PHOSPHORUS: A Task-Based Agent Matchmaker

Yolanda Gil
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, U.S.A.
gil@isi.edu

Surya Ramachandran
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, U.S.A.
surya@isi.edu

ABSTRACT

PHOSPHORUS is an agent matchmaking service that exploits domain ontologies, description logic, and a highly declarative language to reason about task-related agent capabilities. PHOSPHORUS uses the EXPECT goal language to represent the tasks that agents are capable of accomplishing, as well as requests to locate agents with a required capability. PHOSPHORUS supports matching through subsumption, reverse subsumption, and several kinds of goal reformulation.

1. INTRODUCTION

Multi-agent architectures typically offer matchmaking services that can be queried to find what agents can fulfill a requested service. For example, a route planning agent may query about threat detection agents and invoke them in order to make a safe choice among all possible routes. Typically, simple string matching suffices since the agent communities are relatively small and the agents that need to issue a request can be told beforehand what other agents are available and how they have to be invoked. In large and heterogeneous communities of agents, where the agent that formulates the request would have no idea of whether and how another agent has advertised relevant services, there is a need for more sophisticated matchmaking mechanisms.

Our research on agent matchmaking draws from previous work on matching problem solving goals and methods within the EXPECT architecture [5]. In our approach, task descriptions are tightly integrated with domain ontologies, in our case specified in Loom, a knowledge representation system based on description logic. We use EXPECT's language to express agent capabilities and requests. These are automatically translated into Loom descriptions, which are then organized in a subsumption hierarchy that exploits the descriptions in the domain ontologies [3]. In contrast with other work in matchmaking using logic descriptions [4, 6], PHOSPHORUS reasons not just with the parameters but with the whole expression that specifies capabilities and requests, including any constraints on the parameters of a capability.

PHOSPHORUS is an agent matchmaking service that we are developing as part of our work in the DARPA-funded Electric Elves (E-Elves) project [2], an agent-based environment that integrates agent organizations and human organizations. Typical tasks in this environment involve planning a schedule for a visitor, setting up and attending meetings, 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.

AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.
Copyright 2001 ACM 1-58113-326-X/01/0005 ...$5.00.

2. REPRESENTING AGENT CAPABILITIES

Agent capabilities and requests are represented in the same language, which is EXPECT's language for goals and capabilities [5]. They are represented as verb clauses with typed arguments, as in a case grammar. Each argument has a name and a parameter. Argument names are typically a preposition. A parameter may be: 1) a specific instance, e.g., the Phosphorus project, represented as `Phosphorus`; 2) an abstract concept, e.g., a demonstration, represented as `(spec-of demonstration)`; 3) an instance type, e.g., vcr equipment, represented as `(inst-of vcr)`; 4) a set of instances, intensionally or extensionally specified, e.g., projects concerning agents, represented as `(set-of (inst-of agent-related-projects))` or `(PHOSPHORUS ARIADNE TEAMCORE)`; 5) a set of concepts, intensionally or extensionally specified, e.g., vcr and lcd projectors, represented as `(vcr lcd)`. Here are some examples of agent capabilities advertised using this representation:

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

"agents that can demo Phosphorus"
((capability (demo (obj Phosphorus)))
 (agents (jihie surya)))

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

Notice that an important feature of this language is that it allows a more declarative description of task qualification parameters (e.g., processing the reimbursement of a given receipt versus processing the validity of a given receipt) in addition to data parameters.

Requests are formulated in the same language. For example, a request for an agent that can reimburse a traveller for two trips can be formulated as: `(process (obj (spec-of reimbursement)) (for (Washington-trip-receipt Palo-Alto-trip-receipt)))`. Requests do not have to be instantiated, for example their arguments can query about general types of instances as in `(process (obj (spec-of reimbursement)) (for (set-of (inst-of receipt))))`.

3. MATCHING REQUESTS

Agent capabilities and requests are translated into LOOM definitions, following an algorithm described in [3]. For example, the request above for reimbursement of two trips is translated into:

```
(defconcept process-reimbursement38
  :is (:and process
        (:the obj (:and concept-description reimbursement))
        (:the of (:and number extensional-instance-set
                     (:filled-by instance-name
                        Washington-trip-receipt)
                     (:filled-by instance-name
                        Palo-Alto-trip-receipt)))))
```

LOOM's classifier is now able to reason 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. As a result, the capability to "setup equipment" will subsume one to "setup a vcr", because according to the domain ontologies equipment subsumes vcr.

