In-Context Information Management through Adaptive Collaboration of Intelligent Agents*

Katia Sycara
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA. 15213
katia@cs.cmu.edu

November 24, 1998

Abstract

Although the number and availability of electronic information sources are increasing, current information technology requires manual manipulation and user-specification of all details. Once accessed, information must be filtered in the context of the user's task. Current systems lack the ability to get contextual information or use it to automate filtering. At Carnegie Mellon University, we have been engaged in the RETSINA project, which aims to develop a reusable multiagent software infrastructure that allows heterogeneous agents on the Internet, possibly developed by different designers, to collaborate with each other to manage information in the context of user-specified tasks. In this chapter, we will provide a brief overview of the whole system and then focus on its capability for in-context information management.

*This research has been supported in part by DARPA contract F30602-98-2-0138, and by ONR Grant N00014-96-1222.

1 Introduction

The Web is full of information resources. This abundance of information holds great potential for being brought to bear at the right time on decision making tasks. Current practice in automated finding and filtering of information from the Web, however, is far from helping accomplish this goal. The context of the task for which the information is needed remains in the user's head; moreover, the user must spend much time manually searching for relevant information. Task context is currently expressed only through keywords that are manually submitted to search engines. Keywords are a very impoverished expression of task context making it difficult for the user to accurately express requirements for information relevant to the task under consideration. Currently, there are no formal languages that allow representation of task context to guide information retrieval. Search engines don't adapt their search strategies according to different users. Moreover, the problem is exacerbated because the information sources have high "noise", i.e. most of the pages are irrelevant to the interests of a particular user. Research on intelligent software agents is under way to address these issues. Intelligent agents are programs that act on behalf of their human users to perform laborious information-gathering tasks [31] and they are one of the "hot" topics in Information Systems R&D at the moment. The last ten years have seen a marked interest in agent-oriented technology, spanning applications as diverse as information retrieval, user interface design and network management.

We have been developing the RETSINA multiagent infrastructure to aid users in decision making and information management (information gathering, filtering, integration) tasks [31]. Users can delegate tasks to RETSINA agents who coordinate with each other to fulfill the delegated tasks. The RETSINA multiagent infrastructure provides a reusable framework for structuring agents that operate in an open Information environment (e.g. the Internet), where they form adaptive teams on demand in order to solve decision making and information management tasks delegated by users. RETSINA agents provide capabilities for in-context information gathering and filtering at different levels of automation and sophistication.

The simplest model of task context for information retrieval is expressed by a set of user-supplied keywords. In current practice, these keywords are supplied by the user to search engines that return a set of documents that are indexed by the provided keywords. Unfortunately, when this simple keyword-based context is used, current search engines return far too many documents to be useful. Meta search engines (e.g. MetaCrawler) try to improve the situation by taking the intersection of documents returned by each engine using a given set of keywords. Document intersection increases search relevance somewhat but does not solve the problem by any means. Besides these two very simple in-context information retrieval strategies, RETSINA agents provide more sophisticated ways to define the task context for information retrieval. We briefly mention these mechanisms in the rest of this introduction and present them in more detail in the rest of this chapter.

Instead of a user trying for each task to use ad hoc keywords for information retrieval, a more robust sense of context can be established by automatically supplying refinement keywords that can constrain the search, thus hopefully increasing the relevance of the returned information. RETSINA agents use two methods to automatically supply refinement keywords to establish such a context: using the trigger pair model and document similarity assessment based on relevance feedback.

An even more reliable information context that RETSINA agents provide can be established as an agent monitors user's actions and learns and tracks user interests as the user executes manual information searches. Then the learned user profile can serve as the information context. This type of context has the advantage that it has been learned from repeated information search trials and by utilizing user feedback. It still suffers from the fact that the task for which the information gathering was being performed is still implicit in the user's head.

RETSINA supports explicit task and goal representation and delegation to agents. A user can specify goals to be solved for. Agents form teams on demand to cooperatively form task-based plans to fulfill the delegated goals. The planning takes place in an open information environment where information sources, agents or communication links can change, appear or disappear dynamically. Hence information access and monitoring is an integral part of the planning process. Thus, the context of information gathering is the current task that an agent is planning for on the user's behalf. We present the computational mechanisms that allow an agent or a set of agents to cooperatively plan to fulfill user-delegated goals and tasks. In addition, we provide tools that allow a user to specify and delegate tasks to RETSINA agents. This is accomplished through two main tools, the Task Editor and the Agent Editor. The Task Editor allows explicit representation of tasks in hierarchical task networks. Task networks represent plan skeletons and are stored in an Editor task library. Through interactions with the usser, task networks can be retrieved and instantiated or combined with newly provided task fragments to represent new tasks. Once the task networks for a new agent have been specified, the Agent Editor instantiates the new agent automatically.

Providing an explicit computational formalism to express task context is extremely important since it (a) guides information searches so that only relevant task related information is returned, and (b) saves the user great amounts of manual effort.

The paper is organized as follows. Section 2 presents an overview of the RETSINA infrastructure. Section 2.1 presents the basic architecture of a RETSINA agent that, as is discussed in subsequent sections, supports the explicit task context representation. Section 3 presents the methods that an interface RETSINA agent uses for automatically supplying keywords to refine user-supplied information management context. User interests and information profile could be a more reliable context for information gathering and filtering. Implemented methods for learning and keeping track of multiple user interests are presented in Section 4. Section 5 presents how RETSINA agents represent task context

explicitly and how the context is utilized in collaborative information gathering tasks to support user goals. Finally, 6 presents a summary of the chapter and future work.

2 Brief Overview of RETSINA

Our work on the computational framework of intelligent agents, has been motivated by a number of considerations.

