Designing Behaviors for Information Agents

Keith Decker
Computer and Information Sciences
University of Delaware
decker@cis.udel.edu

Anandeep Pannu, Katia Sycara and **Mike Williamson**
The Robotics Institute, Carnegie-Mellon University
(pannu,mikew,sycara)@cs.cmu.edu

Abstract

To facilitate the rapid development and open system interoperability of autonomous agents we need to carefully specify and effectively implement various classes of agent behaviors. Our current focus is on the behaviors and underlying architecture of WWW-based autonomous software agents that collect and supply information to humans and other computational agents. This paper discusses a set of architectural building blocks that support the specification of behaviors for these *information agents* in a way that allows periodic actions, interleaving of planning and execution, and the concurrent activation of multiple behaviors with asynchronous components. We present an initial set of information agent behaviors, including responding to repetitive queries, monitoring information sources, advertising capabilities, and self cloning. We have implemented and tested these behaviors on the WWW in the context of WARREN, an open multi-agent organization for financial portfolio management.

Introduction

As the number of agents that appear on the Web rapidly increases, it is clearly desirable to develop frameworks for rationalizing agent engineering, (a) so each agent does not have to be constructed from scratch in ad hoc ways, and (b) to facilitate the interaction and interoperability among agents that have been developed by different designers. One way to make progress towards this goal is through the specification of reusable agent architectural components that support reusable agent behaviors. By "behaviors", we mean the execution by an agent of partially-ordered sequences of basic actions. By "reusable", we mean that the behaviors are specified in terms of domain-independent abstractions and can be reused in building an agent for a new domain or task. In our view, each agent category (e.g. information agents) will have a fixed set of behaviors that characterize it. These behaviors are supported by the agent's architectural infrastructure consisting of a "standardized" generic set of components. Such an agent may perform several instantiated behaviors concurrently, and each behavior may consist of possibly asynchronous components spread out in time. We believe that specifying a generic set of architectural agent components and agent behaviors for each class of agents is important because they can tremendously reduce programming burden, they can facilitate understanding of agent characteristics thus leading to a typology of agents, and they can support interactions among agents that have been developed by different agent designers.

Our framework explicitly admits the notion that agents are part of multi-agent systems. This has the implication that agent coordination is not an afterthought but part of our system and agent architectural commitments. 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 environment. Our framework comprises *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 (Sycara & Zeng 1996). These three broad agent categories share common architectural components but have different sets of agent behaviors. The behaviors in each class of agent are reusable. We have identified the behaviors that would be desirable for an autonomous information agent functioning in the context of the WWW. An agent has to provide a description of the information that it provides, requiring what we call an *advertising behavior*. It must be constantly aware of its environment and other agents. On the Internet this means that it has to rely on network communication using its *message polling* behavior. The capability of responding to all kinds of information requests whether transient, periodic or based on conditions on the underlying information, needs to be encapsulated within the agent. *Information monitoring* and *query answering* are behaviors that fulfill this need. Efficiency considerations are part and parcel of the agent and these are taken into account while serving up information. When an agent is overloaded, it can clone itself to share its' load (*cloning behavior*). In this paper we give a brief description of the underlying generic architectural agent infrastructure and then focus on how we achieved the specification of reusable behaviors for information agents.

In particular, we will focus on the behaviors of *basic informa-*

tion agents that encapsulate a single (or very closely coupled) information source; we will also briefly discuss how these can be extended into a class of *multi-source information agents*. Each class of information agents has a fixed set of behaviors. When a new information agent is created/instantiated the programmer does *not* choose or program behaviors, instead he/she specifies a domain-level data model and a domain-specific access method.This greatly simplifies the process of creating new information agents and facilitates agent interactions.

Information agents do not function in a vacuum. They have to make themselves known as information providers to the larger agent society. When an agent is created, it advertises itself to some entity such as a matchmaker or broker (Genesereth & Katchpel 1994; Finin *et al.* 1994; Kuokka & Harada 1995; Decker, Williamson, & Sycara 1996b). This advertisement, expressed in terms of an agent's information base schema, specifies the information services that the agent is making available, the associated ontology(ies) and any associated query limitations. This advertisement acts as a commitment by the agent to respond to appropriate requests in the future. In general, the process of matchmaking allows one agent with some objective to learn the name of another agent that could take on that objective. The reader may have already guessed that the matchmaker agent, which provides yellow pages information (agent name-finding), i.e. associating information agents and the information that they provide, is itself an information agent. One of the reusable behaviors inherited by a matchmaker is the ability to process persistent queries as described in (Kuokka & Harada 1995).

