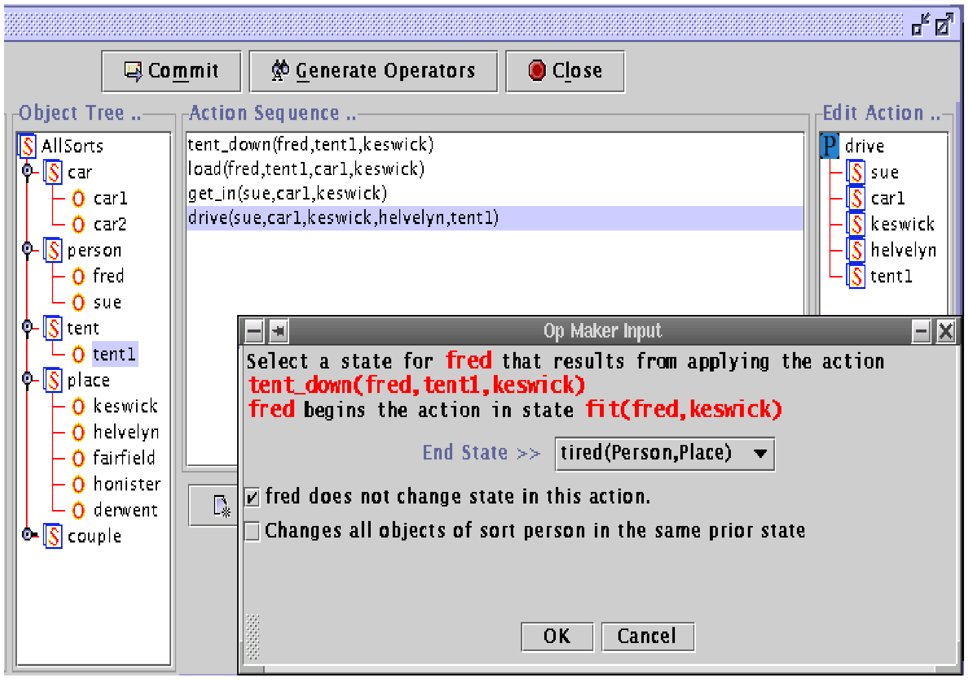
6. Student Learning Opportunities

From this stage in process, the students learn the difficulty in acquiring knowledge about actions, and how using machine learning techniques one can potentially avoid the need to hand craft action knowledge. They can compare these acquisition methods: After OpMaker has produced the set of induced operators, another learning opportunity for the student is to compare this set with the handcrafted set supplied with GIPO. They can explore the problems and limitations of learning from examples to do with convergence of generalisations, the need for knowledge refinement and the importance of 'good' examples in learning.

The construction of operators provides a good opportunity to compare the planning model with work in formal specification of software. For example, many of our students used the 'B-toolkit' (B-Core (UK) Ltd) to create software specifications. The pre- and post-condition version of a GIPO operator and an operation specified in B have great similarities as they both specify deterministic, instantaneous actions in terms of predicate descriptions.

7. Validation in GIPO

Continuing the analogy with formal specification of software, once the student has built up an initial model of the world it is natural to want to validate it. As in formal



tent_down(fred,tent1,keswick)
load(fred,tent1,car1,keswick)
get_in(sue,car1,keswick)
drive(sue,car1,keswick,helvelyn,tent1)

Select a state for fred that results from applying the action
tent_down(fred,tent1,keswick)
fred begins the action in state fit(fred,keswick)

End State >> tired(Person,Place)

[✓] fred does not change state in this action.
[ ] Changes all objects of sort person in the same prior state

Figure 4: OpMaker Induction Tool

specification, this splits into *internal validation*: checking the model for inconsistencies between component parts, and *external validation* checking the model's accuracy with respect to what is being modelled. With tools such as GIPO, 'local' consistency checks on name uniqueness and hierarchy definition are automatic when these components are being built. Additionally, internal validation includes checks on *global consistency* through various forms of 'static' validation. Most effective in GIPO are the checks which verify that operator definitions do not compromise the invariants.