- *Distributed information sources:* Information sources available on-line are inherently distributed.
- *Sharability:* Typically, user applications need to access several services or resources in an asynchronous manner in support of a variety of tasks. It is desirable that the architecture support sharability of agent capabilities and retrieved information.
- *Complexity hiding:* Often information retrieval in support of a task involves quite complex coordination of many different agents. To avoid overloading users with a confusing array of different agents and agent interfaces, it is necessary to develop an architecture that hides the underlying distributed problem solving complexity. Complexity hiding, while alleviating cognitive overload, should not come at the price of making agents' expertise opaque. To alleviate the agent opacity problem, we have committed to explicit knowledge representation of agent behaviors, and explanation capabilities.
- *Modularity and Reusability:* One of the basic ideas behind our distributed agent-based approach is that agents should be kept simple for ease of maintenance, initialization and customization. Another facet of reusability is that it should be relatively straightforward to incorporate access to pre-existing information services.
- *Flexibility:* Intelligent agents should be able to interact in new configurations "on-demand", depending on the requirements of a particular decision making task. This includes the capability to migrate to other platforms to take advantage of locally accessible resources.
- *Robustness:* When information and control is distributed, the system should be able to degrade gracefully even when some of the agents, the information sources or the communication links are out of service temporarily.

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 distributed agent-based architecture has three types of agents (see Figure 1): *interface* agents, *task* agents and *information* agents. This is a simplification of the possible types of agent categories both for purposes of exposition and purposes of determining in a principled manner particular agent reusable behaviors that can serve as agent software building blocks. These three broad agent categories share common architectural components (section 2.1) but have different sets of agent behaviors. The behaviors in each class of agent are reusable.

Interface agents interact with the user receiving user specifications and delivering results. They acquire, model and utilize user preferences to guide system coordination in support of the user's tasks. For example, an agent that filters electronic mail according to its user's preferences is an interface agent. The main functions of an interface agent include: (1) collecting relevant information from the user to initiate a task, (2) presenting relevant information including results and explanations, (3) asking the user for additional information during problem solving, and (4) asking for user confirmation, when necessary. Interacting only through a relevant interface agent for a task hides the underlying distributed information gathering and problem solving complexity. For example, in our WARREN system[32] for financial portfolio management, more than 10 agents are involved. However, the user interacts directly only with the portfolio management interface agent.

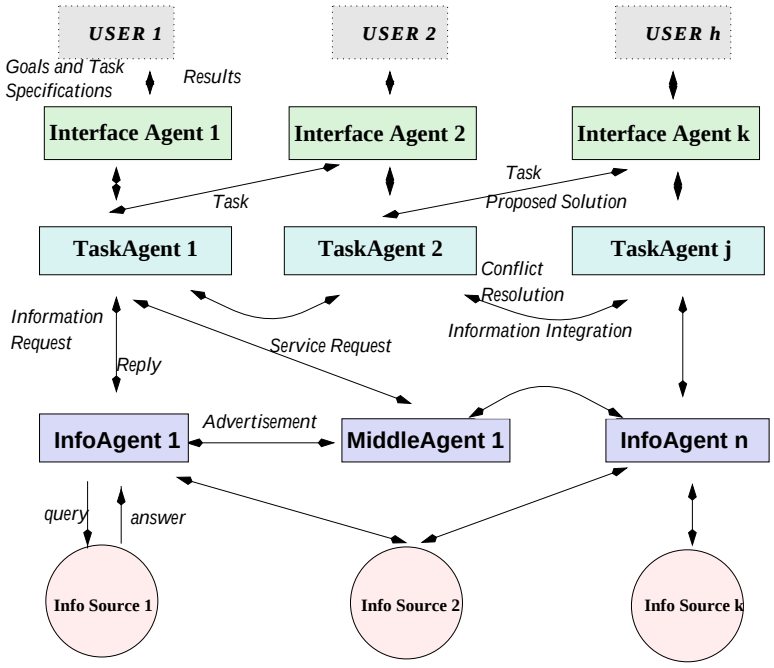

Figure 1: Distributed System Architecture

Task agents support decision making by formulating problem solving plans and carrying them out through querying and exchanging information with other software agents. Task

agents have knowledge of the task domain, and which other task agents or information agents are relevant to performing various parts of the task. In addition, task agents have strategies for resolving conflicts and fusing information retrieved by information agents. For example, an agent that makes stock buy or sell recommendations is a task agent. A task agent (1) receives user delegated task specifications from an interface agent, (2) interprets the specifications and extracts problem solving goals, (3) forms plans to satisfy these goals, (4) identifies information seeking subgoals that are present in its plans, (5) decomposes the plans and coordinates with appropriate task agents or information agents for plan execution, monitoring and results composition. This type of intelligent agent differs from traditional AI systems since information-seeking and communication during problem solving is an inherently built-in part of the system. This enables it to deal with open world environments (new information from the environment is incorporated during the agent's problem solving). Planning and execution are interleaved since retrieved information may change the planner's view of the outside world or alter the planner's inner belief system.

The main function of an information agent is to process intelligently and efficiently information retrieval and information monitoring requests. These requests come externally from other agents; the information used to fulfill these requests comes from arbitrary external information sources. Typically, a single information agent will serve the information needs of many humans or machine 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.

In order to allow agents to find others in an open environment such as the Internet, we have developed a set of agents, called *middle agents*. Middle agents receive advertisements of agent capabilities and store these advertisements in an internal data base. When an agent (a requester or consumer agent) would like to find a service provider agent that possesses certain desired capabilities, it sends a request to a middle agent. The middle agent matches the request to its data base of received advertisements to determine whether an agent whose capabilities math the request is known. We have identified different types of middle agents (e.g. matchmakers, brokers) and have reported experimental results that show different performance tradeoffs of these agents [22]. In addition, we have developed the language LARKS (Language for Advertisement and Request for Knowledge Sharing)[30].

