Designing Reusable Behaviors for Information Agents

Keith Decker and Katia Sycara
The Robotics Institute, Carnegie-Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
decker@cs.cmu.edu

January 8, 1996

Abstract

An important issue in multi-agent systems is the specification and effective implementation of various classes of computational agent behaviors. One such class of behaviors involves collecting and supplying information to other computational agents or humans. We call an agent that exhibits such behaviors an *information agent*. Information agents can play an important role in many larger mixed human- and computational- agent organizations. Much previous work has focussed on the language one might use to communicate with such an agent (e.g., KQML), but not on the set of behaviors the agent needs in order to constructively respond to such communications—reusable behaviors that are independent of the particular problem-solving domain. This paper discusses a set of architectural building blocks that support the specification of reusable behaviors for information agents. We present an initial set of information agent behaviors, including responding to one-shot or repetitive queries, proactive monitoring of changing information sources for new occurrences of given patterns, notification of relevant agents, and automatic self cloning of an agent to achieve increased levels of service and efficient use of system resources. We have implemented these reusable information agent behaviors and tested them experimentally on the World Wide Web in several domains: tracking stock prices, extracting news stories, and monitoring airfares.

Content Areas: Distributed AI, Multiagent Systems, Software Agents

Word Count: 6840

Tracking Number: A601
This paper has not been submitted for review elsewhere.

1 Introduction

It has been clear for some time that the Internet is a viable medium for supplying the data needed for making various types of decisions. Most of the interfaces used to access that data, such as WWW browsers, are sensibly constructed to be easily used by humans. As more and more useful data become

available at different times and in multiple locations, however, it becomes more difficult and time-consuming for a person to collect and evaluate that data. Most current agent-oriented approaches to this problem have focussed on *interface agents*—a single agent with general knowledge and capabilities to perform a wide range of user-delegated information-finding tasks (e.g., [7]). We believe that such centralized approaches have several limitations: the need for an enormous amount of knowledge in order to provide coverage for a variety of tasks; the implied centralized processing bottleneck; the inability of most such single agents to deal dynamically with the appearance of new agents and information sources. Another proposed solution is to use multi-agent computer systems to access, filter, evaluate, and integrate this information [16, 14]. Such multi-agent systems can compartmentalize specialized task knowledge, organize themselves to avoid processing bottlenecks, and can be built expressly to deal with dynamic changes in the agent and information-source landscape. Such systems may comprise *interface agents* interacting with an individual user, *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. An information agent is different from an interface agent in that an information agent is tied more closely to the data that it is providing, while an interface agent is closely tied to the goals of a single user. Typically, a single information agent will serve the information needs of many other agents (humans or computational agents). An information agent is also quite different from a typical 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). Information agents can carry out long-term interactions that involve monitoring for particular conditions, as well as simple information updating.

This paper is about the specification of, and experimentation with, reusable behaviors for information agents. By "behaviors", we mean the execution of partially ordered sequences of basic actions. By "reusable", we mean that the behaviors are specified in terms of domain-independent abstractions. There are two categories of information agent behaviors we will consider here: *domain level* behaviors, and *reflective*, performance enhancing behaviors. An example of a domain level behavior is processing an information query; an example of a reflective behavior is self-cloning.

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. Primary interactions with an information agent are then in terms of one-shot, persistent, and change-monitoring queries on the local database. The local database allows all information agents to present an internally consistent, domain-independent interface to other agents and re-use behaviors, even in very different information environments (e.g. streams of numerical sensor information vs. news articles). The local database is constructed from a schema described in a database definition language that we have developed. This schema also specifies the services provided by a particular information agent.

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 information agents. To instantiate a new information agent, one needs only to define its database schema and develop a small piece of site specific code to handle a portion of the external data access. The architecture presented here is consistent with formal BDI agent theory [1, 15].

The agents in our system communicate using KQML [8]. Our focus on long-term behaviors, such as periodic queries and database monitoring, has required us to extend the language with performative parameters to allow the specification of deadlines, task frequencies, and other temporal behavioral constraints. Although we have not added commisives (promises and commitments) [2] to KQML, we (and many others) believe that commitments are the key to coordinated activity in a multi-agent system [1, 12, 5]. Coordination of information agents is accomplished by placing them in an organizational context that provides implicit commitments: each agent takes on an organizational role that specifies certain long-term commitments to certain classes of actions. Thus, these simpler agents can work effectively with one another as well as with more complex agents, such as task agents, that reason about commitments explicitly to produce coordinated behavior [5].

