How Can a Structured Representation of Capabilities Help in Planning?

Yolanda Gil and Jim Blythe
University of Southern California / Information Sciences Institute
Marina del Rey, CA 90292
gil@isi.edu, blythe@isi.edu

Abstract

In order to support a wide range of planning-related activities, we argue that plan and action representations must move to a more expressive language for goals and capabilities than is found in most current systems. A structured representation for capabilities can make explicit a hierarchy of capabilities based on subsumption, resulting in benefits for reasoning, representing, and acquiring operators and plans. By making capabilities more easily understandable to humans, such a representation can also benefit mixed-initiative approaches. We present a structured representation of capabilities and a subsumption-based matcher for it. We then describe three existing systems that use this approach in different kinds of planning tasks and tools. We finish with a discussion of how plan generation systems can benefit from using this representation.

Introduction

An important component of plans, processes, and activities is the description of the capabilities (or goals, or tasks, or objectives) that an agent or component within a system can achieve. Typically, a capability is described as a flat predicate with a predicate name and several arguments and only limited reasoning is done about them. As part of our work within the EXPECT project at USC/ISI, we have been using a structured representation of capabilities that has been useful in several tasks and tools, including a problem solver, a plan editor, a plan evaluation critiquing tool, and an agent matchmaker. Although our work concentrates on a particular aspect of the overall representation of plans and tasks, we believe that it raises some issues that must be taken into account in the development of plan representations that can address the challenges of real-world, knowledge-intensive domains.

The main features of our approach are:

- exploit domain ontologies that can be used to reason about the objectives
- represent explicitly *task qualification parameters* that are part of the capability description in addition to data needed to achieve the capability
- support flexible matching techniques that go beyond exact match, such as subsumption and reformulation.
- it is human understandable as well as transparent to machines
- support self-organizing libraries of capabilities

In this paper, we describe our approach to represent capabilities and present three systems where we have used it that address planning-related tasks of a different nature and in different domains.

Structured Representations of Capabilities

In our approach, capabilities are represented as verb clauses using a case-grammar style of formalism (Fillmore 1968). Each capability consists of a verb, that specifies what is to be done, and a number of roles, or slots, which specify the parameters to be used in the action. The parameters use terms that are defined in a domain ontology, in our case specified in Loom (MacGregor 1987). For example, the goal of estimating the closure date of a particular transportation movement would be specified roughly as:

```
estimate OBJ closure-date
         OF transportation-movement-1
```

Here, `estimate` is the verb, and the roles are indicated in upper case. The roles are filled by concepts and instances taken from the domain ontology.

Roles can be filled in several different ways, which allows considerable flexibility in specifying a task to be performed. A role can be filled by a specific *instance:*

```
add OBJ 3 TO 5
```

which allows us to specify particular instances that are to be used in an action. A role can be filled by a *concept:*

```
compute OBJ (spec-of factorial) of 7
```

In this case, the concept `factorial` is used to specify the kind of task that is to be performed. The data required to perform the computation are specified as parameters (in this case the number 7), while these additional task parameters allow us to express what needs to be done with that data in an explicit way and are not strictly necessary to perform the computation itself. **The fact that roles can be used both to specify the parameters or objects that will be involved in a task *and* to further describe or specify the task itself is one of the key capabilities that our representation supports, providing us with a rich language for specifying goals.**

Roles can be a type of an instance, as in:

```
divide OBJ (inst-of number) BY 2
```

This expresses a generic goal that can be instantiated with any elements of that type.

Roles can also be filled by extensional sets as in:

```
find OBJ (spec-of maximum) OF (42 2 99)
```

or they can be filled by intensional sets, where the set is described by a concept:

```
find OBJ (set-of (spec-of violated-constraint))
     IN (inst-of configuration)
```

Finally, it is possible to use descriptions (which are similar to the definitions of Loom concepts) in roles:

```
estimate OBJ support-personnel
         IN (and location (exactly 0 seaports))
```

This is a goal to estimate the support personnel in a location with no seaports.

This approach provides us with a rich language for specifying behaviors. The case grammar formalism makes it relatively straightforward to paraphrase the goals into natural language, helping to make them more understandable (Swartout, Paris, & Moore 1991).

