, , 1–24 ()

Intelligent Adaptive Information Agents

KEITH S. DECKER decker@cis.udel.edu
Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716

KATIA SYCARA sycara@cs.cmu.edu
The Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA 15213

Editor: Ibrahim Imam and Larry Kerschberg

Abstract. Adaptation in open, multi-agent information gathering systems is important for several reasons. These reasons include the inability to accurately predict future problem-solving workloads, future changes in existing information requests, future failures and additions of agents and data supply resources, and other future task environment characteristic changes that require system reorganization. We have developed a multi-agent distributed system infrastructure, RETSINA (REusable Task Structure-based Intelligent Network Agents) that handles adaptation in an open Internet environment. Adaptation occurs both at the individual agent level as well as at the overall agent organization level. The RETSINA system has three types of agents. *Interface agents* interact with the user receiving user specifications and delivering results. They acquire, model, and utilize user preferences to guide system coordination in support of the user's tasks. *Task agents* help users perform tasks by formulating problem solving plans and carrying out these plans through querying and exchanging information with other software agents. *Information agents* provide intelligent access to a heterogeneous collection of information sources. In this paper, we concentrate on the adaptive architecture of the information agents. We use as the domain of application WARREN, a multi-agent financial portfolio management system that we have implemented within the RETSINA framework.

Keywords: Multi-Agent Systems, Intelligent Agents, Distributed AI, Agent Architectures, Information Gathering

1. Introduction

Due to the current nature of the World Wide Web, information is becoming increasingly difficult for a person or machine system to collect, filter, evaluate, and use in problem solving. The notion of Intelligent Software Agents (e.g., [1, 55, 32, 49, 35, 36, 18]), has been proposed to address this challenge. Although a precise definition of an intelligent agent is still forthcoming, the current working notion is that Intelligent Software Agents are programs that act on behalf of their human users in order to perform laborious information gathering tasks, such as locating and accessing information from various on-line information sources, resolving inconsistencies in the retrieved information, filtering away irrelevant or unwanted information, integrating information from heterogeneous information sources and adapting over time to their human users' information needs and the shape of the Infosphere.

We have developed a multi-agent system infrastructure, called RETSINA [48] where multi-agents compartmentalize specialized task knowledge and coordinate among themselves to gather and filter information in response to user-initiated problem solving. We have developed applications in different domains using the RETSINA framework. Of particular interest for this paper is WARREN, a multi-agent system for financial portfolio management. In RETSINA, there are three types of agents: *interface agents* tied closely to an individual human's goals, *task agents* involved in the processes associated with arbitrary problem-solving tasks, and *information agents* that are closely tied to a source or sources of data. Typically, a single information agent will serve the information needs of many other agents (humans or intelligent software agents). An information agent is also quite different from a typical World Wide Web (WWW) service that provides data to multiple users. Besides the obvious interface differences, an information agent can reason about the way it will handle external requests and the order in which it will carry them out (WWW services are typically blindly concurrent). Moreover, information agents not only perform information gathering in response to queries but also can carry out long-term interactions that involve *monitoring* the Infosphere for particular conditions, as well as information updating. The agents communicate through message passing using the KQML [19] communication language. Since RETSINA is an *open agent society* where agents may join (e.g. new agents are put on the Internet every day), or leave (e.g. agents fail intermittently) at any time, the agents utilize *middle agents*, e.g. matchmakers and brokers to find each other.

The RETSINA agents have been designed expressly to handle adaptation. Adaptation is behavior of an agent in response to unexpected (i.e., low probability) events in a dynamic environment. Examples of unexpected events include the unscheduled failure of an agent, an agent's computational platform, or its underlying information sources. Examples of dynamic environments include the occurrence of events that are expected but it is not known *when* (e.g., an information agent may reasonably expect to become at some point overloaded with information requests), events whose importance fluctuates widely (e.g., price information on a stock is much more important while a transaction is in progress, and even more so if certain types of news become available), the appearance of new information sources and agents, and finally underlying environmental uncertainty (e.g., not knowing beforehand precisely how long it will take to answer a particular query).

The RETSINA agents handle adaptation at several different levels, from the high-level multi-agent organization down to the monitoring of execution of individual actions. Such multi-faceted adaptation is needed especially in *open* information environments, where agents and information sources may appear or disappear, and where communication links may fail. Use of middle agents helps the RETSINA agents organize themselves in a flexible and adaptive fashion that makes the overall system robust to failures of information sources and other agents. In addition, the agents are able to adapt at the individual level. This adaptation is facilitated by the reusable underlying individual agent architecture, comprised of a number of modules that operate asynchronously to handle the various agent tasks. These modules are: communicator, planner, scheduler, and execution monitor. Besides capabilities for handling its particular tasks, an agent is endowed with *self monitoring* behavior that allows it to adapt to fluctuations in performance through *cloning*.

In this paper, we concentrate on information agents and describe their implemented adaptation capabilities both at the individual agent level as well as at the organizational level. We will use the functioning of information agents in the WARREN application to illustrate our points. We will present an overview of the individual agent architecture, agent organization scheme and give specific examples and experimental results of adaptation. In the next section we will discuss the individual architecture of these agents. In section 2, we will discuss the reusable components of an individual agent. In sections 3 and 4, we will present the internal adaptation schemes for some of these agent components. Adaptation between agents at the organizational level will be presented in section 5 and conclusions and future work in section 6.

2. Agent Architecture

Most of our work in the information gathering domain to date has been centered on the most basic type of intelligent agent: the *information* agent, which is tied closely to a single data source. An information agent accesses information from Web-based information sources in response to a request from another agent or a human. The dominant domain level behaviors of an information agent are: retrieving information from external information sources in response to one shot queries (e.g. "retrieve the current price of IBM stock"); requests for periodic information (e.g. "give me the price of IBM every 30 minutes"); monitoring external information sources for the occurrence of given information patterns, called change-monitoring requests, (e.g. "notify me when IBM's price increases by 10% over $80"). Information originates from external sources. Because an information agent does not have control over these external information sources, it must extract, possibly integrate, and store relevant pieces of information in a database local to the agent. The agent's information processing mechanisms then process the information in the local database to service information requests received from other agents or human users. Other simple behaviors that are used by all information agents include advertising their capabilities to middle agents, managing and rebuilding the local database when necessary, and polling for KQML messages from other agents.

A large amount of previous work has concentrated on how to access and integrate information from heterogeneous databases (e.g. relational databases) containing structured information. Many problems arise due to semantic schema conflicts and ontological mismatches [26]. The work described here is focussing on WWW-based information where most of the information is unstructured and where an information agent does not have access to the contents of the whole information source at once but only through an external query interface. Projects such as Carnot [5] have shown that different types of traditional databases (e.g. relational, object-oriented) can be mapped via articulation axioms to a shared global context language (in Carnot's case, based on CYC). Such an approach is compatible with ours and can be used to add traditional structured database external sources to our basic information agents. The ontological mismatch problem (e.g. [24]) is still a difficult one and is outside the scope of this paper.