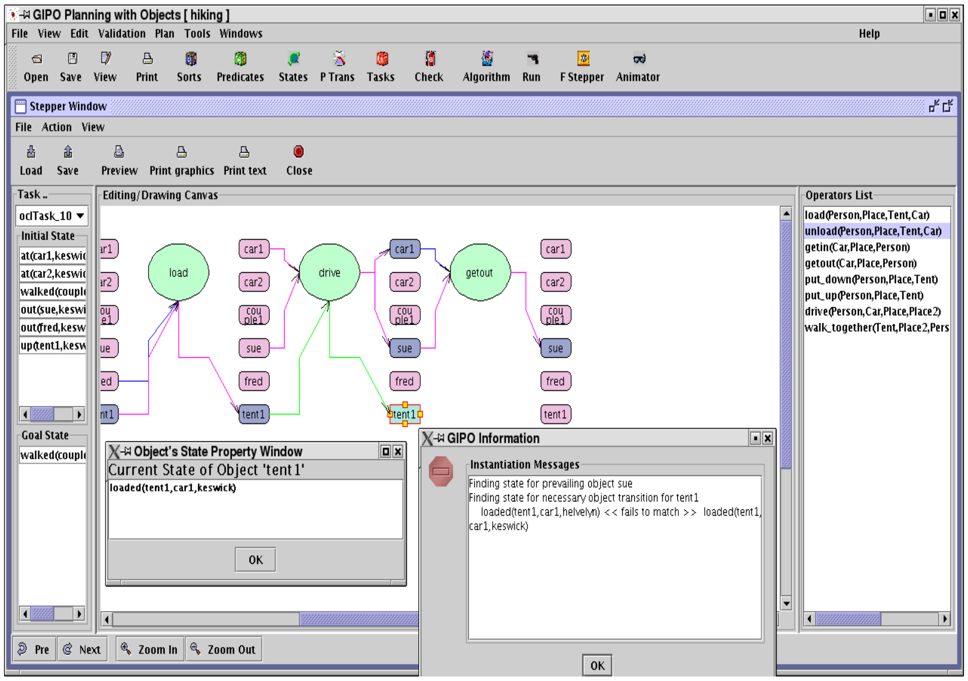
8. Dynamic Validation

The student has several opportunities to learn about and carry out 'dynamic' internal and external validation resulting in the repair of inconsistent, inaccurate or incorrect knowledge.

- the *reachability analysis* tool: after the student has specified state invariants and an operator set, the student can use this tool to check whether the operator set is sufficent to reach all states allowed by the invariants. The reachability tool can be used in conjunction with OpMaker: it can indicate if the deduced operators do not give an adequate coverage. This is shown by the existence of defined states that are not referenced by any operators.

- the *plan stepper*: the student can dynamically check a domain is adequately specified against a set of problems by using the plan stepper. The student uses drag and drop to select operators, and pop-up menus to instantiate them, effectively attempting to solve their own planning problems using the model. Each operator is applied in the current state to generate the consequent state. The student proceeds in this manner to verify that the domain and operator definitions do support the known plans for given problems within the domain. The stepper operates as a manual forward planner, with results of each object transition caused by an operator shown graphically (see Figure 5). This tool is very useful for checking *hierarchical* operators - that is those that encapsulate other operators. In Figure 6 we show a snapshot of the use of the stepper to execute plans invovling hierachical operators.

- running planning engines: the student can, of course, execute one of the supplied planners within GIPO on specified tasks. Such is the intractable nature of planning problems that his has to be care-


GIPO Planning with Objects [hiking]
File View Edit Validation Plan Tools Windows Help
Open Save View Print Sorts Predicates States P Trans Tasks Check Algorithm Run F Stepper Animator
Stepper Window
File Action View
Load Save Preview Print graphics Print text Close
Task ..
oclTask_10
Initial State
Goal State
Editing/Drawing Canvas
load
drive
getout
car1
car2
couple1
sue
fred
tent1
Operators List
Object's State Property Window
GIPO Information
Instantiation Messages
OK
Pre Next Zoom In Zoom Out

at(car1,keswic
at(car2,keswic
walked(couple
out(sue,keswi
out(fred,kesw
up(tent1,kesw

walked(couple

load(Person,Place,Tent,Car)
unload(Person,Place,Tent,Car)
getin(Car,Place,Person)
getout(Car,Place,Person)
put_down(Person,Place,Tent)
put_up(Person,Place,Tent)
drive(Person,Car,Place,Place2)
walk_together(Tent,Place2,Pers