An important aspect of the systems we have implemented that use this expressive representation is how they reason with it, exploiting *subsumption* and *reformulation* as we describe next. Further details can be found in (Swartout & Gil 1995; Gil & Melz 1996). Capabilities are translated into Loom definitions, following an algorithm described in (Gil & Gonzalez 1996). For example,

```
compute OBJ (spec-of factorial) OF (5 7)
```

is translated into:

```
(defconcept compute-factorial-of-numbers
    :is (:and compute
              (:the obj (:and concept-description factorial))
              (:the of (:and number extensional-instance-set
                             (:filled-by instance-name 5)
                             (:filled-by instance-name 7)))))
```

Loom's classifier reasons about these definitions and places them in a lattice, where more general definitions subsume more specific ones. Notice that this subsumption reasoning uses the definitions of the domain terms and ontologies that are contained in the domain knowledge bases. For example, the capability to "move cargo with a vehicle" will subsume one to "move cargo with an aircraft", because according to the domain ontologies vehicle subsumes aircraft. This is illustrated in the hierarchy shown in Figure 1. As a result, the capabilities are automatically organized according to subsumption, and they can be compared based on their place in the lattice.

Subsumption matching can help find suitable capabilities when presented with a query, but in some cases no subsuming capability has been added to the knowledge base. In these cases it may be possible to fulfill the request through a *goal reformulation*. This allows a more flexible matching than is possible if one required an exact match for goals and methods. A goal reformulation is a form of divide and conquer. It transforms a goal into a set of goals that partition the original goal. For example, suppose a goal of estimating support personnel has been posted, and that the domain ontology indicates that the concept support personnel is partitioned into unloading personnel, seaport support

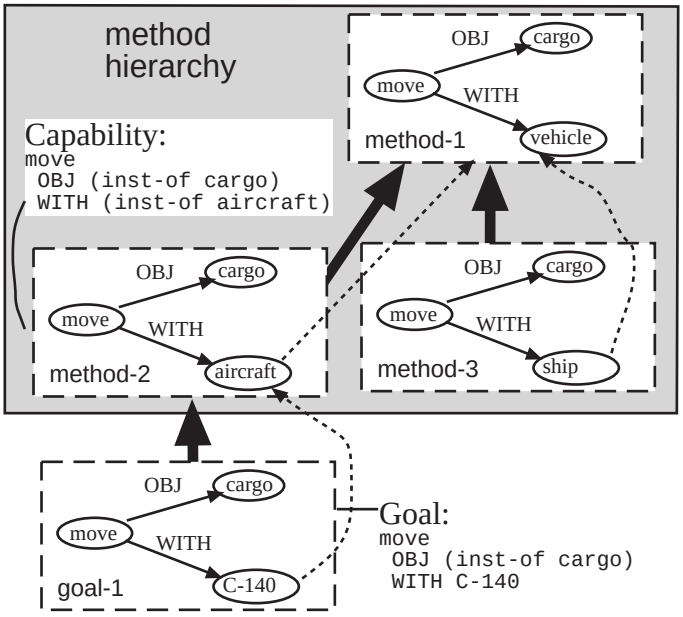

Figure 1: Translating capabilities into Loom descriptions

personnel and airport support personnel. The original goal can then be reformulated into three new goals to estimate each type of personnel in the partition. In our approach we support several types of reformulations. For more details, see (Swartout & Gil 1995).

Goal reformulations allow us to state the description of method capabilities more independently from the descriptions of the goals that are posted by other methods or by the user. The benefit is a more loose coupling between methods and tasks, i.e., between what is to be accomplished and what are possible ways to accomplish it.

We have used these structured representations of objectives in several systems that we now describe.

EXPECT

EXPECT is a reasoning system that supports acquisition of problem-solving knowledge through several different techniques. These include maintaining a dependency model of any knowledge-based system (KBS) that is built with EXPECT, scripting tools that can guide a user through a multi-step modification to a KBS and the use of background knowledge about generic tasks. Here we focus on EXPECT's representation of tasks and subtasks. Details of the overall reasoning and knowledge acquisition tools can be found in (Swartout & Gil 1995; Gil & Melz 1996; Tallis & Gil 1999; Kim & Gil 1999; Blythe & Gil 2000).