The presence of the middle agents in the RETSINA framework allows adaptive agent organization. The high level goals and tasks imparted by the user form the context within which agents can adaptively (with the help of middle agents) form teams/coalitions so that this collaboration will fulfill the goals and tasks. This adaptive collaboration is also supported by the internal architecture of RETSINA agents (see section ?refsec:agent-arch). Overall system robustness is also facilitated through the use of middle agents. Agents can have replicated capabilities. For example in the Warren financial portfolio management application[32], there are different information agents that can find stock quotes from the

Web (e.g. Security APL, Gault). If a particular service provider disappears, a requester agent can find another one with same/similar capabilities by interrogating appropriate middle agents.

When an agent is created, it advertises itself to some entity such as a matchmaker or broker [26, 25, 28, 23]. This advertisement, expressed in terms of an agent's information base schema (section 5.4), 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. In addition, an 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.

The agents in our system communicate using KQML [25]. 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. 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 [24].

2.1 Overview of Single Agent Architecture

Currently the RETSINA framework provides an abstract basic agent architecture consisting of and integrating reusable modules, as depicted in Figure 2.1, and a concrete implementation of each module. Each agent consists of multithreaded Java code, and each module of an agent operates asynchronously. This enables the agent to be responsive to changing requirements and events. Each RETSINA agent consists of the following reusable modules:

- *communication and coordination* module that accepts and interprets messages from other agents, and sends replies.
- *planner* module that produces a plan that satisfies the agent's goals and tasks.
- *scheduler* module that schedules plan actions.
- *execution monitoring* module that initiates and monitors action execution.

The modules use well defined interfaces to interact with data structures that are used for keeping process information and for control purposes. In addition, each agent has the following knowledge components:

- *a set of domain independent and domain dependent plan fragments (Task Schemas)* indexed by goals. These plan fragments are retrieved and incrementally instantiated according to the current input parameters.
- *Belief Database* that contains facts, constraints and other knowledge reflecting the agent's current model of the environment and the state of execution.
- *Schedule* that depicts the sequence of actions that have been scheduled for execution.

We have provided an implementation of each of these modules and the associated data structures. Since we characterize agents from the structural point of view we are not committed to particular implementations of the planner, scheduler or any other module. The interfaces between the modules and the control flow paths are well defined. Different modules for planning or scheduling, for example can be "plugged-in" as long as they provide specific interface functionality. For a more extensive description of the agent architecture, see [31]. Depending on the agent type, there could be additional modules present. For example, information agents contain in addition, a local information database.

3 Automated Information Context Refinement by an Interface Agent

Single keywords are usually ambiguous, or too general. Moreover, they can occur in vast quantities of documents, thus making the search return hundreds of hits, most of which are irrelevant to the intended user query. The single keywords are not a very useful context for information retrieval. Giving additional keywords can refine the context and constrain the search providing considerable improvement in the retrieval results. Good refinement words must have meanings that help disambiguate or make more specific the original search word. For example, the word "stock" has more than 10 definition in the WordNet[1] including "the capital raised by a corporation through the issue of shares entitling holders to partial ownership", "gun-stock", "inventory", "stock certificate", etc. Providing the refinement words that correspond to each one of those meanings, would help a search engine, for example, to prune out documents where the word is used with any of its other meanings. There are three ways to expand the query: manual query expansion, semi-manual query expansion, and automatic query expansion [12]. No matter which method is used, the key point is to get the best refinement words. In manual query expansion, although the user

