Intelligent Adaptive Information Agents

Keith Decker, Katia Sycara, and Mike Williamson
The Robotics Institute, Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
(decker,sycara,mikew)@cs.cmu.edu

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 are developing a multi-agent financial portfolio management system that must deal with all of these problems. This paper will briefly describe our approaches and solutions at several different levels within the agents: adaptation at the organizational, planning, scheduling, and execution levels. We discuss our solution for execution-level adaptation in detail, and present empirical evidence backing up the theory behind the solution.

Introduction

Adaptation is behavior of an agent in response to unexpected (i.e., low probability) events or dynamic environments. Examples of unexpected events include the unscheduled failure of an agent, an agent's computational platform, or 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).

We have been involved in designing, building, and analyzing multi-agent systems that exist in these types of dynamic and partially unpredictable environments. These agents handle adaptation at several different levels, from the high-level multi-agent organization down to the monitoring of individual method executions. In the next section we will discuss the individual architecture of these agents. Then, in the section entitled "Agent Adaptation" we will discuss the problems and solutions to agent adaptation at the organizational, planning, scheduling, and execution monitoring levels. In particular, we will discuss how our architecture supports organizational and planning-level adaptation currently and what areas are still under active investigation. We will discuss schedule adaptation only in passing and refer the interested reader to work elsewhere. Finally, we will present a detailed model and some experiments with one particular behavior, agent self-cloning, for execution-level adaptation.

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. 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, managing and rebuilding the local database when necessary, and polling for KQML messages from other agents.

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 an instantiation of the DECAF (Distributed, Environment-Centered Agent Framework) architecture (Decker *et al.* 1995).

Control: Planning, Scheduling, and Action Execution

The control process for information agents includes steps for planning to achieve local or non-local objectives, scheduling the actions within these plans, and actually carrying out these actions. In addition, the agent has a shutdown and an initialization process. The agent executes the initialization process upon startup; it bootstraps the agent by giving it initial objectives 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 uncontinueable error signal. The shutdown process assures that messages are sent from the terminating agent asserting goal dissolution to client agents and requesting goal dissolution to server agents (see the section on planning adaptation).

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 $\mathcal{G}_\backslash$), and the set of current task structures (plan instances) $\mathcal{T}$. It produces a new set of current task structures (Williamson, Decker, & Sycara 1996).

- 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 $\mathbf{A}$. Currently, tasks and actions are related by how information flows from the *outcomes* of one task or action to the *provisions* of anther task or action. Subtasks may inherit provisions from their parents and provide outcomes to their parents. Each action also 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 (Cohen & Levesque 1990). 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. An information agent receives in messages from other agents three important types of goals:

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").

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.
- Periodic actions must be executed at least once during their period (as measured from the previous execution instance) (technically, this is a max invocation separation constraint, not a "period").
- 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) (Garvey & Lesser 1995).
- 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 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 are recorded, downstream actions are passed provisions if so indicated, and runtime statistics are collected.