An information agent's reusable behaviors are facilitated by its reusable agent architecture, i.e. the domain-independent abstraction of the local database schema, and a set of generic software components for knowledge representation, agent control, and interaction with other agents. The generic software components are common to all agents, from the simple information agents to more complex multi-source information agents, task agents, and interface agents. The design of useful basic agent behaviors for all types of agents rests on a deeper specification of agents themselves, and is embodied in an agent architecture. Our current agent architecture is part of the RETSINA (REusable, Task Structure-based Intelligent Network Agents) approach [48], partly based on earlier work on the DECAF architecture [7, 40].

In the RETSINA approach, an agent's control architecture consists of the following modules: communicator, planner, scheduler and execution monitor. The architecture presented here is consistent with BDI-style agent theory [3, 42]. These generic software components are common to *all* classes of agents, not just information agents. This differentiates our approach from approaches such as SIMS [28] that are focused on providing only multi-source information agent behaviors. The focus of our architecture is the ability to interleave computational actions from many concurrent behaviors, to interleave planning and execution, to schedule periodic activities and activities that have deadlines, and to handle behaviors that are strung out in time and where the next step may be externally, asynchronously enabled.[1]

In developing the discrete control architecture of our software agents, we rely on a shared representation of the structure of the tasks an agent is carrying out. The planner/plan retriever creates these structures for objectives determined by the communicator; the scheduler actively manages the agenda of executable actions in the structures; the execution monitor takes care of individual action executions. The task structure representation we use here has features derived from earlier hierarchical task network planning work, as well as task structure representations such as TCA/TCX [45] and TÆMS [14].

2.1. Agent Control Components

The control process for information agents includes steps for communicating agent goals or objectives, planning to achieve local or non-local objectives, scheduling the actions within these plans, and actually carrying out these actions (see Figure 1). In addition, the agent has a shutdown and an initialization process. The initialization process, executed by the agent at startup bootstraps the agent by giving it initial objectives, i.e. to poll for messages from other agents and to advertise its capabilities. The shutdown process is executed when the agent either chooses to terminate or receives an error signal that cannot be handled. The shutdown process assures that messages are sent to (1) the agents that have outstanding information requests with this agent, and (2) to the middle agents with whom the agent has advertised. These messages inform of the agent's impending unavailability (see Section 3).

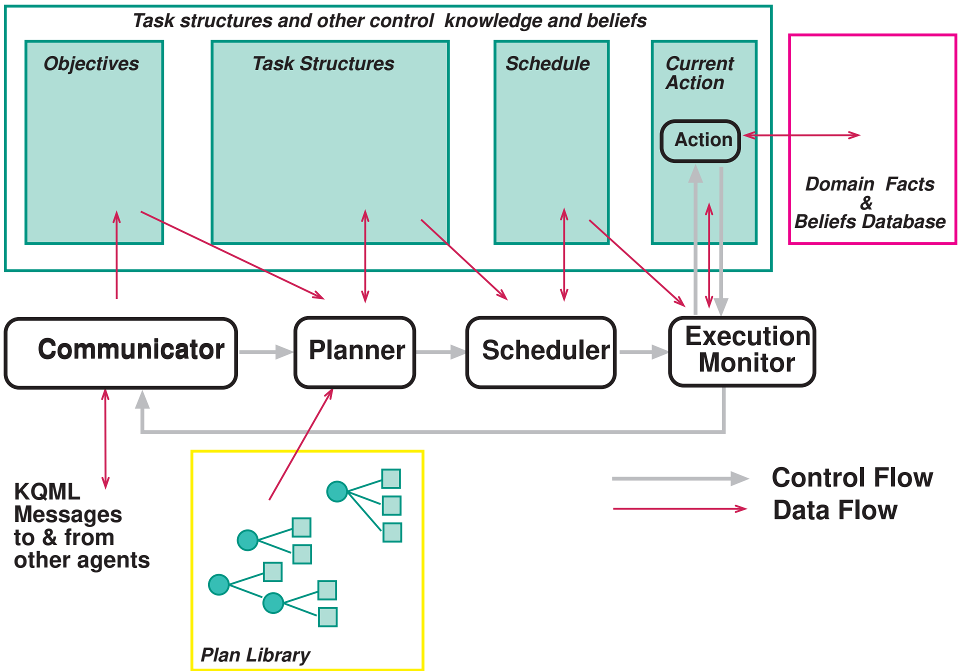

Figure 1. Overall view of data and control flow in an information agent.

2.2. Communication

An agent communicates with others through messages using KQML (Knowledge Query and Manipulation Language) [19, 31]. KQML is an emerging agent communication language standard that focuses on an extensible set of "performatives", based on speech acts [44], such as TELL, ASK, and ADVERTISE. Each communicative act consists of an well-known *outer* language that includes the performative and several message "parameters" such as :SENDER, :RECEIVER, :REPLY-WITH, :IN-REPLY-TO, :LANGUAGE, :ONTOLOGY, and :CONTENT. The *inner* or *content* language is specified by the :LANGUAGE and :ONTOLOGY and can thus be domain-specific. KQML currently only has informal semantics for most of its performatives and lacks commissives [4], however, these problems are being addressed [30, 47].

For example, a requestor agent makes information requests via ASK to an information provider agent using the :REPLY-WITH parameter as a conversation marker. Eventually, it receives replies via REPLY or SORRY messages with the a matching :IN-REPLY-TO parameter. Receipt of requests from a requestor engender internal goals to be fulfilled by the recipient information agent. There are three types of information seeking goals that an information agent receives:

1. Answering a one-shot query about the associated database.
2. Setting up a periodic query on the database, that will be run repeatedly, and the results sent to the requester each time (e.g., "tell me the price of IBM every 30 minutes").
3. Monitoring a database for a change in a record, or the addition of a new record (e.g., "tell me if the price of IBM drops below $80 within 15 minutes of its occurrence").

In addition, message communication is used by an agent to *advertise* its capabilities to middle agents when it first joins the agent society, and *unadvertise* when it leaves the society. (see our description of *matchmaking* in section 5.1 and of *advertising* in section 3.1).

2.3. Planning

The focus of planning in our systems is on explicating the basic information flow relationships between tasks, and other relationships that affect control-flow decisions. Most control relationships are *derivative* of these more basic relationships. Final action selection, sequencing, and timing are left up to the agent's local scheduler (see the next subsection). Thus the agent planning process (see Figure 1) takes as input the agent's current set of goals $\mathcal{G}$ (including any new, unplanned-for goals $\mathcal{G}_n$), and the set of current task structures (plan instances) $\mathcal{T}$. It produces a new set of current task structures.

- Each individual task T represents an instantiated approach to achieving one or more of the agent's goals $\mathcal{G}$—it is a unit of goal-directed behavior. Every task has an (optional) deadline.

- Each task consists of a partially ordered set of subtasks and/or basic actions **A**. Currently, tasks and actions are related by how information flows from the *outcomes* of one task or action to the *provisions* of another task or action. Subtasks may inherit provisions from their parents and provide outcomes to their parents. In addition, each action has an optional deadline and an optional period. If an action has both a period and a deadline, the deadline is interpreted as the one for the next periodic execution of the basic action.

The most important constraint that the planning/plan retrieval algorithm needs to meet (as part of the agent's overall properties) is to guarantee at least one task for every goal until the goal is accomplished, removed, or believed to be unachievable [3]. For information agents, a common reason that a goal in unachievable is that its specification is malformed, in which case a task to respond with the appropriate KQML error message is instantiated. For more information on the structure of the planning module, see [54]. In section 3, we present planning adaptation.