[1] http://www.cogsci.princeton.edu/~wn/

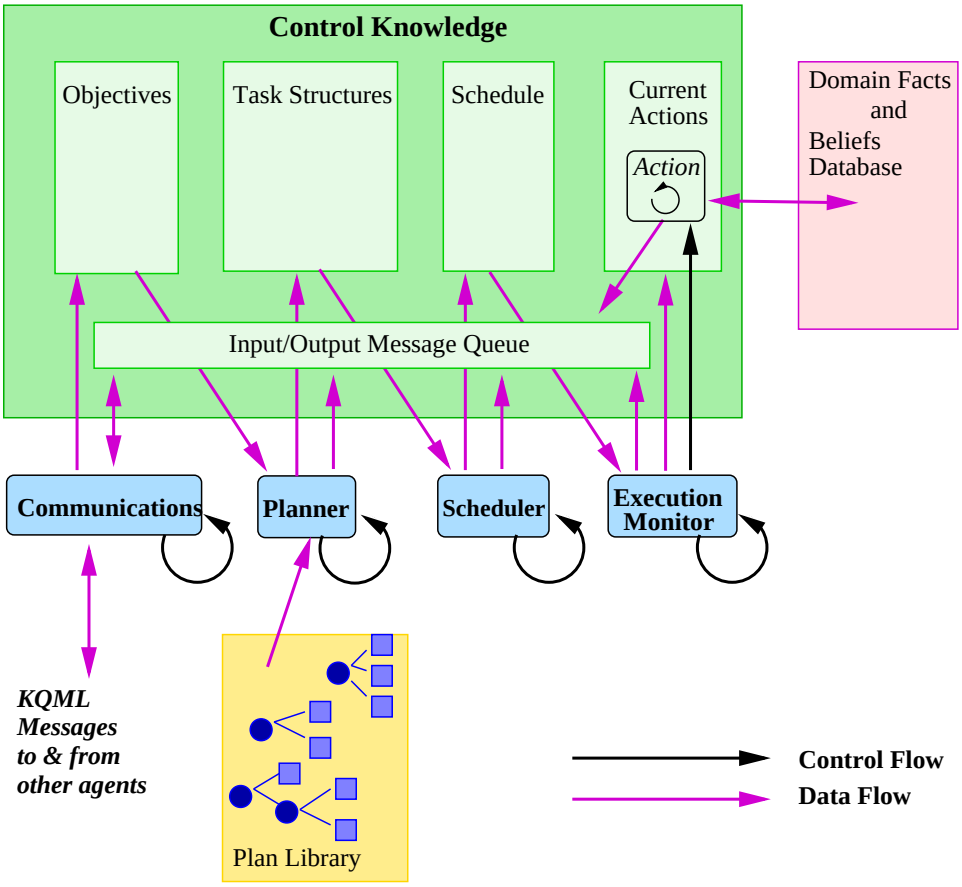

Figure 2: The RETSINA Basic Agent Architecture

knows the intended meaning of the keyword she is using, she may not be able to provide the best refinement words. "Best" here means refinement words that most frequently co-occur with the word in its intended meaning in large number of documents. In other words, one of the characteristics of good refinement words is that they be domain specific. In this section we present methods for automatically finding appropriate keywords to constrain and refine search for relevant documents. These methods are: (a) using trigger pairs for automated keyword expansion, and (b) relevance feedback. The methods have been implemented in WebMate[2], an interface agent in the RETSINA system.

WebMate is composed of a stand-alone proxy that can monitor a user's actions to provide information for learning and search refinement, and an applet controller that interacts with a user. The stand-alone proxy is an HTTP proxy that sits between a user's web browser and the World-Wide Web. All HTTP transactions pass through WebMate which can monitor a user's browsing and searching activities and learn from them. The applet controller is the interface between the user and the stand-alone proxy. Through it, the user can express his interests when he browses and provide relevance feedback when he searches. In addition, through the applet controller, the user receives intelligent help from WebMate.

3.1 Automatic Keyword Refinement

WebMate uses the Trigger Pairs model to automatically generate refinement words. The Trigger Pairs Model [13, 14] is as follows:. If a word S is significantly correlated with another word T, then (S, T) is considered a "trigger pair", with S being the trigger and T the triggered word. When S occurs in the document, it triggers T, causing its probability estimate to change. That is, when we see the word S appearing at some point in a text, we expect the word T to appear somewhere after S with some confidence[3]. The mutual information (MI) that considers the word order is a measure of the correlation and used to extract trigger pairs from large corpus. The mutual information is given by the following formula:

$$\mathcal{MI}(s,t) = \mathcal{P}(s,t) \log \frac{\mathcal{P}(s,t)}{\mathcal{P}(s)\mathcal{P}(t)}$$

By experimenting with language corpora of varying domain specificity, we found that trigger pairs are domain specific. For example, the triggers to "Stock" in news and media domain (Broadcast News Corpus, 140M words) are {company, bond, buy, business, bank, dow, earning, composite, cent, analyst, big, chrysler, investor, cash, average, economy, close, capital, chip, ...}. However, in business and Economic (Wall Street Journal Corpus,

[2] Work on WebMate is joint with Liren Chen.

[3] In the Trigger Pairs Model, (S,T) is different from (T,S), so the Trigger Pairs Model is different from the method of using co-occurrence of two words that is generally used in other keywords expansion experiments[12]

1M words) the triggers are {share, investor, index, exchange, price, dow, market, buy, point, jone, trade, trader, average, cent, industrial, gain, shareholder, company, board, ...}

The trigger pair method can provide several candidate refinement keywords. An additional question is, how many and which ones to use under any given circumstances. For a search with only one keyword, the top several triggers to the keyword are used to expand the search. But for a search with more than 2 keywords, the choice becomes more complicated. Let us assume that the keywords are $K_1, K_2, \ldots, K_m$, and the expected number of refinement words is N. We use an algorithm that involves finding the trigger pairs of all keywords with the highest mutual information and take subsets of their intersections. The detailed algorithm is reported in [21].

Besides allowing automated selection of refinement keywords to constrain search, this method also provides disambiguation information for ambiguous query words. We present in some deail a typical example of how our refinement method indeed helps improve retrieval results. Suppose the user is interested in documents where the word "stock" appears in its financial meaning. Inputting simply the keyword "stock" to Lycos and Altavista returns the following results.

From Lycos:

1) YOSEMITE STOCK PHOTOS, ROCK CLIMBING, Daniela Masetti PHOTOS

2) YOSEMITE STOCK PHOTOS, ROCK CLIMBING PHOTOS

3) YOSEMITE STOCK PHOTOS, FISHING PHOTO

*4) Stock information Java Applet

5) STOCK GRAPHICS & PHOTOS

*6) American Stock Transfer & Trust Home Page

*7) STOCK CHARTS

*8) GROWTH STOCK ADVISOR FULL DISCLAIMER

*9) Stock information Java Applet

10) Ocean Stock

Only 5 hits are relevant to the financial meaning of "stock" in the top 10.

From Altavista:

1. E. coli Genetic Stock Center

2. Michael Paras Photography: Photographs, Photography, stock photos,stock photo

*3. iGOLF Features - Stocks & Industry - Stock Report: Tuesday,September 5, 1995

4. Cedar Stock Resort Trinity Center Marina

*5. Stock 4 Art: HOME PAGE!

6. NET INFO - Luc Sala - Myster - stock footage

*7. The Official Vancouver Stock Exchange

*8. Stock Club

*9. NIAGARA MOHAWK DECLARES PREFERRED STOCK DIVIDEND

*10. The Italian Stock Exchange

There are 6 hits that are relevant to the financial meaning of the "stock" in the top 10. At this point, although the user may not be satisfied with the returned search results, it is difficult for him/her to figure out what words should be used to refine the search. So the trigger pairs can be used to expand the current search. The triggers to "stock" are {share, investor, index, exchange, price, dow, market, buy, point, jone, trade, trader, average, cent, industrial, gain, shareholder, company, board, ...}. If we use the first word "share" in the ranked triggers list to expand the keyword "stock" and send {stock share} to the above two search engines, all the top 10 hits returned are relevant to the financial meaning of "stock". We can see that these results are better than before. We can also refine the search "stock share" if the results are still not satisfactory. The intersection of the triggers sets of "stock" and "share" can be used for such search refinement.