Subsumption matching can help find suitable capabilities when presented with a query, but in some cases no subsuming capability is offered by any agent. PHOSPHORUS uses reverse subsumption-based matching to find agents whose capabilities are subsumed by the request, and so they can satisfy some aspect of the original request. In other cases it may be possible to fulfill the request by decomposing it expressing it in different terms, resulting in a more flexible matching service. PHOSPHORUS supports several types of reformulations. A *covering reformulation* is a form of divide and conquer. It transforms a request into a set of requests that partition the original one. If all the requests in the set are achieved, the intent of the original request is achieved. Here is an example based on the breakdown of research projects:

```
Request:
   (breakdown (discuss (obj (spec-of isi-coabs-project))))
Reply:
   (COVERING -name ARIADNE-PROJECT
                  -matches  KNOBLOCK MINTON LERMAN
             -name PHOSPHORUS-PROJECT
                  -matches  GIL SURYA HANS TAR
             -name TEAMCORE-PROJECT
                  -matches
                     COVERING
                     (-name MULTIAGENT-LEARNING-PROJECT
                            -matches  MODI TAMBE
                      -name PDA-PROJECT
                            -matches  TAMBE PYNADATH LINGDEI
                      -name ADJUSTABLE-AUTONOMY-PROJECT
                            -matches  TAMBE SCERRI PYNADATH
                      -name TEAMWORK-PROJECT
                            -matches  TAMBE PYNADATH MODI)
             -name ROSETTA-PROJECT
                  -matches  GIL HANS TAR)
```

A *set reformulation* is like a covering reformulation except that it involves a request over a set of objects which is reformulated into a set of requests over individual objects. An *input reformulation* occurs when a request is specified with a general parameter and no single agent is available at a sufficiently general level to handle the parameter. In that case, the goal can be reformulated into cases based on the subtypes of the parameter given in the ontologies. Reformulations allow stating the description of required capabilities more independently from the descriptions of the requests that are posted. The benefit is a more loosely coupling between agent capabilities and requests, i.e., between what is to be accomplished and what are possible ways to accomplish it.

Requests can be sent to PHOSPHORUS by another software agent, as shown above. PHOSPHORUS can also be used interactively through an interface that allows a person to formulate a request and see what agents have declared to have a matching capability. This interface is based on EXPECT's knowledge acquisition tools [1]. We are currently extending this interface to allow people to specify and update their capabilities.

In summary, the main features of our approach are:

- exploit domain ontologies to represent capabilities and requests
- 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

There are many interesting challenges lying ahead in this work that will enable us to further investigate the use of structured representations for representing tasks, goals, activities, capabilities, and objectives. Including people as agents presents additional challenges regarding capability representations. For example, in principle anyone has the capability to call a taxi for a visitor (and will do so if necessary), but project assistants are the preferred option. Depending on upcoming deadlines, a researcher may be capable but not willing to participate in a visitor's schedule. Many such issues need to be addressed in an agent architecture that includes people.

Acknowledgements

We would like to thank other members of the EXPECT and the Electric Elves projects for their contributions to this work. We gratefully acknowledge the support of DARPA with grant F30602-97-C-0068 as part of the DARPA Control of Agent-Based Systems program.

4. REFERENCES

[1] Blythe, J., Kim, J., Ramachandran, S., Gil, Y. An Integrated Environment for Knowledge Acquisition. In *Proc. of the International Conference on Intelligent User Interfaces (IUI-2001)*, Santa Fe, NM, January 2001.

[2] Chalupsky, H., Gil, Y., Knoblock, C., Lerman, K., Oh, J., Pynadath, D. V., Russ, T. A., Tambe, M. Electric Elves: Applying Agent Technology to Support Human Organizations. In *Proc. of the Thirteenth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-2001)*, Seattle, WA, August 2001.

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

[4] Kuokka, D. and Harada, L. Modeling Web Sources for Information Integration. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada, 1995.

[5] Swartout, B., and Gil, Y. EXPECT: Explicit representations for flexible acquisition. In *Proc. Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop*.

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