The agents in our system communicate using KQML (Finin *et al.* 1994). Our focus on long-term behaviors, such as periodic queries and information monitoring, has required us to extend the language with performative parameters to allow the specification of deadlines, task frequencies, and other temporal behavioral constraints. 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 (see our discussion of Advertising). 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 (Decker & Lesser 1995). In particular, in the WARREN portfolio management multi-agent system, basic and multi-source information agents work alongside more complex task- and interface-agents.

We have used (and reused) the architectural components and agent behaviors for multiple, coordinating agents in various domains. These domains span projects such as PLEIADES, in the domain of organizational decision making (Sycara & Zeng 1996) and WARREN, in the financial portfolio management domain (Decker, Sycara, & Zeng 1996)

A Functional Overview of Information Agents

Typically, a single information agent will serve the information needs of many other machine or human agents. An information agent is 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).

The main function of an information agent and hence its dominant domain-level behaviors are: retrieving information from external information sources in response to *one shot* queries (e.g. "retrieve the current price of IBM stock"); fulfilling 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 *information monitoring* requests, (e.g. "notify me when IBM's price increases by 10% over $80"). Multi-source information agents add the capability of breaking apart a request and planning for its efficient fulfillment, potentially using multiple information sources through interaction with basic information agents. In either case these requests come externally from other agents. In Figure 1, Task Agents 1 and 2 have come to know about Information Agent 1 via a matchmaker. The task agents interact with the information agent by sending one-shot queries or *registering* to receive the results of periodic queries and information monitoring queries. An information agent may at any time be responsible for several active queries from each of several agents.

Since our agents operate on the Web, the information they access is from external information 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. We call this local agent database, the agent's "infobase" to avoid confusion with external data bases. The agent's information processing mechanisms then process the information in the infobase to service information requests received from other agents or human users.

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 (Kim & Seo 1991). 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 (Collet, Huhns, & Shen 1991) 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. (Gruber 1993)) is still a difficult one and is outside the scope of this paper.

Figure 1 shows an abstract view of a single information agent in relation to its environment. An information agent has three conceptual functional parts: the agent's current activity information, the agent's local infobase, and the problem-solving library that includes site-specific external interface code. The current activity information supports agent communication

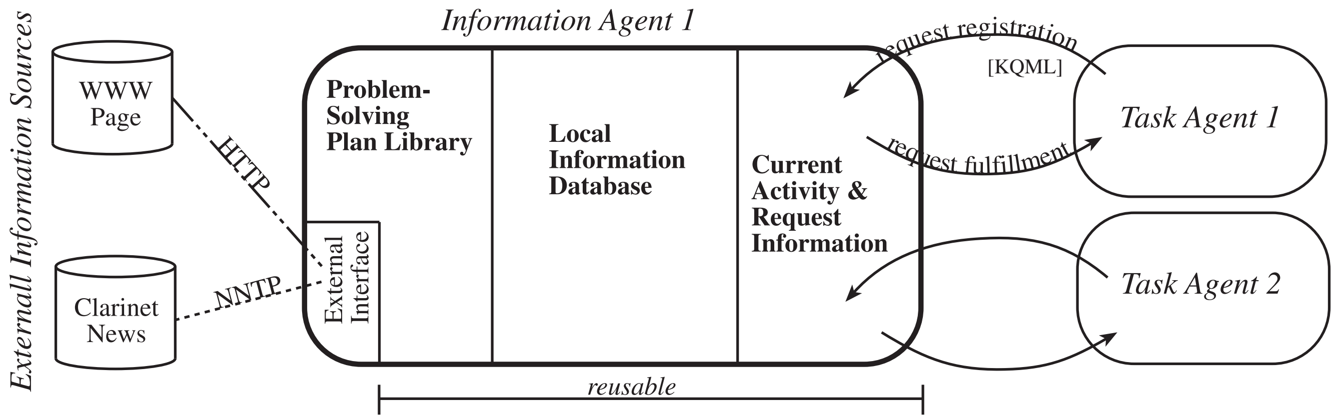

Figure 1: An information agent's knowledge structures and environment.

and planning. It keeps track of which customer 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).