3.2 Relevance Feedback

One of the most important ways in which current information retrieval technology supports refining searches is relevance feedback. Relevance feedback is a process where users identify relevant documents in an initial list of retrieved documents, and the system then creates a new query based on those sample relevant documents [14]. The idea is that since the newly formed query is based on documents that are similar to the desired relevant documents, the returned documents will indeed be similar. The central problems in relevance feedback are selecting "features" (words, phrases) from relevant documents and calculating weights for these features in the context of a new query [8].

Given a relevant page, WebMate first looks for the keywords (assume K_i is one of the keywords) and context of the keywords. The context is composed of the words that come before or after the given keyword. For example, the 5-context of the keyword K_i is $\ldots W_{-5}W_{-4}W_{-3}W_{-2}W_{-1}K_iW_1W_2W_3W_4W_5 \ldots$. For each keyword $K_(i)$, the system then extracts the chunks of 5 words $W_{-5}W_{-4}W_{-3}W_{-2}W_{-1}$ before K_i and the chunks of 5 words $W_1W_2W_3W_4W_5$ after K_i until all the keywords in the query are processed. Then, a bag of chunks are collected and passed to the processes of deleting the stop words and calculating the frequency. After that, the top several frequent words are used to expand the current search keywords.

For example, the following text is part of the overview of our Intelligent Agents project at CMU[4]. Suppose a user gives this text as a relevance feedback to the search keywords "intelligent agent".

[4]The URL of our project is: http//www.cs.cmu.edu/~softagents.

Intelligent Software Agents

The voluminous and readily available information on the Internet has given rise to exploration of Intelligent Agent technology for accessing, filtering, evaluating and integrating information.

In contrast to most current research that has investigated single-agent approaches, we are developing a collection of multiple agents that team up on demand—depending on the user, task, and situation—to access, filter and integrate information in support of user tasks. We are investigating techniques for developing distributed adaptive collections of information agents that coordinate to retrieve, filter and fuse information relevant to the user, task and situation, as well as anticipate user's information needs.

Approach is based on:

adaptable user and task models

flexible organizational structuring

a reusable agent architecture

Underlying Technology

Our intra-agent architecture and inter-agent organization is based on the RETSINA multiagent reusable infrastructure that we are developing.

Using our method, the refinement words extracted from the text are {software, structure, reusable, architecture, technology, organizational, network, schedule, research, rise}. Most of the refinement words reflect the characteristic of the project well. But, if instead of using the context method, we considered the whole content of the page when calculating the frequency, then the expanding words would be {software, information, task, area, application, technology, user, current, develop, underlying}. Obviously, the context of the search keywords can reflect the relevance better than the whole content of the web page. Subsequently, we used the top 5 words {software structure reusable architecture technology} to expand the search "intelligent agent". Of the top 10 results returned by Lycos, 7 were relevant to the query.

4 Learning of Information Retrieval Context

The automatically learned and continuously updated user profile can possibly serve as a more reliable indicator of information retrieval context than user-supplied keywords. There are several machine learning approaches that can be used to learn a user profile, such as Bayesian classifier, Nearest Neighbor, PEBLS, Decision Trees, TF-IDF, Neural Nets [4, 5]. In order for a particular technique to be effective, it should match the characteristics of the task and the user.

The filtering task for our agent involves judging whether an article is relevant or irrelevant to the user based on the user profile, in an environment where the prior probability of

encountering a relevant document is very low compared to the probability of encountering an irrelevant document. In such an environment, it would be very frustrating and time consuming for a user to interact with an agent that starts with no knowledge but must obtain a set of positive and negative examples from user feedback. When a user browses, he does not want to evaluate all web pages that might contain potentially interesting information. To reduce user evaluation burden, WebMate collects only examples that are interesting to the user (only positive training examples). This kind of interaction presents potential problems since the documents that a user might label as "I like It" might fall into many distinct domains (e.g fishing, computer science, soccer). Those subclasses correspond to the different interests a user has. There have been two methods to address the problem of multiple user interests. The first is to keep a single user profile where the keywords might come from different domains but are "averaged'. This method has the disadvantage that averaging the vectors from the different documents might decrease too much the weights of words that are important for only a few of the interest categories. The second method is to ask the user to explicitly provide labels for the sub-categories of interest. Instead, WebMate learns the categories automatically.

WebMate utilizes the TF-IDF method [7] with multiple vectors representation. We have developed an algorithm for *multi TF-IDF* vector learning that we summarize here. For more details, see [21]. Let N be the assumed number of user interests. For each of the first N documents that the user marks "I like it", extract the TF-IDF vector and place it in a different "interest profile bucket". For each subsequent document, calculate the cosine similarity between every two TF-IDF vectors (including the vectors already in the profile and the new vector) so that the old vectors and the new vector can be placed in the appropriate "interests profile bucket". This algorithm is run whenever a user marks a document as "I like it". Thus, the user profile is incrementally, unobtrusively and continuously updated.

After a profile has been learned, it can be used for document filtering and anticipating user information requests. In particular, we have used the continuously updated user profile to compile a personal newspaper [9, 10, 11] and conducted informal experiments to determine the effectiveness of the method. In our experiments, the system monitors 14 news sites that contain articles about high technology including LAN time news[5], Media Central [6], PC magazine online [7], etc. Experimental results show that the average accuracy (relevance rate) that the recommended news is relevant to our interests is between 50% and 60% in the top 10 news articles . Generally the system will spide more than 500 pieces of news each day. In the whole recommended news, the average accuracy is about 30%. But if the news are randomly chosen from 500 pieces of news in which we assume there are 100 interesting news to us (this is based on our observation that for a typical news site

[5] http://www.lantimes.com/

[6] http://www.mediacentral.com/Magazines/MediaDaily/Archive

[7] http://www8.zdnet.com/pcmag/

such as LinkExchange, there are about 10 out of 50 pieces of news that are interesting to us in any given day), the default accuracy in the whole news is about 20%. So a 50% to 60% accuracy, achieved by WebMate, represents a two to three-fold accuracy increase.