The problem-solving knowledge of an EXPECT KBS consists of a set of methods. Each method has a capability that declares what task can be achieved by the method, a body that describes how the capability is achieved and a return type that characterizes what the method produces. The method body is written in a programming language that in-

cludes basic constructs such as a conditional test and can also include other goals. These goals may be matched by the capabilities of other methods, which will be used when the method is applied, resulting in a tree of methods.

EXPECT method capability descriptions are specified in a similar way to goals, except that variables may appear in the capability descriptions. These are bound when the capability descriptions are matched with goals. The following is an example of a method and its capability:

```
((name calculate-total-cargo-weight-objects)
 (capability (calculate (obj (?w is (spec-of weight)))
    (of  (?fms is (set-of (inst-of object))))))
 (result-type (inst-of weight))
 (method (sum (obj (r-weight ?fms)))))
```

Because it uses structured representations of method capabilities, EXPECT can reason about how different methods relate to each other. This is useful for organizing method libraries (Swartout, Gil, & Valente 1999) as well as supporting the acquisition of new problem-solving methods (Kim & Gil 1999).

We mentioned earlier that the representations were developed to support natural language paraphrasing to support explanation generation. This can support knowledge acquisition (KA) from domain experts, since the paraphrase of a capability can be generated automatically from the computer representation and corresponds closely to it. As part of our work on KA tools, we have developed an English-based method editor that takes advantage of this feature, and allows users to select meaningful portions of a paraphrase and replace them by other constructs proposed by the system based on the grammar and the existing content of the knowledge base (Blythe & Gil 2000; Blythe & Ramachandran 1999). Figure 2 shows a snapshot of part of this editor.

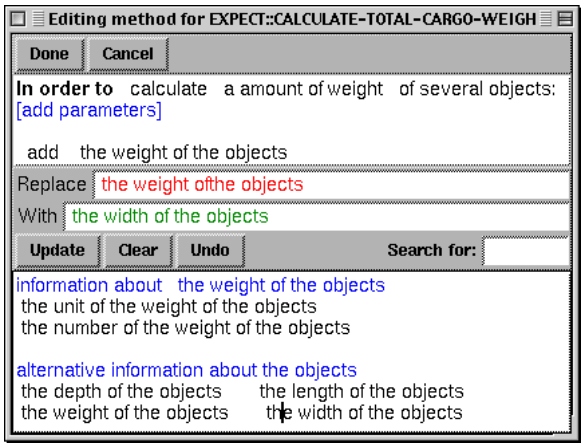

Figure 2: English-based EXPECT method editor

Using the techniques described in the previous section, EXPECT creates Loom definitions for the capabilities of all the methods that are defined in the knowledge base. EXPECT exploits the representation of goals and capabilities for matching method capabilities with the goals that arise during problem solving.

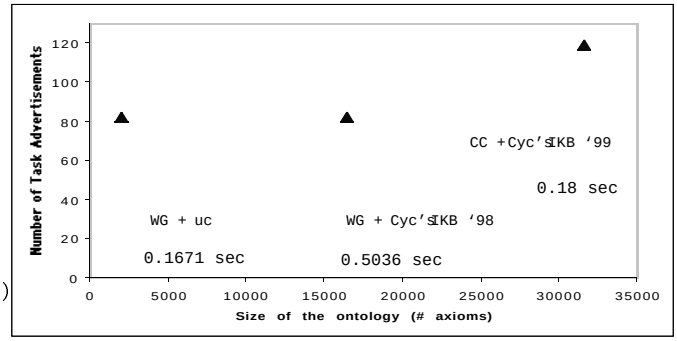

Figure 3: Performance of the Loom-based EXPECT Matcher.

```
          |
National Military Objective       Deter/defeat large-scale,
          |                       regional, military aggression
          V
  Campaign Objective              Attain air superiority
          |
          V
 Operational Objective            Suppress enemy air-sortie
          |                           generation
          V
   Operational Task               Destroy key hardened airbase-support
                                  facilities
                       OPERATION  CONCEPT: F15E squadrons with LGBS
```

Figure 4: Strategies-to-Tasks Example (from [Todd 94])

We have been using the representations and matching approach described here for almost a decade within EXPECT. Figure 3 shows the average matcher performance for several problems in different domains with quite large knowledge bases. We have found that it scales up well, due in no small part to Loom's efficiency.

