Ontologies and Planners
A Statement of Interest

Robert P. Goldman
SIFT, LLC
2119 Oliver Avenue South Minneapolis, MN 55405
rpgoldman@sift.info

I am currently involved in research on planning and execution for multiple, heterogeneous automated vehicles. There are a number of challenges for this kind of planning that ontology research might assist with:

- code reuse, in this case plan library reuse in particular;
- overcoming problems in plan library development;
- more easily importing background information for planner domains;
- sharing plan information between planner and execution components; and
- decoupling planner user interface from planner and executive software components, by declarative specifications of information.

In this statement of interest, I will briefly discuss challenges and opportunities in each of these areas. I will also briefly discuss the de facto standard ontology, PDDL (Ghallab *et al.* 1998; McDermott 2000; Fox & Long 2002), and how it relates to these topics. I have tried to capture some of my own puzzlement about how to make use of the riches of knowledge representation offered by ontology researchers. Ontology research seems to hold out the promise to greatly reduce and simplify the amount of domain engineering needed to build a practical planning system. And yet....

Planning Background

My research aims to use AI planning to provide an intelligent, goal-oriented interface to complex autonomous systems, such as teams of uninhabited aerial vehicles (UAVs), spacecraft, and flexible manufacturing plants. Users should be able to give high-level tasks to such systems, phrased in terms familiar to them, and the planner should provide a bridge to lower levels of functionality. For this reason, we favor Hierarchical Task Network (HTN), or decomposition planning (Erol, Hendler, & Nau 1994b; 1994a; Currie & Tate 1991; Wilkins 1988), over first-principles planning based on precondition chaining (Weld 1999). Our planner should build plans by hierarchical decomposition that correspond to task models of human task performers, so that the plans generated and executed will meet with human approval (Goldman *et al.* 2000; Miller & Goldman 1997).

Plan Library Reuse

The state of the art in ontologies for plan library reuse is the PDDL effort (Ghallab *et al.* 1998; McDermott 2000; Fox & Long 2002), which has as its special objective support for the AIPS (now ICAPS) planning competition. By and large, the planning competition entrants are small teams of individual programmers, rather than large industrial organizations. Their efforts are aimed at demonstrating the power of planning algorithms themselves, rather than at demonstrating the utility of planners as tools to solve some problem in the real world. So, for example, we cannot expect great efforts on usability, nor can we expect these teams to be able to invest a great deal of effort into importing complex ontologies with extensive semantic baggage.

Even in its more recent extensions, PDDL is extremely easy to parse, and carries relatively little semantic baggage (e.g., it has a very simple typing scheme). It would be asking a great deal for one of these teams to incorporate a complex domain description in OWL, for example, while maintaining its full semantic content. Furthermore, little effort has been expended to make it convenient to author PDDL domain descriptions, especially to the extent that this would complicate parsing a PDDL domain file. This is almost certainly the correct approach, since the effort of designing a competition domain in PDDL is amortized over all of the participants in the competition and those researchers who use the competition domains as benchmarks. On the other hand, the parsers will not be amortized over the individual programmers, who use a diversity of algorithms, internal representations, data structures, and programming languages. Recently, Frank, Golden, and Jónsson (2003) have discussed some of the shortcomings of PDDL as a tool for practical applications of planning technology.

For applied purposes, we are less interested in direct reuse of planner domain descriptions than we are in exploiting task-oriented models developed for other purposes. At the moment, there simply aren't enough planners in use to make reuse of their domains a worthwhile project (from an application standpoint). Instead, we would like to exploit task models that capture standard operating procedures executed by people, and models of tasks that automated systems can perform.

We have been interested in leveraging existing task models, encoded in various forms of ontology, for use as the basis of a planner's operator library. One problem we have found in relating task models to plan libraries has to do with the need to chunk up different parts of the task model into methods and primitive tasks. Most planners, for example, are unable to reason well about control structures other than simple sequential composition or, possibly, conditionals. Iteration constructs are beyond the reasoning power of most planners at this time, so iteration blocks are best treated as, to some extent, atomic (possibly as pairs of start and end activities).