The agent's problem-solving part contains the planner's task reduction library and some site-specific interface code. Task reductions, indexed by the goals they achieve, are retrieved from the library by the planner (see Figure 2) and instantiated. The primitive actions in the instantiated reductions are then scheduled and executed. The way that an information agent accesses external information sources and creates local infobase records from them in response to a 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 activities—behaviors—is reusable, i.e. domain independent and fixed for the class of information agents. This includes the behaviors of advertising, listening for messages, and accepting, recording, running, and responding to queries.

The information agent's local infobase contains records that have been retrieved from external information sources in relation to one or more queries. 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 infobase. Since agent advertisement is done with the local infobase schema, the infobase allows all information agents to present a consistent, domain-independent interface to other agents that specifies the services provided by the agent. In addition, 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 infobase acts to buffer the requesting agents from unexpected problems with the external information sources. Third, the local infobase provides attribute enhancements, such as storing historical data for each record field. This type of caching is not without tradeoffs; it uses more memory, and one must be careful to not introduce inconsistencies between the external source and the cache. A recent discussion of some of these tradeoffs in the context of higher-level multi-source information agent caching can be found in (Arens & Knoblock 1994).

Agent Architecture: Building Blocks for Agent Behaviors

An information agent's reusable behaviors are facilitated by its reusable agent architecture, i.e. the domain-independent abstraction of the local infobase schema, and a set of generic software components for knowledge representation, agent control, and interaction with other agents. The generic software components, based on DECAF (Decker & Lesser 1995; Oates *et al.* 1995), are common to all classes of information agents as well as to the task and interface agents we have built. The architecture presented here is consistent with formal BDI agent theory (Cohen & Levesque 1990; Rao & Georgeff 1995). The larger part of the architecture (outside of the local infobase cache) is shared by *all* of our software agent classes. This differentiates our approach from approaches such as SIMS (Knoblock, Arens, & Hsu 1994) that are focused on providing only what we would call multi-source information agent behaviors (discussed later). 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]

[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 (Etzioni, Lesh, & Segal 1992), rather

The control process for information agents includes, *communication* of planning goals, *planning* to achieve internal or external goals, *scheduling* the actions within these plans, actually carrying out these actions (see Figure 2), and *monitoring action execution*. In addition, the agent has a shutdown and an initialization process. At startup, the agent executes the initialization process which bootstraps the agent by giving it initial goals, i.e. to poll for messages from other agents and to advertise itself with a matchmaker or broker. The shutdown process is executed when the agent either chooses to terminate or receives an error signal. The shutdown process sends messages from the terminating agent to any current customers and the matchmaker or broker informing them of service interruption.

An information agent's external goals, i.e. answering a one-shot query, setting up periodic queries and monitoring information changes are communicated to it in KQML messages originating from other agents. The communication module receives and parses messages, extracts the information goals and passes them to the planner.

Planning

The agent planning process is based on a hierarchical task network (HTN) planning formalism. It takes as input the agent's current set of goals, the current set of task structures, and a library of task reduction schemas. A task structure is a partially instantiated task network, including inter-task information flow relationships and descriptions of potential task outcomes. A task reduction schema presents a way of carrying out a task by specifying a set of sub-tasks/actions and describing the information-flow relationships between them. That is, the reduction may specify that the result of one sub-task (e.g. deciding the name of an agent) be provided as an input to another sub-task (e.g. sending a message). Such relationships are represented as *provision links* within an instantiated task network. Actions may require that certain information be provided before they can be executed, and may also produce information upon execution. For example, the act of sending a KQML messages requires the name of the recipient and the content of the message, while the act of deciding to whom to send some message would produce the name of an agent. An action is *enabled* when all the required inputs have been provided. (See the example reduction in Figure 3, or see (Williamson, Decker, & Sycara 1996) for a complete description of our task structure representation.) The task reduction schemas are retrieved and instantiated by the planner to provide task structures that are subsequently scheduled and executed. The execution of the resulting task structures composes the agent's behaviors. Thus, the specification of a particular set of task reduction schemas defines a class of behaviors which will be shared by all agents which have those schemas in their libraries.

Since arrival of goals, planning and execution are on-going and interleaved, the planning process modifies the current set of task structures—by removing tasks, further reducing existing tasks, or instantiating new tasks—to account for any changes to the agent's goals. Task reduction is incremental, and is interleaved with execution.