Representing Objectives and Tasks for Air Campaign Planning

We have also used structured representations to represent objectives in air campaign plans. An air campaign plan specifies what air operations will be conducted in a military campaign. The strategies-to-tasks methodology (Todd 1994; Thaler 1993) provides a framework to think about air campaign plans where low-level military tasks are derived from higher level national and campaign goals. Figure 4 illustrates the hierarchy of objectives and how it links low level operational activities to higher level objectives.

The Air Campaign Planning Tool (ACPT) was the first in a series of plan editors that allow a user to define objectives using this methodology, possibly invoking automated plan generation tools to flesh out the plan at the lower levels. Plan editors are useful at the higher levels of planning to enable users to design the overall strategy and structure of the plan. However, this process is prone to error since it is mostly manual and can involve several hundreds of interdependent objectives and tasks contributed by a number of different people. We developed INSPECT (Valente *et al.* 1999), a knowledge-based system built with EXPECT that analyzes a

manually created air campaign plan and checks for common flaws, including incompleteness, problems with plan structure, and insufficient resources. Here we focus on the structured representation of objectives that was developed in this work.

In ACPT the objectives were described as strings, which lack the structure that an automated tool such as INSPECT needs in order to reason about them. We embarked on an effort to provide a formalization of air campaign objectives, which improved the editor and was also useful to other air campaign planning tools. Operational users found our structured representations very useful, because an editor would enable them to be more precise in representing objectives and because they resulted in standard statements of objectives that could be shared and understood by everyone. For example, ACPT allowed them to state an objective as "conduct operations", which is too vague, or "gain air superiority", which is imprecise because it does not specify the geographical area within the theater that is intended.

Working with Air Force experts from the "Checkmate cell" at the Pentagon, we developed a structured representation of air campaign objectives (Valente, Swartout, & Gil 1996) based on the EXPECT representation. We later developed grammars for other kinds of objectives, including force support and defensive air operations. In our initial analysis of offensive air operations, less than 30 different verbs were used. Each verb had a precise meaning that the experts would agree upon after some discussion. Once the main verbs were agreed to, we discovered that the verb itself put considerable constraints on the roles that were applicable, making a case grammar an appropriate way to structure the objectives. Several editors that make use of our grammars have been built as extensions to ACPT.

Our air campaign planning tool was successfully demonstrated in ARPI's Fourth Integrated Feasibility Demonstration (IFD-4) in 1996, was part of the DARPA Joint Force Air Component Commander (JFACC) Jumpstart Demonstration in 1997, and was integrated within ARPI's Multi-Agent Planning and Visualization System (MAPViS) demonstrated at ISTI-98 and EFX-98. The structured representations of air campaign plan objectives were used by other systems besides INSPECT in several of these demonstrations. They continue to be well received by operational users, and there is interest in using them to develop a formalization of several standardization efforts such as the Unified Joint Task List (UJTL) and the US Air Force METL. We are currently involved in a technology transition effort as part of the Joint Defense Planner (JDP).

Phosphorus: Describing and Matching Agent Capabilities

We are also using structured representations of capabilities and ontologies for a new project on agent matchmaking[1]. Multi-agent architectures typically offer matchmaking services that an agent can query to find what other agents can perform a given task. For example, a route planning agent may invoke threat detection agents in order to make a safe choice among all possible routes. Simple string matching often suffices when the agent communities are relatively small and the agents that need to issue a request know beforehand what other agents are available and how they are invoked. Most current multi-agent systems also assume that an agent can perform a few tasks (often just a single task), where the advertisements and invocations of agents are negotiated in advance by the agent designers. In large and heterogeneous communities of agents, where the agent that formulates the request would not know whether and how another agent has advertised relevant services, there is a need for more sophisticated matchmaking mechanisms. A language is required to support descriptions of agent capabilities that enable communication among agents that had no previous knowledge of each other and thus need to provide enough information about themselves to agree to joint activities. The kinds of structured representations discussed in this paper provide a richer language for advertising the capabilities of agents and would support more flexible matching algorithms.