Current State of Object 'tent1'
loaded(tent1,car1,keswick)

```
Finding state for prevailing object sue
Finding state for necessary object transition for tent1
   loaded(tent1,car1,helvelyn) << fails to match >>  loaded(tent1,
car1,keswick)
```

Figure 5: The GIPO Plan Stepper

fully controlled by the tutor and GIPO. Depending on the planner / task combination chosen, the solution may not be found for a good period of time. GIPO has its own planning engines, but third party planners are easy to integrate (we often use the FF (Hoffmann 2000) planer, which was a past winner of the International Planning Competition). After a planner has returned a solution, the student can step through the solution using GIPO's *animator* tool. This takes the results of a planner and produces a graphical representation of the object transitions using the same layout as the stepper.

We have outlined the main components of GIPO above - more details can be found in the AI Planning literature e.g (McCluskey, Richardson, & Simpson 2002). To enable GIPO to be used as a general domain modelling tool we have developed translators between our internal language and the planning domain language PDDL (Simpson *et al.* 2000). The API enables external planning systems to interface to the tools, to provide scope for testing and fielding alternative planning algorithms to those internal to GIPO.

9. Using GIPO in Teaching and Learning

GIPO has been used in the teaching of intermediate and final year undergraduates, in both introductory and advanced AI modules since its creation in 2001. It offers a wide range of learning opportunities in AI, through knowledge acquisition, knowledge formulation, validation and maintenance of domain models, inductive learning and automated plan generation. Although numbers of student groups (typically 15-20) are too small to make any statistical claims, anecdotally GIPO seems to help students to *integrate* AI knowledge learned in lectures, and to reach a *deeper* level of understanding of 'dry' subject matter on for example the acquisition and engineering of symbolic knowledge.

Students are supported by an online three part tutorial, which introduces them to the subject matter in a step by step fashion, by leading them to develop a simple example domain model. Part one of the tutorial introduces the 'flat' model, where operators are primitive and separate. Part two introduces a hierarchical model of plan operators, which amounts to a principled approach to HTN planning. Finally part three introduces the OpMaker operator learning