2.4. Scheduling

The agent scheduling process in general takes as input the agent's current set of task structures $\mathcal{T}$, in particular, the set of all basic actions, and decides which basic action, if any, is to be executed next. This action is then identified as a fixed intention until it is actually carried out (by the execution component). Constraints on the scheduler include:

- No action can be intended unless it is *enabled* (all of its provisions are present).
- Periodic actions must be executed at least once during their period (as measured from the previous execution instance)[2]
- Actions must begin execution before their deadline.
- Actions that miss either their period or deadline are considered to have failed; the scheduler must report all failed actions. Sophisticated schedulers will report such failures (or probable failures) before they occur by reasoning about action durations (and possibly commitments from other agents) [21].
- The scheduler attempts to maximize some predefined utility function defined on the set of task structures. For the information agents, we use a very simple notion of utility—every action needs to be executed in order to achieve a task, and every task has an equal utility value.

In our initial implementation, we use a simple earliest-deadline-first scheduling heuristic. A list of all actions is constructed (the schedule), and the earliest deadline action that is enabled is chosen. Enabled actions that have missed their deadlines are still executed but the missed deadline is recorded and the start of the next period for the task is adjusted to help it meet the next period deadline. When a periodic task is chosen for execution, it is reinserted into the schedule with a deadline equal to the current time plus the action's period. The architecture can support more complex scheduling [20, 50].

2.5. Execution Monitoring

Since the RETSINA agents operate in a dynamic environment, the results of their actions cannot be predicted with certainty. Therefore, the agent architecture includes an execution monitoring module. The execution monitoring process takes the agent's next intended action and prepares, monitors, and completes its execution. The execution monitor prepares an action for execution by setting up a context (including the results of previous actions, etc.) for the action. It monitors the action by optionally providing the associated computation-limited resources—for example, the action may be allowed only a certain amount of time and if the action does not complete before that time is up, the computation is interrupted and the action is marked as having failed. Upon completion of an action, results, whether or not the action has executed successfuly or has failed, are recorded and runtime statistics are collected.

Besides monitoring execution of actions in the environment that is external to the agent, execution monitoring includes a self-reflective phase where execution of actions internal to the agent are monitored. Self reflection allows the agent to adapt to fluctuating performance requirements. For example, an agent could have been tasked with so many information requests that its efficiency decreases resulting in missed deadlines. To monitor performance fluctuations, the start and finish time of each action is recorded as well as a running average duration for that action class. A periodic task is created to carry out the calculations. Adaptation to performance fluctuations is handled through *cloning*, i.e. the agent makes a copy of itself and allocates to it part of its tasks. Cloning as an adaptive response will be discussed in section 4. In the next two sections we will consider the adaptation of various internal agent components. In particular, we will discuss adaptation of planning and execution monitoring.

3. Planning Adaptation

The adaptive capabilities we have developed in RETSINA for our task-reduction planner are based on recent work in planning which has tried to go beyond the limited definition of a plan as a (partially ordered) *sequence* of actions. Recently there has been a strong interest in new plan representations that support sophisticated control flow, such as parallel execution [27], conditional branching [41, 15] and loops [46, 34, 39, 23]. These developments have gone hand-in-hand with the creation of models for *informative* (a.k.a. "sensing", "information-gathering") actions [37, 16]. The two developments are closely interrelated, since adaptive contingencies in a plan are only meaningful if new information from the changing outside world is made available to the planner. Conversely, sensing the world is most useful when doing so does in fact cause the agent to adapt its course of action.

Our initial focus for handling planner adaptation has been on two additional kinds of control flow in plans: 1) *periodic actions*, which are performed repeatedly at specific intervals, and 2) *triggered actions*, which are performed (perhaps repeatedly) in response to external events. Existing planning formalisms explicitly describe control flow in terms of *ordering relationships* between actions, (i.e. $A \prec B$ denoting that action A must be performed before action B). But these relationships are insufficient to distinguish between a number of distinct control relationships that might arise in plans with repetitive actions. We hold the position that most control flow relationships are *derivative* of other, more basic relationships such as required information flows, effects of one task on the duration or result quality of another, etc. We have been working towards an integrated representation for information and control flow in hierarchical task structures [13, 54, 53].

Some types of adaptation expressed by an agent at this level include:

Adapting to failures: At any time, any agent in the system might be unavailable or might go off-line (even if you are in the middle of a long term monitoring situation with that agent). Our planner's task reductions handle these situations so that such failures are dealt with smoothly. If alternate agents are available, they will be contacted and the subproblem restarted (note that unless there are some sort of partial solutions, this could still be expensive). If no alternate agent is available, the task will have to wait. In the future, such failures will signal the planner for an opportunity to replan.

Multiple reductions: Each task can potentially be reduced in several different ways, depending on the current situation. Thus even simple tasks such as answering a query may result in very different sequences of actions (looking up an agent at the matchmaker; using a already known agent, using a cached previous answer).

Interleaved planning and execution: The reduction of some tasks can be delayed until other, "information gathering" tasks, are completed.

The planner architecture is designed so that it can handle these types of adaptation in a natural way.

3.1. Example: Advertising

We illustrate the planner's adaptive capabilities using as an example the functinality of advertising an agents' capabilities to middle agents. The advertisement behavior is shared by all RETSINA agents. Figure 2 shows the task structure which results from the basic planner reduction of the "advertise" task.

The reduction is best understood in two parts: the first is the initial communication to the matchmaker, the second are the actions associated with the "shutdown" of the advertise task (Make Un-Advertisement and the SendKQML on the lower right). Note that the task structure emphasizes the representation of information flows in a plan. The planner fleshes out these more basic relationships; the agent's scheduler then uses this information to produce a schedule (or schedules in a dynamic multi-criteria decision-making situation [20]) that selects, orders, and locates in time specific instantiated actions in a specific context.

The three actions "Make Advertisement", "Get Matchmaker Name", and the topmost instance of "SendKQML" are involved in sending the advertising message.

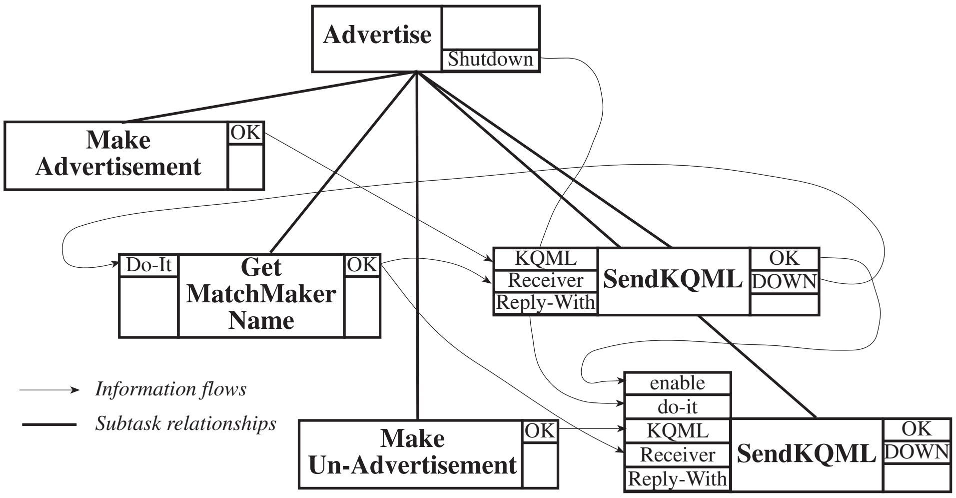

Figure 2. One planner task structure reduction for the "advertise" task.