Our group is collaborating with the Loom, TEAMCORE, and ARIADNE projects at ISI[2] to develop an agent-based environment that integrates agent organizations and human organizations. Typical tasks involve planning a schedule for a visitor and organizing off-site demonstrations and visits. Researchers, students, technical support personnel, and project assistants play different roles in each of these tasks and each person has different capabilities to offer in the organization. For example, only certain project assistants can process receipt reimbursements and only certain people within a project can give demos of it. Agent capabilities are advertised using our structured representations:

```
((capability (process (obj (spec-of reimbursement))
             (for (?r is (set-of (inst-of receipt))))))
 (agents (katya fanny tanya)))

((capability (demo (obj Phosphorus)))
 (agents (surya)))

((capability (take (obj (?v is (inst-of visitor)))
                   (to (spec-of lunch))))
 (agents (tambe knoblock minton chalupsky gil)))
```

Using ontologies, we represent information about projects, (their members, funding agencies, software, etc.), equipment, etc. The agent capabilities are translated into Loom descriptions as before. The matchmaker uses subsumption, reverse subsumption, and several kinds of reformulations to find agents relevant to a request. Figure 5 illustrates the different kinds of match.

Requests are formulated in the same language. They can be issued by a software component or by a person through an interactive interface that uses a structured editor that guides a user to formulate a request using the grammar. A snapshot of this interface is shown in Figure 6. Here, the user has issued a request for agents that can set up equipment, and through a covering reformulation the matcher has found

[1] see http://www.isi.edu/expect/projects/agents/phosphorus.

[2] see the project pages http://www.isi.edu/isd/LOOM, http://www.isi.edu/teamcore and http://www.isi.edu/ariadne .

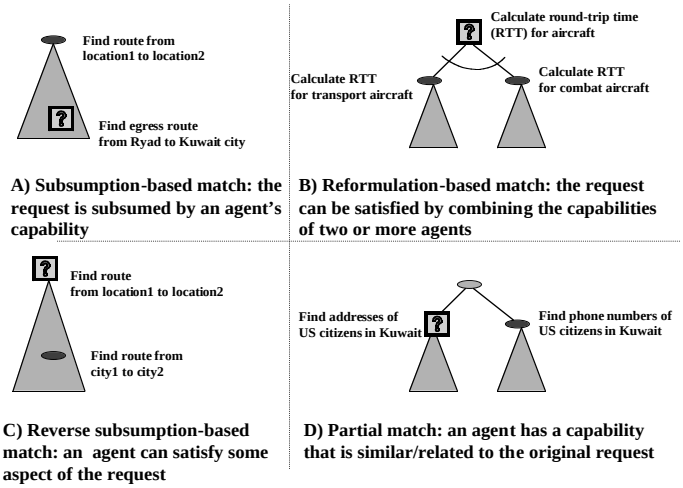

Figure 5: Agent capability matching

sets of agents that can set up different kinds of equipment. Work is under way to develop an interface that allows individual researchers to specify their capabilities so they can be advertised through the matcher.

We have integrated our matcher with the TEAMCORE architecture for rapid assembly of agent teams, which needs to reason about agent capabilities in order to support team work and configure initial teams.

This work will enable us to further investigate the use of structured representations for tasks, goals, activities, capabilities, and objectives. Currently, only a few agents are represented in the system, but as the project progresses and more agents are incorporated, we expect the system to benefit from the flexibility provided by our approach. We plan to build on work that has been done in other areas, including CSCW, agent-based systems, workflow and planning, and software reuse (Tissot & Gruninger 1999; Sycara *et al.* 1999; Fensel *et al.* 1999; Ghallab *et al.* 1998; Wilkins & Myers 1995; Tate 1998).

Discussion

We have shown how a structured objective representation approach can help with planning-related tasks such as plan evaluation and capability matching. The approach can also provide benefits for classical AI plan generation systems, as we now describe. In some cases these benefits are in modelling a domain, helping to keep the model manageable and help knowledge acquisition, and in some cases they are algorithmic, allowing the planner to make use of more powerful tools.