5 Explicit Context Representation in Task Schemas

If agents have an explicit representation of the user's task that they can computationally reason about, then they could use this task representation as the information gathering and filtering context. Making use of this context would support the user's decision making more effectively. One wasy to explicitly specify the task context is to simply give an agent a set of goals it should fulfill. This assumes that the agent has the capability of producing plans to satisfy the goals. In addition, simply specifying high level goals might result in ambiguity and misunderstandings.

A more detailed specification includes, in addition to goals, a set of skeletal plan fragments that can be composed and instantiated to fulfill the given goals. In RETSINA, a task and its subtasks along with other useful computational parameters are represented using the formalism of Hierarchical Task Networks (HTNs)[33]. An HTN has nodes that denote abstract tasks, links between parent and children nodes that denote subtask relationships, and links between sibling nodes that denote precedence orderings. Leaf nodes represent *actions* that can be scheduled and executed. A task is *reduced* by instantiating a set of subtasks. Planning using a HTN formalism 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 reduction schema presents a way of carrying out a task by specifying a set of sub-tasks/actions. In addition, Task Reduction schemas specify preconditions and postconditions for task reductions, and different types of constraints (state, temporal and resource constraints). These prescribe task interdependencies and dictate static control flow. In addition to these static relations, there are also runtime relations describing the information flow between subtasks. 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). Runtime information flow is expressed by *provisions* whose variables are bound during runtime reflecting information coming in from the environment (e.g. sensors) or supplied by other agents.

Actions may require that certain information be provided before they can be executed, and may also produce information upon execution. Action execution produces an *outcome* to indicate different results of action execution (e.g. completion, different types of failures) and a *result* that expresses particular information about a returned outcome. The outcome is one of a finite set of predefined symbols. The results can be any arbitrary piece of information returned by the code object that implements the action. An example will make clear the intended difference between outcomes and results. Consider the action of

retrieving a web page. The outcomes of such an action might be OK or ERROR, depending on whether the action succeeds on fetching the page or not. If the outcome is OK, the result of the action would be the web page itself. If the outcome is an ERROR, the result might be some description of the error. Such outcomes of one subtask reduction can be propagated at runtime to match provisions of another subtask. 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. If the value(s) of the outcome(s) matches the value(s) of the provision, and if all the required inputs have been supplied, then the action is *enabled* and can be scheduled for execution. Final action selection, sequencing, and timing are left up to the agent's local scheduler.

These mechanisms support the representation and efficient execution of plans *with periodic actions, externally triggered actions and loops* [33]. Task schemas and alternative task reductions are blueprints for representing agent requirements, specification information and processing structure. This representation enables an agent to understand its requirements and reason about how its behavior can satisfy the requirements.

A task typically has alternative task reductions, that is alternative execution paths that can satisfy it depending on the parameters of the current situation. Thus, even simple tasks such as answering a query may result in very different sequences of actions (asking a middle agent, using an already known agent, using a cached previous answer).

5.1 Agent Advertisement Behavior

An 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. An agent's task reduction schemas are retrieved and incrementally instantiated by its planner module 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. For example, since the behavior of advertising its capabilities is shared by all RETSINA agents, each agent contains the same task structures and standard reductions for this behavior, shown in Figure 3.

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 Match-

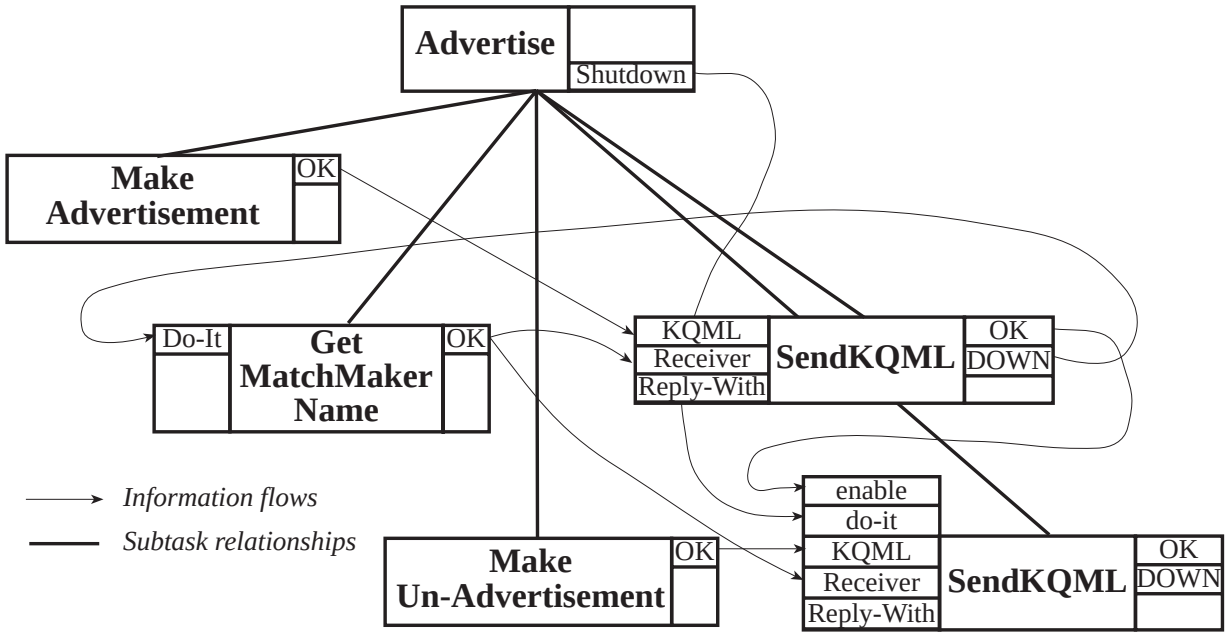

Figure 3: The standard task reduction for the "advertise" task.

maker Name" is to use a predefined name. [8] 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 signals a "DOWN" outcome, which provides a new signal to Get Matchmaker name, and the two tasks are rescheduled (they are periodic) and rerun.