Both "Get Matchmaker Name" and this instance of "SendKQML" are periodic. All three tasks have an initial deadline of "as soon as possible". "Make Advertisement" constructs the KQML advertisement message content (using the agent's local database schema plus execution information gathered and persistently stored from previous invocations) and provides it to SendKQML. The typical first reduction for "Get Matchmaker Name" is to use a predefined name; alternate reductions involving persistent memory or more complex network level protocols are possible adaptations to failures. If no matchmaker currently exists or some other network error occurs, the SendKQML signals a "DOWN" outcome, which provides a new signal to Get Matchmaker name, and the two tasks are rescheduled (they are periodic) and rerun. In general, the planner strives to produce structures such that agents can appear on the scene at any time and in any order (as well as disappear, which see next).

The two action instances "Make Un-Advertisement" and the second, lower right SendKQML instance comprise the *shutdown* actions for this task. A task is shutdown whenever:

1. The planner removes the current reduction (because, for instance, it has failed). This would not typically happen for advertisement, but does for other tasks.

2. The agent itself intentionally (via a "Termination" action) or unintentionally (unrecoverable error/signal) goes off-line.

Shutdown actions are placed on both the regular and a special shutdown scheduling queue. Both actions are non-periodic and have a deadline of "only execute if there's nothing better to do". Actions on the shutdown queue are architecturally guaranteed to execute at least once, so in particular, the "Make Un-Advertisement" action will either execute during normal processing when the agent would otherwise be idle, or during shutdown if the agent never managed to have a second to spare. The SendKQML that actually passes the advertisement retraction on to the matchmaker has two extra enablement conditions: first, that the initial advertisement actually completed without error, and secondly that the task is being shutdown.

4. Execution Adaptation

Within similar architectures, previous execution-time adaptation has focussed on monitoring actions, or trying to determine if things are going badly before it is too late to correct the problem [21, 2]. In RETSINA, we have begun looking at adaptive load-balancing/rebalancing behaviors such as agent cloning.

Cloning is one of an information agent's possible responses to overloaded conditions. When an information agent recognizes via self-reflection that it is becoming overloaded, it can remove itself from actively pursuing new queries ("unadvertising" its services in KQML) and create a new information agent that is a clone of itself. To do this, it uses a simple model of how it's ability to meet new deadlines is related to the characteristics of it's current queries and other tasks. It compares this model to a hypothetical situation that describes the effect of adding a new agent. In this way, the information agent can make a rational meta-control decision about whether or not it should undertake a cloning behavior.

The key to modeling the agent's load behavior is its current task structures. Since one-shot queries are transient, and simple repeated queries are just a subcase of database monitoring queries, we focus on database monitoring queries only. Each monitoring goal is met by a task that consists of three activities; run-query, check-triggers, and send-results. Run-query's duration is mostly that of the external query interface function. Check-triggers, which is executed whenever the local DB is updated and which thus is an activity shared by all database monitoring tasks, takes time proportional to the number of queries. Send-results takes time proportional to the number of returned results. Predicting performance of an information agent with n database monitoring queries would thus involve a quadratic function, but we can make a simplification by observing that the external query interface functions in all of the information agents we have implemented so far using the Internet (e.g., stock tickers, news, airfares) take an order of magnitude more time than any other part of the system (including measured planning and scheduling overhead). If we let E be the average time to process an external query, then with n queries of average period p, we can predict an idle percentage of:

$$I\% = \frac{p - En}{p} \tag{1}$$

We validate this model in Section 4.1.

When an information agent gets cloned, the clone could be set up to use the resources of another processor (via an 'agent server', or a migratable Java or Telescript program). However, in the case of information agents that already spend the majority of their processing time in network I/O wait states, an overhead proportion $O < 1$ of the En time units each period are available for processing.[3] Thus, as a single agent becomes overloaded as it reaches p/E queries, a new agent can be cloned on the same system to handle another $m = On$ queries. When the second agent runs on a separate processor, $O = 1$. This can continue, with the i^{th} agent on the same processor handling $m_i = O^i m_{i-1}$ queries (note the diminishing returns). We also demonstrate this experimentally in Section 4.1. For two agents, the idle percentage should then follow the model

$$I_{1+2}\% = \frac{(p - En) + (OEn - Em)}{p + OEn} \tag{2}$$

It is important to note how our architecture supports this type of introspection and on-the-fly agent creation. The execution monitoring component of the architecture computes and stores timing information about each agent action, so that the agent learns a good estimate for the value of E. The scheduler, even the simple earliest-deadline-first scheduler, knows the actions and their periods, and so can compute the idle percentage $I\%$. In the systems we have been building, new queries arrive slowly and periods are fairly long, in comparison to E, so the cloning rule waits until there are $(p/E - 1)$ queries before cloning. In a faster environment, with new queries arriving at a rate r and with cloning taking duration C, the cloning behavior should be begun when the number of queries reaches

$$\frac{p}{E} - \lceil rc \rceil$$

4.1. Experimental Results: Execution Adaptation

We undertook an empirical study to measure the baseline performance of our information agents, and to empirically verify the load models presented in the previous section for both a single information agent without the cloning behavior, and an information agent that can clone onto the same processor. We also wanted to verify our work in the context of a real application (monitoring stock prices).

Our first set of experiments were oriented toward the measurement of the baseline performance of an information agent. Figure 3 shows the average idle percentage, and the average percentage of actions that had deadlines and that missed them, for various task loads. The query period was fixed at 60 seconds, and the external query time fixed at 10 seconds (but nothing else within the agent was fixed). Each experiment was run for 10 minutes and repeated 5 times. As expected, the idle time decreases and the number of missed deadlines increases, especially after the predicted saturation point ($n = 6$). The graph also shows the average amount of time by which an action misses its deadline.

The next step was to verify our model of single information agent loading behavior (Equation 1). We first used a partially simulated information agent to minimize variation factors external to the information agent architecture. Later, we used a

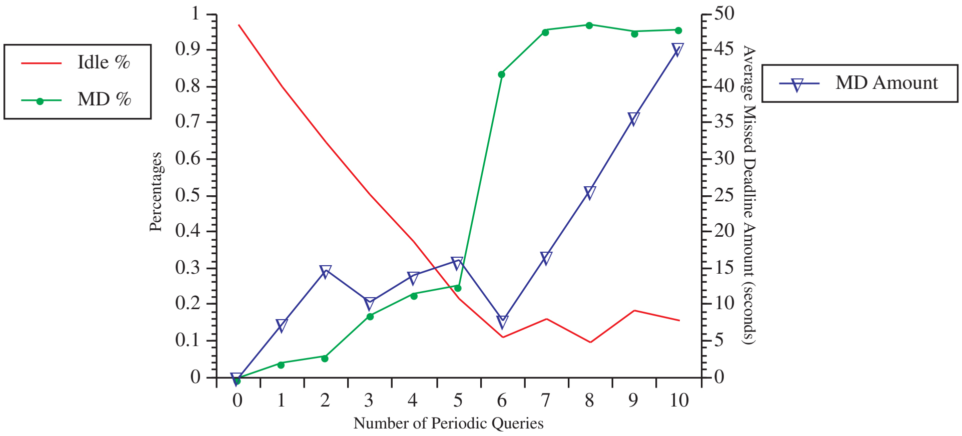

Figure 3. A graph of the average percentage idle time and average percentage of actions with deadlines that missed them for various loads (left Y axis). Superimposed on the graph, and keyed to the right axis, are the average number of seconds by which a missed deadline is missed.

completely real agent with a real external query interface (the Security APL stock ticker agent).

