Task Planning Agents in the UMDL
extended abstract

José M. Vidal and Edmund H. Durfee
Artificial Intelligence Laboratory, University of Michigan
1101 Beal Avenue, Ann Arbor, MI 48109-2110
jmvidal@umich.edu

September 22, 1995

1 Introduction

The University of Michigan Digital Library (UMDL) project [1] is a large-scale, multidisciplinary effort to design and build a flexible, scalable infrastructure for rendering library services in a digital networked environment. In order to meet these goals, we chose to implement the library as a collection of interacting agents, each specialized to perform a particular task and all of them acting in an artificial economy. This architecture promotes modularity by distributing the responsibility of achieving tasks among the different agents. In this paper, we will be talking about one particular class of agents, the Task Planner Agents (TPA), who are responsible for decomposing tasks and forming teams of agents to accomplish them. We will also concentrate on a particular instance of a TPA that specializes in planning query tasks, and is currently under development.

2 Task Planning in the UMDL Architecture

A fundamental activity in the University of Michigan Digital Library (UMDL) is connecting people/agents that need help accomplishing their tasks with people/agents that are capable of doing those tasks. A canonical example is that library users in search of information need the help of appropriate collections that contain the information. The job of a task planning agent (TPA) is to forge these connections between agents - to help agents team up with the right agents in a large, open, and constantly evolving network of agents.

We envision that there will be many TPAs within the UMDL. They will have in common the general role of finding resources (agents with capabilities and/or content of interest), but each will possess specific knowledge and procedures for doing so, depending on the characteristics of the tasks that are in need of resources and of the agents that possess the tasks. The class of tasks that we are initially focusing on are, not surprisingly, query answering tasks. Thus, in this paper we will consider the subclass of TPAs that are specialized for query tasks, and will focus specifically on how we have designed and built an instance of a TPA for query planning.

A query planning TPA is ultimately responsible for finding one or more collections based on the needs of a user. The TPA receives the specification of the user's needs, along with parameters concerning task planning such as constraints on the number of collections to find or the effort to expend in finding them, from the User Interface Agent (UIA) which interacts directly with the user. The TPA needs to examine the query task to identify the characteristics of collections that might satisfy it, and from this could formulate a query to the Registry Agent (RA) to see whether such collections exist. The TPA can inspect the responses from the RA and might simply forward (a subset of) them back to the UIA - but, more often, what is returned does not satisfy the constraints of the query task. The TPA possesses procedural knowledge about how to handle such cases, which could in turn involve enlisting the help of other agents in the network, such as agents that can broaden/narrow topics of search (Broad System of Ordering Agents (BSOAs)) and agents that can provide synonymous terms (Thesaurus Agents (TAs)). Moreover, for a candidate collection, the TPA can contact the Collection Interface Agent (CIA) associated with it, to probe more deeply into its capabilities, content, or availability. Ultimately, the TPA formulates a candidate set of collections for the user to contact, and returns the set to the UIA.

2.1 Task Planner Agent

When developing an agent for a multiagent infrastructure such as the UMDL, there are at least three issues to consider. First we must decide on an internal architecture for the agent, one that is "natural" for the activities of the agent, allowing the modification and use of the relevant knowledge to be quick and easy. Second, we must specify the language that the agent uses for communicating with other agents. In the UMDL, all agents are required to communicate, at the highest level, in a KQML-based language [2]. This language supports the fundamental requirement of agents to be able to convey descriptions of what they can do, without which registration and team formation cannot occur. However, the agents are still free to define their own command or task languages to be used among members of an agent team. Third, we need to determine what domain knowledge is necessary for an agent, and how to represent it. As we shall see, this decision depends on and influences the first one.

The next sections explain our decisions along these lines. To keep the discussion grounded, we will consider throughout the realization of these ideas as manifested in our query planning TPA. In particular, an agent must possess languages for conducting its specialized interactions with others, and knowledge associated with its particular domain of expertise, so any instantiation of a TPA must be tailored to fill a particular niche in the UMDL. However, it is our hope and expectation that the TPA architecture that we have developed will be able to support the construction of a wide variety of TPAs.

2.1.1 TPA architecture

The basic TPA architecture has been implemented following the Beliefs, Desires and Intentions (BDI) paradigm [4], and using the University of Michigan Procedural Reasoning System (UM-PRS) platform [3]. This approach was deemed appropriate because of the type of tasks we expect a TPA to perform. Namely, a TPA has to be able to act in both an event-driven manner (e.g. when an UIA sends it a task to perform), and in a goal-driven fashion (e.g. it will always have the goals of "learning more about the other agents" and we hope to implement other learning mechanisms that will allow the TPA to modify its plans to better fit the current environment). A TPA will also need