Almost all AI planning algorithms include a step to find suitable operators given a goal or capability specification. In subgoaling planners such as UCPOP (Penberthy & Weld 1992) and Prodigy (Minton *et al.* 1989), this usually finds all operators with an effect that can unify exactly with the goal. In HTN planners such as SIPE (Wilkins 1988) and UMCP (Erol, Nau, & Hendler 1994), all templates that can match the capability description are found. Using subsumption matching with a structured representation, one can increase the flexibility of these algorithms and this can have two distinct advantages. First, more planning knowledge can be represented in a domain-independent manner, resulting in more compact domains that are less likely to have errors and are easier to understand. Second, the matching mechanisms can be more explicit than the equivalent planner-dependent inference, leading to a clearer domain representation. These advantages can be increased if more sophisticated matching techniques are used, for example using goal reformulations.

For example, suppose that a planning domain has two operators for clearing rubble from an area, that the first calls for a generic bulldozer in its preconditions and the second calls for a particular kind of bulldozer. The second is more specific and, when it can be matched, is more reliable, so we would like the planner to choose it when it is applicable. One might use a domain-dependent select-operator preference rule to do this, but because the rule is opaque, one can only record that the reason for the choice is specificity through comments that are only read by humans. If this effect is instead achieved by explicitly preferring the operator with the more specific objective, there are at least two benefits. First, less planner code has to be written, since the operator preference is now expressed in a domain-independent manner. This makes for both a more readable and a more bug-free domain. Second, the nature of the preference rule, preferring more specific operators, is now made explicit.

Another important area where structured representations are useful is the development of verification tools for planning domains. Even if not used explicitly in the algorithm, the extra semantic information can provide error checking capabilities, *e.g.* if some operator's capability is more specific than another, but the first could not be used to solve the second's goals, or this fact is not somehow reflected in more specific preconditions.

Acknowledgements

We would like to thank present and past members of the EXPECT project who have been involved in the work described here, especially Bill Swartout, Marcelo Tallis, Andre Valente, Surya Ramachandran and Jihie Kim. We gratefully acknowledge the support from DARPA under contracts DABT63-95-C-0059 as part of the DARPA/Rome Laboratory Planning Initiative, F30602-97-1-0195 from the High Performance Knowledge Bases (HPKB) program, F30602-97-C-0118 from the Joint Forces Air Component Commander (JFACC) program, and F30602-97-C-0068 from the Control of Agent Based Systems (CoABS) program.

References

Blythe, J., and Gil, Y. 2000. Extending the role-limiting approach: Supporting end-users to acquire problem-solving knowledge. Technical report, Expect internal report.

Blythe, J., and Ramachandran, S. 1999. Knowledge acquisition using and english-based method editor. In *Proc. Twelfth Knowledge Acquisition for Knowledge-Based Systems Workshop*.