Scheduling

The agent scheduling process takes as input the agent's current set of task structures, particularly the set of all primitive actions, and decides which primitive 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 complete execution before their deadline.
- Actions that miss either their period or deadline are considered to have failed.
- 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.

Execution Monitoring

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 provided to downstream actions by provision links in the task network, and statistics collected.

The Local Agent Infobase

The agent's local infobase is defined in terms of an *ontology*, a set of *attributes*, a *language*, and a *schema*. The infobase ontology links concept-names to their data types and domain-specific meaning (and must match on any incoming KQML message). Infobase attributes are meta-information stored about a field in a 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 infobase is formed from information gathered from multiple external

than by adding a seperate local scheduling component.

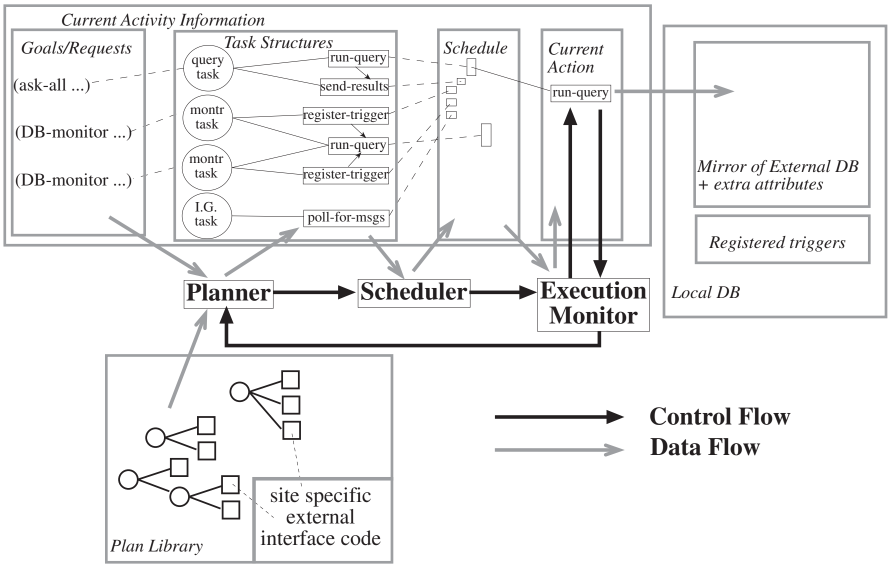

Figure 2: Overall view of data and control flow in an information agent.

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 infobase update.

The local infobase is constructed from a simple data definition language description, which includes the meta-information mentioned above and a fairly traditional database schema. This description also serves as a way to describe the services a particular information agent offers to other agents. The important parts of the schema description, defined for each field in the infobase, are:

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

selectable flag: A field is flagged "selectable" if it possible to use that field to select records in a query. This is especially important when dealing with WWW information that is *not* interfaced to a traditional database. For example, our ticker agents must indicate that they do handle queries like "Tell me all the stocks that cost $30".

primary key: Indicates that this field is the primary key of the local agent infobase.

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.

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 one record, 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 (the local infobase schema + meta information) is:

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

Information agents receive queries in KQML messages.

Since an information agent does not have control over an external information source, it executes the query on the cached information in its local infobase. The query is expressed in the :CONTENT slot of the message. The "simple-query" query language is a set of predicate clauses on field values and attributes, joined into an implicit conjunction.

An example of a monitoring query, "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 the local infobase specification and the underlying infobase manipulation functions are reusable. As an example of a new infobase specification, we present the Dow Jones news feed that provides unstructured text information.

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

Below, we give 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 infobase schema definition as described above, and a site-specific function for querying new external information source(s).[2] Everything else is shared and reused between information agent instances (and thus improvements and new functionality are shared as well).

[2]This is similar to SIMS (Knoblock, Arens, & Hsu 1994).

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 sub-tasks or primitive action instances, and the information-flow relationships between them. In this section we describe an information agent's fixed set of reusable behaviors.

As described above, the task structure for a given goal is instantiated incrementally and dynamically by the planner. The scheduler then determines when individual action instances should be executed, depending on their deadline and on the availablity of required information. As actions are executed and new information is produced, it is routed (as stipulated by information-flow links in the task structure) to other tasks. Thus, the control flow of individual actions within a behavior is determined by information flow.