The graph on the left of Figure 4 depicts the actual and predicted idle times for an information agent that monitors a simulated external information source that takes a constant 10 seconds.[4] The information agent being examined was given tasks by a second experiment-driver agent. Each experiment consisted of a sequence of 0 through 10 tasks (n) given to the information agent at the start. Each task had a period of 60 seconds, and each complete experiment was repeated 5 times. Each experiment lasted 10 minutes. The figure clearly shows how the agent reaches saturation after the 6th task as predicted by the model ($p/E = 6$). The idle time never quite drops below 10% because the first minute is spent idling between startup activities (e.g., making the initial connection and sending the batch of tasks). After adding in this extra base idle time, our model predicts the actual utilization quite well ($R^2 = 0.97$; R^2 is a measure of the total variance explained by the model).

We also ran this set of experiments using a real external interface, that of the Security APL stock ticker. The results are shown graphically on the graph in the right in Figure 4. Five experiments were again run with a period of 60 seconds (much faster than normal operation) and 1 through 10 tasks. Our utilization model also correctly predicted the performance of this real system, with $R^2 = 0.96$ and the differences between the model and the experimental results were not significant by either t-tests or non-parametric signed-rank tests. The odd utilization results that occurred while testing $n = 7, 8, 9$ were caused by network delays that significantly changed the average value of E (the duration of the external query). However, since the agent's execution monitor measures this value during problem solving, the agent can still react appropriately (the model still fits fine).

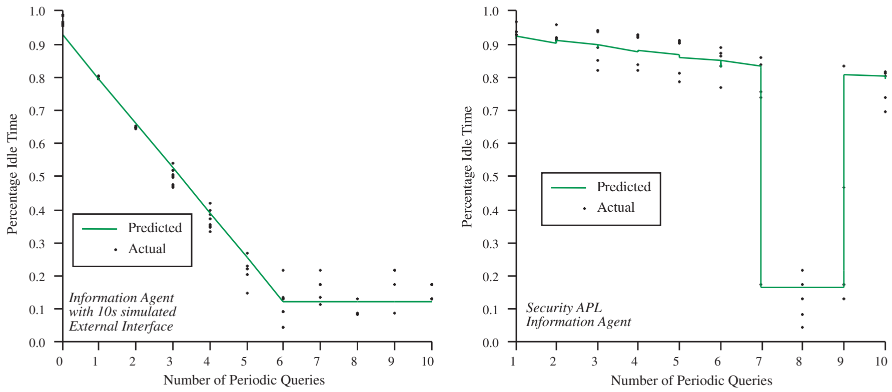

Figure 4. On the left, graph of predicted and actual utilization for a real information agent with a simulated external query interface. On the right, the same graph for the Security APL stock ticker agent.

Finally, we extended our model to predict the utilization for a system of two agents with the cloning behavior. Figure 5 shows the predicted and actual results over loads of 1 to 10 tasks with periods of 60 seconds, $E = 10$, and 5 repetitions. Agent 1 clones itself onto the same processor when $n > 5$. In this case, model $R^2 = 0.89$, and the differences between the model and the measured values are not significant by t-test or signed-ranks. The same graph shows the predicted curve for one agent (from the left side of Figure 4) as a comparison.[5]

5. Organizational Adaptation

The previous two sections have discussed adaptation within an agent, this section deals with adaptation at the multi-agent, organizational level. It has been clear to organizational theorists since at least the 60's that there is no one good organizational structure for human organizations [33]. Organizations must instead be chosen and adapted to the task environment at hand. Most important are the different types and qualities of uncertainty present in the environment (e.g., uncertainty associated with inputs and output measurements, uncertainty associated with causal relationships in the environment, the time span of definitive feedback after making a decision [43]).

In multi-agent information systems, one of the most important sources of uncertainty revolves around what information is available from whom (and at what cost). Our organizational model relies on three basic roles: that of the requester, the middle-agent, and the provider. Any one agent in a domain system might take on multiple roles, for example an agent that requests basic info from several providers, does some complex integration, and then serves the integrated info to other requesters. In this model, communicative acts are limited to requests, replies,

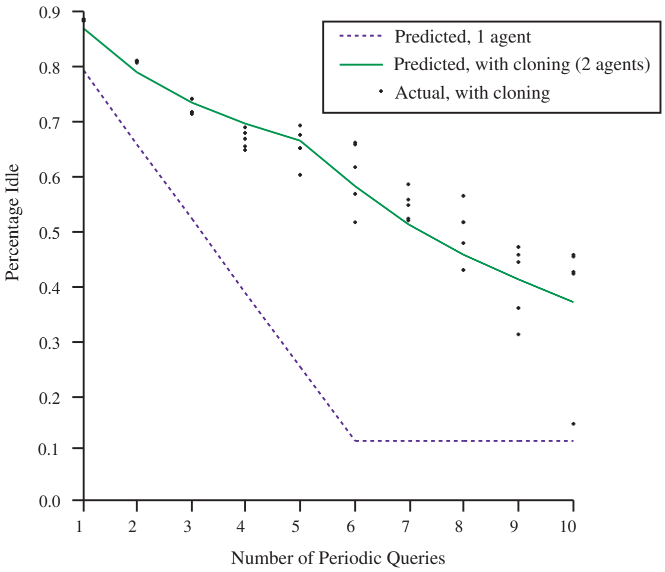

Figure 5. Predicted idle percentages for a single non cloning agent, and an agent with the cloning behavior across various task loads. Plotted points are the measured idle percentages from experimental data including cloning agents.

and commitments. This has two benefits: first, the semantics of requests and commitments are well-understood [19, 4], and second, such a model allows us to build simpler agents that can work in a open environment.

We say that a requester agent has *preferences*, and that a provider agent has *capabilities*. A specific *request* is an instance of an agent's preferences, and a specific *reply* or action in service of a request is an instance of an agent's capabilities. Furthermore, an agent can have a mental state with respect to a particular specification of a preference or capability. An advertisement is a capability specification such that the agent creating the advertisement is committed [3, 25] to servicing any request that satisfies the advertisement's constraints.

We have developed a standard basic advertising behavior that allows agents to encapsulate a model of their capabilities and send it to a "matchmaker" or "yellow-pages" middle agent [29]. Such a matchmaker agent can then be used by a multi-agent system to form several different organizational structures[9]:

Uncoordinated Team: agents use a basic shared behavior for asking questions that first queries the matchmaker as to who might answer the query, and then chooses an agent randomly for the target query. Very low overhead, but potentially unbalanced loads, reliability limited by individual data sources, and problems linking queries across multiple ontologies.