2 A Functional Overview of Information Agents

The main function of an information agent is to process intelligently and efficiently one-shot, periodic, and information monitoring requests. These requests come externally from other agents; the information used to fulfill these requests comes from arbitrary external information sources. Figure 1 shows an abstract view of a single information agent in relation to its environment. An information agent has three distinct types of knowledge: knowledge about potential problem-solving behaviors (including knowledge about how to access the external information sources), a local information database that contains information used to satisfy recent requests by other agents, and information about the agent's current activities and pending requests. The problem-solving knowledge and the processes that support and reason about the other types of information are reusable in different domains, except for the external access interface.

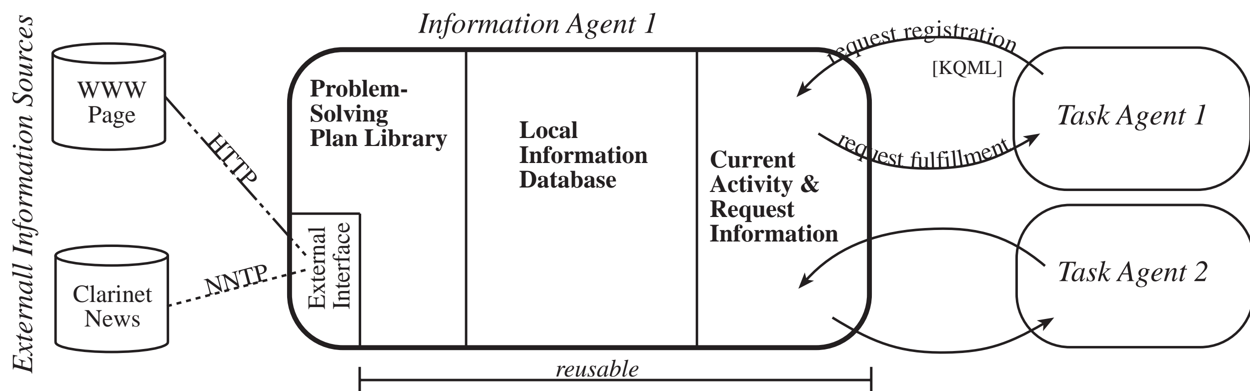

Figure 1: An information agent's knowledge structures and environment.

When an agent is created, it registers itself with some entity such as a communication facilitator [10]. This registration is really an advertisement specifying the information services that the agent is

making available, the associated ontology(ies) and any associated query limitations.[1] This advertisement acts as a commitment by the agent to respond to appropriate requests in the future (see Section 3.3). In Figure 1 Task Agents 1 and 2 have come to know about Information Agent 1 via a facilitator or some other means (see [10, 8, 13]). The task agents interact with the information agent by sending one-shot queries or registering to receive the results of periodic queries and change monitoring queries. An information agent may at any time be responsible for several active queries from each of several agents.

As shown in Figure 1, an information agent has three conceptual functional parts: the agent's current activity information, the agent's local database, and the problem-solving library that includes site-specific external interface code. The current activity information supports agent communication and planning. It keeps track of which agent has registered for which kind of information service, and with what reply deadline. It supports scheduling of the tasks necessary to fulfill an information request and monitoring of the execution of a current action. The information agent's knowledge about its current activities and responsibilities is also important because it is crucial to the agent's ability to reflect, introspect and adapt its behavior (e.g., by self-cloning or politely refusing requests, see Section 4).

The agent's problem-solving part contains the plan library and some site-specific interface code. Plan fragments, consisting of task structures and indexed by information requests/goals, are retrieved from the library by the planner (see Figure 2) and instantiated. The actions in the task structures are then scheduled and executed. The way that an information agent accesses external information sources and creates local database records from them in response to some requesting agent(s) query(ies) is called the external interface and is the only non-reusable portion of an information agent's knowledge structures. However, the bulk of an information agent's knowledge about possible problem-solving actions—behaviors—is reusable. This includes the behaviors of registration, listening for messages, and accepting, recording, running, and responding to queries (see Section 4).