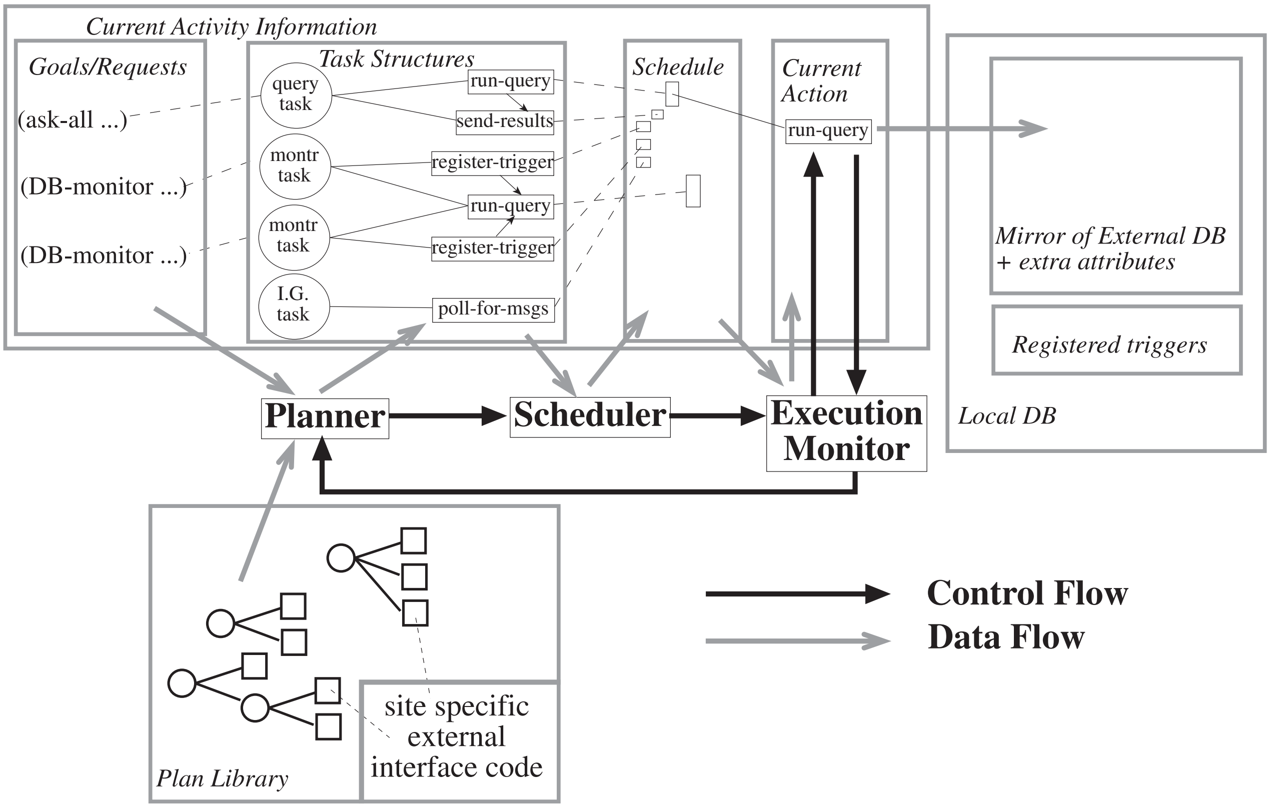

Figure 1: Overall view of data and control flow in an information agent.

Agent Adaptation

In this section we briefly consider several types of adaptation supported by this individual agent architecture in our current and previous work. These types include organizational, planning, scheduling, and execution-time adaptation. We are currently actively involved in expanding an agent's adaptation choices at the organizational and planning levels—in this short paper we will only describe how our architecture supports organizational and planning-level adaptation, what we have currently implemented, and what directions we are currently pursuing. We have not, in our current work, done much with schedule adaptation; instead we indicate future potential by pointing to earlier work within this general architecture that addresses precisely schedule adaptation. Finally, we present a fairly comprehensive account of one type of execution-time adaptation ("self-cloning").

Organizational Adaptation

It has been clear to organizational theorists since at least the 60's that there is no one good organizational structure for human organizations (Lawrence & Lorsch 1967). 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 (Scott 1987)). Recently, researchers have proposed that organizations grow toward, and structure themselves around, sources of information that are important to them because they are sources of news about how the future is (evidently) turning out (Stinchcombe 1990).

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). We have developed a standard basic advertising behavior that allows agents to encapsulate a model of their capabilities and send it to a "matchmaker" information agent (Kuokka & Harada 1995). Such a matchmaker agent can then be used by a multi-agent system to form several different organizational structures(Decker, Williamson, & Sycara 1996):

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. Our initial implementation used this organization exclusively.

Federations: (e.g., (Wiederhold, Wegner, & Cefi 1992; Genesereth & Katchpel 1994; Finin *et al.* 1994)) 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. We have constructed general purpose brokering agents, and are currently conducting an empirical study of matchmaking vs. brokering behavior. Of course, a hybrid organization is both possible and compelling in many situations.

Economic Markets: (e.g., (Wellman 1993)) 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). Both brokers and matchmakers can be used in market-based systems (corresponding to centralized and decentralized markets, respectively).

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. "Managing" can be viewed as brokering with special constraints on worker behavior brought about by the manager-worker authority relationship.

This is not an exhaustive list. Our architecture has supported other explorations into understanding the effects of organizational structures (Decker 1996).

Planning Adaptation

The "planner" portion of our agent architecture consists of a new hierarchical task network based planner using a plan formalism that admits sophisticated control structures such as looping and periodic tasks (Williamson, Decker, & Sycara 1996). It has features derived from earlier classical planning work, as well as task structure representations such as TCA/TCX (Simmons 1994) and TÆMS (Decker & Lesser 1995). The focus of planning in our system 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).

Some types of adaptation expressed by our agents at this level in our current implementation 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 be 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.

Previous work has focussed on coordination mechanisms alone. In particular, the Generalized Partial Global Planning family of coordination mechanisms is a domain-independent approach to multi-agent scheduling- and planning-level coordination that works in conjunction with an agent's existing local scheduler to adapt a plan by adding certain constraints (Decker & Lesser 1995). These include commitments to do a task with a minimum level of quality, or commitments to do a task by a certain deadline. If the resulting plan can be successfully scheduled, these local commitments can be communicated to other agents where they become non-local commitments to those agent's local schedulers. Not all mechanisms are needed in all environments. Nagendra-Prasad has begun work on learning which mechanisms are needed in an environment automatically (Prasad & Lesser 1996).

Scheduling Adaptation

In our current work, we have been using a fairly simple earliest deadline first scheduler that does little adaptation besides adjusting the deadlines of periodic (technically "max invocation separation constrained") actions that miss or are about to miss their initial deadlines. Also, agents can dynamically change their information request periods which affect only the scheduling of the related actions.

Earlier work within this architecture has used a more sophisticated "Design-to-Time" scheduling algorithm, which adapts the local schedule in an attempt to maximize schedule quality while minimizing missed deadlines (Garvey & Lesser 1995; Decker & Lesser 1995). In doing so, the scheduler may choose from both "multiple methods" (different algorithms that represent difference action duration/result quality tradeoffs) and anytime algorithms (summarized by duration/quality probability distribution tables (Zilberstein & Russell 1992)).

Execution Adaptation

Within this architecture, previous execution-time adaptation has focussed on monitoring actions (Garvey & Lesser 1995). Recently, we have begun looking at load-balancing/rebalancing behaviors such as agent cloning.

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.

This self-reflection phase is a part of the agent's execution monitoring process. 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 required by the model described below.

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 the next section.

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.[1] 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 the next section. 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

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

Execution Adaptation: Experimental Results

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

On the left of Figure 3 is a graph of the actual and predicted idle times for an information agent that monitors a simulated external information source that takes a constant 10 seconds.[2] 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 right in Figure 3. 5 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).

Finally, we extended our model to predict the utilization for a system of agents with the cloning behavior, as indicated in the previous section. Figure 4 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

[1] 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 uniprocessor 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.

[2] All the experiments described here were done on a standard timesharing Unix workstation while connected to the network.

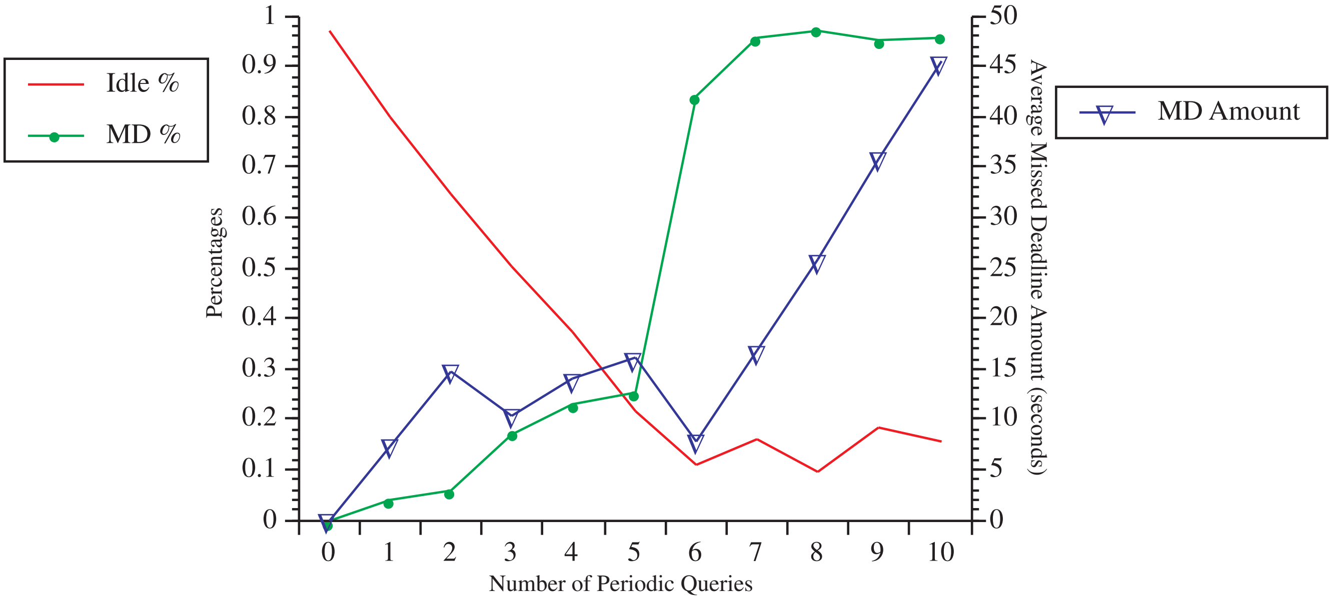

Figure 2: A graph of the average percentage idle time and average percentage of actions with deadlines that missed them for various loads (left Y axis). Superimposed on the graph, and keyed to the right axis, are the average number of seconds by which a missed deadline is missed.

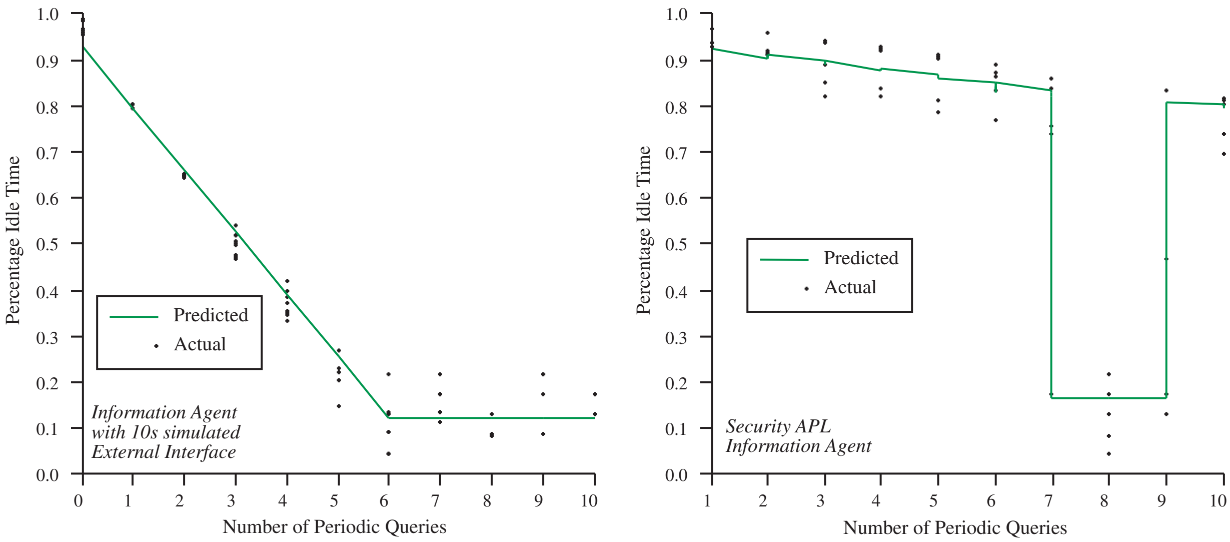

Figure 3: On the left, graph of predicted and actual utilization for a real information agent with a simulated external query interface. On the right, the same graph for the Security APL stock ticker agent.

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 3) as a comparison.[3]

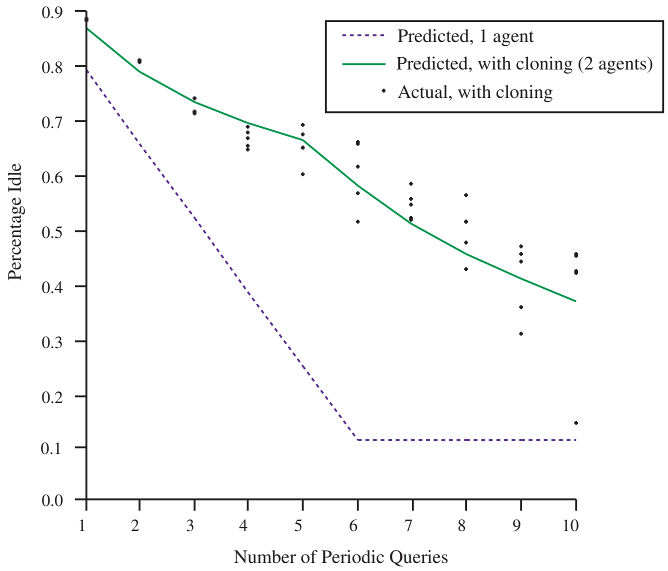

Figure 4: Predicted idle percentages for a single non cloning agent, and an agent with the cloning behavior across various task loads. Plotted points are the measured idle percentages from experimental data including cloning agents.