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

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

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

Shutdown actions are placed on both the regular and a special shutdown scheduling queue. Both actions are non-periodic and have a deadline of "only execute if there's nothing better to do". Actions on the shutdown queue are architecturally guaranteed to execute at least once, so in particular, the "Make Un-Advertisement" action will either execute during

[8] 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.

normal processing when the agent would otherwise be idle, or during shutdown if the agent never had any spare time. The SendKQML that actually passes the advertisement retraction on to the matchmaker has two extra enablement conditions: first, that the initial advertisement actually completed without error, and second that the task is being shutdown. This basic advertising behavior is shared (reused) for all information agents (simple and multi-source).

5.2 Forms of Task Delegation

To facilitate the interactive specification of task structures and task reductions by human agent designers, we have implemented the Task Editor and the Agent Editor. The Task Editor provides a library of task schemas and task reduction schemas that are indexed by the goals they accomplish and can be automatically retrieved. A user may specify new schemas, and/or new reductions needed to define the behaviors of a new agent. The Task Editor possesses a Graphical User Interface that shows graphically the tasks and task reductions. In addition, it has facilities for checking the consistency of specifications. When a user is finished with specifying the tasks and reductions for a new agent, the Agent Editor is invoked. The Agent Editor automatically composes the specified task and reductions and instantiates the new agent. We are currently designing tools to increase the user-friendliness of the Task Editor so that it can be used as a task delegation mechanism by end users.

A task/sub-task schema includes notation as to whether the agent itself is capable of reducing the task schema and executing the leaf actions, or the agent must delegate this to another agent[9] that has the requisite capabilities. If, during planning, an agent finds out that it should delegate particular task reductions or actions to other agents, it automatically generates a message to be routed to the appropriate middle agent [10] requesting agents that have the capability required by the current (sub)-task reduction. In this way, task delegation among agents is dynamic, and takes into account the openness of the environment. Alternatively, for reasons of efficiency, an agent may have cached the names and locations of agents with whom it has collaborated in the past. In this case, the agent does not need to send requests to middle agents.

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. The flexible representation and incremental instantiation of task schemas and task reduction schemas and the presence of middle agents act synergistically to allow adaptive interleaving of information management (e.g. information gathering, filtering and integration) and execution in the context of the tasks that are being planned for. For example, at

[9] This operationalizes the *intent-to* and *intent-that* predicates of [27].

[10] In the current implementation of RETSINA, such middle agents are matchmakers.

any time, any agent in the system might be unavailable or might go off-line. The agent's task reductions and execution monitoring ability handle these situations so that such failures are dealt with smoothly. If alternate agents are available, they will be contacted and the subproblem restarted. If no alternate agent is available, the agent will replan. Such replanning may result in choosing alternative task reductions that mey require a different set of service provider agents.

In the next section, we take a more detailed look at how information agents are specified and how they operate.

5.3 Functional Overview of Information Agents

Typically, a single information agent will serve the information needs of many other agents (humans or intelligent software agents). An information agent can reason about the way it will handle external requests and adapt to different environmental events. Moreover, information agents not only perform information gathering in response to queries but also can carry out long-term interactions that involve *monitoring* the Infosphere for particular conditions, as well as information updating. The RETSINA agents communicate through message passing using the KQML [25] communication language.

In particular, we will focus on the behaviors of *basic information 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.

The dominant domain level behaviors of an information agent are: retrieving information from external information sources in response to one shot queries (e.g. "retrieve the current price of IBM stock"); requests for periodic information (e.g. "give me the price of IBM every 30 minutes"); monitoring external information sources for the occurrence of given information patterns, called change-monitoring requests, (e.g. "notify me when IBM's price increases by 10% over $80"). Information originates from external sources. Because an information agent does not have control over these external information sources, it must extract, possibly integrate, and store relevant pieces of information in a database local to the agent. The agent's information processing mechanisms then process the information in the local database to service information requests received from other agents or human users. Other simple behaviors that are used by all information agents include advertising their capabilities, managing and rebuilding the local database when necessary, and checking for KQML messages from other agents.

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 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 [29].

Since information agents share the basic agent architecture of a RETSINA agent, they too have a library of task schemas and task reduction schemas. These task structures allow information agents to plan about how to execute information 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 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 particular class of information agent. This includes the behaviors of advertising, listening for messages, and accepting, recording, running, and responding to queries.

5.4 Local Information Agent Infobase

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, The information agent's local infobase contains records that have been retrieved from external information sources in relation to one or more queries. This local infobase is important because it allows the information agent to become more than just a fancy wrapper. 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 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 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".

The local infobase is constructed from a data definition language description. This description also serves as a way to describe the services a particular information agent offers to other agents. Since agent advertisement is done through the local infobase schema, the infobase allows all information agents to present an internally consistent, domain-independent interface to other agents that specifies the capabilities of the agent, or alternatively, the services provided by the agent. In addition, the local infobase 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 [20].

5.5 Examples of Information Queries

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 query is expressed in the :CONTENT slot of the message. The communication module receives and parses messages, extracts the information goals and passes them to its planner module. 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. An example of a one-shot query, "give me the most recent GH record" is:

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

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). Everything else is shared and reused between information agent instances (and thus improvements and new functionality are shared as well).