Economic Markets: (e.g., [51]) Agents use price, reliability, and other utility characteristics with which to choose another agent. The matchmaker can supply to each agent the appropriate updated pricing information as new agents enter and exit the system, or alter their advertisements. Agents can dynamically adjust their organization as often as necessary, limited by transaction costs. Potentially such organizations provide efficient load balancing and the ability to provide truly expensive services (expensive in terms of the resources required).

Federations: (e.g., [52, 22, 19]) Agents give up individual autonomy over choosing who they will do business with to a locally centralized "facilitator" (an extension of the matchmaker concept) that brokers requests. Centralization of message traffic potentially allows greater load balancing and the provision of automatic translation and mediation services.

Bureaucratic Functional Units: Traditional manager/employee groups of a single multi-source information agent (manager) and several simple information agent (employees). By organizing into functional units, i.e., related information sources, such organizations concentrate on providing higher reliability (by using multiple underlying sources), simple information integration (from partially overlapping information), and load balancing.

This is not an exhaustive list. Our general architecture has supported other explorations into understanding the effects of organizational structures [11, 12, 10].

5.1. Example: Matchmaking and Brokering

As an example of organizational adaptation, let us compare the failure recovery characteristics of matchmade and bureaucratic manager ("brokered") information organizations. Both organizations are possible solutions to the connection problem [6]—finding the other agents that might have the information and capabilities you need.

In a matchmade organization (see Figure 6), providers advertise[6] their capabilities with a matchmaker. If those capabilities change, or the agent exits the open system, the provider *unadvertises*. A matchmaker stores these advertisements in a local database. A requester wishing to ask a query first formulates a meta-query asking for advertisements from agents that could respond to the query. This meta-query is asked of a matchmaker, which responds with a set of matching advertisements. The requester can then use its full preferences to choose a provider, and make its request directly to the chosen provider. Furthermore, if this type of query is asked often, then the requester can *subscribe* to updated advertisements from a matchmaker, and keep a local cache of the current advertisements.

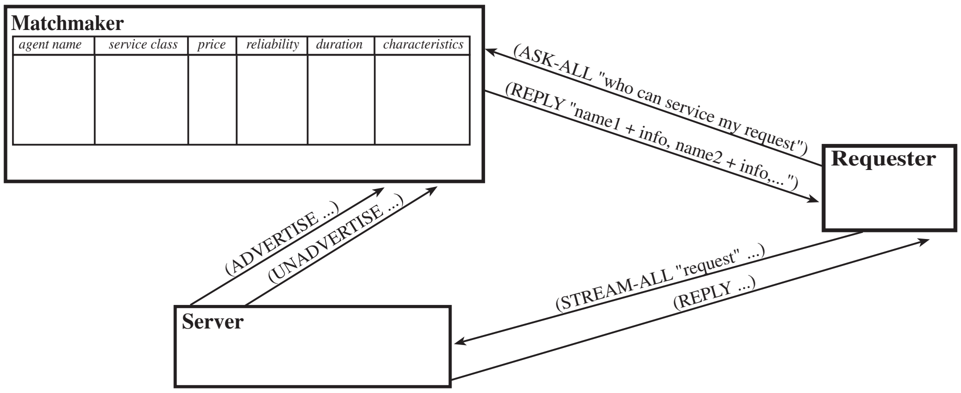

Figure 6. Communications between three agents taking on the roles of matchmaker, requester, and server.

In brokered organizations (Figure 7), requester behaviors remain the same. In a *pure* brokered organization the brokers are generally known by all the agents, just like a matchmaker is. However, for practicality in an open system *hybrid* brokered organizations use a matchmaker so that providers and requesters can find the appropriate broker. Providers query a matchmaker to find an appropriate broker, and then advertise with one broker. Brokers advertise summary capabilities built from all the providers that have advertised with them; these capabilities are advertised in turn to the matchmaker. When a request comes in, the broker matches it with a provider and sends it on; the reply is then sent back through the broker to the original requester.

The main differences between matchmakers and brokers are these: Matchmakers reveal to requesters the capabilities of agents that have advertised with them. They maintain the privacy of requesters. Requesters select the provider that vest fits their constraints and communicate with that provider directly. In contrast, brokers protect the privacy of both requesters and providers. The broker knows both the preferences and capabilities and routes both requests and replies appropriately. In a brokered system requesters and providers do not communicate directly with each other but only through brokers.

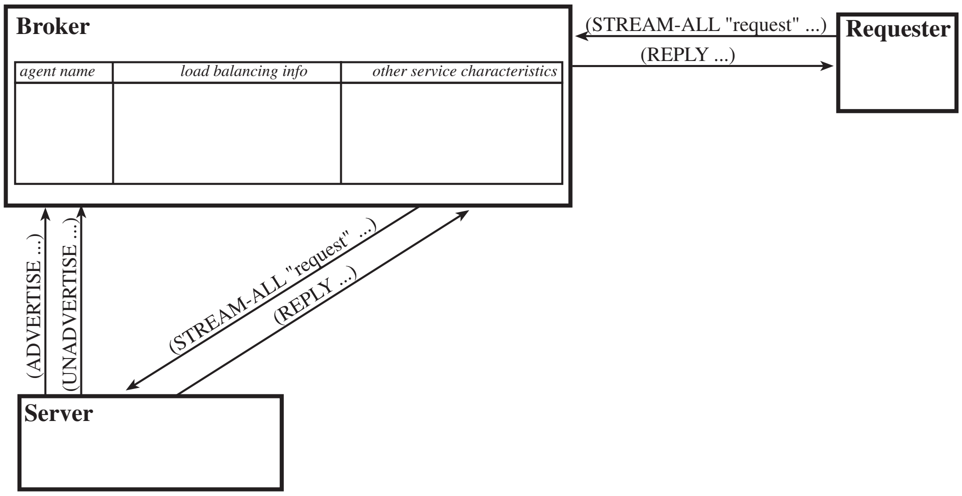

Figure 7. Communications between three agents taking on the roles of broker, requester, and brokered server.

The ability of an organization to quickly adapt to changing capabilities (i.e., the entry or exit of providers) is a function of the distance that the information has to travel, and the costs of keeping that information up-to-date. Elsewhere, the authors have analyzed and developed an analytical model of what happens in an open system to the maximum service time when providers come and go [8]. We have also verified these results empirically using the Warren system, where the agents experience real communication and processor latencies, etc. while running on multiple serial processors.

5.2. Experimental Results: Organizational Adaptation

In this section we report on some experimental results that provide insights into the adaptive capabilities and particular performance tradeoffs between matchmade and brokered systems. In the experiments we used one broker and one matchmaker. The broker and matchmaker are the same agents used in the actual portfolio management system. Providers and requesters are instances of Warren providers and requesters. The actual service time for each request and the period between requests were generated randomly; the service times were distributed normally around the mean T, and the request generation period was distributed exponentially around the mean P. But because we used real rather than simulated agents, some parameters were beyond our control, such as the inter-agent communication latency, the computational needs of the broker or matchmaker, and the amount of time spent by the providers and requesters on planning, scheduling, and other internal operations.

We recreate one of these experiments here, where we investigate the effect of provider failure and recovery on our two systems—demonstrating simple adaptive organizational behavior. We began with three providers, and fixed the service time and request generation period at 15 and 10 seconds, respectively. After five minutes, we killed one of the providers, and after five more minutes, we killed a second one. Five minutes after that, we brought one of the severs back on line, and then ten minutes later the third one returned. When a provider dies, it sends a SORRY message for each outstanding request. Each of these requests must be reallocated (by either the broker or the original requester) to another provider.