The information agent's local information database contains records that have been retrieved from external information sources in relation to one or more queries. This local database is important because it allows the information agent to become more than just a fancy database wrapper. In an information agent, retrieval of external data is separate from query processing. This allows for a domain-independent specification of an information agent in terms of the abstract schema of its local database, it allows an information agent to potentially tie together multiple external information sources, and also provides three performance enhancements. First, multiple related queries can be grouped to limit access to the external information source and free up processing bandwidth. Second, the local database acts to buffer the requesting agents from unexpected problems with the external information sources. Third, the local database provides attribute enhancement such as storing historical data for each record field.

3 Agent Architecture: Building Blocks for Agent Behavior

The design of reusable information agent behaviors rests on a deeper specification of agents themselves, and is embodied in an agent architecture. The information agent architecture is a simplification of the DECAF (Distributed, Environment-Centered Agent Framework) architecture [6]. The task structure

[1] The reader may have already guessed that such a communication facilitation agent, which provides yellow pages information, i.e. a database associating information agents and the information that they provide, is itself an information agent that we have implemented. One of the reusable behaviors inherited by this matchmaker is the ability to process persistent queries as described in [13].

representation is simplified to use only one level of abstraction and only one coordination relationship between tasks (*enables*). This allows us to use much simpler plan retrieval and scheduling algorithms, at the cost of disallowing complex, multi-level, abstract plans and sophisticated coordination across plans. These tradeoffs seem to be reasonable for an information agent (where they would not be for a higher-level task agent). Our information agent architecture also extends the TÆMS task structure representation [4] to allow periodic, repeating actions.

3.1 Control: Planning, Scheduling, and Action Execution

The simplified control process for information agents includes steps for planning to achieve local or non-local goals, 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 goals to poll for messages from other agents and register itself with a KQML facilitator. 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 Section 3.3.

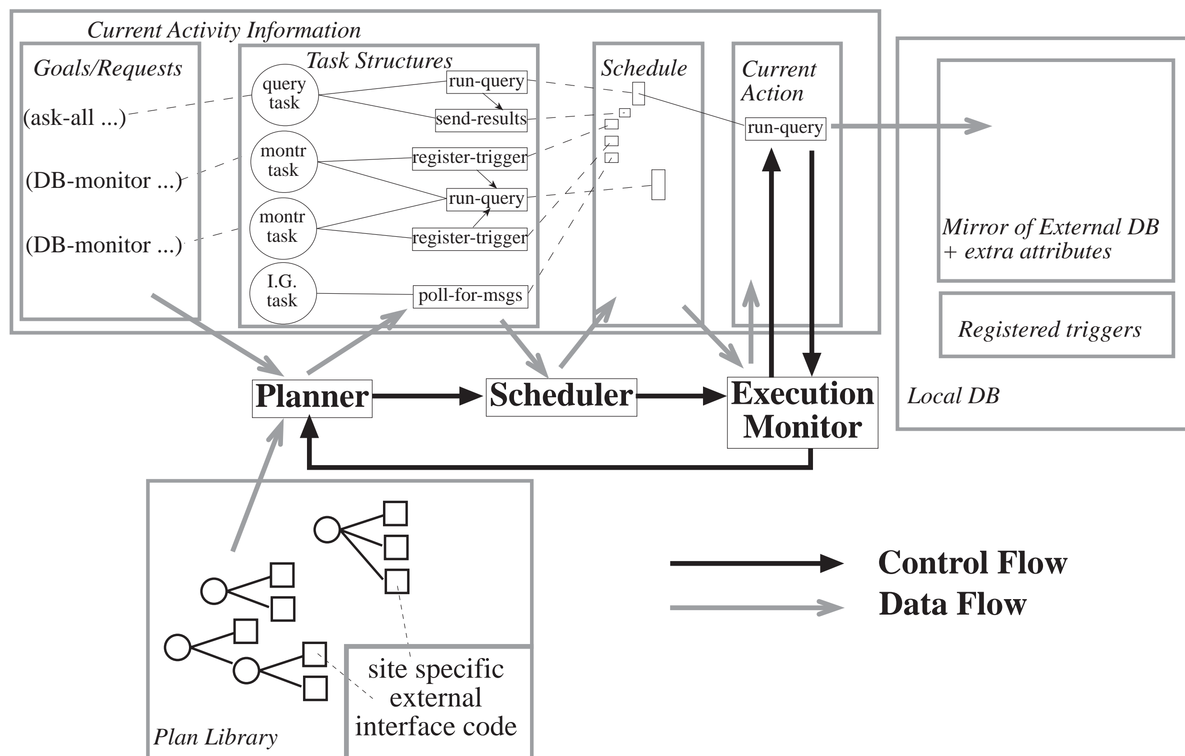

Figure 2: Overall view of data and control flow in an information agent.

The agent planning process (see Figure 2) 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. Information agents use simple task structures (compare to [4]):

- 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 basic actions $\mathbf{A}$. These actions are ordered on a single relation: *enables*. If action A_1 enables action A_2 then A_1 must explicitly signal that A_2 may execute. 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 [1]. 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").

In our initial implementation, the information agent planning component is a simple plan retrieval mechanism that instantiates a new task structure for each goal, i.e. for each query request.

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)
- 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) [9].
- 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 enabled if so indicated, and statistics collected.

3.2 Knowledge: Goals, Local Information DB, and Plan Templates

There are three major components to an information agent's knowledge and belief structures: the representation of the agent's current goals, the agent's internal representation of information present in the external database(s) with which it is associated, and a library of plan templates indexed by goal type. These plan templates are retrieved and instantiated to provide task structures that are subsequently scheduled and executed as explained in the previous section. The execution of these plan fragments composes the agent's behaviors.

Each of the agent's external goals (e.g., one-shot query, etc.) has associated information about the agent making the request, the KQML reply-with ID, the communication format desired for the reply, and the query itself. Other goals, such as polling for KQML messages ("I.G. task" in figure 2), are internal and are initiated by the agent itself. In fact, external goals are added to the agent by the actions of the message-polling task.

The agent's local database is defined in terms of an *ontology*, a set of *attributes*, a *language*, and a *schema*. The database ontology links concept-names to their data types and domain-specific meaning (and must match on any incoming KQML message).[2] Database *attributes* are meta-informations stored about a field in a database record. By default, information agents will keep track of two attributes: timestamp and previous value. The timestamp indicates when the field value in the record was last updated. Note that when a local agent database is formed from information gathered from multiple external information sources, the timestamps on every field in a record may be different. The previous value is the value the field had before the last database update. No absolute temporal relationship between the current value and previous value can be assumed—only the relative temporal relationship ("before").[3] The database *query language* specifies a pre-determined format for the KQML "content" slot. The "simple-query" query language is a set of predicate clauses on field values and attributes, joined with an implicit "AND". Other database description modifiers, such as advertised monitoring costs (charging different amounts depending on how frequent updates need to be) could be added.

The local database is constructed from a database definition language description. This description also serves as a way to describe the services a particular information agent offers to other agents. The important parts of the description, defined for each field in the database, are:

concept-name: the column or field name, the term referring to what the value in the database represents. A concept-name plus the ontology implies a data-type (string, integer, float, data/time,

[2] This paper will not discuss ontological representation, see for example [11].

[3] This restriction can be removed either by adding another reusable behavior to an information agent that creates in effect a new "value-history" attribute, or by simply adding value-histories to the local database definition itself. The first alternative is more conservative of space.

enumerated, etc.)

advertised flag: A field is flagged "advertised" if it possible to use that field to select records in a query. If the external database is a full-fledged modern relational DB, then usually all fields will be flagged "advertised". However, many of the databases available over the WWW do not allow record selection by every field, and so the agent needs to include this in its specification.

unique flag: A field is flagged "unique" if every record in the database must differ on that field. Every database definition must have at least one unique field. This field does not have to be advertised. If necessary, it can be internally generated.

predicates: *optional.* If present, it limits the predicates that can be used in queries on this field. If absent, all of the natural predicates associated with the data-type for the concept-name can be used. Typically, a key field might limit the operators to EQUAL only.

range: *optional.* Gives information on any range limits associated with the field in question. If not present, the natural limits for the field's concept-name are assumed.

For example, the Security APL server is an Internet information source that has stock prices. The only key is a company's ticker symbol. The following table shows the Security APL data for the General Host Corporation (ticker symbol GH).

Symbol	GH
Exchange	New York Stock Exchange (NYSE)
Description	GENERAL HOST CORP
Last Traded at	5.6250
Date/Time	Oct 11 11:52:45
$ Change	-0.1250
% Change	-2.17
Volume	13100
# of Trades	6
Day Low	5.6250
Day High	5.7500
52 Week Low	3.7500
52 Week High	7.6250

Our Security APL DDL specification is:

```
(DATABASE security-apl
 :ONTOLOGY financial-stocks
 :ATTRIBUTES previous-value timestamp
 :QUERY-LANGUAGE simple-query
 :SCHEMA
  (symbol :ADVERTISED :UNIQUE :PREDICATES =)
  (company :ADVERTISED)
  (exchange :RANGE (NYSE AMEX NASDAQ))
  (reference-date :RANGE *today*)
  (price )
  (volume )
  (day-high )
  (day-low )
```

```
  (52-week-high )
  (52-week-low )
  (beta )
  (shares )
  (dividend-yield )
  (EPS )
  (market-capitalization )
)
```

A one-shot query specification of “give me the most recent GH record” is:

```
(query security-apl :CLAUSES (eq $symbol "GH") :OUTPUT :ALL)
```

A monitoring query specification of “tell me whenever Oracle reaches a new 52-week high price” is:

```
(query security-apl
   :CLAUSES
   (eq $symbol "ORCL")
   (> $52-week-high (previous-value $52-week-high))
   :OUTPUT :ALL)
```

For a different information domain, such as news articles, a new ontology must be specified, but an information agent’s database specification can be done in similar way to the one for security APL. The following specification is for the Dow Jones news feed.

```
(DATABASE dow-jones-news
 :ONTOLOGY news
 :ATTRIBUTES nil
 :QUERY-LANGUAGE simple-query
 :SCHEMA
  (newsgroups :ADVERTISED :RANGE "dow-jones.*")
  (message-id :UNIQUE)
  (subject)
  (date :RANGE (> (- *today* 5days)))
  (body))
```

This is an example of a monitoring query KQML message (including sender and receiver agents) for every article on IBM earnings from the dow jones news.

```
(monitor
 :SENDER barney
 :RECEIVER news-agent
 :LANGUAGE simple-query
 :ONTOLOGY news
 :REPLY-WITH ibm-query-2
 :NOTIFICATION-DEADLINE (30 minutes)
 :CONTENT (query news
                :CLAUSES
                (=~ $newsgroups "dow-jones.indust.earnings-projections")
                (=~ subject "IBM")
                :OUTPUT :ALL))
```

Creating an instance of a totally new information agent for an information source in any domain requires only that the agent be provided with a database schema definition as described above, and an external query function. Everything else is shared and reused between information agent instances (and thus improvements and new functionality are shared as well). We created information agents to answer queries on current stock prices (from publicly available delayed ticker feeds), Usenet news articles (including CMU's Clarinet and Dow-Jones news feeds), lowest-cost airfares, and the capabilities of other agents (matchmaking).

3.3 Communication: Making, Breaking, and Fulfilling Commitments

The final part of our information agent architecture is a set of communication principles that constrain how and when an agent accepts goal requests, and also constrain the behaviors used by the agent to carry out those requests. How can we guarantee that information agents act in ways that are predictable and useful enough for them to take part in larger multi-agent systems and organizations? We, along with others believe that commitments of various types are the key to coordinated multi-agent behavior [12, 1, 5, 15]. KQML, however, has no commisive performatives, and we have not added any. In the case of information agents, we resolve this clash by designing information agents to abide by a set of *implicit* commitments. First, such implicit commitments allow us to build organizations of information agents that work together in a coordinated fashion. Second, they allow more sophisticated task agents that reason about commitments explicitly to dynamically recruit the services of information agents.

An information agent sends and receives three basic classes of messages: those about goal creation, those about goal cessation, and transmitting results (non-local action enablement). Both goal creation and cessation may be assertive (telling another agent a belief) or directive (requesting another agent to do something: achieve a goal, answer a query in a certain way etc) speech acts[2]. Here is a list of these basic communicative classes and their meaning:

Assertion of goal creation: An information agent may assert to another agent that it has a goal (i.e., to answer any queries on its database). This is the implicit commisive communication. When an information agent advertises its database by sending the database specification to, for example, some facilitator, it is making an implicit commitment to accept requests on the class of query goals valid for that database. Any other agent can then assume that if it sends a query request to that agent, the agent will accept that goal and work for its achievement until it is either achieved or the accepting information agent believes it cannot be achieved.

Request for goal creation: An agent may request that an information agent adopt the goal of answering some query Q, e.g., "Tell me the price of IBM every 10 minutes".

Assertion of goal dissolution: An information agent may assert that a goal, accepted from another agent, is being removed. This assertion normally carries with it a reason—typically, the query was malformed, the agent is unable to meet it's implicit commitment because it is too busy, or the agent is shutting down due to circumstances beyond its control. "I can no longer provide the price of IBM every 10 minutes due to a shutdown for preventive maintenance".

Request for goal dissolution: An agent my request that an information agent give up a goal previously requested by the agent. "Cancel the goal of telling me the price of IBM every 10 minutes".

Assertion of an action result: Such an assertion, for example "The price of IBM at Dec 21 9:51:46 AM is 90 5/8". may result in non-local enablement of an action at the requesting agent

Information agent behaviors that deal directly with goals from other agents, then, must obey these implicit commitments. They can do this because they are built on an agent architecture that provides planning and scheduling mechanisms that detect or predict action failures.

4 Reusable Behaviors

An information agent behavior is a particular approach to accomplishing a goal. Behavior instances are represented by a task instance, a set of basic action instances, and the relationships between them. We describe an information agent's reusable behaviors.

4.1 Message Polling

Message Polling is the simplest information agent behavior. The information agent initialization process asserts a goal for the agent to collect and process incoming KQML messages, for which the planner retrieves a simple task structure with one periodic action. This task does non-blocking checks on the agent's incoming network communication channel and retrieves any messages that have arrived since the last periodic invocation. Each message is parsed at the KQML level and stored as a planning goal.

This basic behavior is necessary for guaranteeing that the agent keeps its (implicit) commitments. Remember that by advertising its database, an information agent makes an implicit commitment to accept queries on the database (requests-for-goals) from other agents. This behavior makes sure that any communications that contain such requests actually cause the creation of the necessary goal.

4.2 Answering Simple Queries

A simple query is one that is simply applied to a database and the results returned to the query-initiator. The query might be a one-shot question or it might be a request for periodic query applications with a given frequency. A task to respond to a simple query consists of two actions: running the query and sending the results to the query-initiator; the running action enables the sending action. The only difference between one-shot and periodic simple queries is whether the constituent actions are themselves periodic.

Running a query consists of passing the query to the external interface. When the external interface process returns, the local DB is assumed to be updated to contain any records that have been returned by the query. Next, any database triggering conditions are checked (see the next section, "Database Monitoring"). The query is then run on the local database, and the resulting selected records (if any) become the "result" of the action and enable the execution of the send-results action whenever it is scheduled to occur. The send-results action then packages up the result records according to the output specification in the :CONTENT part of the original query, and then sends out the properly formatted KQML REPLY messages depending on the original query performative (ask-one, ask-all, stream-all, etc.) and any :REPLY-WITH information.

4.3 Database Monitoring

A database monitoring query is one that is interpreted as expressing a condition that when true will trigger the transmission of a selected record or records. The default task to respond to a database monitoring goal is a slightly different behavior from the response to a simple query. It consists of three actions: run-query and send-results, which are the same as in a periodic simple query, but also a check-trigger action. Unlike in the simple query task, send-results is enabled by check-trigger, not run-query. The check-trigger action is effectively run anytime the local database is updated (i.e., whenever any run-query is executed).

4.4 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 modeling and decision process uses the methodology developed in [3].

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 5.

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.[4] Thus,

[4] Another way to recoup this time is to run the blocking external query in a seperate 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.

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 ith 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 5. For two agents, the idle percentage should then follow the model

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

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

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

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

On the left of Figure 4 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.[5] 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