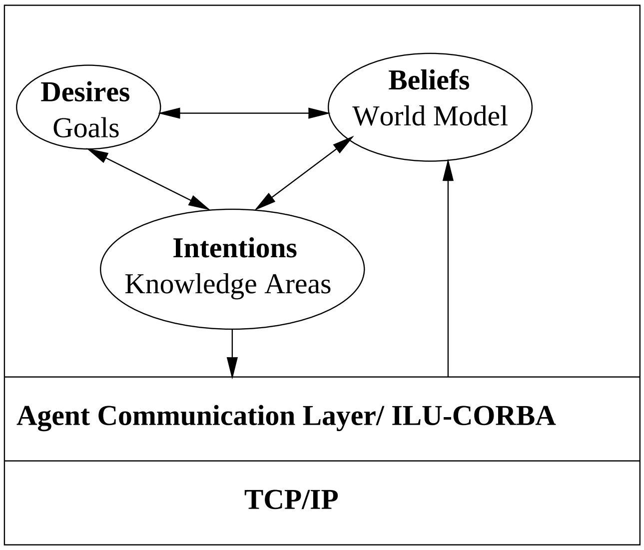

Figure 1: Diagram of the TPA architecture. Messages from other agents go through the communications layers and become part of the agent's world model. The agent's knowledge areas are responsible for generating the messages to other agents.

to dynamically switch between plans for achieving one goal versus another as when, for example, a goal is posted that has higher priority than the one the TPA is currently working on. We also wanted a TPA to be pro-active so that it could actively search for other agents that might help it carry out its task, and not just wait for them to come to him.

The levels of our TPA architecture can be seen in Figure 1. The TPA communicates asynchronously with a communication layer which is responsible for the low-level message handling. The messages that arrive from other agents pass through the communications layer and are posted into the TPA's world model (i.e. they become part of the TPA's beliefs). If the message is a new task, then the TPA will immediately generate a top-level goal of achieving it. Otherwise the message has to be a reply to some previously sent message, which means that there are already active goals in the TPA that will trigger actions based on this reply.

When using a BDI logic, one must decide between a myriad of intention semantics (i.e. what does it mean to intend something? when do intentions happen? when are they retracted? etc.). In UM-PRS, the intention semantics are summarized using a numerical priority and a selection function. We chose to use a simple selection function that basically states that, among the Knowledge Areas (KAs) that match the TPAs goals and beliefs, the one with the highest priority is always intended. When executing, the intended KA provides a blueprint for action, specifying what the TPA should do to achieve its goals within the current context of beliefs.

This simple method for prioritizing action selection can easily lead to a very messy control structure, one where the programmer loses track of the expected behavior of the agent. In order to alleviate this problem, we also chose to divide the goals along three priority levels, (low, medium, high) and two achievability types (persistent and immediate). We find that we can fit all of the type of goals we want the agent to achieve into one of the categories formed by these two dimensions, as seen here:

	Low	Medium	High
Immediate		do-task	
Persistent	garbage-collect learning		opportunistic-goals

The *do-task* goals are activated when a task comes in, and they form the basic behavior of the TPA. The *garbage-collect* and *learning* goals are done whenever the agent is not busy answering a query. The *opportunistic* goals activate KAs that can recognize a specific problem and provide a quick way of solving it, e.g. recognizing that a query will match a particular CIA and adding it to the result set. They are a good way of implementing either learned knowledge or quick shortcuts. The division of the priorities into three classes does not rule out the use of much more fine-grained priority values. In fact, we believe that continuous priorities will be very useful, as long as they stay within the limits dictated by the coarser division.

Thus, our TPA architecture supports both pro-active and reactive agent behavior, based on internalized goals and beliefs of the agent. In contrast to other "simpler" agents that could be in the UMDL—agents which are purely reactive such as sitting idle until asked to service a specific request—our TPA architecture provides general functionality for integrating persistent, adaptable agents within the UMDL. We therefore expect our architecture to be widely applicable to various task planning, and other, knowledge-based services that must be an integral part of a successful large-scale digital library.

2.1.2 TPA language

As we mentioned before, general message passing among agents in UMDL is supported by insisting that the agents follow the standard conventions of our KQML-based protocol with associated performatives. The performatives are used to tell the agent what the message is about (e.g. recommend-one, ask-about, reply, error) and provide a basic level of interaction. Inside the KQML message, however, we can include strings written in the agent's task language. This is a language specific to the agent and tailored for its domain.

The content of queries to a TPA must be in the Task Description Language (TDL), which is currently just a list of argument-value pairs specifying different parameters of the task. For example, a task to find collections for a user to investigate can be characterized by features of the query (e.g. topic(s), dates, keywords, level of presentation) and features of the search process (e.g. the maximum time that can be spent doing it, number of collections to return). At its simplest, a query planning TPA must be able to forward the features of the query to the Registry Agent, and return the results back. Our query planning TPA is envisaged to do much more, however, and already has added functionality such as:

- The TDL has a `:max-time` slot which tells the TPA not to exceed a certain delay, or absolute time, when answering the query.
- The TDL also has a `:min-hits` and `:max-hits` slots which it uses to determine the quality of the result that it has generated for the given query. If the quality is lower than a reasonable value, then the TPA will try to improve it, if it can.

2.1.3 TPA domain knowledge

Implementing the domain knowledge of the TPA raises a lot of questions. First of all, it is clear that the choice of the agents' architecture is closely tied to the the choice of a domain knowledge representation. We believe that our choice of a BDI architecture will allow us to use various domain knowledge architectures (e.g. the agent can spend its idle time running learning algorithms on cached data while still being reactive to incoming messages, the agent can also create KAs that are opportunistic and take over when appropriate), and still have the agent behave in a rational way (i.e. act when appropriate). But, there are still many questions about the nature of the domain knowledge that we are working on: How specialized or general should the agent's knowledge be? What are the appropriate areas for specialization? How adaptive should the agent be? What kind of knowledge and learning mechanism (e.g. reinforcement learning, genetic algorithms, Bayesian networks, etc.) will work best? We do not expect to answer all of these question (in fact, we expect that the economic market will give the last word on what works and what doesn't), but we hope to investigate some of them.

Currently the domain knowledge is implemented directly into the agent's KAs. We assume that a query is answered by the achievement of successive goals; that is, we wrote a plan for achieving the task of answering a query within the given task parameters. A subset of the goal structure formed by our TPA is shown in Figure 2. The different goals and associated KAs have been gathered from continuing meetings with librarians and other Information and Library Sciences faculty. From these meetings, it has become clear that the knowledge used by a librarian to guide a student to the appropriate collections is sometimes elusive. This implies that, even though there does seem to be a process or plan that can reliably map queries to the appropriate collections, the execution of this plan often requires the use of common-sense knowledge or of shallow domain knowledge. We believe that the amount of common-sense knowledge required for our limited domain is small enough that it can be encoded in the TPA's KAs and, we are hoping that the other "specialist" agents will provide the needed domain knowledge.

3 Summary

In this paper, we have outlined in broad brushstrokes the philosophy behind our multiagent infrastructure for large-scale digital libraries, with an emphasis on the importance of having agents within that infrastructure that have knowledge and strategies for finding resources and forming agent teams. We have described task planning agents as a class of such agents, and have described our current efforts in developing the architecture, language, and domain knowledge for such agents - and in particular, for task planning agents that support query planning. While there are many unresolved issues in our design, we believe that we have a good starting point for developing such

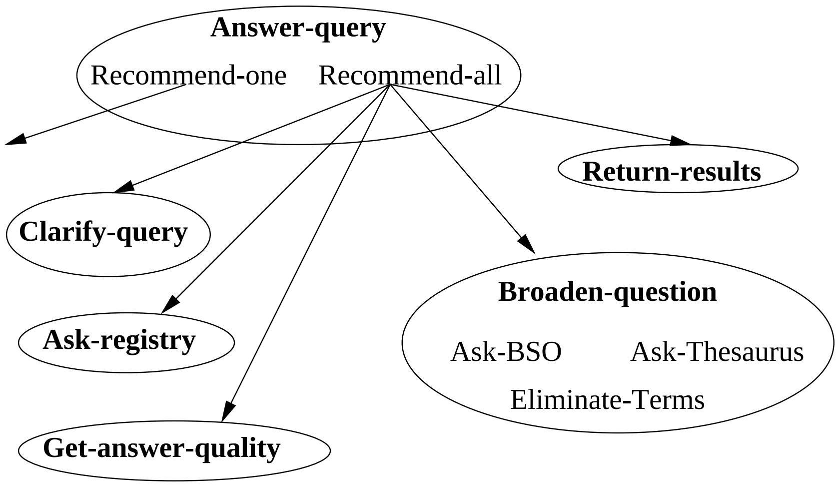

Figure 2: A subset of the goal structure that is formed when a request to recommend-all comes in. The arrows represent the goal/sub-goal relationships. The first names in the nodes are the goals, while the others are the names of KAs that can achieve these goals.

agents, and we have already begun to incorporate and evaluate prototype versions of our TPAs within the UMDL system.

References

[1] William P. Birmingham, K. M. Drabenstott, , C. O. Frost, A. J. Warner, and K. Willis. The university of michigan digital library: This is not your father's library. In *Proceedings of Digital Libraries*, pages 53–60, June 1994. http://http2.sils.umich.edu/UMDL/HomePage.html .

[2] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Managment.* ACM Press, 1994.

[3] Jaeho Lee, Marcus J. Huber, Edmund H. Durfee, and Patrick G. Kenny. UM-PRS: An implementation of the procedural reasoning system for multirobot applications. In *Proceeding of the AIAA/NASA Conferencce on*

Intelligent Robotics in Field, Factory Service and Space, pages 842–859, March 1994.

[4] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 312–319, 1995.