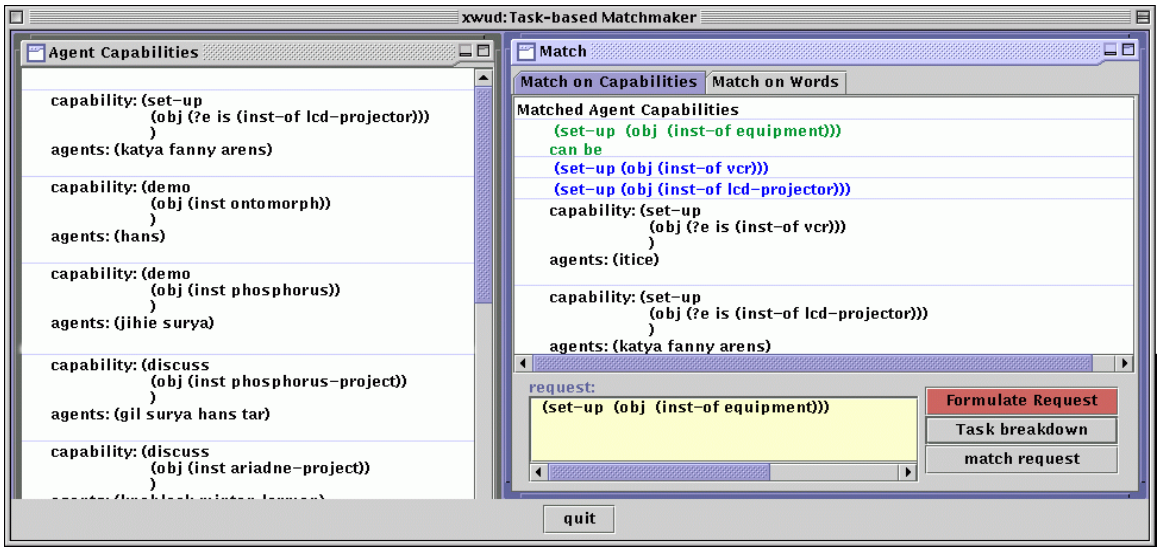

Figure 6: Interactive Formulation of a Request

Erol, K.; Nau, D. S.; and Hendler, J. 1994. Umcp: A sound and complete planning procedure for hierarchical task-network planning. In Hammond, K., ed., *Proc. Second International Conference on Artificial Intelligence Planning Systems*.

Fensel, D.; Benjamins, V. R.; Motta, E.; and Wielinga, B. 1999. Upml: A framework for knowledge system reuse. In *Proc. 16th International Joint Conference on Artificial Intelligence*.

Fillmore, C. 1968. The case for case. In Bach, E., and Harms, R. T., eds., *Universals in Linguistic Theory*. New York: Holt.

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. Available at http://www.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz.

Gil, Y., and Gonzalez, P. 1996. Subsumption-based matching: Bringing semantics to goals. In *International Workshop on Description Logics*.

Gil, Y., and Melz, E. 1996. Explicit representations of problem-solving strategies to support knowledge acquisition. In *Proc. Thirteenth National Conference on Artificial Intelligence*.

Kim, J., and Gil, Y. 1999. Deriving expectations to guide knowledge-base creation. In *Proc. Sixteenth National Conference on Artificial Intelligence*.

MacGregor, R. M. 1987. A deductive pattern matcher. In *Proc. Seventh National Conference on Artificial Intelligence*, 403–408.

Minton, S.; Carbonell, J. G.; Knoblock, C. A.; Kuokka, D. R.; Etzioni, O.; and Gil, Y. 1989. Explanation-based learning: A problem solving perspective. *Artificial Intelligence* 40:63–118.

Penberthy, J. S., and Weld, D. S. 1992. Ucpop: A sound, complete, partial order planner for adl. In *Third International Conference on Principles of Knowledge Representation and Reasoning*, 103–114.

Swartout, W. R., and Gil, Y. 1995. Expect: Explicit representations for flexible acquisition. In *Proc. Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop*.

Swartout, B.; Gil, Y.; and Valente, A. 1999. Representing capabilities of problem-solving methods. In Benjamins, V. R.; Chandrasekaran, B.; Gomez-Perez, A.; Guarino, N.; and Uschold, M., eds., *IJCAI 99 Workshop on Ontologies and Problem-Solving Methods*.

Swartout, W. R.; Paris, C. L.; and Moore, J. D. 1991. Design for explainable expert systems. *IEEE Expert* 6(3):58–64.

Sycara, K.; Lu, J.; Klush, M.; and Widoff, S. 1999. Matchmaking among heterogeneous agents in the internet. In *AAAI Spring Symposium on Intelligent Agents in Cyberspace*.

Tallis, M., and Gil, Y. 1999. Designing scripts to guide users in modifying knowledge-based systems. In *Proc. Sixteenth National Conference on Artificial Intelligence*.

Tate, A. 1998. Roots of spar - shared planning and activity representation. *Knowledge Engineering Review* 13(1):121–128.

Thaler, D. 1993. Strategies to tasks, a framework for linking means and ends. technical report, RAND Corporation.

Tissot, F., and Gruninger, M. 1999. Nist process specification language. Technical report, NIST.

Todd, D. 1994. Strategies-to-tasks baseline for usaf planning. Internal document, Strategic Planning Division, HQ United States Air Force.

Valente, A.; Blythe, J.; Gil, Y.; and Swartout, W. 1999. On the role of humans in enterprise control systems: the experience of inspect. In *Proc. of the JFACC Symposium on Advances in Enterprise Control*.

Valente, A.; Swartout, W.; and Gil, Y. 1996. A representation and library for objectives in air campaign plans. Technical report, USC – Information Sciences Institute.

Wilkins, D. E., and Myers, K. L. 1995. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation* 5(6):731–761.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann.