Task Planning Agents in the UMDL

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

February 17, 1996

1 Introduction

The University of Michigan Digital Library (UMDL) project [2] 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, and flexibility by allowing the formation of different teams of agents to accomplish specific tasks. This, in turn, requires that some of the agents have the ability to exploit the flexibility of the architecture and the properties of other agents. In this paper, we will be talking about such a class of agents, the Task Planner Agents (TPAs), 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 of a library user in search of information who needs to contact the appropriate collections which 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

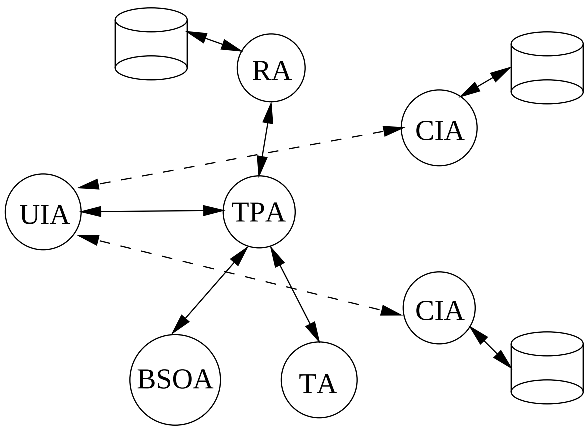

Figure 1: Schematic view of the UMDL architecture showing the agents in it and the communications between them. The solid lines represent the first part of the search, where the UIA looks for applicable collections. The dashed lines are the second part, when the UIA queries the collections that were found. Future versions will have several instantiations of each agent.

content of interest), but each will possess specific knowledge and procedures for doing so, depending both on the characteristics of the tasks that are in need of resources, and on the resources available for doing the search (e.g. monetary funds, user patience, etc.). 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; in order to do so, it communicates with many agents in the UMDL, as seen in Figure 1. 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 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 could contact the Collection Interface Agent (CIA) associated with it, to probe more deeply into its capabilities, content, price, or availability. Ultimately, the TPA formulates a candidate set of collections for the user to contact, and returns the set to the UIA. The UIA might then proceed to contact some of these CIAs directly and ask for the particular documents, before returning the final answer to the user.

2.1 Task Planner Agent

When developing an agent for a multi-agent 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 [4]. 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 knowledge the agent will need. We have divided this knowledge into two categories, namely "world knowledge" and "domain knowledge". World knowledge includes everything the agent needs to know about the universe it inhabits — in our case, the UMDL architecture. Domain knowledge is expert knowledge that the agent will use to achieve the particular tasks it claims to be able to perform. As we shall see, determining what knowledge is needed and how to represent it influences our choice of architecture.

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 [10], and using the University of Michigan Procedural Reasoning System (UM-PRS) platform [7]. 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 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.

UM-PRS also allows the programmer to write simple sequential procedures to achieve the necessary goals, while the UM-PRS scheduler worries about when to execute them, when to preempt them for another higher priority event, and when to reinstate them. A good example of this is the *get-reply* goal. A common occurrence is for the TPA to send a message to another agent and then wait for a reply for some amount of time before timing out. In UM-PRS, the programmer can instruct the system to achieve the goals *send-message* and *get-reply* in sequence, as part of, say, some higher-level goal. The system will try to achieve them as such but, since the second one requires that the reply message has actually arrived, the system will block on this goal until the message does arrive. During this time, the system will not sit idly waiting but, if it has other goals that have been blocked or are merely waiting to be executed, (e.g. another message comes in, there are messages that need to be deleted from memory, etc.) it will change its focus of attention and start (or continue) to try to achieve them. Similarly, it will later return and handle the reply when it finally arrives. In the same vein, we could also send several messages and work on local tasks while waiting for the replies , automatically handling each reply in the order in which it comes in.

The levels of our TPA architecture can be seen in Figure 2. 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. In the event that the message is neither a new task nor an expected reply, it is marked for deletion.