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

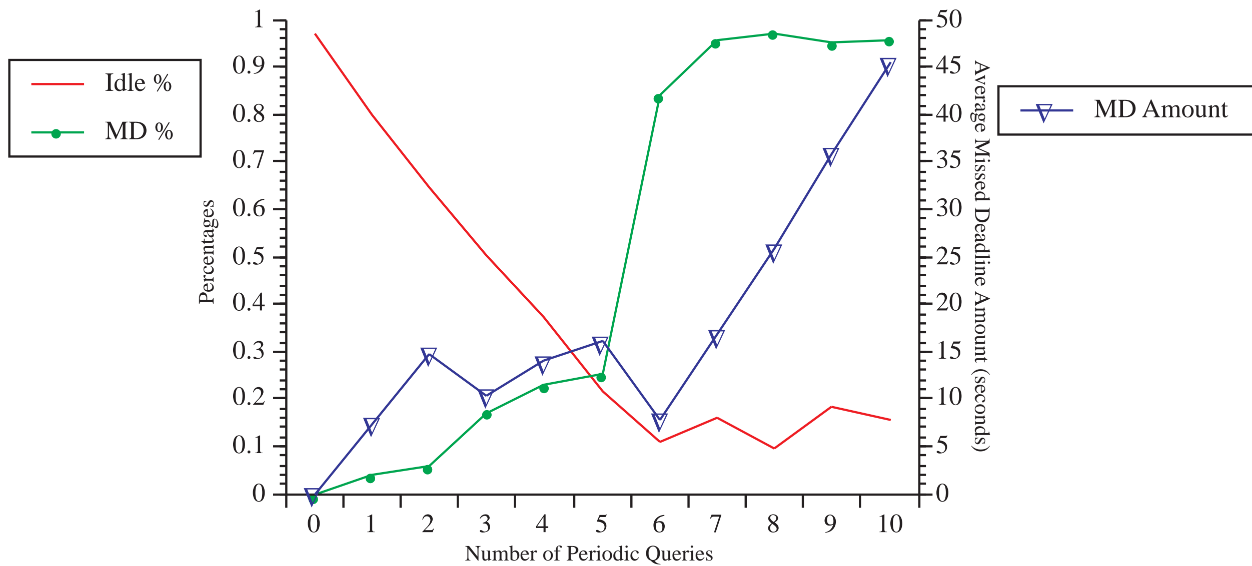

Figure 3: A graph of the average percentage idle time and average percentage of actions with deadlines that missed them for various loads (left Y axis). Superimposed on the graph, and keyed to the right axis, are the average number of seconds by which a missed deadline is missed.

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 4. 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 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.[6]

[6] 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.

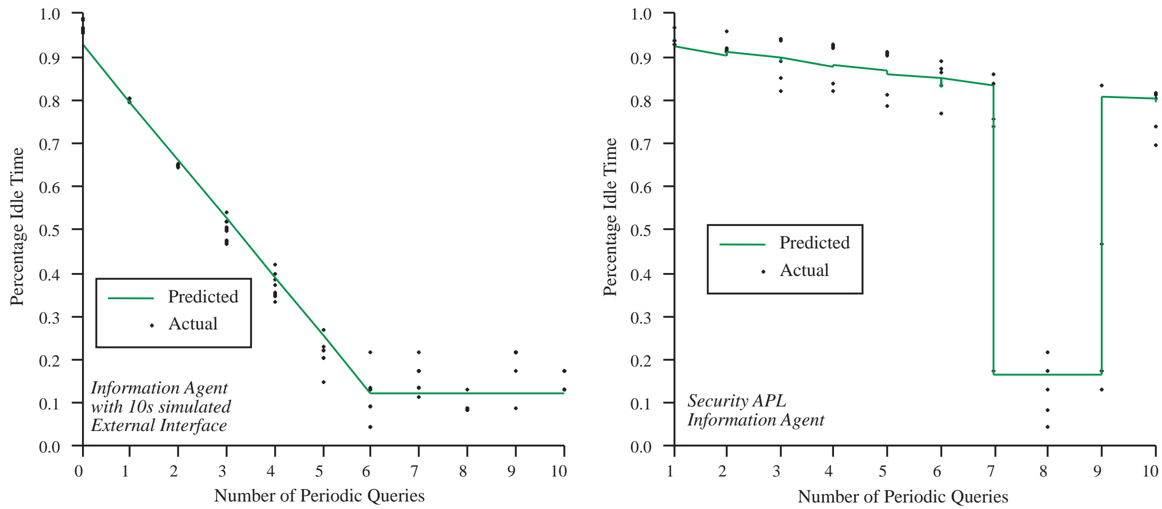

Figure 4: On the left, graph of predicted and actual utilization for a real information agent with a simulated external query interface. On the right, the same graph for the Security APL stock ticker agent.