Figures 8a and 8b show the results of this experiment in each of the two basic organizational regimes. Each point represents the completion of a service request. The response-time superiority of the brokered system stems from the difference in behavior of the two systems when the failed providers come back online. When there is only one provider left operating, the system is saturated, so that provider begins to build up a large backlog of requests. When the second and third providers become available again, the requesters in the matchmade system continue to allocate one-half or one-third of their requests to the overloaded provider, so the backlog persists for a long time.[7] In the brokered system, on the other hand, all new requests are allocated to the new provider, allowing the backlog at the congested provider to quickly dissipate.

6. Conclusions

This paper has discussed adaptation in the RETSINA system of intelligent agents. We have implemented adaptation schemes at the individual agent level as well as at the organizational level. Organizational adaptation includes behaviors by which agents deal with an open, dynamically changing society. Some of our work has involved building and verifying analytical models of the properties of some simple organizations.

The adaptive behavior at the individual agent level is supported by the reusable agent architecture. We have reported examples and experimental results at the level of planning and execution. At the planning level, we discussed adapting to failures, the use of multiple task reductions, and interleaved planning and execution. Our task reduction planner, which focuses on indicating the underlying information flows between tasks, rather than directly indicating task ordering, was described. Current work includes the integration of a highly adaptive scheduling component (e.g., [20, 50]).

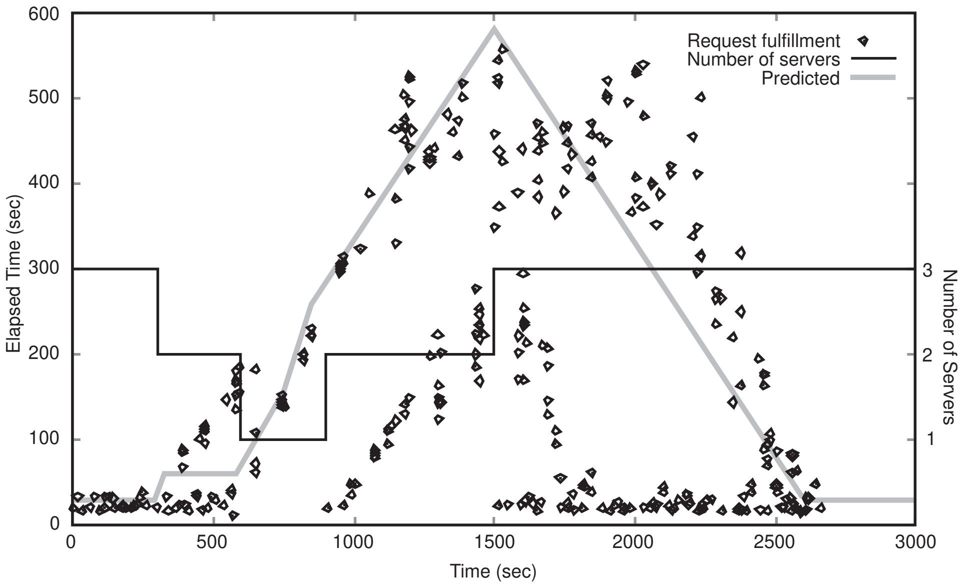

Figure 8a. Predicted and actual effect of provider failure and recovery in a matchmade system

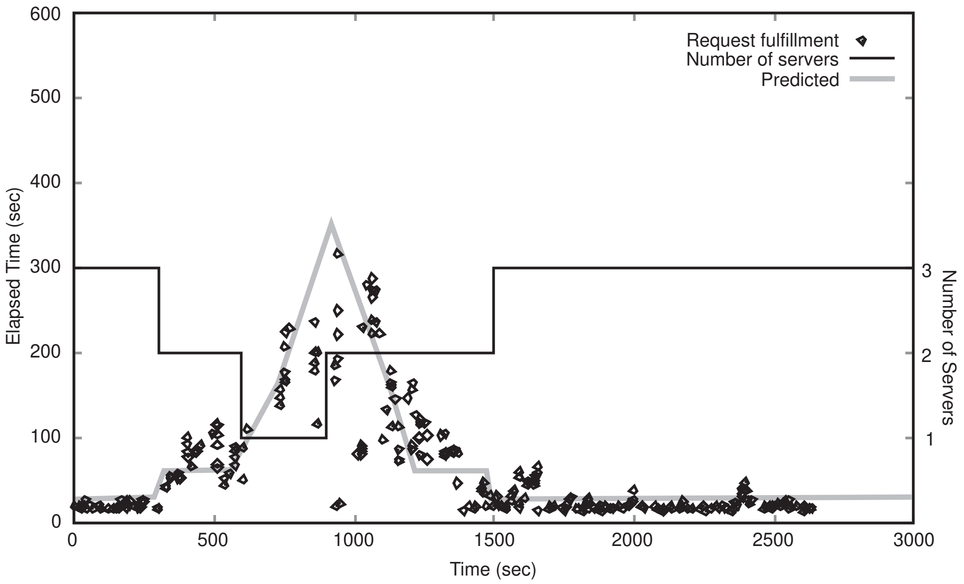

Figure 8b. Predicted and actual effect of provider failure and recovery in a brokered system

This paper also discussed a fairly detailed model of, and experimentation with, a simple cloning behavior we have implemented. This behavior is an agent's adaptive response to fluctuating performance requirements. Several extensions to this cloning model are being considered. In particular, there are several more intelligent ways with which to divide up the tasks when cloning occurs in order to use resources more efficiently (and to keep queries balanced after a cloning event occurs). These include:

1. partitioning existing tasks by time/periodicity, so that the resulting agents have a balanced, schedulable set of tasks,
2. partitioning tasks by client so that all tasks from agent 1 end up at the same clone, and
3. partitioning tasks by class/type/content so that all tasks about one subject (e.g., the stock price of IBM) end up at the same clone.

Acknowledgments

We would like to thank Mike Williamson for his help in developing the WARREN task reduction planner and in running some of the experiments.

Notes

1. While SIMS agents do not have most of these capabilities, it is interesting to note that Etzioni's softbots provide for some of them—periodic and externally enabled actions—by adding a programming language layer above the planner [17], rather than by adding a seperate local scheduling component.
2. Here, the term *periodic* refers to to the fact that the task must be repeated at intervals that cannot be any longer than the specified period, i.e., "max invocation separation constrained" [38].
3. Another way to recoup this time is to run the blocking external query in a separate process, breaking run-query into two parts. We are currently comparing the overhead of these two different uni-processor solutions—in any case we stress that both behaviors are reusable and can be used by any existing information agent without reprogramming. Cloning to another processor still has the desired effect.
4. All the experiments described here were done on a standard timesharing Unix workstation while connected to the network.
5. Since the potential second agent would, if it existed, be totally idle from $1 < n < 6$, the idle curve differs there in the cloning case.
6. All communications here are done via the appropriate KQML performatives or extensions [19].
7. This effect could be reduced if requesters make an effort at active load balancing.

References

1. P. R. Cohen and H. J. Levesque. Intention=choice + commitment. In *Proceedings of AAAI-87*, pages 410–415, Seattle, WA., 1987. AAAI.