Advertising Behaviors

Upon startup, every agent creates an internal goal to advertise itself. An agent advertises itself by sending a matchmaker or broker its infobase schema, which contains all the information needed for describing the information services that the advertising agent can provide. Figure 3 shows the task structure which results from the planner's standard reduction of the "advertise" task.

The reduction is best understood in two parts: the first is the initial communication to the matchmaker, the second contains the actions associated with the "shutdown" of the advertise task (Make Un-Advertisement and the SendKQML on the lower right). The three actions "Make Advertisement", "Get Matchmaker Name", and the topmost instance of "SendKQML" are involved in sending the advertising message. 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 infobase schema plus execution information gathered and persistently stored from previous invocations) and provides it to SendKQML. In our current implementation, the typical first reduction for "Get Matchmaker Name" is to use a predefined name.[3] The matchmaker name is supplied as the RECEIVER of the KQML message, and the REPLY-WITH field is supplied by the planner at task-reduction time. If no matchmaker currently exists or some other network error occurs, the SendKQML action 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 the system is robust in the face of appearance or disappearance of agents.)

The two action instances "Make Un-Advertisement" and the second, lower right SendKQML instance comprise the *shutdown* actions for this task. This allows the agent to be shutdown gracefully and inform any other agents that it has

[3]In our system, the matchmaker is the only agent with a known name. The default matchmaker name can be changed by a Unix environment variable, thus our group can have agents working in multiple logical agent namespaces even though they share a small set of physical processors.

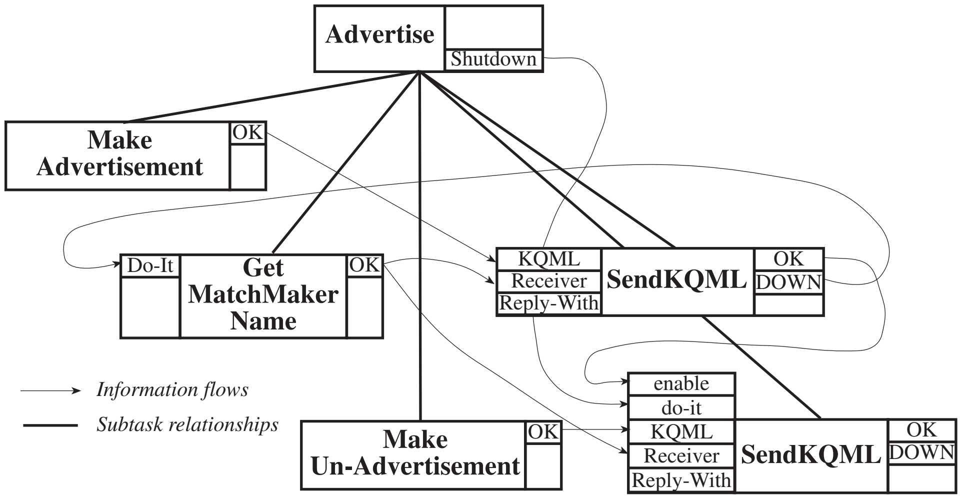

Figure 3: The standard task reduction for the "advertise" task.

made a commitment to of its inability to meet the commitment. This basic advertising behavior is shared (reused) for all information agents (simple and multi-source).

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. The Message Polling task reduction has only one periodic action, "poll-for-messages", which is instantiated by the planner. Executing the action results in non-blocking checks on the agent's incoming network communication channel and retrieval of any messages that have arrived since the last periodic invocation. Each message is parsed at the KQML level and any contained information request is stored as a planning goal.[4]

This basic behavior is necessary for guaranteeing that the agent keeps its (implicit) commitments. By advertising its infobase schema, an information agent makes an implicit commitment to accept queries on the infobase from other agents. The message polling behavior makes sure that any communications that contain such requests actually cause the creation of an appropriate goal.

Answering Simple Queries

A simple query is one that is simply applied to the agent's infobase 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.

[4]Unsupported or illegal communications result in a KQML ERROR message to be sent to the SENDER.

Running a query consists of passing the query to the function that accesses the external information source. When the external interface process returns, the local infobase is assumed to be updated to contain any records that have been returned by the query. Next, any infobase triggering conditions are checked (see below). The query is then run on the local infobase, 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 according to the output specification in the :CONTENT part of the original query, and then sends out the properly formatted KQML *reply* messages.