Current & Future Work

This paper has discussed adaptation in a system of intelligent agents at four different levels: organizational, planning, scheduling, and execution. Work at the organizational and planning levels is a current, active pursuit; we expect to return to schedule adaptation as time and resources permit. Currently, we are conducting an empirical study into matchmakers, brokers, and related hybrid organizations.

This paper also discussed a fairly detailed model of, and experimentation with, a simple cloning behavior we have implemented. 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:

- Partitioning existing tasks by time/periodicity, so that the resulting agents have a balanced, schedulable set of tasks.
- Partitioning tasks by client: all tasks from agent 1 end up at the same clone.
- Partitioning tasks by class/type/content: all tasks about one subject (e.g., the stock price of IBM) end up at the same clone.
- For multi-source information agents, partitioning tasks by data source: all tasks requiring the use of source A end up at the same clone.

[3]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.

Acknowledgements

The authors would like to thank the reviewers for their helpful comments. This work has been supported in part by ARPA contract F33615–93–1–1330, in part by ONR contract N00014–95–1–1092, and in part by NSF contract IRI–9508191.

References

Cohen, P. R., and Levesque, H. J. 1990. Intention is choice with commitment. *Artificial Intelligence* 42(3):213–261.

Decker, K. S., and Lesser, V. R. 1995. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems*, 73–80. San Francisco: AAAI Press. Longer version available as UMass CS-TR 94–14.

Decker, K.; Lesser, V.; Prasad, M. N.; and Wagner, T. 1995. MACRON: an architecture for multi-agent cooperative information gathering. In *Proccedings of the CIKM-95 Workshop on Intelligent Information Agents.*

Decker, K.; Williamson, M.; and Sycara, K. 1996. Modeling information agents: Advertisements, organizational roles, and dynamic behavior. In *Proceedings of the AAAI-96 Workshop on Agent Modeling.*

Decker, K. S. 1996. Task environment centered simulation. In Prietula, M.; Carley, K.; and Gasser, L., eds., *Simulating Organizations: Computational Models of Institutions and Groups.* AAAI Press/MIT Press. Forthcoming.

Finin, T.; Fritzson, R.; McKay, D.; and McEntire, R. 1994. KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management CIKM'94.* ACM Press.

Garvey, A., and Lesser, V. 1995. Representing and scheduling satisficing tasks. In Natarajan, S., ed., *Imprecise and Approximate Computation.* Norwell, MA: Kluwer Academic Publishers. 23–34.

Genesereth, M., and Katchpel, S. 1994. Software agents. *Communications of the ACM* 37(7):48–53,147.

Kuokka, D., and Harada, L. 1995. On using KQML for matchmaking. In *Proceedings of the First International Conference on Multi-Agent Systems*, 239–245. San Francisco: AAAI Press.

Lawrence, P., and Lorsch, J. 1967. *Organization and Environment.* Cambridge, MA: Harvard University Press.

Prasad, M. N., and Lesser, V. 1996. Learning situation-specific coordination in generalized partial global planning. In *AAAI Spring Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*.

Scott, W. R. 1987. *Organizations: Rational, Natural, and Open Systems.* Englewood Cliffs, NJ: Prentice-Hall, Inc.

Simmons, R. 1994. Structured control for autonomous robots. *IEEE Trans. on Robotics and Automation* 10(1).

Stinchcombe, A. L. 1990. *Information and Organizations.* Berkeley, CA: University of California Press.

Wellman, M. 1993. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research* 1:1–23.

Wiederhold, G.; Wegner, P.; and Cefi, S. 1992. Toward megaprogramming. *Communications of the ACM* 33(11):89–99.

Williamson, M.; Decker, K.; and Sycara, K. 1996. Unified information and control flow in hierarchical task networks. In *Proceedings of the AAAI-96 workshop on Theories of Planning, Action, and Control*.

Zilberstein, S., and Russell, S. J. 1992. Constructing utility-driven real-time systems using anytime algorithms. In *Proceedings of the IEEE Workshop on Imprecise and Approximate Computation*, 6–10.