2. Paul Cohen, Michael Greenberg, David Hart, and Adele Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3):33–48, Fall 1989. Also COINS-TR-89-61.
3. Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.
4. P.R. Cohen and H.J. Levesque. Communicative actions for artificial agents. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 65–72, San Francisco, June 1995. AAAI Press.
5. C. Collet, M.N. Huhns, and W. Shen. Resource integration using a large knowledge base in Carnot. *Computer*, pages 55–62, December 1991.
6. R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, January 1983.
7. K. S. Decker, V.R. Lesser, M.V. Nagendra Prasad, and T. Wagner. MACRON: an architecture for multi-agent cooperative information gathering. In *Proccedings of the CIKM-95 Workshop on Intelligent Information Agents*, Baltimore, MD, 1995.
8. K. S. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, Nagoya, Japan, August 1997.
9. K. S. Decker, M. Williamson, and K. Sycara. Modeling information agents: Advertisements, organizational roles, and dynamic behavior. In *Proceedings of the AAAI-96 Workshop on Agent Modeling*, 1996. AAAI Report WS-96-02.
10. Keith S. Decker. Task environment centered simulation. In M. Prietula, K. Carley, and L. Gasser, editors, *Simulating Organizations: Computational Models of Institutions and Groups*. AAAI Press/MIT Press, 1997. Forthcoming.
11. Keith S. Decker and Victor R. Lesser. An approach to analyzing the need for meta-level communication. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 360–366, Chambéry, France, August 1993.
12. Keith S. Decker and Victor R. Lesser. A one-shot dynamic coordination algorithm for distributed sensor networks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 210–216, Washington, July 1993.
13. Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224, Washington, July 1993.
14. Keith S. Decker and Victor R. Lesser. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 73–80, San Francisco, June 1995. AAAI Press. Longer version available as UMass CS-TR 94–14.
15. D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proc. 2nd Intl. Conf. on A.I. Planning Systems*, June 1994.
16. O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An Approach to Planning with Incomplete Information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, San Francisco, CA, October 1992. Morgan Kaufmann. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.
17. Oren Etzioni, Neal Lesh, and Richard Segal. Building softbots for unix (preliminary report). Technical Report softbots-tr.ps, University of Washington, 1992.
18. Oren Etzioni and Daniel Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37(7), July 1994.
19. 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 Management CIKM'94*. ACM Press, November 1994.
20. Alan Garvey, Marty Humphrey, and Victor Lesser. Task interdependencies in design-to-time real-time scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 580–585, Washington, July 1993.
21. Alan Garvey and Victor Lesser. Representing and scheduling satisficing tasks. In Swaminathan Natarajan, editor, *Imprecise and Approximate Computation*, pages 23–34. Kluwer Academic Publishers, Norwell, MA, 1995.
22. M.R. Genesereth and S.P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53,147, 1994.

23. R. Goodwin. Using loops in decision-theoretic refinement planners. In *Proc. 3rd Intl. Conf. on A.I. Planning Systems*, 1996.
24. T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Technical Report KSL-93-4, Knowledge Systems Laboratory, Stanford University, 1993.
25. N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.
26. W. Kim and J. Seo. Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, pages 12–18, December 1991.
27. C. Knoblock. Generating parallel execution plans with a partial-order planner. In *Proc. 2nd Intl. Conf. on A.I. Planning Systems*, pages 98–103, June 1994.
28. C.A. Knoblock, Y. Arens, and C. Hsu. Cooperating agents for information retrieval. In *Proc. 2nd Intl. Conf. on Cooperative Information Systems*. Univ. of Toronto Press, 1994.
29. D. Kuokka and L. Harada. On using KQML for matchmaking. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 239–245, San Francisco, June 1995. AAAI Press.
30. Y. Labrou and T. Finin. A semantics approach for KQML. In *Proceedings of the Third International Conference on Information and Knowledge Management CIKM'94*. ACM Press, November 1994.
31. Y. Labrou and T. Finin. A proposal for a new kqml specification. CSEE Technical Report TR CS–97–03, University of Maryland Baltimore County, August 1997.
32. Kan Lang. Newsweeder: Learning to filter netnews. In *Proceedings of Machine Learning Conference*, 1995.
33. Paul Lawrence and Jay Lorsch. *Organization and Environment*. Harvard University Press, Cambridge, MA, 1967.
34. S. Lin and T. Dean. Generating optimal policies for markov decision processes formulated as plans with conditional branches and loops. In *Proc. 2nd European Planning Workshop*, 1995.
35. Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7), July 1994.
36. Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7), July 1994.
37. R. Moore. A Formal Theory of Knowledge and Action. In J. Hobbs and R. Moore, editors, *Formal Theories of the Commonsense World*. Ablex, Norwood, NJ, 1985.
38. D. J. Musliner. Scheduling issues arising from automated real-time system design. Technical Report CS-TR-3364, UMIACS-TR-94-118, Department of Computer Science, University of Maryland, 1994.
39. L. Ngo and P. Haddawy. Representing iterative loops for decision-theoretic planning. In *Working Notes of the AAAI Spring Symposium on Extending Theories of Action*, 1995.
40. Tim Oates, M. V. Nagendra Prasad, Victor R. Lesser, and Keith S. Decker. A distributed problem solving approach to cooperative information gathering. In *AAAI Spring Symposium on Information Gathering in Distributed Environments*, Stanford University, March 1995.
41. M. Peot and D. Smith. Conditional Nonlinear Planning. In *Proc. 1st Intl. Conf. on A.I. Planning Systems*, pages 189–197, June 1992.
42. A.S. Rao and M.P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 312–319, San Francisco, June 1995. AAAI Press.
43. W. Richard Scott. *Organizations: Rational, Natural, and Open Systems*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.
44. J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, 1969.
45. R. Simmons. Structured control for autonomous robots. *IEEE Trans. on Robotics and Automation*, 10(1), February 1994.
46. D. Smith and M. Williamson. Representation and evaluation of plans with loops. In *Working Notes of the AAAI Spring Symposium on Extended Theories of Action: Formal Theory and Practical Applications*, Stanford, CA, 1995.

47. I.A. Smith and P.R. Cohen. Toward a semantics for an agent communication language based on speech acts. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 24–31, August 1996.
48. K. Sycara, K. S. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, December 1996.
49. Katia Sycara and Dajun Zeng. Towards an intelligent electronic secretary. In *Proceedings of the CIKM-94 (International Conference on Information and Knowledge Management) Workshop on Intelligent Information Agents*, National Institute of Standards and Technology, Gaithersburg, Maryland, December 1994.
50. T. Wagner, A. Garvey, and V. Lesser. Complex goal criteria and its application in design-to-criteria scheduling. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, July 1997.
51. Michael Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.
52. G. Wiederhold, P. Wegner, and S. Cefi. Toward megaprogramming. *Communications of the ACM*, 33(11):89–99, 1992.
53. M. Williamson, K. S. Decker, and K. Sycara. Executing decision-theoretic plans in multi-agent environments. In *AAAI Fall Symposium on Plan Execution*, November 1996. AAAI Report FS-96-01.
54. M. Williamson, K. S. Decker, and K. Sycara. Unified information and control flow in hierarchical task networks. In *Proceedings of the AAAI-96 workshop on Theories of Planning, Action, and Control*, 1996.
55. M. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.