Information Monitoring

An information 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 an information 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 infobase is updated (i.e., whenever any run-query is executed).

Cloning

Cloning is one of an information agent's possible responses to overloaded conditions. An information agent could recognize via self-reflection that it is becoming overloaded. To do this, it uses a simple model of how its ability to meet new deadlines is related to the characteristics of its current queries and other tasks. It compares this model to a hypothetical situation that describes the effect of adding a new agent. If this evaluation suggests that adding a new agent would be beneficial, the agent removes itself (temporarily) from actively pursuing new

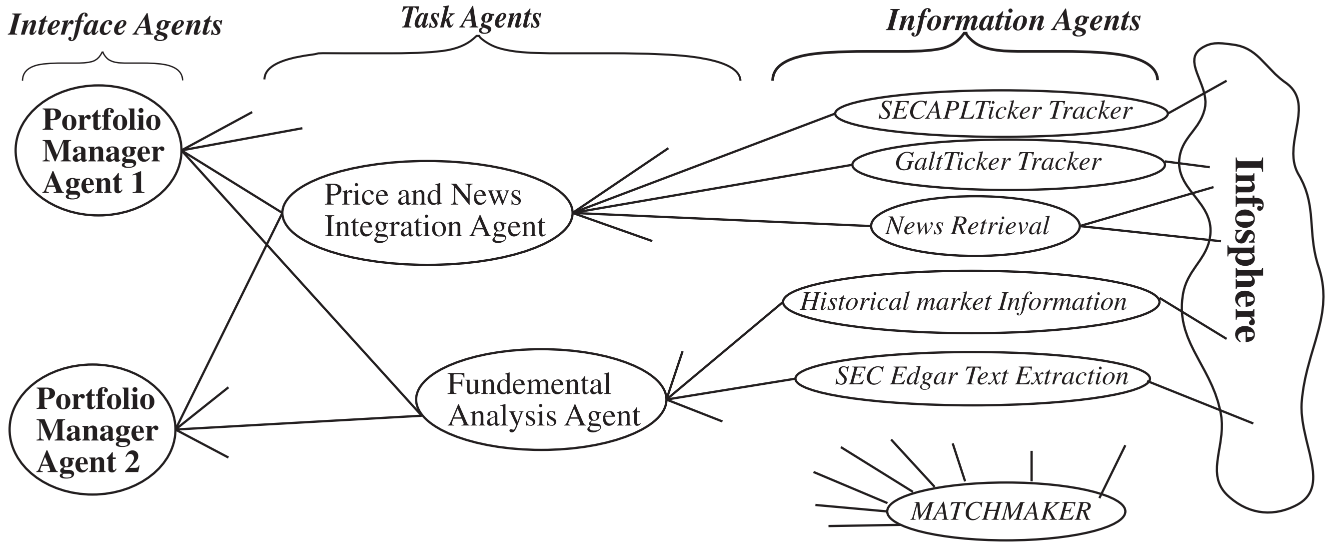

Figure 4: The WARREN portfolio management system.

queries (by "unadvertising" its services to the matchmaker) and creates a new information agent that is a clone of itself. In this way, the information agent can make a rational meta-control decision about whether or not it should undertake a cloning behavior. A formal presentation of the cloning criteria and the results of empirical evaluation can be found in (Decker, Williamson, & Sycara 1996a).

Multi-Source Information Agents

Basic information agents are certainly useful, but there is a great deal of reusable behavior that is involved with manipulating queries themselves, independent of the ultimate sources of data. We are currently constructing a new class of *multi-source information agents* by both adding new behaviors, e.g. for dealing with conflict resolution, and extending the existing behaviors (i.e. adding new task reductions allowing the agent to achieve the same objective in different ways). A query brokering agent is an example of such a multi-source information agent. A query brokering agent takes a request from a task agent and farms it out to one or more basic information agents, insulating the requesting task agent from certain types of source errors and efficiently balancing the computational load (Decker, Williamson, & Sycara 1996b). A newly instantiated query broker needs only to be provided the names of the basic information agents for which it is responsible. It can then contact the appropriate basic information agents, have them unadvertise with the matchmaker, advertise itself to the matchmaker, and begin providing query brokering services.