6 Summary and Future Research

As the Web is used my increasing numbers of users for different tasks, there is a need for information query management (e.g. retrieval, filtering, integration) to take into consideration the user's current task. Such in-context information retrieval can increase relevance of retrieved information. Currently, the task is in the user's head, therefore information management is through user searching and browsing. Search engines have notoriously low accuracy in finding directly relevant task information. We have presented different techniques implemented in the RETSINA multiagent infrastructure for in-context information retrieval. Task context can be implicitly defined through a list of keywords directly supplied by the user, or augmented automatically through provision of refinement keywords by an

agent. A more robust form of information retrieval and filtering context consists of the user interests that can be automatically learned and tracked over time. Explicitly representing task context in computational forms and allowing agents to reason about it, originate and delegate information requests to others is the most advanced way of taking task context into consideration. In RETSINA, the representational vehicle is task structures in Hierarchical Task Networks. We have presented how the task structures can be used to represent task context and agent behaviors, and how information retrieval and filtering requests are accomplished though adaptive collaboration of agents. In future work we plan to rigorously evaluate these different techniques.

References

[1] Katia Sycara, Anandeep Pannu, Mike Williamson, Dajun Zeng, Keih Decker. *1996, Distributed Intelligent Agents.* Published in IEEE Expert, Intelligent Systems & their applications, Dec, 1996

[2] Shaw Green, Leon Hurst, Brenda Nangle, Padraig Cunninggham, Fergal Somers, Richard Evans. *1997, Software Agents: A review.* http://www.cs.tcd.id/Brenda.Nangle/iag.html

[3] Marko Balabanovic, Yoav Shaham. *1995, Learning Information Retrieval Agents: Experiments with Automated Web Browsing.* Proceedings of the AAAI Spring Symposium Series on Information Gathering from Heterogeneous, Distributed Environments: 13-18.

[4] M. Pazzani, J. Muramatsu, D. Billsus. *1996, Syskill & webert: Identifying interesting web sites.* In AAAI conference, Portland, 1996

[5] Pannu, A. and Sycara, K. *1996, A Personal Text Filtering Agent.* Proceedings of the AAAI Stanford Spring Symposium on Machine Learning and Information Access, Stanford, CA, March 25-27, 1996.

[6] K.Lang. *1995, NewsWeeder: Learning to filter Netnews.* Proceedings of Machine Learning, Morgan Kaufman, San Francisco, 1995

[7] G.Salton and M.J.McGill. *1983, Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983

[8] Gerard Salton, Chris Buckley. *1988, Improving Retrieval Performance by Relevance Feedback.* Cornell University, 88-898

[9] Marbo Balabonovic, Yoav Shoham. *1997, Combining Content-Based and Collaborative Recommendation.* Communications of the ACM, March, 1997

[10] Peter W. Foltz, Susan T. Dumais. *1992, Personalized Information Delivery: An Analysis of Information Filtering Methods.* Published in Comminucations of the ACM, 35(12), 51-60, 1992

[11] Paul Resnick. *1997, Filtering Information on the Internet.* http://www.sciam.com/0397issue/0397resnick.html

[12] Chengfeng Han, Hideo Fujii, W. Bruce Croft. *1994 Automatic Query Expansion for Japanese Text Retrieval.* UMass Technical Report

[13] Ronald Rosenfeld. *1994, Adaptive Statistical Language Modeling: A Maximum Entropy Approach.* Carnegie Mellon University, Ph.D. Thesis

[14] Susan Gauch, Robert P. Futrelle. *Experiments in Automatic Word Class and Word Sense Identification for Information Retrieval.* Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval

[15] Kathleen Webster, Kathryn Paul. *1996, Beyond Surfing: Tools and Techniques for Searching the Web.* Jan. 1996, Information Technology

[16] Thorsten Joachims, Dayne Freitag, Tom Mitchell. *1997, WebWatcher: A Tour Guide for the World Wide Web.* Proceedings of IJCAI97, August 1997.

[17] H.Lieberman. *1995, Letizia: An agent that assists web browsing.* In International Joint Conference of Artificial Intelligence, Montreal, Aug. 1995

[18] Bernardo A. Huberman, Michael Kaminsky. *1996, Beehive: A System for Cooperative Filtering and Sharing of Information*

[19] URL: *Collection of Bots in the Web*, http://www.botspot.com/

[20] Y. Arens, C. Y. Chee, C.-N. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):127–58, June 1993.

[21] Liren Chen and K. Sycara. Webmate : A personal agent for browsing and searching. In *Proceedings of the Second International Conference on Autonomous Agents (Agents 98)*, 1998.

[22] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of IJCAI-97*, 1997.

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

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

[25] 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.

[26] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7), July 1994.

[27] Barbara Grosz and Sarit Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86(2):269–3571, 1996.

[28] Daniel Kuokka and Larry Harada. Supporting information retrieval via matchmaking. In Craig Knoblock and Alon Levy, editors, *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, Stanford, CA, March 1995. AAAI.

[29] O. Shehory, K. Sycara, P. Chalasani, and S. Jha. Agent cloning. In *ICMAS-98*, Paris, France, July, 1998.

[30] K. Sycara, J. Lu, and M. Klusch. Interoperability among heterogeneous software agents on the internet. Technical Report CMU-RI-TR-98-22, Carnegie Mellon University, 1998.

[31] Katia Sycara, Keith Decker, Anandeep Pannu, Mike Williamson, and Dajun Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6), December 1996.

[32] Katia Sycara, Keith Decker, and Dajun Zeng. Intelligent agents in portfolio management. In N. Jennings and M. Woolridge, editors, *Agent Technology: Foundations, Applications, and Markets*, chapter 14, pages 267–283. Springer, 1998.

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