A second problem with importing task models is the way planners combine search control hints with other aspects of action modeling. For example, in hierarchical planners, operator and method parameters are used for long-distance information passing in the plan. Preconditions are used as much to bind variables as to provide information about method or operator applicability. All of these features suggest that a planner should provide additional information as a kind of "mixin" to existing task model ontologies. Ideally, such a mixin would make heavy use of the information already in the ontology.

Improving Planner Domain Design

Typical AI planner domain models include the following: models of the operators that can be composed into plans, and some ancillary mechanism for deduction that compensate for the very limited expressive power of the state representation. One possible advantage of the wider adoption of ontologies would be the ability to better engineer these domain descriptions, which are currently very difficult to author and maintain.

One obstacle to the wider use of ontologies in planning systems is that many of the most popular ontology frameworks are object-centered, rather than formula-centered. While these obstacles may have little theoretical import — there's little, if any, difference in expressive power between frame-based and description logic systems, and the kind of axiom and backward-chaining rules that we see in planners — they have a profound effect on the nature of domain engineering.

Along with the use of formula-based representations comes the use of unification and backward-chaining as inference methods. Further, the STRIPS assumption (Lifschitz 1990) carries with it an obligation to flatten representations to primitive state descriptors, to avoid the need to handle ramifications. This requirement is only somewhat relaxed by the addition of backward-chaining systems and hooks into special purpose external problem-solvers.

From a software engineering standpoint, the use of explicit formulas as data structures is not an advantage of contemporary planning systems. For one thing, domain descriptions for such planning systems are typically *extremely* difficult to validate and debug, and have little type discipline. Confusion about the appropriate level of detail at which system state is to be described can cause grave difficulties in modifying or extending a planner domain. Finally, the formula-centered view makes planners difficult to integrate with other software components, which are typically object-centered, and makes them difficult to integrate with the thought processes of developers of conventional software.

There are a number of practical obstacles to simply abandoning formula-centric approaches and adopting object-centered representations from ontologies. One is simple cultural inertia. More substantial issues include the reification of formulas in planners so that techniques like SAT-solving and Mixed-integer linear programming can be used to derive plans.

On the other hand, it's not a simple task to translate of object-based ontologies into formulas for use by planners. A single ontology might be interpreted very differently by different planner applications, since each planner application might use different formulas as its primitive fluents. Furthermore, aspects of the ontology that do not map onto primitive fluents impose consistency (state) constraints that may be difficult for planners to enforce. Ideally, the powerful type inference and consistency restrictions incorporated in most modern ontologies would provide valuable assistance in establishing the correctness of planner libraries. It would be particularly valuable if the existing constraints, which typically capture facts about acceptable inter-object relationships, were expanded to also capture information about units of measurement, which are still a plague to software correctness.

One issue in the use of task model ontologies is the relationship between the method decomposition relationship in HTN planners and the subsumption (isa) relationship in object-centered ontologies. Many of the ontology systems have grown out of frame-based and description logic systems. In such a framework, it is logical to express something very like the method decomposition (to-do) relationship in the planner, as an is-a relationship. For example, we might say that an `aerial-reconnaissance` isa `reconnaissance` in our ontology, but we wish our planner to carefully control when a `reconnaissance` task is reduced to an `aerial-reconnaissance` one; we don't want this to be automatically done "behind the planner's back" by a subsumption reasoner that has noticed that the agent of the `reconnaissance` task has been bound to an `air-vehicle`.

Pragmatically, we hope to see ontology engineering tools that we can adapt to our own purposes of planner domain engineering. We are particularly interested in graphical environments which could enable people with less planning expertise to engineer planning domains. Unfortunately, to date we have found that such tools (e.g., Protegé(Musen *et al.* 2000)) do not provide good support for the graphical structuring of constrained procedures or for the kinds of coreference specification needed for variable binding.