When using the BDI paradigm, 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

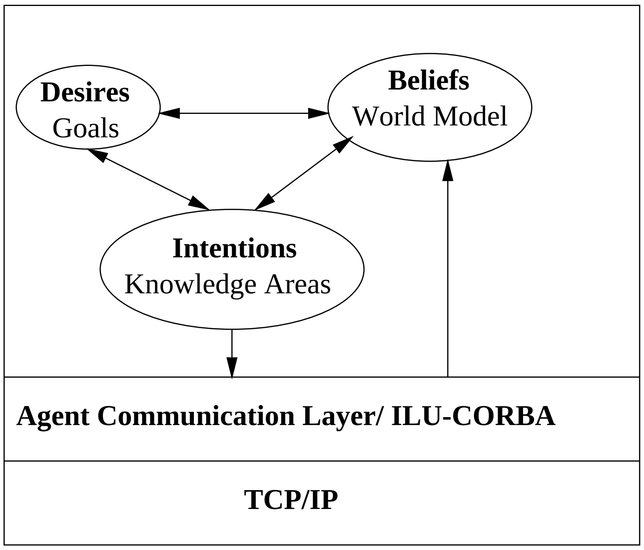

Figure 2: Diagram of the TPA architecture. Messages from other agents go through the communications layers and become part of the agent's world model. The communication layer is implemented using the Inter-Language Unification (ILU) system [1], which is compatible with the Common Object Request Broker Architecture (CORBA). The agent's knowledge areas are responsible for generating the messages to other agents.

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 (as can also occur in blackboard architectures, which have a similar control paradigm). 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). Persistent goals are those that are never achieved. For example, the *garbage-collect* goal finishes executing when there are no more messages left to collect. However, the goal is never achieved because there is always the possibility that new messages will arrive. Immediate goals, on the other hand, will eventually be, either achieved or will fail. 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	find-other-agents	do-task	
Persistent	garbage-collect learning		opportunistic-goals dispatch-new-msgs

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 carrying out a task. Sometimes, while trying to achieve a task, the TPA might realize that the agents it thought existed to carry out a certain goal are no longer available or reachable. In these cases, the TPA will set up a *find-other-agents* goal that will, when there is time, try to find the desired agents (i.e. ask the registry for the specific agent types). As we mentioned, the *dispatch-new-msgs* goal deals with new incoming messages, it determines their priority and instantiates other *do-task* goals if necessary. 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 and sit 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 earlier, general message passing among agents in the 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 a list of argument-value pairs specifying different parameters of the task, plus a query given in a (currently) simplified version of SQL. The query specifies the type of content that the user is interested in and the specific information about the collections that he wishes to receive. 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 to the sender. Our query planning TPA is envisaged to do much more, however, and already has added functionality such as:

- The TDL has a `:max-delay` slot which tells the TPA not to exceed a certain time delay when answering the query. If it fails to accomplish the task in the specified time it returns a *sorry* message along with an explanation and whatever results it has managed to accumulate.
- 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 using the techniques mentioned in Section 2.1.3.

We are currently trying to expand the language to include more information about the user. The plan is for the UIA to gather a profile of the user either by having her fill out a questionnaire, by monitoring her behavior, or by using defaults. This profile will include the user's preferences with respect to the type of medium and format she prefers (e.g. papers in html format), the conceptual level at which she usually searches (e.g. introductory, scholarly), what level she is at (e.g. juvenile, student, adult), etc. The TPA will use this information to make better decisions when trying to achieve the user's task.

2.1.3 TPA 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 repertoire of plans for achieving the task of answering a query within various sets of task parameters. A subset of the goal structure formed by our TPA is shown in Figure 3. 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.

Our basic strategy consists of iterating over a three step loop. The first step takes the current query and delivers it to the RA (in the future it might be delivered, instead, to specialized domain agents). The second step is to ascertain the quality of the results gathered so far. This is done by counting the number of CIA names that have been gathered so far, along with any relevant information about them, and using a function to determine how well they match the task parameters. If the quality is deemed high enough then the results are returned. Otherwise, we proceed to the third step where the query is modified in an effort to produce a new query that will produce the desired or missing results. In order to do this modification, a few simple techniques have been implemented.