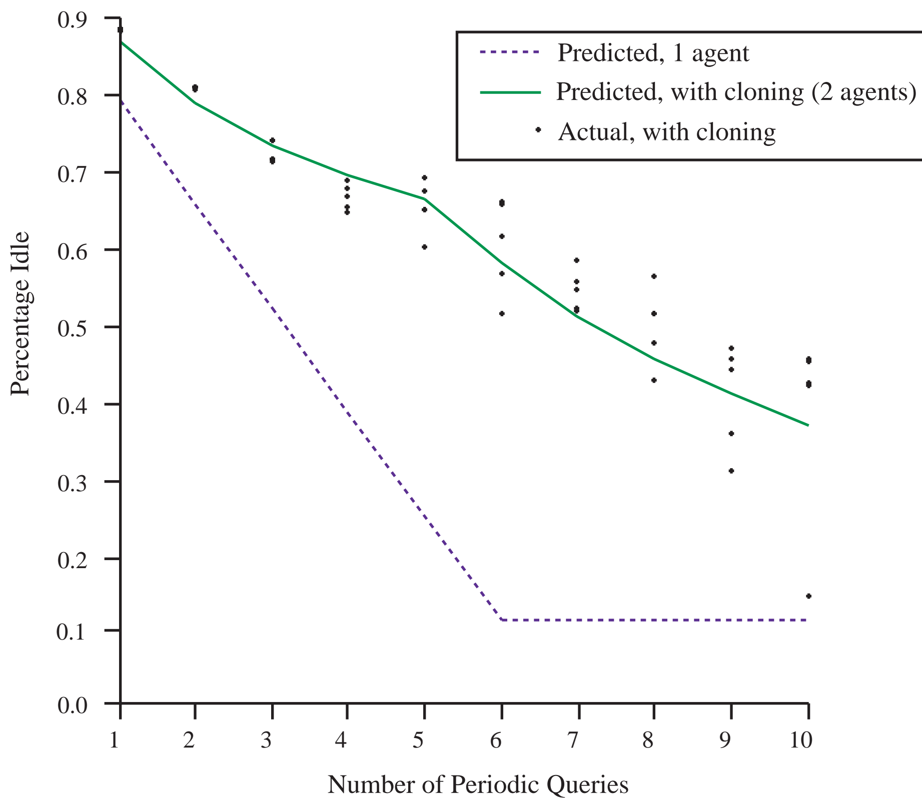

Figure 5: Predicted idle percentages for a single non cloning agent, and an agent with the cloning behavior across various task loads. Plotted points are the measured idle percentages from experimental data including cloning agents.

6 Conclusions

This paper has discussed the design and implementation of *information agents*, a class of autonomous computational agent that is tied closely to a source or sources of data. Information agents are just one part of larger multi-agent systems that may include independent task agents and interface agents responsive to the goals of a human user. We presented a detailed, existing architecture for these information agents. Each component provides functionality that assures that an information agent will accept properly stated goals from other agents and will work toward their achievement unless they prove impossible. With the addition of a few other components (for coordination and decision-making) such an architecture can support a full range of sophisticated autonomous agent behaviors. We also discussed how information agents obey a set of implicit commitments to carry out a certain class of actions, and can thus take part in more complex multi-agent systems.

Using this architecture as a base, we then described a set of reusable behaviors for information agents. Such behaviors are reusable in the sense that they are described and implemented independent of the underlying knowledge domain. An important goal of this work was to develop such a set of behaviors so that new information agents can be created rapidly, even dynamically, with little or no programming. Such high levels of reusability have additional advantages—as new behaviors or refinements to old ones are developed, all information agents in all domains benefit. Our initial basic set of information agent behaviors includes polling for messages, answering one-shot queries, reporting the results of a repeated query, monitoring a database for some change or new pattern, and self-cloning. Finally, we presented and empirically verify a simple load model that is used to drive the agent's cloning behavior.

We believe that this work can serve as one bridge between the software agent community and existing DAI agent theories. We hope that this paper will encourage the discussion and analysis of reusable agent behaviors for information agents and other general classes of agents.

References

[1] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.

[2] Philip R. Cohen and Hector 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Oren Etzioni and Daniel Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37(7), July 1994.

[8] 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.

[9] 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, D.C., July 1993.

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

[11] Tom R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Technical Report KSL-93-4, Knowledge Systems Laboratory, Stanford University, 1993.

[12] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[13] 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.

[14] 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.

[15] 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.

[16] Katia Sycara and Dajun Zeng. Task-based multi-agent coordination for information gathering. In *AAAI Spring Symposium on Information Gathering in Distributed Environments*, Stanford University, March 1995.