Importing domain information

One particular way that ontologies might help with the development of planner domain models would be providing information about settings for the planner. For example, in plans for large organizations, it would be very helpful if one could import details of the structures of those organizations and the resources at their disposal. Here many of the issues discussed in the previous section arise, but they seem more tractable. For example, it seems more straightforward to smash a structured, object-based representation of a set of objects into a flat set of formulas the planner can manipulate. Some form of selection of relevant items and attributes can be combined with implication closure (over inheritance relations) to make the ontology knowledge base “vivid,” (Levesque 1984) for the benefit of a formula-based, STRIPS-style planner.

Other components of the planning architecture

We assume that our planning system will be part of an overall system that also includes a user interface, and an execution monitoring component. The excution monitor will dispatch plans to lower-level control software on devices, and will monitor execution to detect the need to replan. One possible role for ontologies will be to provide data interchange between these layers. For example, a unified ontology, or cross-ontology links could coordinate the planner and executive views of the tasks in the plan. Typically the planner will be more interested in a projective, black-box model of tasks characterized by pre- and postconditions. The executive on the other hand, will need information describing how disturbances are to be handled, when a state has left the controllable region, etc. The user interface provides a different opportunity to exploit ontologies. One could easily imagine a declarative model of the interface content providing a way to permit multiple UIs to connect to the planner, whose precise content might vary depending on user experience levels, user interest, user workload, and characteristics of the interface devices.

Summary

I have presented some preliminary thoughts about how planning systems might profit from the research community’s investment in ontologies. As the available ontologies become more sophisticated and well-populated, we may hope they will become of greater and greater interest to planners, as well as web service programmers.

Disclaimer

The opinions expressed here do not reflect those of the author’s employers, funders, etc. Possibly not even those of the author.

References

Currie, K., and Tate, A. 1991. O-Plan: the open planning architecture. *Artificial Intelligence* 52:49–86.

Erol, K.; Hendler, J.; and Nau, D. S. 1994a. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1123–1128. Menlo Park, CA: AAAI Press/MIT Press.

Erol, K.; Hendler, J.; and Nau, D. S. 1994b. UMCP: A sound and complete procedure for hierarchical task network planning. In Hammond, K. J., ed., *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference*, 249–254. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Fox, M., and Long, D. 2002. PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*. `http://www.dur.ac.uk/computer.science/research/stanstuff/html/dpgpublications.html`.

Frank, J.; Golden, K.; and Jónsson, A. 2003. The Loyal Opposition Comments on Plan Domain Description Languages . In *Proceedings of ICAPS’03 Workshop on PDDL*.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—the planning domain definition language.

Goldman, R. P.; Haigh, K. Z.; Musliner, D. J.; and Pelican, M. J. S. 2000. MACBeth: A multi-agent constraint-based planner. Number WS-00-02 in AAAI Technical Report, 11–17. American Association for Artificial Intelligence.

Levesque, H. 1984. Making believers out of computers. *Artificial Intelligence* 30(1):81–108.

Lifschitz, V. 1990. On the semantics of STRIPS. In Allen, J.; Hendler, J.; and Tate, A., eds., *Readings in Planning*. Los Altos, CA: Morgan Kaufmann Publishers, Inc. 523–530. Reprinted from *Reasoning about Actions and Plans*.

McDermott, D. V. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–55.

Miller, C., and Goldman, R. P. 1997. “Tasking” interfaces; associates that know who’s the boss. In *Proceedings of the Fourth USAF/RAF/GAF Conference on Human/Electronic Crewmembers*.

Musen, M. A.; Fergerson, R. W.; Grosso, W. E.; Noy, N. F.; Crubezy, M.; and Gennari, J. H. 2000. Component-based support for building knowledge-acquisition systems. In *Conference on Intelligent Information Processing (IIP 2000) of the International Federation for Information Processing World Computer Congress (WCC 2000)*.

Weld, D. S. 1999. Recent advances in AI planning. *AI Magazine* 20(2):93–123.

Wilkins, D. 1988. *Practical Planning*. Morgan Kaufmann Publishers, Inc.