A common technique is to find a broader or related topic to search to the one(s) given. For example, if the query was about "clouds", this modification technique might also add "water vapor", or "storms", or both. The semantic knowledge of what terms are related, and which terms are broader or narrower than others, is given by the BSOA. It is important to note that these relations

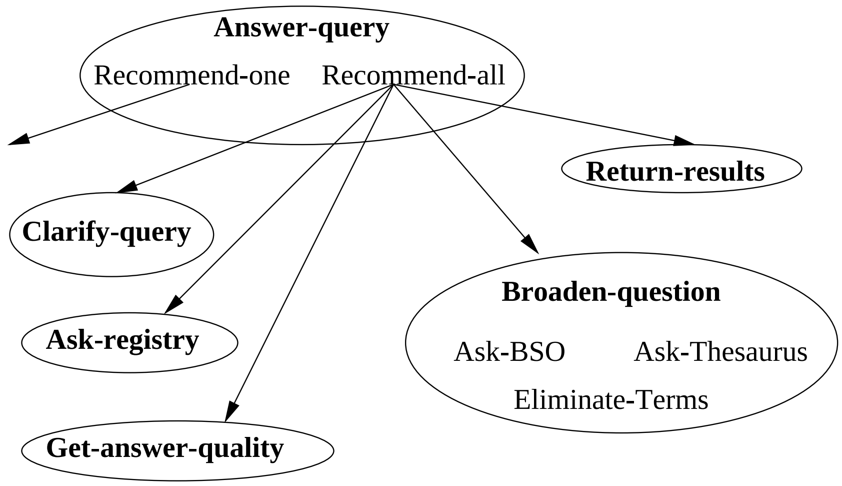

Figure 3: A subset of the goal structure that is formed when a request to recommend-all comes in. The arrows represent the goal/sub-goal relationships. The first names in the nodes are the goals, while the others are the names of KAs that can achieve these goals.

can only be trusted within a specific knowledge domain. That is, our BSOA only deals with themes related to "space and earth sciences." If the user was interested in literature and asked about "clouds" then a BSOA agent specializing in literature might give related terms such as "obscures," "vague." The TPA will need to possess enough domain knowledge and profile information on the user to determine the general area to which the query probably refers.

Another similar technique is to ask the Thesaurus Agent for synonyms or words related to the ones in the query. The TA can also be specialized in some subject area, and its content is considered to be much more extensive and flexible than the BSOA. For example, the TA contains terms like "cirrus clouds" and "dust clouds" while the BSOA is limited to much more general terms. This technique can be used when the UIA provides terms that are too specific to be subjects but are likely to be words in the actual documents (i.e. keywords). Another way of achieving this same effect is by asking the collections themselves if they contain the specified words. However, this is only feasible if we have trimmed the number of applicable collections.

We can also apply similar techniques to other fields in the query such as "conceptual level," "audience," "language," etc. Eventually, these techniques will be combined to work better together. For example, if a high-school student asks about the moon then the process will first go through general references and introductory texts (or collections of such) about the moon and planetary bodies in general, and only after this will it try more advanced or scholarly articles on the moon and astrophysics.

Finally, if all else fails, we can eliminate terms from the specified query. Note that we are *not* allowed to eliminate terms from other parts of the task description, only from the part of it that contains the query.

The TPA's **world knowledge** consists of the knowledge the agent needs for achieving its goals of effectively communicating with the other agents, forming deals and, eventually, turning a profit. Specifically, the TPA needs to know the following:

- The mapping between some, if not all, of the possible agent types to their associated capabilities and vice-versa. The more mappings it knows, the more effective it can be.
- The type of deals that can be established. These will probably cover a spectrum from one-time deals to long term contracts for service at a fixed price.
- Basic knowledge about the world the agent inhabits (e.g. messages can take arbitrarily long to arrive, agents can go down, etc).