These types of new behaviors can be achieved by supplying new reductions to the existing query answering behaviors. More extensive multi-source integration (for example, breaking up queries, making multiple requests, and integrating the results (Knoblock, Arens, & Hsu 1994)) might require not only new reductions, but additional new behaviors.

WARREN

WARREN, our multi-agent portfolio management system, currently consists of a matchmaker, two task agents, 6 information agents and one interface agent, the Portfolio agent, for each user (see Figure 4.) The Portfolio agent (via a WWW interface) displays the standard information about a user's portfolio, allows the user to (simulate) buying and selling shares, and displays recent pricing and news information. It also provides access to the reports produced by the two task agents, either continuously updated or on demand. The information agents are: two stock ticker agents using different WWW sources, a news agent for Clarinet and Dow-Jones news articles, an agent that can extract current and historical sales and earnings-per-share data from the SEC Edgar electronic annual report information source, and another for certain textual portions of annual reports. The two task agents provide:

1. a graphical integration of stock prices and news stories about a single stock over time on a WWW page—the body of the news stories can be accessed by hyperlinks, and the information is stored persistently so that that it survives the task agent going off-line
2. a simple fundamental analysis of a stock with respect to its historical sales and earnings data, along with an accompanying graph.

Our experimental evaluations have included studies of both cloning behaviors (Decker, Williamson, & Sycara 1996a), and matchmaking vs. brokering behaviors (Decker, Williamson, & Sycara 1996b). More information and demos can be found on the WWW at `http://www.cs.cmu.edu/~softagents /warren/warren.html`.

Conclusions & Future Work

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, currently implemented 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. Such an architecture can support a full range of sophisticated autonomous agent behaviors.

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, and are fixed for a class of agents. An important goal of this work was to develop such a set of behaviors so that information agents for new information sources and domains 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 agent service advertising, polling for messages, answering one-shot queries, reporting the results of a repeated query, monitoring an information source for changes or new information patterns, and self-cloning.

The most important reason for our approach is that behavior specification is the proper level for allowing people to construct new classes of software agents in a structured, well-defined way. We are currently working on an *Agent Behavior Editor* which will allow more rapid construction of new classes of agents through the reuse and combination of existing behaviors, specification of new behaviors, and new task reductions. Other current work includes a release of our information agent construction kit so that anyone may advertise and provide data over the WWW using the above behaviors. We are currently formalizing and documenting our advertisement ontology (and expanding it to non-information agent services), as well as adding new agents to WARREN.

References

Arens, Y., and Knoblock, C. 1994. Intelligent caching: Selecting, representing, and reusing data in an information server. In *Proc. 3rd Intl. Conf. on Information and Knowledge Management*.

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

Collet, C.; Huhns, M.; and Shen, W. 1991. Resource integration using a large knowledge base in Carnot. *Computer* 55–62.

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.; Sycara, K.; and Zeng, D. 1996. Designing a multi-agent portfolio management system. In *Proceedings of the AAAI Workshop on Internet Information Systems*.

Decker, K.; Williamson, M.; and Sycara, K. 1996a. Intelligent adaptive information agents. In *Proceedings of the AAAI-96 Workshop on Intelligent Adaptive Agents*. AAAI Press Tech Report WS-96-04.

Decker, K.; Williamson, M.; and Sycara, K. 1996b. Matchmaking and brokering [poster]. In *Proceedings of the 2nd Intl. Conf. on Multi-Agent Systems*.

Etzioni, O.; Lesh, N.; and Segal, R. 1992. Building softbots for unix (preliminary report). Technical Report softbots-tr.ps, University of Washington.

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.

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

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

Kim, W., and Seo, J. 1991. Classifying schematic and data heterogeneity in multidatabase systems. *Computer* 12–18.

Knoblock, C.; Arens, Y.; and Hsu, C. 1994. Cooperating agents for information retrieval. In *Proc. 2nd Intl. Conf. on Cooperative Information Systems*. Univ. of Toronto Press.

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.

Oates, T.; Prasad, M. V. N.; Lesser, V. R.; and Decker, K. S. 1995. A distributed problem solving approach to cooperative information gathering. In *AAAI Spring Symposium on Information Gathering in Distributed Environments*.

Rao, A., and Georgeff, M. 1995. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*, 312–319. San Francisco: AAAI Press.

Sycara, K., and Zeng, D. 1996. Coordination of multiple intelligent software agents. *International Journal of Intelligent and Cooperative Information Systems*.

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*.