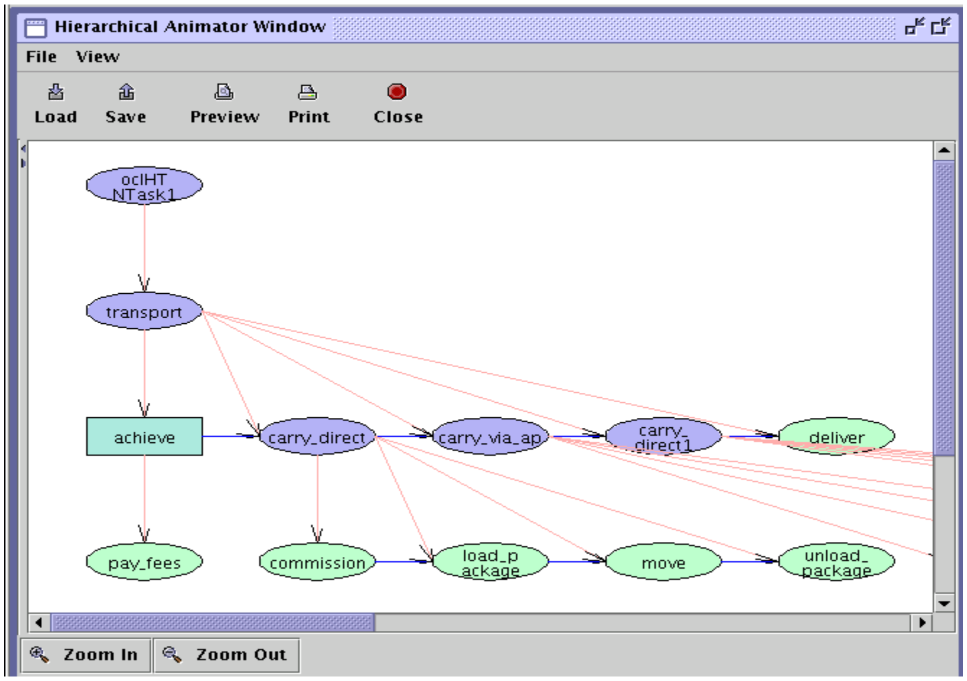


Figure 6: Using the Plan Stepper in Hierarchical Domains

method. Whereas the tutorials lead the student through the features methodically, for learning about specific features GIPO has an online, hyperlinked user manual. For those students who need to dig deeper (for example final year project students) GIPO also has a language manual which defines the underlying knowledge representation language.

10. Conclusions

In this paper we have illustrated the use of the GIPO tool, and shown how it helps students apply AI theory that they have learned during lectures. Its interface and underlying language uses the object metaphor similar to other tools that students use in the computing curriculum. Students are able to use it both to gain experience of a wide range of AI topics (knowledge acquisition, automated planning, learning from examples) and to obtain a deep knowledge of topics in these areas. For example, a student may learn about algorithms for learning from examples, and representations for planning operators, but without application the knowledge is somewhat stale. Using GIPO the student can use the OpMaker tool to induce planning operators, thus both sustaining their knowledge of these areas and integrating the two together. Additionally, we have argued that GIPO helps students see the commonalities between AI with other subject areas, helping them to integrate new knowledge with other parts of the curriculum.

This year our final year undergraduates will be using the award winning GIPO III software on the AI module. Amongst other innovations, this version has an interface based on *object life histories* This forms another knowledge acquisition input into the tool (in the same way as OpMaker). It allows the student to enter a diagram recording the transitions of objects, and it automatically creates domain operators. It also allows the user to re-use pre-stored object patterns that represent typical dynamic objects. For example, Lazy Hiking domain object behaviour can be derived from a combination of generic objects we call *mobile*, *bistate* and *portable* (see (McCluskey & Simpson 2004) for details).

References

B-Core (UK) Ltd. http://www.b-core.com/.

Gennari, J. H.; Musen, M. A.; Fergerson, R. W.; Grosso, W. E.; Crubezy, M.; Eriksson, H.; Noy, N. F.; and Tu, S. W.

2003. The evolution of Protege: an environment for knowledge-based systems development. *Int. J. Hum.-Comput. Stud.* 58.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Hoffmann, J. 2000. A Heuristic for Domain Independent Planning and its Use in an Enforced Hill-climbing Algorithm. In *Proceedings of the 14th Workshop on Planning and Configuration - New Results in Planning, Scheduling and Design*.

Liu, D., and McCluskey, T. L. 2000. The OCL Language Manual, Version 1.2. Technical report, Department of Computing and Mathematical Sciences, University of Huddersfield .

McCluskey, T. L., and Porteous, J. M. 1997. Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency. *Artificial Intelligence* 95:1–65.

McCluskey, T. L., and Simpson, R. M. 2004. Knowledge Formulation for AI Planning. Proceedings of 4th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2004) Whittlebury Hall, Northamptonshire, UK, 2004. Published by Springer in the LNAI series.

McCluskey, T. L.; Richardson, N. E.; and Simpson, R. M. 2002. An Interactive Method for Inducing Operator Descriptions. In *The Sixth International Conference on Artificial Intelligence Planning Systems*.

Planform. 1999. An open environment for building planners. http://scom.hud.ac.uk/planform.

Simpson, R. M.; McCluskey, T. L.; Liu, D.; and Kitchin, D. E. 2000. Knowledge Representation in Planning: A PDDL to OCL_h Translation. In *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems*.

Simpson, R. M.; McCluskey, T. L.; Zhao, W.; Aylett, R. S.; and Doniat, C. 2001. GIPO: An Integrated Graphical Tool to support Knowledge Engineering in AI Planning. In *Proceedings of the 6th European Conference on Planning*.