The first item is a first step towards the modeling of other agents. The TPA might find that simply having a mapping between agent types and capabilities is not enough and that it needs more detailed models of other agents (e.g. a TA might be designed primarily for grade-school students, a BSO agent might always take a long time to answer, a TPA might always deliver results of poor quality). For these cases the TPA will need to expand its knowledge base to include models of the other agents. These models should include, the agents' specialty, how long they took to do their task, the quality of their results and the price they charged for their services. The TPA will probably start with some basic generic models which it will improve and specify using its observations. Of course, if we assume that the other agents are just as complex, then we must also assume that they are building similar models. This situation leads to a potentially infinite recursive modeling. The TPA will need to be able establish tradeoffs between modeling and taking action. It must determine who, when, and how deep to model, and when to use this knowledge.

3 Implementation

There is a working prototype of the UMDL and of a TPA. The current UMDL contains several hundred collections. There is a BSOA that uses a locally developed subject hierarchy specializing in earth and space sciences, and a TA that uses data from the NASA thesaurus. The implementation of the TPA basically follows the description given in this paper except for the economic or deal-making aspects of it, which have not yet been implemented. The TPA is capable of receiving a query from the UIA, consulting the RA, BSOA, and TA, and returning a set of appropriate collections based on the information it gathered and on its local knowledge. It can also talk to several UIAs at the same time, interleaving the different conversations and handling messages as they arrive. The current working prototype can be accessed at [2].

This implementation served to verify the effectiveness of our agent architecture and the use of UM-PRS as a basic platform for developing software agents. As we described before, the query answering plans are rather simple, and the semantic knowledge sources (i.e. the BSOA and TA) are also simplistic, of narrow scope and limited in number. Also, the collection topics are currently registered using only a limited subset of the BSO language. All this implies that the search carried out by the TPA is slower and less accurate than we would want. We are currently working on increasing the TPA's search knowledge, while other members of the UMDL are working on improving and adding more agents like the BSOA and TA, and on expanding the descriptions given to the collections. Given these improvements, we hope that our next version will be faster and much more tailored to the specific user's needs. We also plan to include some basic pricing/buying strategies and more complex contracting/team-forming capabilities into the TPA and other agents. These changes should start to move the UMDL towards becoming a real economic society.

4 Summary

In this paper, we have outlined in broad brushstrokes the philosophy behind our multi-agent 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, world and domain knowledge, and language 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 agents, and we have already begun to incorporate and evaluate prototype versions of our TPAs within the UMDL system.

Acknowledgments

We would like to thank the whole UMDL team. Special thanks to the Conspectus Search and Retrieval group (Amy Warner, Elke Rudensteiner, Anisoara Nica, Steve Kirk, Spencer Thomas, Fritz Freiheit, Eric Glover, Edmund Durfee, and José Vidal) for coming up with the techniques that the TPA uses for answering a query. This work was supported by the NSF/ARPA/NASA Digital

Library initiative.

References

[1] Judy Anderson, Antony Courtney, Doug Cutting, Bill Janssen, Dan Larner, Denis Severson, Bridget Spitznagel, Mike Spreitzer, Mark Stefik, and Farrell Wymore. Inter-language unification. ftp://ftp.parc.xerox.com/pub/ilu/ilu.html .

[2] 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 .

[3] Edmund H. Durfee, Piotr J. Gmytrasiewicz, and Jeffrey S. Rosenschein. The utility of embedded communications and the emergence of protocols. In *Proceedings of the 13th International Distributed Artificial Intelligence Workshop*, 1994.

[4] 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.

[5] Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. Integrating and accessing heterogeneous informatino sources in tSIMMIS. In *Information Gathering from Heterogeneous Distributed Environments — AAAI Spring Symposium*, 1995.

[6] Michael P. Georgeff and Anand S. Rao. The semantics of intention maintenance for rational agents. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[7] 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.

[8] Tracy. Mullen and Michael. P. Wellman. Some issues in the design of market-oriented agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents Volume II — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996.

[9] Tim Oates, M. V. Nagendra Prasad, Victor R. Lesser, and Keith Decker. A distributed problem solving approach to cooperative information gathering. In *Information Gathering from Heterogeneous Distributed Environments — AAAI Spring Symposium*, 1995.

[10] 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.

[11] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[12] José M. Vidal and Edmund H. Durfee. Recursive agent modeling using limited rationality. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 125–132, 1995.

[13] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–48, March 1992.